Network Working Group
Internet-Draft
Obsoletes: 5201 (if approved)
Intended status: Standards Track
Expires: January 1, 2014

R. Moskowitz, Ed.
Verizon
T. Heer
Hirschmann Automation and Control
P. Jokela
Ericsson Research NomadicLab
T. Henderson
The Boeing Company
June 30, 2013

Host Identity Protocol Version 2 (HIPv2)
draft-ietf-hip-rfc5201-bis-12

Abstract

This document specifies the details of the Host Identity Protocol (HIP). HIP allows consenting hosts to securely establish and maintain shared IP-layer state, allowing separation of the identifier and locator roles of IP addresses, thereby enabling continuity of communications across IP address changes. HIP is based on a SIGMA-compliant Diffie-Hellman key exchange, using public key identifiers from a new Host Identity namespace for mutual peer authentication. The protocol is designed to be resistant to denial-of-service (DoS) and man-in-the-middle (MitM) attacks. When used together with another suitable security protocol, such as the Encapsulated Security Payload (ESP), it provides integrity protection and optional encryption for upper-layer protocols, such as TCP and UDP.

This document obsoletes RFC 5201 and addresses the concerns raised by the IESG, particularly that of crypto agility. It also incorporates lessons learned from the implementations of RFC 5201.

Status of This Memo

This Internet-Draft will expire on January 1, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

This document specifies the details of the Host Identity Protocol (HIP). A high-level description of the protocol and the underlying architectural thinking is available in the separate HIP architecture description [I-D.ietf-hip-rfc4423-bis]. Briefly, the HIP architecture proposes an alternative to the dual use of IP addresses as "locators" (routing labels) and "identifiers" (endpoint, or host, identifiers). In HIP, public cryptographic keys, of a public/private key pair, are used as Host Identifiers, to which higher layer protocols are bound instead of an IP address. By using public keys (and their representations) as host identifiers, dynamic changes to IP address sets can be directly authenticated between hosts, and if desired, strong authentication between hosts at the TCP/IP stack level can be obtained.

This memo specifies the base HIP protocol ("base exchange") used between hosts to establish an IP-layer communications context, called a HIP association, prior to communications. It also defines a packet format and procedures for updating an active HIP association. Other elements of the HIP architecture are specified in other documents, such as:

- "Using the Encapsulating Security Payload (ESP) transport format with the Host Identity Protocol (HIP)" [I-D.ietf-hip-rfc5202-bis]: how to use the Encapsulating Security Payload (ESP) for integrity protection and optional encryption
- "Host Mobility with the Host Identity Protocol" [I-D.ietf-hip-rfc5206-bis]: how to support host mobility in HIP
- "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions" [I-D.ietf-hip-rfc5205-bis]: how to extend DNS to contain Host Identity information
- "Host Identity Protocol (HIP) Rendezvous Extension" [I-D.ietf-hip-rfc5204-bis]: using a rendezvous mechanism to contact mobile HIP hosts

Since the HIP Base Exchange was first developed, there have been a few advances in cryptography and attacks against cryptographic systems. As a result, all cryptographic protocols need to be agile. That is, it should be a part of the protocol to be able to switch from one cryptographic primitive to another. It is important to support a reasonable set of mainstream algorithms to cater for different use cases and allow moving away from algorithms that are later discovered to be vulnerable This update to the Base Exchange includes this needed cryptographic agility while addressing the

downgrade attacks that such flexibility introduces. In particular, Elliptic Curve support by Elliptic Curve DSA (ECDSA) and Elliptic Curve Diffie-Hellman (ECDH) and alternative hash functions have been added.

1.1. A New Namespace and Identifiers

The Host Identity Protocol introduces a new namespace, the Host Identity namespace. Some ramifications of this new namespace are explained in the HIP architecture description [I-D.ietf-hip-rfc4423-bis].

There are two main representations of the Host Identity, the full Host Identity (HI) and the Host Identity Tag (HIT). The HI is a public key and directly represents the Identity of a host. Since there are different public key algorithms that can be used with different key lengths, the HI, as such, is unsuitable for use as a packet identifier, or as an index into the various state-related implementation structures needed to support HIP. Consequently, a hash of the HI, the Host Identity Tag (HIT), is used as the operational representation. The HIT is 128 bits long and is used in the HIP headers and to index the corresponding state in the end hosts. The HIT has an important security property in that it is self-certifying (see Section 3).

1.2. The HIP Base Exchange (BEX)

The HIP base exchange is a two-party cryptographic protocol used to establish communications context between hosts. The base exchange is a SIGMA-compliant [KRA03] four-packet exchange. The first party is called the Initiator and the second party the Responder. The protocol exchanges Diffie-Hellman [DIF76] keys in the 2nd and 3rd packets, and authenticates the parties in the 3rd and 4th packets. The four-packet design helps to make HIP DoS resilient. It allows the Responder to stay stateless until the IP address and the cryptographic puzzle is verified. The Responder starts the puzzle exchange in the 2nd packet, with the Initiator completing it in the 3rd packet before the Responder stores any state from the exchange.

The exchange can use the Diffie-Hellman output to encrypt the Host Identity of the Initiator in the 3rd packet (although Aura, et al., [AUR03] notes that such operation may interfere with packet-inspecting middleboxes), or the Host Identity may instead be sent unencrypted. The Responder's Host Identity is not protected. It should be noted, however, that both the Initiator's and the Responder's HITs are transported as such (in cleartext) in the packets, allowing an eavesdropper with a priori knowledge about the parties to identify them by their HITs. Hence, encrypting the HI of

any party does not provide privacy against such attacker.

Data packets start to flow after the 4th packet. The 3rd and 4th HIP packets may carry a data payload in the future. However, the details of this may be defined later.

An existing HIP association can be updated using the update mechanism defined in this document, and when the association is no longer needed, it can be closed using the defined closing mechanism.

Finally, HIP is designed as an end-to-end authentication and key establishment protocol, to be used with Encapsulated Security Payload (ESP) [I-D.ietf-hip-rfc5202-bis] and other end-to-end security protocols. The base protocol does not cover all the fine-grained policy control found in Internet Key Exchange (IKE) [RFC4306] that allows IKE to support complex gateway policies. Thus, HIP is not a complete replacement for IKE.

1.3. Memo Structure

The rest of this memo is structured as follows. Section 2 defines the central keywords, notation, and terms used throughout the rest of the document. Section 3 defines the structure of the Host Identity and its various representations. Section 4 gives an overview of the HIP base exchange protocol. Sections 5 and 6 define the detailed packet formats and rules for packet processing. Finally, Sections 7, 8, and 9 discuss policy, security, and IANA considerations, respectively.

2. Terms and Definitions

2.1. Requirements Terminology

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

2.2. Notation

[x] indicates that x is optional.

{x} indicates that x is encrypted.

X(y) indicates that y is a parameter of X.

<x>i indicates that x exists i times.

--> signifies "Initiator to Responder" communication (requests).

<-- signifies "Responder to Initiator" communication (replies).

| signifies concatenation of information (e.g., X | Y is the concatenation of X with Y).

Ltrunc (H(x), K) denotes the lowest order #K bits of the result of the hash function H on the input x.

2.3. Definitions

HIP Base Exchange (BEX): the handshake for establishing a new HIP association.

Host Identity (HI): The public key of the signature algorithm that represents the identity of the host. In HIP, a host proves its identity by creating a signature with the private key belonging to its HI (c.f. Section 3).

Host Identity Tag (HIT): A shorthand for the HI in IPv6 format. It is generated by hashing the HI (c.f. Section 3.1).

HIT Suite: A HIT Suite groups all cryptographic algorithms that are required to generate and use an HI and its HIT. In particular, these algorithms are: 1) the public key signature algorithm and 2) the hash function, 3) the truncation (c.f. Appendix E).

HIP association: The shared state between two peers after completion of the BEX.

Initiator: The host that initiates the BEX. This role is typically forgotten once the BEX is completed.

Responder: The host that responds to the Initiator in the BEX. This role is typically forgotten once the BEX is completed.

Responder's HIT Hash Algorithm (RHASH): The Hash algorithm used for various hash calculations in this document. The algorithm is the same as is used to generate the Responder's HIT. The RHASH is the hash function defined by the HIT Suite of the Responder's HIT (c.f. Appendix E).

Length of the Responder's HIT Hash Algorithm (RHASH_len): The natural output length of RHASH in bits.

Signed data: Data that is signed is protected by a digital signature that was created by the sender of the data by using the private key of its HI.

KDF: The Key Derivation Function (KDF) is used for deriving the symmetric keys from the Diffie-Hellman key exchange.

KEYMAT: The keying material derived from the Diffie-Hellman key exchange by using the KDF. Symmetric keys for encryption and integrity protection of HIP control and payload packets are drawn from this keying material.

3. Host Identity (HI) and its Structure

In this section, the properties of the Host Identity and Host Identity Tag are discussed, and the exact format for them is defined. In HIP, the public key of an asymmetric key pair is used as the Host Identity (HI). Correspondingly, the host itself is defined as the entity that holds the private key of the key pair. See the HIP architecture specification [I-D.ietf-hip-rfc4423-bis] for more details on the difference between an identity and the corresponding identifier.

HIP implementations MUST support the Rivest Shamir Adelman (RSA) [RFC3110] public key algorithm and the Elliptic Curve Digital Signature Algorithm (ECDSA) for generating the HI as defined in Section 5.2.9. Additional algorithms MAY be supported.

A hashed encoding of the HI, the Host Identity Tag (HIT), is used in protocols to represent the Host Identity. The HIT is 128 bits long and has the following three key properties: i) it is the same length as an IPv6 address and can be used in fixed address-sized fields in APIs and protocols, ii) it is self-certifying (i.e., given a HIT, it is computationally hard to find a Host Identity key that matches the HIT), and iii) the probability of a HIT collision between two hosts is very low, hence, it is infeasible for an attacker to find a collision with a HIT that is in use. For details on the security properties of the HIT see [I-D.ietf-hip-rfc4423-bis].

The structure of the HIT is defined in [I-D.ietf-hip-rfc4843-bis]. The HIT is an Overlay Routable Cryptographic Hash Identifier (ORCHID) and consists of three parts: first, an IANA assigned prefix to distinguish it from other IPv6 addresses. Second, a four-bit encoding of the algorithms that were used for generating the HI and the hashed representation of HI. Third, a 96-bit hashed

representation of the Host Identity. The encoding of the ORCHID generation algorithm and the exact algorithm for generating the hashed representation is specified in Appendix E.

Carrying HIs and HITs in the header of user data packets would increase the overhead of packets. Thus, it is not expected that they are carried in every packet, but other methods are used to map the data packets to the corresponding HIs. In some cases, this makes it possible to use HIP without any additional headers in the user data packets. For example, if ESP is used to protect data traffic, the Security Parameter Index (SPI) carried in the ESP header can be used to map the encrypted data packet to the correct HIP association.

3.1. Host Identity Tag (HIT)

The Host Identity Tag is a 128-bit value -- a hashed encoding of the Host Identifier. There are two advantages of using a hashed encoding over the actual variable-sized Host Identity public key in protocols. First, the fixed length of the HIT keeps packet sizes manageable and eases protocol coding. Second, it presents a consistent format for the protocol, independent of the underlying identity technology in use.

RFC 4843-bis [I-D.ietf-hip-rfc4843-bis] specifies 128-bit hash-based identifiers, called Overlay Routable Cryptographic Hash Identifiers, ORCHIDs. Their prefix, allocated from the IPv6 address block, is defined in [I-D.ietf-hip-rfc4843-bis]. The Host Identity Tag is one type of an ORCHID.

This document extends the original, experimental HIP specification [RFC5201] with measures to support crypto agility. One of these measures is to allow different hash functions for creating a HIT. HIT Suites group the sets of algorithms that are required to generate and use a particular HIT. The Suites are encoded in HIT Suite IDs. These HIT Suite IDs are transmitted in the ORCHID Generation Algorithm (OGA) field in the ORCHID. With the HIT Suite ID in the OGA field, a hosts can tell from another host's HIT, whether it supports the necessary hash and signature algorithms to establish a HIP association with that host.

3.2. Generating a HIT from an HI

The HIT MUST be generated according to the ORCHID generation method described in [I-D.ietf-hip-rfc4843-bis] using a context ID value of 0xF0EF F02F BFF4 3D0F E793 0C3C 6E61 74EA (this tag value has been generated randomly by the editor of this specification), and an input that encodes the Host Identity field (see Section 5.2.9) present in a HIP payload packet. The set of hash function, signature algorithm,

and the algorithm used for generating the HIT from the HI depends on the HIT Suite (see Appendix E) and is indicated by the four bits of the ORCHID Generation Algorithm (OGA) field in the ORCHID. Currently, truncated SHA-1, truncated SHA-384, and truncated SHA-256 [FIPS.180-2.2002] are defined as hashes for generating a HIT.

For identities that are either RSA, Digital Signature Algorithm (DSA), or Elliptic Curve DSA (ECDSA) public keys, the ORCHID input consists of the public key encoding as specified for the Host Identity field of the HOST_ID parameter (see Section 5.2.9). This document defines four algorithm profiles: RSA, DSA, ECDSA, and ECDSA_LOW. The ECDSA_LOW profile is meant for devices with low computational capabilities. Hence, one of the following applies:

> The RSA public key is encoded as defined in [RFC3110] Section 2, taking the exponent length (e_len), exponent (e), and modulus (n) fields concatenated. The length (n_len) of the modulus (n) can be determined from the total HI Length and the preceding HI fields including the exponent (e). Thus, the data that serves as input for the HIT generation has the same length as the HI. The fields MUST be encoded in network byte order, as defined in [RFC3110].
>
> The DSA public key is encoded as defined in [RFC2536] Section 2, taking the fields T, Q, P, G, and Y, concatenated as input. Thus, 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]. The size parameter T, affecting the field lengths, MUST be selected as the minimum value that is long enough to accommodate P, G, and Y. The fields MUST be encoded in network byte order, as defined in [RFC2536].
>
> The ECDSA public keys are encoded as defined in [RFC6090] Section 4.2 and 6.

In Appendix B, the public key encoding process is illustrated using pseudo-code.

4. Protocol Overview

This section is a simplified overview of the HIP protocol operation, and does not contain all the details of the packet formats or the packet processing steps. Sections 5 and 6 describe in more detail the packet formats and packet processing steps, respectively, and are normative in case of any conflicts with this section.

The protocol number 139 has been assigned by IANA to the Host Identity Protocol.

The HIP payload (Section 5.1) header could be carried in every IP datagram. However, since HIP headers are relatively large (40 bytes), it is desirable to 'compress' the HIP header so that the HIP header only occurs in control packets used to establish or change HIP association state. The actual method for header 'compression' and for matching data packets with existing HIP associations (if any) is defined in separate documents, describing transport formats and methods. All HIP implementations MUST implement, at minimum, the ESP transport format for HIP [I-D.ietf-hip-rfc5202-bis].

4.1. Creating a HIP Association

By definition, the system initiating a HIP base exchange is the Initiator, and the peer is the Responder. This distinction is typically forgotten once the base exchange completes, and either party can become the Initiator in future communications.

The HIP base exchange serves to manage the establishment of state between an Initiator and a Responder. The first packet, I1, initiates the exchange, and the last three packets, R1, I2, and R2, constitute an authenticated Diffie-Hellman [DIF76] key exchange for session-key generation. In the first two packets, the hosts agree on a set of cryptographic identifiers and algorithms that are then used in and after the exchange. During the Diffie-Hellman key exchange, a piece of keying material is generated. The HIP association keys are drawn from this keying material by using a Key Derivation Function (KDF). If other cryptographic keys are needed, e.g., to be used with ESP, they are expected to be drawn from the same keying material by using the KDF.

The Initiator first sends a trigger packet, I1, to the Responder. The packet contains the HIT of the Initiator and possibly the HIT of the Responder, if it is known. Moreover, the I1 packet initializes the negotiation of the Diffie-Hellman group that is used for generating the keying material. Therefore, the I1 packet contains a list of Diffie Hellman Group IDs supported by the Initiator. Note that in some cases it may be possible to replace this trigger packet by some other form of a trigger, in which case the protocol starts with the Responder sending the R1 packet. In such cases, another mechanism to convey the Initiator's supported DH Groups (e.g., by using a default group) must be specified.

The second packet, R1, starts the actual authenticated Diffie-Hellman exchange. It contains a puzzle -- a cryptographic challenge that the Initiator must solve before continuing the exchange. The level of difficulty of the puzzle can be adjusted based on level of trust with the Initiator, current load, or other factors. In addition, the R1 contains the Responder's Diffie-Hellman parameter and lists of

cryptographic algorithms supported by the Responder. Based on these lists, the Initiator can continue, abort, or restart the base exchange with a different selection of cryptographic algorithms. Also, the R1 packet contains a signature that covers selected parts of the message. Some fields are left outside the signature to support pre-created R1s.

In the I2 packet, the Initiator MUST display the solution to the received puzzle. Without a correct solution, the I2 message is discarded. The I2 packet also contains a Diffie-Hellman parameter that carries needed information for the Responder. The I2 packet is signed by the Initiator.

The R2 packet acknowledges the receipt of the I2 packet and completes the base exchange. The packet is signed by the Responder.

The base exchange is illustrated below in Figure 1. The term "key" refers to the Host Identity public key, and "sig" represents a signature using such a key. The packets contain other parameters not shown in this figure.

```
   Initiator                              Responder

                I1: DH list
              -------------------------->
                                          select precomputed R1
                R1: puzzle, DH, key, sig
              <-------------------------
check sig                                 remain stateless
solve puzzle
              I2: solution, DH, {key}, sig
              -------------------------->
compute DH                                check puzzle
                                          check sig
                        R2: sig
              <--------------------------
check sig                                 compute DH
```

Figure 1

4.1.1. HIP Puzzle Mechanism

The purpose of the HIP puzzle mechanism is to protect the Responder from a number of denial-of-service threats. It allows the Responder to delay state creation until receiving the I2 packet. Furthermore, the puzzle allows the Responder to use a fairly cheap calculation to check that the Initiator is "sincere" in the sense that it has

churned enough CPU cycles in solving the puzzle.

The puzzle allows a Responder implementation to completely delay session-specific state creation until a valid I2 packet is received. An I2 packet without valid puzzle solution can be rejected immediately once the Responder has checked the solution by computing only one hash function before state is created and CPU-intensive public-key signature verification and Diffie-Hellman key generation are performed. By varying the difficulty of the puzzle, the Responder can frustrate CPU or memory targeted DoS attacks.

The Responder can remain stateless and drop most spoofed I2 packets because puzzle calculation is based on the Initiator's Host Identity Tag. 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 the I1 packet. When the Responder then later receives the I2 packet, it can verify that the puzzle has been solved using the Initiator's HIT. This makes it impractical for the attacker to first exchange one I1/R1 packet, and then generate a large number of spoofed I2 packets that seemingly come from different HITs. This method does not protect the Responder from an attacker that uses fixed HITs, though. Against such an attacker, a viable approach may be to create a piece of local state, and remember that the puzzle check has previously failed. See Appendix A for one possible implementation. Responder implementations SHOULD include sufficient randomness in the puzzle values so that algorithmic complexity attacks become impossible [CRO03].

The Responder can set the puzzle difficulty for the Initiator, based on its level of trust of the Initiator. Because the puzzle is not included in the signature calculation, the Responder can use pre-calculated R1 packets and include the puzzle just before sending the R1 to the Initiator. The Responder SHOULD use heuristics to determine when it is under a denial-of-service attack, and set the puzzle difficulty value #K appropriately as explained later.

4.1.2. Puzzle Exchange

The Responder starts the puzzle exchange when it receives an I1 packet. The Responder supplies a random number #I, and requires the Initiator to find a number J. To select a proper #J, the Initiator must create the concatenation of #I, the HITs of the parties, and #J, and calculate a hash over this concatenation using the RHASH algorithm. The lowest order #K bits of the result MUST be zeros. The value #K sets the difficulty of the puzzle.

To generate a proper number #J, the Initiator will have to generate a number of Js until one produces the hash target of zeros. The

Initiator SHOULD give up after exceeding the puzzle Lifetime in the PUZZLE parameter (as described in Section 5.2.4). The Responder needs to re-create the concatenation of #I, the HITs, and the provided #J, and compute the hash once to prove that the Initiator completed its assigned task.

To prevent precomputation attacks, the Responder MUST select the number #I in such a way that the Initiator cannot guess it. Furthermore, the construction MUST allow the Responder to verify that the value #I was indeed selected by it and not by the Initiator. See Appendix A for an example on how to implement this.

Using the Opaque data field in the PUZZLE (see Section 5.2.4), in an ECHO_REQUEST_SIGNED (see Section 5.2.20) or in an ECHO_REQUEST_UNSIGNED parameter (see Section 5.2.21), the Responder can include some data in R1 that the Initiator MUST copy unmodified in the corresponding I2 packet. The Responder can use the opaque data to transfer a piece of local state information to the Initiator and back, for example to recognize that the I2 is a response to a previously sent R1. The Responder can generate the Opaque data in various ways; e.g., using encryption or hashing with some secret, the sent #I, and possibly using other related data. With the same secret, the received #I (from the I2 packet), and the other related data (if any), the Responder can verify that it has itself sent the #I to the Initiator. The Responder MUST periodically change such a secret.

It is RECOMMENDED that the Responder generates new secrets for the puzzle and new R1s once every few minutes. Furthermore, it is RECOMMENDED that the Responder is able to verify valid puzzle solution at least Lifetime seconds after the puzzle secret has been deprecated. This time value guarantees that the puzzle is valid for at least Lifetime and at most 2 * Lifetime seconds. This limits the usability that an old, solved puzzle has to an attacker. Moreover, it avoids problems with the validity of puzzles if the lifetime is relatively short compared to the network delay and the time for solving the puzzle.

The puzzle value #I and the solution #J are inputs for deriving the keying material from the Diffie-Hellman key exchange (see Section 6.5). Therefore, a Responder SHOULD NOT use the same puzzle #I with the same DH keys for the same Initiator twice to ensure that the derived keying material differs. Such uniqueness can be achieved, for example, by using a counter as an additional input for generating #I. This counter can be increased for each processed I1 packet. The state of the counter can be transmitted in the Opaque data field in the PUZZLE (see Section 5.2.4), in an ECHO_REQUEST_SIGNED (see Section 5.2.20) or in an

ECHO_REQUEST_UNSIGNED parameter (see Section 5.2.21) without the need to establish state.

NOTE: The protocol developers explicitly considered whether R1 should include a timestamp in order to protect the Initiator from replay attacks. The decision was to NOT include a timestamp to avoid problems with global time synchronization.

NOTE: The protocol developers explicitly considered whether a memory bound function should be used for the puzzle instead of a CPU-bound function. The decision was not to use memory-bound functions.

4.1.3. Authenticated Diffie-Hellman Protocol with DH Group Negotiation

The packets R1, I2, and R2 implement a standard authenticated Diffie-Hellman exchange. The Responder sends one of its public Diffie-Hellman keys and its public authentication key, i.e., its Host Identity, in R1. The signature in the R1 packet allows the Initiator to verify that the R1 has been once generated by the Responder. However, since the R1 is precomputed and therefore does not cover association-specific information in the I1 packet, it does not protect from replay attacks.

Before the actual authenticated Diffie-Hellman exchange, the Initiator expresses its preference regarding its choice of the DH groups in the I1 packet. The preference is expressed as a sorted list of DH Group IDs. The I1 packet is not protected by a signature. Therefore, this list is sent in an unauthenticated way to avoid costly computations for processing the I1 packet at the Responder side. Based on the preferences of the Initiator, the Responder sends an R1 packet containing its most suitable public DH value. The Responder also attaches a list of its own preferences to the R1 to convey the basis for the DH group selection to the Initiator. This list is carried in the signed part of the R1 packet. If the choice of the DH group value in the R1 does not match the preferences of the Initiator and the Responder, the Initiator can detect that the list of DH Group IDs in the I1 was manipulated (see below for details).

If none of the DH Group IDs in the I1 packet is supported by the Responder, the Responder selects the DH Group most suitable for it regardless of the Initiator's preference. It then sends the R1 containing this DH Group and its list of supported DH Group IDs to the Initiator.

When the Initiator receives an R1, it receives one of the Responder's public Diffie-Hellman values and the list of DH Group IDs supported by the Responder. This list is covered by the signature in the R1 packet to avoid forgery. The Initiator compares the Group ID of the

public DH value in the R1 packet to the list of supported DH Group IDs in the R1 packets and to its own preferences expressed in the list of supported DH Group IDs. The Initiator continues the BEX only if the Group ID of the public DH value of the Responder is the most preferred of the IDs supported by both the Initiator and Responder. Otherwise, the communication is subject of a downgrade attack and the Initiator MUST either restart the base exchange with a new I1 packet or abort the base exchange. If the Responder's choice of the DH Group is not supported by the Initiator, the Initiator MAY abort the handshake or send a new I1 packet with a different list of supported DH Groups. However, the Initiator MUST verify the signature of the R1 packet before restarting or aborting the handshake. It MUST silently ignore the R1 packet if the signature is not valid.

If the preferences regarding the DH Group ID match, the Initiator computes the Diffie-Hellman session key (Kij). The Initiator creates a HIP association using keying material from the session key (see Section 6.5), and may use the HIP association to encrypt its public authentication key, i.e., the Host Identity. The resulting I2 packet contains the Initiator's Diffie-Hellman key and its (optionally encrypted) public authentication key. The signature of the I2 message covers all parameters of the signed parameter ranges (see Section 5.2) in the packet without exceptions as in the R1.

The Responder extracts the Initiator's Diffie-Hellman public key from the I2 packet, computes the Diffie-Hellman session key, creates a corresponding HIP association, and decrypts the Initiator's public authentication key. It can then verify the signature using the authentication key.

The final message, R2, completes the BEX and protects the Initiator against replay attacks because the Responder uses the shared key from the Diffie-Hellman exchange to create an HMAC as well as uses the private key of its Host Identity to sign the packet contents.

4.1.4. HIP Replay Protection

The HIP protocol includes the following mechanisms to protect against malicious packet replays. Responders are protected against replays of I1 packets by virtue of the stateless response to I1 packets with pre-signed R1 messages. Initiators are protected against R1 replays by a monotonically increasing "R1 generation counter" included in the R1. Responders are protected against replays of forged I2 packets by the puzzle mechanism (see Section 4.1.1 above), and optional use of opaque data. Hosts are protected against replays of R2 packets and UPDATEs by use of a less expensive HMAC verification preceding the HIP signature verification.

The R1 generation counter is a monotonically increasing 64-bit counter that may be initialized to any value. The scope of the counter MAY be system-wide but there SHOULD be a separate counter for each Host Identity, if there is more than one local host identity. The value of this counter SHOULD be preserved across system reboots and invocations of the HIP base exchange. This counter indicates the current generation of puzzles. Implementations MUST accept puzzles from the current generation and MAY accept puzzles from earlier generations. A system's local counter MUST be incremented at least as often as every time old R1s cease to be valid. The local counter SHOULD never be decremented, otherwise the host exposes its peers to the replay of previously generated, higher numbered R1s.

The R1 generation counter may roll over or may become reset. It is important for an Initiator to be robust to the loss of state about the R1 generation counter of a peer, or to a reset of the peer's counter. It is recommended that, when choosing between multiple R1s, the Initiator prefer to use the R1 that corresponds to the current R1 generation counter, but that if it is unable to make progress with that R1, the Initiator may try the other R1s beginning with the R1 packet with the highest counter.

A host may receive more than one R1, either due to sending multiple I1 packets (see Section 6.6.1) or due to a replay of an old R1. When sending multiple I1 packets to the same host, an Initiator SHOULD wait for a small amount of time (a reasonable time may be 2 * expected RTT) after the first R1 reception to allow possibly multiple R1s to arrive, and it SHOULD respond to an R1 among the set with the largest R1 generation counter. If an Initiator is processing an R1 or has already sent an I2 packet (still waiting for the R2 packet) and it receives another R1 with a larger R1 generation counter, it MAY elect to restart R1 processing with the fresher R1, as if it were the first R1 to arrive.

4.1.5. Refusing a HIP Base Exchange

A HIP-aware host may choose not to accept a HIP base exchange. If the host's policy is to only be an Initiator, it should begin its own HIP base exchange. A host MAY choose to have such a policy since only the privacy of the Initiator's HI is protected in the exchange. It should be noted that such behavior can introduce the risk of a race condition if each host's policy is to only be an Initiator, at which point the HIP base 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 'Destination 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. A HIP NOTIFY message is not used because no HIP session exists between the two hosts at that time.

4.1.6. Aborting a HIP Base Exchange

Two HIP hosts may encounter situations in which they cannot complete a HIP base exchange because of insufficient support for cryptographic algorithms, in particular the HIT Suites and DH Groups. After receiving the R1 packet, the Initiator can determine whether the Responder supports the required cryptographic operations to successfully establish a HIP association. The Initiator can abort the BEX silently after receiving an R1 packet that indicates an unsupported set of algorithms. The specific conditions are described below.

The R1 packet contains a signed list of HIT Suite IDs as supported by the Responder. Therefore, the Initiator can determine whether its source HIT is supported by the Responder. If the HIT Suite ID of the Initiator's HIT is not contained in the list of HIT Suites in the R1, the Initiator MAY abort the handshake silently or MAY restart the handshake with a new I1 packet that contains a source HIT supported by the Responder.

During the Handshake, the Initiator and the Responder agree on a single DH Group. The Responder selects the DH Group and its DH public value in the R1 based on the list of DH Suite IDs in the I1 packet. If the responder supports none of the DH Groups requested by the Initiator, the Responder selects an arbitrary DH and replies with an R1 containing its list of supported DH Group IDs. In such case, the Initiator receives an R1 packet containing the DH public value for an unrequested DH Group and also the Responder's DH Group list in the signed part of the R1 packet. At this point, the Initiator MAY abort the handshake or MAY restart the handshake by sending a new I1 packet containing a selection of DH Group IDs that is supported by the Responder.

4.1.7. HIP Downgrade Protection

In a downgrade attack, an attacker attempts to unnoticeably manipulate the packets of an Initiator and/or a Responder to influence the result of the cryptographic negotiations in the BEX to its favor. As a result, the victims select weaker cryptographic algorithms than they would otherwise have selected without the attacker's interference. Downgrade attacks can only be successful if they remain un-detected by the victims and the victims falsely assume a secure communication channel.

In HIP, almost all packet parameters related to cryptographic

negotiations are covered by signatures. These parameters cannot be directly manipulated in a downgrade attack without invalidating the signature. However, signed packets can be subject to replay attacks. In such a replay attack, the attacker could use an old BEX packet with an outdated and weak selection of cryptographic algorithms and replay it instead of a more recent packet with a collection of stronger cryptographic algorithms. Signed packets that could be subject to this replay attack are the R1 and I2 packet. However, replayed R1 and I2 packets cannot be used to successfully establish a HIP BEX because these packets also contain the public DH values of the Initiator and the Responder. Old DH values from replayed packets lead to invalid keying material and mismatching shared secrets because the attacker is unable to derive valid keying material from the DH public keys in the R1 and cannot generate a valid HMAC and signature for a replayed I2.

In contrast to the first version of HIP [RFC5201],the version 2 of HIP defined in this document begins the negotiation of the DH Groups already in the first BEX packet, the I1. The I1 packet is, by intention, not protected by a signature to avoid CPU-intensive cryptographic operations for processing floods of I1 packets targeted at the Responder. Hence, the list of DH Group IDs in the I1 packet is vulnerable to forgery and manipulation. To thwart an unnoticed manipulation of the I1 packet, the Responder chooses the DH Group deterministically and includes its own list of DH Group IDs in the signed part of the R1 packet. The Initiator can detect an attempted downgrade attack by comparing the list of DH Group IDs in the R1 packet to its own preferences in the I1 packet. If the choice of the DH Group in the R1 packet does not equal to the best match of the two lists (the highest priority DH ID of the Responder that is present in the Initiator's DH list), the Initiator can conclude that its list in the I1 packet was altered by an attacker. In this case, the Initiator can restart or abort the BEX. As mentioned before, the detection of the downgrade attack is sufficient to prevent it.

4.1.8. HIP Opportunistic Mode

It is possible to initiate a HIP BEX even if the Responder's HI (and HIT) is unknown. In this case, the initial I1 packet contains all zeros as the destination HIT. This kind of connection setup is called opportunistic mode.

The Responder may have multiple HITs due to multiple supported HIT Suites. Since the Responder's HIT Suite in the opportunistic mode is not determined by the destination HIT of the I1 packet, the Responder can freely select a HIT of any HIT Suite. The complete set of HIT Suites supported by the Initiator is not known to the Responder. Therefore, the Responder SHOULD should select its HIT from the same

HIT Suite as the Initiator's HIT (indicated by the HIT suite information in the OGA field of the Initiator's HIT) because this HIT Suite is obviously supported by the Initiator. If the Responder selects a different HIT that is not supported by the Initiator, the Initiator MAY restart the BEX with an I1 packet with a source HIT that is contained in the list of the Responder's HIT Suites in the R1 packet.

Note that the Initiator cannot verify the signature of the R1 packet if the Responder's HIT Suite is not supported. Therefore, the Initiator MUST treat R1 packets with unsupported Responder HITs as potentially forged and MUST NOT use any parameters from the unverified R1 besides the HIT Suite List. Moreover, an Initiator that uses an unverified HIT Suite List from an R1 packet to determine a possible source HIT MUST verify that the HIT_SUITE_LIST in the first unverified R1 packet matches the HIT_SUITE_LIST in the second R1 packet for which the Initiator supports the signature algorithm. The Initiator MUST restart the BEX with a new I1 packet for which the algorithm was mentioned in the verifiable R1 if the two lists do not match. This procedure is necessary to mitigate downgrade attacks.

There are both security and API issues involved with the opportunistic mode. These issues are described in the reminder of this section.

Given that the Responder's HI is not known by the Initiator, there must be suitable API calls that allow the Initiator to request, directly or indirectly, that the underlying system initiates the HIP base exchange solely based on locators. The Responder's HI will be tentatively available in the R1 packet, and in an authenticated form once the R2 packet has been received and verified. Hence, the Responder's HIT could be communicated to the application via new API mechanisms. However, with a backwards-compatible API the application sees only the locators used for the initial contact. Depending on the desired semantics of the API, this can raise the following issues:

- The actual locators may later change if an UPDATE message is used, even if from the API perspective the session still appears to be between two specific locators. However, the locator update is still secure and the session is still between the same nodes.

- Different sessions between the same two locators may result in connections to different nodes, if the implementation no longer remembers which identifier the peer had in an earlier session. This is possible when the peer's locator has changed for legitimate reasons or when an attacker pretends to be a node that has the peer's locator. Therefore, when using opportunistic mode,

HIP implementations MUST NOT place any expectation that the peer's HI returned in the R1 message matches any HI previously seen from that address.

If the HIP implementation and application do not have the same understanding of what constitutes a session, this may even happen within the same session. For instance, an implementation may not know when HIP state can be purged for UDP-based applications.

- As with all HIP base exchanges, the handling of locator-based or interface-based policy is unclear for HIP in opportunistic mode. An application may create a connection to a specific locator because the application has knowledge of the security properties along the network to that locator. If one of the nodes moves and the locators are updated, these security properties may not be maintained. Depending on the security policy of the application, this may be a problem. This is an area of ongoing study. As an example, there is work to create an API that applications can use to specify their security requirements in a similar context [I-D.ietf-btns-c-api].

In addition, the following security considerations apply. The generation counter mechanism will be less efficient in protecting against replays of the R1 packet, given that the Responder can choose a replay that uses an arbitrary HI, not just the one given in the I1 packet.

More importantly, the opportunistic exchange is vulnerable to man-in-the-middle attacks, because the Initiator does not have any public key information about the peer. To assess the impacts of this vulnerability, we compare it to vulnerabilities in current, non-HIP-capable communications.

An attacker on the path between the two peers can insert itself as a man-in-the-middle by providing its own identifier to the Initiator and then initiating another HIP session towards the Responder. For this to be possible, the Initiator must employ opportunistic mode, and the Responder must be configured to accept a connection from any HIP-enabled node.

An attacker outside the path will be unable to do so, given that it cannot respond to the messages in the base exchange.

These security properties are characteristic also of communications in the current Internet. A client contacting a server without employing end-to-end security may find itself talking to the server via a man-in-the-middle, assuming again that the server is willing to talk to anyone.

If end-to-end security is in place, then the worst that can happen in both the opportunistic HIP and non-HIP (normal IP) cases is denial-of-service; an entity on the path can disrupt communications, but will be unable to successfully insert itself as a man-in-the-middle.

However, once the opportunistic exchange has successfully completed, HIP provides confidentiality and integrity protection for the communications, and can securely change the locators of the endpoints.

As a result, it is believed that the HIP opportunistic mode is at least as secure as current IP.

4.2. Updating a HIP Association

A HIP association between two hosts may need to be updated over time. Examples include the need to rekey expiring security associations, add new security associations, or change IP addresses associated with hosts. The UPDATE packet is used for those and other similar purposes. This document only specifies the UPDATE packet format and basic processing rules, with mandatory parameters. The actual usage is defined in separate specifications.

HIP provides a general purpose UPDATE packet, which can carry multiple HIP parameters, for updating the HIP state between two peers. The UPDATE mechanism has the following properties:

> UPDATE messages carry a monotonically increasing sequence number and are explicitly acknowledged by the peer. Lost UPDATEs or acknowledgments may be recovered via retransmission. Multiple UPDATE messages may be outstanding under certain circumstances.
>
> UPDATE is protected by both HIP_MAC and HIP_SIGNATURE parameters, since processing UPDATE signatures alone is a potential DoS attack against intermediate systems.
>
> UPDATE packets are explicitly acknowledged by the use of an acknowledgment parameter that echoes an individual sequence number received from the peer. A single UPDATE packet may contain both a sequence number and one or more acknowledgment numbers (i.e., piggybacked acknowledgment(s) for the peer's UPDATE).

The UPDATE packet is defined in Section 5.3.5.

4.3. Error Processing

HIP error processing behavior depends on whether or not there exists an active HIP association. In general, if a HIP association exists

between the sender and receiver of a packet causing an error condition, the receiver SHOULD respond with a NOTIFY packet. On the other hand, if there are no existing HIP associations between the sender and receiver, or the receiver cannot reasonably determine the identity of the sender, the receiver MAY respond with a suitable ICMP message; see Section 5.4 for more details.

The HIP protocol and state machine are 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 the standard four-packet base exchange, establishing the HIP association.

The system with data to send has no state with the receiver, but the receiver has a residual HIP association.

The system with data to send is the Initiator. The Initiator acts as in no prior state, sending an I1 packet and receiving an R1 packet. When the Responder receives a valid I2 packet, the old association is 'discovered' and deleted, and the new association is established.

The system with data to send has a HIP association, but the receiver does not.

The system sends data on the outbound user data security association. The receiver 'detects' the situation when it receives a user data packet that it cannot match to any HIP association. The receiving host MUST discard this packet.

Optionally, the receiving host MAY send an ICMP packet, with the type Parameter Problem, to inform the sender that the HIP association does not exist (see Section 5.4), and it MAY initiate a new HIP BEX. However, responding with these optional mechanisms is implementation or policy dependent.

4.4. HIP State Machine

The HIP protocol itself has little state. In the HIP base exchange, there is an Initiator and a Responder. Once the security associations (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 symmetric and is presented in a single system view, representing either an Initiator or a Responder. The state machine is not a full representation of the processing logic. Additional processing rules are presented in the packet definitions. Hence, both are needed to completely implement HIP.

This document extends the state machine as defined in [RFC5201] and introduces a restart option to allow for the negotiation of cryptographic algorithms. The extension to the previous state machine in [RFC5201] is a transition from state I1-SENT to I1-SENT - the restart option. An Initiator is required to restart the HIP base exchange if the Responder does not support the HIT Suite of the Initiator. In this case, the Initiator restarts the HIP base exchange by sending a new I1 packet with a source HIT supported by the Responder.

Implementors must understand that the state machine, as described here, is informational. Specific implementations are free to implement the actual processing logic differently. Section 6 describes the packet processing rules in more detail. This state machine focuses on the HIP I1, R1, I2, and R2 packets only. New states and state transitions may be introduced by mechanisms in other specifications (such as mobility and multihoming).

4.4.1. State Machine Terminology

Unused Association Lifetime (UAL): Implementation-specific time for which, if no packet is sent or received for this time interval, a host MAY begin to tear down an active HIP association.

Maximum Segment Lifetime (MSL): Maximum time that a TCP segment is expected to spend in the network.

Exchange Complete (EC): Time that the host spends at the R2-SENT state before it moves to the ESTABLISHED state. The time is n * I2 retransmission timeout, where n is about I2_RETRIES_MAX.

Receive ANYOTHER: Any received packet for which no state transitions or processing rules are defined for a given state.

4.4.2. HIP States

State	Explanation
UNASSOCIATED	State machine start
I1-SENT	Initiating base exchange
I2-SENT	Waiting to complete base exchange
R2-SENT	Waiting to complete base exchange
ESTABLISHED	HIP association established
CLOSING	HIP association closing, no data can be sent
CLOSED	HIP association closed, no data can be sent
E-FAILED	HIP base exchange failed

Table 1: HIP States

4.4.3. HIP State Processes

System behavior in state UNASSOCIATED, Table 2.

Trigger	Action
User data to send, requiring a new HIP association	Send I1 and go to I1-SENT
Receive I1	Send R1 and stay at UNASSOCIATED
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at UNASSOCIATED
Receive user data for an unknown HIP association	Optionally send ICMP as defined in Section 5.4 and stay at UNASSOCIATED
Receive CLOSE	Optionally send ICMP Parameter Problem and stay at UNASSOCIATED

Receive ANYOTHER	Drop and stay at UNASSOCIATED

Table 2: UNASSOCIATED - Start state

System behavior in state I1-SENT, Table 3.

Trigger	Action
Receive I1 from Responder	If the local HIT is smaller than the peer HIT, drop I1 and stay at I1-SENT (see Section 6.5 for HIT comparison) If the local HIT is greater than the peer HIT, send R1 and stay at I1-SENT
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at I1-SENT
Receive R1, process	If the HIT Suite of the local HIT is not supported by the peer, select supported local HIT, send I1 and stay at I1-SENT If successful, send I2 and go to I2-SENT If fail, stay at I1-SENT
Receive ANYOTHER	Drop and stay at I1-SENT
Timeout	Increment timeout counter If counter is less than I1_RETRIES_MAX, send I1 and stay at I1-SENT If counter is greater than I1_RETRIES_MAX, go to E-FAILED

Table 3: I1-SENT - Initiating the HIP Base Exchange

System behavior in state I2-SENT, Table 4.

Trigger	Action
Receive I1	Send R1 and stay at I2-SENT
Receive R1, process	If successful, send I2 and stay at I2-SENT If fail, stay at I2-SENT
Receive I2, process	If successful and local HIT is smaller than the peer HIT, drop I2 and stay at I2-SENT If successful and local HIT is greater than the peer HIT, send R2 and go to R2-SENT If fail, stay at I2-SENT
Receive R2, process	If successful, go to ESTABLISHED If fail, stay at I2-SENT
Receive CLOSE, process	If successful, send CLOSE_ACK and go to CLOSED If fail, stay at I2-SENT
Receive ANYOTHER	Drop and stay at I2-SENT
Timeout	Increment timeout counter If counter is less than I2_RETRIES_MAX, send I2 and stay at I2-SENT If counter is greater than I2_RETRIES_MAX, go to E-FAILED

Table 4: I2-SENT - Waiting to finish the HIP Base Exchange

System behavior in state R2-SENT, Table 5.

Trigger	Action
Receive I1	Send R1 and stay at R2-SENT
Receive I2, process	If successful, send R2 and stay at R2-SENT If fail, stay at R2-SENT
Receive R1	Drop and stay at R2-SENT
Receive R2	Drop and stay at R2-SENT
Receive data or UPDATE	Move to ESTABLISHED
Exchange Complete Timeout	Move to ESTABLISHED
Receive CLOSE, process	If successful, send CLOSE_ACK and go to CLOSED If fail, stay at ESTABLISHED
Receive CLOSE_ACK	Drop and stay at R2-SENT
Receive NOTIFY	Process and stay at R2-SENT

Table 5: R2-SENT - Waiting to finish HIP

System behavior in state ESTABLISHED, Table 6.

Trigger	Action
Receive I1	Send R1 and stay at ESTABLISHED
Receive I2	Process with puzzle and possible Opaque data verification If successful, send R2, drop old HIP association, establish a new HIP association and go to R2-SENT If fail, stay at ESTABLISHED
Receive R1	Drop and stay at ESTABLISHED
Receive R2	Drop and stay at ESTABLISHED
Receive user data for HIP association	Process and stay at ESTABLISHED
No packet sent/received during UAL minutes	Send CLOSE and go to CLOSING
Receive UPDATE	Process and stay at ESTABLISHED
Receive CLOSE, process	If successful, send CLOSE_ACK and go to CLOSED If fail, stay at ESTABLISHED
Receive CLOSE_ACK	Drop and stay at ESTABLISHED
Receive NOTIFY	Process and stay at ESTABLISHED

Table 6: ESTABLISHED - HIP association established

System behavior in state CLOSING, Table 7.

Trigger	Action
User data to send, requires the creation of another incarnation of the HIP association	Send I1 and stay at CLOSING
Receive I1	Send R1 and stay at CLOSING
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at CLOSING
Receive R1, process	If successful, send I2 and go to I2-SENT If fail, stay at CLOSING
Receive CLOSE, process	If successful, send CLOSE_ACK, discard state and go to CLOSED If fail, stay at CLOSING
Receive CLOSE_ACK, process	If successful, discard state and go to UNASSOCIATED If fail, stay at CLOSING
Receive ANYOTHER	Drop and stay at CLOSING
Timeout	Increment timeout sum and reset timer. If timeout sum is less than UAL+MSL minutes, retransmit CLOSE and stay at CLOSING If timeout sum is greater than UAL+MSL minutes, go to UNASSOCIATED

Table 7: CLOSING - HIP association has not been used for UAL minutes

System behavior in state CLOSED, Table 8.

Trigger	Action
Datagram to send, requires the creation of another incarnation of the HIP association	Send I1, and stay at CLOSED
Receive I1	Send R1 and stay at CLOSED
Receive I2, process	If successful, send R2 and go to R2-SENT If fail, stay at CLOSED
Receive R1, process	If successful, send I2 and go to I2-SENT If fail, stay at CLOSED
Receive CLOSE, process	If successful, send CLOSE_ACK, stay at CLOSED If fail, stay at CLOSED
Receive CLOSE_ACK, process	If successful, discard state and go to UNASSOCIATED If fail, stay at CLOSED
Receive ANYOTHER	Drop and stay at CLOSED
Timeout (UAL+2MSL)	Discard state, and go to UNASSOCIATED

Table 8: CLOSED - CLOSE_ACK sent, resending CLOSE_ACK if necessary

System behavior in state E-FAILED, Table 9.

Trigger	Action
Wait for implementation-specific time	Go to UNASSOCIATED. Re-negotiation is possible after moving to UNASSOCIATED state.

Table 9: E-FAILED - HIP failed to establish association with peer

4.4.4. Simplified HIP State Diagram

The following diagram (Figure 2) shows the major state transitions. Transitions based on received packets implicitly assume that the packets are successfully authenticated or processed.

```
                               +--+       +----------------------------+
              recv I1, send R1 |  |       |                            |
                               |  v       v                            |
                             +--------------+  recv I2, send R2        |
            +----------------| UNASSOCIATED |----------------+         |
    datagram|  +--+          +--------------+                |         |
    to send,|  |  | Alg. not supported,                      |         |
     send I1|  |  | send I1                                  |         |
            v  |  v                                          |         |
         +---------+  recv I2, send R2                       |         |
   +---->| I1-SENT |--------------------------------------+  |         |
   |     +---------+            +----------------------+  |  |         |
   |          | recv R2,        | recv I2, send R2     |  |  |         |
   |          v send I2         |                      v  v  v         |
   |       +---------+          |                    +---------+       |
   |  +--->| I2-SENT |----------+     +--------------| R2-SENT |<---+  |
   |  |    +---------+                |              +---------+    |  |
   |  |          |  |recv R2          |        data or|             |  |
   |  |recv R1,  |  |                 |     EC timeout|             |  |
   |  |send I2   +--|-----------------+               |  receive I2,|  |
   |  |          |  |       +-------------+           |      send R2|  |
   |  |          |  +------>| ESTABLISHED |<----------+             |  |
   |  |          |          +-------------+                         |  |
   |  |          |            |  |  |      receive I2, send R2      |  |
   |  |          +------------+  |  +-------------------------------+  |
   |  |          |               +-----------+                      |  |
   |  |          |    no packet sent/received|    +---+             |  |
   |  |          |    for UAL min, send CLOSE|    |   |timeout      |  |
   |  |          |                           v    v   |(UAL+MSL)    |  |
   |  |          |                        +---------+ |retransmit   |  |
   +--|----------|------------------------| CLOSING |-+CLOSE        |  |
      |          |                        +---------+               |  |
      |          |                         | |   | |                |  |
      +----------|-------------------------+ |   | +----------------+  |
      |          |               +-----------+   +------------------|--+
      |          |               |recv CLOSE,      recv CLOSE_ACK   |  |
      |          +-------------+ |send CLOSE_ACK   or timeout       |  |
      |     recv CLOSE,        | |                 (UAL+MSL)        |  |
      |     send CLOSE_ACK     v v                                  |  |
      |                     +--------+  receive I2, send R2         |  |
      +---------------------| CLOSED |------------------------------+  |
                            +--------+                                 |
                             ^ |  |                                    |
   recv CLOSE, send CLOSE_ACK| |  |              timeout (UAL+2MSL)    |
                             +-+  +------------------------------------+
```

Figure 2

4.5. User Data Considerations

4.5.1. TCP and UDP Pseudo-Header Computation for User Data

When computing TCP and UDP checksums on user data packets that flow through sockets bound to HITs, the IPv6 pseudo-header format [RFC2460] MUST be used, even if the actual addresses in the header of the packet are IPv4 addresses. Additionally, the HITs MUST be used in place of the IPv6 addresses in the IPv6 pseudo-header. Note that the pseudo-header for actual HIP payloads is computed differently; see Section 5.1.1.

4.5.2. Sending Data on HIP Packets

Other documents may define how to include user data in various HIP packets. However, currently the HIP header is a terminal header, and not followed by any other headers.

4.5.3. Transport Formats

The actual data transmission format, used for user data after the HIP base exchange, is not defined in this document. Such transport formats and methods are described in separate specifications. All HIP implementations MUST implement, at minimum, the ESP transport format for HIP [I-D.ietf-hip-rfc5202-bis]. The transport format to be chosen is negotiated in the base exchange. The Responder expresses its preference of the transport format in the TRANSPORT_FORMAT_LIST in the R1 packet and the Initiator selects one transform and adds the respective HIP parameter to the I2 packet.

4.5.4. Reboot, Timeout, and 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 the HIP association times out, it has lost its HIP state. If the host that lost state has a datagram to send to the peer, it simply restarts the HIP base exchange. After the base exchange has completed, the Initiator can create a new payload association and start sending data. The peer does not reset its state until it receives a valid I2 packet.

If a system receives a user data packet that cannot be matched to any existing HIP association, it is possible that it has lost the state and its peer has not. It MAY send an ICMP packet with the Parameter Problem type, and with the pointer pointing to the referred HIP-related association information. Reacting to such traffic depends on

the implementation and the environment where the implementation is used.

If the host, that apparently has lost its state, decides to restart the HIP base exchange, it sends an I1 packet to the peer. After the base exchange has been completed successfully, the Initiator can create a new HIP association and the peer drops its old payload associations and creates a new one.

4.6. Certificate Distribution

This document does not define how to use certificates or how to transfer them between hosts. These functions are expected to be defined in a future specification as for HIP Version 1 [RFC6253]. A parameter type value, meant to be used for carrying certificates, is reserved, though: CERT, Type 768; see Section 5.2.

5. Packet Formats

5.1. Payload Format

All HIP packets start with a fixed header.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   | Header Length |0| Packet Type |Version| RES.|1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Checksum             |           Controls            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Sender's Host Identity Tag (HIT)               |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Receiver's Host Identity Tag (HIT)              |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
/                        HIP Parameters                         /
/                                                               /
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The HIP header is logically an IPv6 extension header. 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 define behavior for also other values. However, current implementations MUST ignore trailing data if an unimplemented Next Header value is received.

The Header Length field contains the combined length of the HIP Header and HIP parameters in 8-byte 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. Note: this sets an additional limit for sizes of parameters included in the Parameters field, independent of the individual parameter maximum lengths.

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 unrecognized packet type, it MUST drop the packet.

The HIP Version field is four bits. The version defined in this document is 2. 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 (see Section 5.1.2) .

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

The two fixed bits in the header are reserved for SHIM6 compatibility [RFC5533], Section 5.3. For implementations adhering (only) to this specification, they MUST be set as shown when sending and MUST be ignored when receiving. This is to ensure optimal forward compatibility. Note that for implementations that implement other compatible specifications in addition to this specification, the corresponding rules may well be different. For example, an implementation that implements both this specification and the SHIM6 protocol may need to check these bits in order to determine how to handle the packet.

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

5.1.1. Checksum

Since the checksum covers the source and destination addresses in the IP header, it MUST be recomputed on HIP-aware NAT devices.

If IPv6 is used to carry the HIP packet, the pseudo-header [RFC2460] contains the source and destination IPv6 addresses, HIP packet length in the pseudo-header length field, a zero field, and the HIP protocol number (see Section 4) 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 IPv4 UDP pseudo-header format [RFC0768] is used. In the pseudo-header, the source and destination addresses are those used in the IP header, the zero field is obviously zero, the protocol is the HIP protocol number (see Section 4), and the length is calculated as in the IPv6 case.

5.1.2. HIP Controls

The HIP Controls field conveys information about the structure of the packet and capabilities of the host.

The following fields have been defined:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | | | | | | | | | | | | | | |A|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A - Anonymous: If this is set, the sender's 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 using anonymous sender HIs. The peer receiving an anonymous HI in an R1 or I2 may choose to refuse it.

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

5.1.3. HIP Fragmentation Support

A HIP implementation MUST support IP fragmentation/reassembly. Fragment reassembly MUST be implemented in both IPv4 and IPv6, but fragment generation is REQUIRED to be implemented in IPv4 (IPv4 stacks and networks will usually do this by default) and RECOMMENDED to be implemented in IPv6. In IPv6 networks, the minimum MTU is larger, 1280 bytes, than in IPv4 networks. 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 IPv4 networks, HIP packets may encounter low MTUs along their routed path. Since basic HIP, as defined in this document, does not

provide a mechanism to use multiple IP datagrams for a single HIP packet, support for path MTU discovery does not bring any value to HIP in IPv4 networks. HIP-aware NAT devices SHOULD perform IPv4 reassembly/fragmentation for HIP control packets.

All HIP implementations have to be careful while employing a reassembly algorithm so that the algorithm is sufficiently resistant to DoS attacks.

Certificate chains can cause the packet to be fragmented and fragmentation can open implementations to denial-of-service attacks [KAU03]. "Hash and URL" schemes as defined in [RFC6253] for HIP version 1 may be used to avoid fragmentation and mitigate resulting DoS attacks.

5.2. HIP Parameters

The HIP parameters carry information that is necessary for establishing and maintaining a HIP association. For example, the peer's public keys as well as the signaling for negotiating ciphers and payload handling are encapsulated in HIP parameters. Additional information, meaningful for end-hosts or middleboxes, may also be included in HIP parameters. The specification of the HIP parameters and their mapping to HIP packets and packet types is flexible to allow HIP extensions to define new parameters and new protocol behavior.

In HIP packets, HIP parameters are ordered according to their numeric type number and encoded in TLV format.

The following parameter types are currently defined.

TLV	Type	Length	Data
R1_COUNTER	129	12	Puzzle generation counter
PUZZLE	257	12	K and Random #I
SOLUTION	321	20	K, Random #I and puzzle solution J
SEQ	385	4	UPDATE packet ID number
ACK	449	variable	UPDATE packet ID number
DH_GROUP_LIST	511	variable	Ordered list of DH Group IDs supported by a host
DIFFIE_HELLMAN	513	variable	public key
HIP_CIPHER	579	variable	List of HIP encryption algorithms
ENCRYPTED	641	variable	Encrypted part of a HIP packet
HOST_ID	705	variable	Host Identity with Fully-Qualified Domain FQDN (Name) or Network Access Identifier (NAI)
HIT_SUITE_LIST	715	variable	Ordered list of the HIT suites supported by the Responder
CERT	768	variable	HI Certificate; used to transfer certificates. Specified in a separate docment.
NOTIFICATION	832	variable	Informational data

ECHO_REQUEST_SIGNED	897	variable	Opaque data to be echoed back; signed
ECHO_RESPONSE_SIGNED	961	variable	Opaque data echoed back by request; signed
TRANSPORT_FORMAT_LIST	2049	Ordered list of preferred HIP transport type numbers	variable
HIP_MAC	61505	variable	HMAC-based message authentication code, with key material from KEYMAT
HIP_MAC_2	61569	variable	HMAC based message authentication code, with key material from KEYMAT. Unlike HIP_MAC, the HOST_ID parameter is included in HIP_MAC_2 calculation.
HIP_SIGNATURE_2	61633	variable	Signature used in R1 packet
HIP_SIGNATURE	61697	variable	Signature of the packet
ECHO_REQUEST_UNSIGNED	63661	variable	Opaque data to be echoed back; after signature
ECHO_RESPONSE_UNSIGNED	63425	variable	Opaque data echoed back by request; after signature

As the ordering (from lowest to highest) of HIP parameters is strictly enforced (see Section 5.2.1), the parameter type values for existing parameters have been spaced to allow for future protocol

extensions.

The following parameter type number ranges are defined.

Type Range	Purpose
0 - 1023	Handshake
1024 - 2047	Reserved
2048 - 4095	Parameters related to HIP transport formats
4096 - 8191	Signed parameters allocated through specification documents
8192 - 32767	Reserved
32768 - 49151	Free for experimentation. Signed parameters.
41952 - 61439	Reserved
61440 - 64443	Signatures and (signed) MACs
62464 - 63487	Parameters that are neither signed nor MACed
63488 - 64511	Rendezvous and relaying
64512 - 65023	Parameters that are neither signed nor MACed
65024 - 65535	Reserved

The process for defining new parameters is described in Section 5.2.2 of this document.

The range between 32768 (2^15) and 49151 (2^15 + 2^14) are free for experimentation. Types from this range SHOULD be selected in a random fashion to reduce the probability of collisions.

5.2.1. 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 in the packet so that their types form an increasing order. If multiple parameters with the same type number are in one packet, the parameters with the same type MUST be consecutive in the packet. If the order does not

follow this rule, the packet is considered to be malformed and it MUST be discarded.

Parameters using type values from 2048 up to 4095 are related to transport formats. Currently, one transport format is defined: the ESP transport format [I-D.ietf-hip-rfc5202-bis].

All of the encoded TLV parameters have a length (that includes the Type and Length fields), which is a multiple of 8 bytes. When needed, padding MUST be added to the end of the parameter so that the total length is a multiple of 8 bytes. This rule ensures proper alignment of data. Any added padding bytes MUST be zeroed by the sender, and their values SHOULD NOT be checked by the receiver.

The Length field indicates the length of the Contents field (in bytes). Consequently, 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;

where % is the modulo operator

```
  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            |C|             Length            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 /                          Contents                             /
 /                                               +-+-+-+-+-+-+-+-+
 |                                               |    Padding    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	Type code for the parameter. 16 bits long, C-bit being part of the Type code.
C	Critical. One if this parameter is critical, and MUST be recognized by the recipient, zero otherwise. The C bit is considered to be a part of the Type field. Consequently, critical parameters are always odd and non-critical ones have an even value.
Length	Length of the Contents, in bytes excluding Type, Length, and Padding.
Contents	Parameter specific, defined by Type
Padding	Padding, 0-7 bytes, added if needed

Critical parameters (indicated by the odd type number) MUST be recognized by the recipient. If a recipient encounters a critical

parameter that it does not recognize, it MUST NOT process the packet any further. It MAY send an ICMP or NOTIFY, as defined in Section 4.3.

Non-critical parameters MAY be safely ignored. If a recipient encounters a non-critical parameter that it does not recognize, it SHOULD proceed as if the parameter was not present in the received packet.

5.2.2. Defining New Parameters

Future specifications may define new parameters as needed. When defining new parameters, care must be taken to ensure that the parameter type values are appropriate and leave suitable space for other future extensions. One must remember that the parameters MUST always be arranged in numerically increasing order by Type code, thereby limiting the order of parameters (see Section 5.2.1).

The following rules MUST be followed when defining new parameters.

1. The low-order bit C of the Type code is used to distinguish between critical and non-critical parameters. Hence, even parameter type numbers indicate non-critical parameters while odd parameter type numbers indicate critical parameters.

2. A new parameter MAY be critical only if an old implementation that ignored it would cause security problems. In general, new parameters SHOULD be defined as non-critical, and expect a reply from the recipient.

3. If a system implements a new critical parameter, it MUST provide the ability to set the associated feature off, such that the critical parameter is not sent at all. The configuration option MUST be well documented. Implementations operating in a mode adhering to this specification MUST disable the sending of new critical parameters by default. In other words, the management interface MUST allow vanilla standards-only mode as a default configuration setting, and MAY allow new critical payloads to be configured on (and off).

4. See Section 9 for allocation rules regarding Type codes.

5.2.3. R1_COUNTER

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       Reserved, 4 bytes                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                R1 generation counter, 8 bytes                 |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

     Type           129
     Length         12
     R1 generation
       counter      The current generation of valid puzzles
```

The R1_COUNTER parameter contains a 64-bit unsigned integer in network-byte order, indicating the current generation of valid puzzles. The sender SHOULD increment this counter periodically. It is RECOMMENDED that the counter value is incremented at least as often as old PUZZLE values are deprecated so that SOLUTIONs to them are no longer accepted.

Support for the R1_COUNTER parameter is mandatory. It SHOULD be included in the R1 (in which case, it is covered by the signature), and if present in the R1, it MUST be echoed (including the Reserved field verbatim) by the Initiator in the I2 packet.

5.2.4. PUZZLE

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  #K, 1 byte   |    Lifetime   |        Opaque, 2 bytes        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Random #I, RHASH_len/8 bytes             |
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	257
Length	4 + RHASH_len / 8
#K	#K is the number of verified bits
Lifetime	puzzle lifetime 2^(value-32) seconds
Opaque	data set by the Responder, indexing the puzzle
Random #I	random number of size RHASH_len bits

Random #I is represented as a n-bit integer (where n is RHASH_len), #K and Lifetime as 8-bit integers, all in network byte order.

The PUZZLE parameter contains the puzzle difficulty #K and a n-bit random integer #I. The Puzzle Lifetime indicates the time during which the puzzle solution is valid, and sets a time limit that should not be exceeded by the Initiator while it attempts to solve the puzzle. The lifetime is indicated as a power of 2 using the formula 2^(Lifetime-32) seconds. A puzzle MAY be augmented with an ECHO_REQUEST_SIGNED or an ECHO_REQUEST_UNSIGNED parameter included in the R1; the contents of the echo request are then echoed back in the ECHO_RESPONSE_SIGNED or in the ECHO_RESPONSE_UNSIGNED parameter, allowing the Responder to use the included information as a part of its puzzle processing.

The Opaque and Random #I field are not covered by the HIP_SIGNATURE_2 parameter.

5.2.5. SOLUTION

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | #K, 1 byte   |   Reserved    |        Opaque, 2 bytes        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Random #I, n bytes                       |
    /                                                               /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |            Puzzle solution #J, RHASH_len/8 bytes              |
    /                                                               /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	321
Length	4 + RHASH_len /4
#K	#K is the number of verified bits
Reserved	zero when sent, ignored when received
Opaque	copied unmodified from the received PUZZLE parameter
Random #I	random number of size RHASH_len bits
Puzzle solution #J	random number of size RHASH_len bits

Random #I and Random #J are represented as n-bit unsigned integers (where n is RHASH_len), #K as an 8-bit unsigned integer, all in network byte order.

The SOLUTION parameter contains a solution to a puzzle. It also echoes back the random difficulty #K, the Opaque field, and the puzzle integer #I.

5.2.6. DH_GROUP_LIST

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DH GROUP ID #1| DH GROUP ID #2| DH GROUP ID #3| DH GROUP ID #4|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| DH GROUP ID #n|                Padding                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	511
Length	number of DH Group IDs
DH GROUP ID	defines a DH GROUP ID supported by the host. The list of IDs is ordered by preference of the host. The list of define DH Group IDs in the DIFFIE_HELLMAN parameter. Each DH Group ID is one octet long.

The DH_GROUP_LIST parameter contains the list of supported DH Group IDs of a host. The Initiator sends the DH_GROUP_LIST in the I1 packet, the Responder sends its own list in the signed part of the R1 packet. The DH Group IDs in the DH_GROUP_LIST are listed in the order of their preference of the host sending the list. DH Group IDs that are listed first are preferred over the DH Group IDs listed later. The information in the DH_GROUP_LIST allows the Responder to select the DH group preferred by itself and supported by the Initiator. Based on the DH_GROUP_LIST in the R1 packet, the Initiator can determine if the Responder has selected the best possible choice based on the Initiator's and Responder's preferences. If the Responder's choice differs from the best choice, the Initiator can conclude that there was an attempted downgrade attack (see Section 4.1.7).

When selecting the DH group for the DIFFIE_HELLMAN parameter in the R1 packet, the Responder MUST select the first DH Group ID in its DH_GROUP_LIST in the R1 packet that is compatible with one of the Suite IDs in the Initiator's DH_GROUP_LIST in the I1 packet. The Responder MUST NOT select any other DH Group ID that is contained in both lists because a downgrade attack cannot be detected then.

In general, hosts SHOULD prefer stronger groups over weaker ones if the computation overhead is not prohibitively high for the intended application.

5.2.7. DIFFIE_HELLMAN

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Group ID    |      Public Value Length      | Public Value  /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                               |            Padding            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	513
Length	length in octets, excluding Type, Length, and Padding
Group ID	defines values for p and g as well as the KDF
Public Value Length	length of the following Public Value in octets
Public Value	the sender's public Diffie-Hellman key

The following Group IDs have been defined:

Group	KDF	Value
Reserved		0
DEPRECATED		1
DEPRECATED		2
1536-bit MODP group [RFC3526]	HKDF [RFC5869]	3
3072-bit MODP group [RFC3526]	HKDF [RFC5869]	4
DEPRECATED		5
DEPRECATED		6
NIST P-256 [RFC5903]	HKDF [RFC5869]	7
NIST P-384 [RFC5903]	HKDF [RFC5869]	8
NIST P-521 [RFC5903]	HKDF [RFC5869]	9
SECP160R1 [SECG]	HKDF [RFC5869]	10

The MODP Diffie-Hellman groups are defined in [RFC3526]. The ECDH groups 7 - 9 are defined in [RFC5903] and [RFC6090]. ECDH group 10 is covered in Appendix D. Any ECDH used with HIP MUST have a co-factor of 1.

The Group ID also defines the key derivation function that is to be used for deriving the symmstric keys for the HMAC and symmetric encryption from the keying material from the Diffie Hellman key exchange (see Section 6.5).

A HIP implementation MUST implement Group ID 3. The 160-bit

SECP160R1 group can be used when lower security is enough (e.g., web surfing) and when the equipment is not powerful enough (e.g., some PDAs). Implementations SHOULD implement Group IDs 4 and 8.

To avoid unnecessary failures during the base exchange, the rest of the groups SHOULD be implemented in hosts where resources are adequate.

5.2.8. HIP_CIPHER

```
    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              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Cipher ID #1         |          Cipher ID #2         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Cipher ID #n         |             Padding           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	579
Length	length in octets, excluding Type, Length, and Padding
Cipher ID	defines the cipher algorithm to be used for encrypting the contents of the ENCRYPTED parameter

The following Cipher IDs are defined:

Suite ID	Value	
RESERVED	0	
NULL-ENCRYPT	1	([RFC2410])
AES-128-CBC	2	([RFC3602])
DEPRECATED	3	
AES-256-CBC	4	([RFC3602])

The sender of a HIP_CIPHER parameter MUST make sure that there are no more than six (6) Cipher IDs in one HIP_CIPHER parameter. Conversely, a recipient MUST be prepared to handle received transport parameters that contain more than six Cipher IDs by accepting the first six Cipher IDs and dropping the rest. The limited number of transforms sets the maximum size of the HIP_CIPHER parameter. As the default configuration, the HIP_CIPHER parameter MUST contain at least one of the mandatory Cipher IDs. There MAY be a configuration option that allows the administrator to override this default.

The Responder lists supported and desired Cipher IDs in order of

preference in the R1, up to the maximum of six Cipher IDs. The Initiator MUST choose only one of the corresponding Cipher IDs. This Cipher ID will be used for generating the ENCRYPTED parameter.

Mandatory implementation: AES-128-CBC. NULL-ENCRYPTION is included for testing purposes.

5.2.9. HOST_ID

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          HI Length            |DI-type|      DI Length        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Algorithm            |         Host Identity         /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                               |       Domain Identifier       /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                                               |    Padding    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	705
Length	length in octets, excluding Type, Length, and Padding
HI Length	length of the Host Identity in octets
DI-type	type of the following Domain Identifier field
DI Length	length of the Domain Identifier field in octets
Algorithm	index to the employed algorithm
Host Identity	actual Host Identity
Domain Identifier	the identifier of the sender

The following DI-types have been defined:

Type	Value
none included	0
FQDN	1
NAI	2

FQDN	Fully Qualified Domain Name, in binary format.
NAI	Network Access Identifier

The format for the FQDN is defined in RFC 1035 [RFC1035] Section 3.1. The format for the NAI is defined in [RFC4282]

A host MAY optionally associate the Host Identity with a single

Domain Identifier in the HOST_ID parameter. If there is no Domain Identifier, i.e., the DI-type field is zero, the DI Length field is set to zero as well.

The following HI Algorithms have been defined:

Algorithm profiles	Values
RESERVED	0
DSA	3 [FIPS 186-3] (RECOMMENDED)
RSA	5 [RFC3447] (REQUIRED)
ECDSA	7 [RFC4754] (REQUIRED)
ECDSA_LOW	9 [SECG] (RECOMMENDED)

For DSA, RSA, and ECDSA key types, profiles containing at least 112 bits of security strength (as defined by [NIST.800-131A.2011]) should be used. For RSA signature padding, the PSS method of padding [RFC3447] MUST be used.

The Host Identity is derived from the DNSKEY format for RSA and DSA. For these, the Public Key field of the RDATA part from RFC 4034 [RFC4034] is used. For ECC we distinguish two different profiles: ECDSA and ECDSA_LOW. ECC contains curves approved by NIST and defined in RFC 4754 [RFC4754]. ECDSA_LOW is defined for devices with low computational capabilities and uses shorter curves from SECG [SECG]. Any ECDSA used with HIP MUST have a co-factor of 1.

For ECDSA and ECDSA_LOW Host Identities are represented by the following fields:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          ECC Curve            |                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                          Public Key                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ECC Curve	Curve label
Public Key	Represented in Octet-string format [RFC6090]

For hosts that implement ECDSA as algorithm the following ECC curves are required:

Algorithm	Curve	Values
ECDSA	RESERVED	0
ECDSA	NIST P-256	1 [RFC4754]
ECDSA	NIST P-384	2 [RFC4754]

For hosts that implement the EDSA_LOW algorithm profile, the following curve is required:

Algorithm	Curve	Values
ECDSA_LOW	RESERVED	0
ECDSA_LOW	SECP160R1	1 [SECG]

5.2.10. HIT_SUITE_LIST

The HIT_SUITE_LIST parameter contains a list of the supported HIT suite IDs of the Responder. The Responder sends the HIT_SUITE_LIST in the signed part of the R1 packet. Based on the HIT_SUITE_LIST, the Initiator can determine which source HITs are supported by the Responder.

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     ID #1     |     ID #2     |     ID #3     |     ID #4     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     ID #n     |                Padding                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	715
Length	number of HIT Suite IDs
ID	defines a HIT Suite ID supported by the host. The list of IDs is ordered by preference of the host. Each HIT Suite ID is one octet long. The four higher-order bits of the ID field correspond to the HIT Suite ID in the ORCHID OGA field. The four lower-order bits are reserved and set to 0 and ignored by the receiver.

The HIT Suite ID indexes a HIT Suite. HIT Suites are composed of signature algorithms as defined in Section 5.2.9 and hash functions.

The ID field in the HIT_SUITE_LIST is defined as eight-bit field as opposed to the four-bit HIT Suite ID and OGA field in the ORCHID. This difference is a measure to accommodate larger HIT Suite IDs if

the 16 available values prove insufficient. In that case, one of the 16 values, zero, will be used to indicate that four additional bits of the ORCHID will be used to encode the HIT Suite ID. Hence, the current four-bit HIT Suite-IDs only use the four higher order bits in the ID field. Future documents may define the use of the four lower-order bits in the ID field.

The following HIT Suites ID are defined:

HIT Suite	ID	
RESERVED	0	
RSA,DSA/SHA-256	1	(REQUIRED)
ECDSA/SHA-384	2	(RECOMMENDED)
ECDSA_LOW/SHA-1	3	(RECOMMENDED)

5.2.11. TRANSPORT_FORMAT_LIST

The TRANSPORT_FORMAT_LIST parameter contains a list of the supported HIP transport formats (TFs) of the Responder. The Responder sends the TRANSPORT_FORMAT_LIST in the signed part of the R1 packet. Based on the TRANSPORT_FORMAT_LIST, the Initiator chooses one suitable transport format and includes the respective HIP transport format parameter in its response packet.

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          TF type #1           |           TF type #2          /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/          TF type #n           |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	2049
Length	2x number of TF types
TF Type	defines a transport format (TF) type supported by the host. The TF type numbers correspond to the HIP parameter type numbers of the respective transform parameters. The list of TF types is ordered by preference of the sender

The TF type numbers index the respective HIP parameters for the transport formats in the type number range between 2050 to 4095. The parameters and their use is defined in separate documents. Currently, the only transport format defined is IPsec ESP [I-D.ietf-hip-rfc5202-bis].

For each listed TF type, the sender of the parameter MUST include the repective transport form parameter in the HIP packet. The TF type in the TRANSPORT_FORM_LIST MUST be ignored if no matching transport form parameter is present in the packet.

5.2.12. HIP_MAC

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    |                             HMAC                              |
    /                                                               /
    /                               +-------------------------------+
    |                               |            Padding            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	61505
Length	length in octets, excluding Type, Length, and Padding
HMAC	HMAC computed over the HIP packet, excluding the HIP_MAC parameter and any following parameters, such as HIP_SIGNATURE, HIP_SIGNATURE_2, ECHO_REQUEST_UNSIGNED, or ECHO_RESPONSE_UNSIGNED. The checksum field MUST be set to zero and the HIP header length in the HIP common header MUST be calculated not to cover any excluded parameters when the HMAC is calculated. The size of the HMAC is the natural size of the hash computation output depending on the used hash function.

The HMAC uses RHASH as hash algorithm. The calculation and verification process is presented in Section 6.4.1.

5.2.13. HIP_MAC_2

The HIP_MAC_2 is a MAC of the packet and the HOST_ID parameter of the sender while only the packet without HOST_ID of the sender is sent. The parameter structure is the same as in Section 5.2.12. The fields are:

Type	61569
Length	length in octets, excluding Type, Length, and Padding
HMAC	HMAC computed over the HIP packet, excluding the HIP_MAC_2 parameter and any following parameters such as HIP_SIGNATURE, HIP_SIGNATURE_2, ECHO_REQUEST_UNSIGNED, or ECHO_RESPONSE_UNSIGNED, and including an additional sender's HOST_ID parameter during the HMAC calculation. The checksum field MUST be set to zero and the HIP header length in the HIP common header MUST be calculated not to cover any excluded parameters when the HMAC is calculated. The size of the HMAC is the natural size of the hash computation output depending on the used hash function.

The HMAC uses RHASH as hash algorithm. The calculation and verification process is presented in Section 6.4.1.

5.2.14. HIP_SIGNATURE

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |    SIG alg                    |            Signature          /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                               |             Padding           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	61697
Length	length in octets, excluding Type, Length, and Padding
SIG alg	signature algorithm
Signature	the signature is calculated over the HIP packet, excluding the HIP_SIGNATURE parameter and any parameters that follow the HIP_SIGNATURE parameter. When the signature is calculated the checksum field MUST be set to zero, and the HIP header length in the HIP common header MUST be calculated only up to the beginning of the HIP_SIGNATURE parameter.

The signature algorithms are defined in Section 5.2.9. The signature in the Signature field is encoded using the method depending on the signature algorithm (e.g., according to [RFC3110] in case of RSA/SHA-1, according to [RFC5702] in case of RSA/SHA-256, according to [RFC2536] in case of DSA, or according to [RFC6090] in case of

ECDSA).

The HIP_SIGNATURE calculation and verification process are presented in Section 6.4.2.

5.2.15. HIP_SIGNATURE_2

The HIP_SIGNATURE_2 excludes the variable parameters in the R1 packet to allow R1 pre-creation. The parameter structure is the same as in Section 5.2.14. The fields are:

Type	61633
Length	length in octets, excluding Type, Length, and Padding
SIG alg	signature algorithm
Signature	Within the R1 packet that contains the HIP_SIGNATURE_2 parameter, the Initiator's HIT, the checksum field, and the Opaque and Random #I fields in the PUZZLE parameter MUST be set to zero while computing the HIP_SIGNATURE_2 signature. Further, the HIP packet length in the HIP header MUST be adjusted as if the HIP_SIGNATURE_2 was not in the packet during the signature calculation, i.e., the HIP packet length points to the beginning of the HIP_SIGNATURE_2 parameter during signing and verification.

Zeroing the Initiator's HIT makes it possible to create R1 packets beforehand, to minimize the effects of possible DoS attacks. Zeroing the Random #I and Opaque fields within the PUZZLE parameter allows these fields to be populated dynamically on precomputed R1s.

Signature calculation and verification follows the process defined in Section 6.4.2.

5.2.16. SEQ

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Update ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Type            385
Length          8
Update ID       32-bit sequence number
```

The Update ID is an unsigned number in network byte order, initialized by a host to zero upon moving to ESTABLISHED state. The Update ID has scope within a single HIP association, and not across multiple associations or multiple hosts. The Update ID is incremented by one before each new UPDATE that is sent by the host; the first UPDATE packet originated by a host has an Update ID of 0.

5.2.17. ACK

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       peer Update ID 1                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                       peer Update ID n                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	449
Length	length in octets, excluding Type and Length
peer Update ID	32-bit sequence number corresponding to the Update ID being ACKed.

The ACK parameter includes one or more Update IDs that have been received from the peer. The number of peer Update IDs can be inferred from the length by dividing it by 4.

5.2.18. ENCRYPTED

```
    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              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Reserved                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              IV                               /
   /                                                               /
   /                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               /
   /                        Encrypted data                         /
   /                                                               /
   /                               +-------------------------------+
   /                               |            Padding            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	641
Length	length in octets, excluding Type, Length, and Padding
Reserved	zero when sent, ignored when received
IV	Initialization vector, if needed, otherwise nonexistent. The length of the IV is inferred from the HIP_CIPHER.
Encrypted data	The data is encrypted using the encryption algorithm defined in the HIP_CIPHER parameter.

The ENCRYPTED parameter encapsulates other parameters, the encrypted data, which holds one or more HIP parameters in block encrypted form.

Consequently, the first fields in the encapsulated parameter(s) are Type and Length of the first such parameter, allowing the contents to be easily parsed after decryption.

The field labeled "Encrypted data" consists of the output of one or more HIP parameters concatenated together that have been passed through an encryption algorithm. Each of these inner parameters is padded according to the rules of Section 5.2.1 for padding individual parameters. As a result, the concatenated parameters will be a block of data that is 8-byte aligned.

Some encryption algorithms require that the data to be encrypted must be a multiple of the cipher algorithm block size. In this case, the above block of data MUST include additional padding, as specified by the encryption algorithm. The size of the extra padding is selected so that the length of the unencrypted data block is a multiple of the

cipher block size. The encryption algorithm may specify padding bytes other than zero; for example, AES [FIPS.197.2001] uses the PKCS5 padding scheme (see section 6.1.1 of [RFC2898]) where the remaining n bytes to fill the block each have the value of n. This yields an "unencrypted data" block that is transformed to an "encrypted data" block by the cipher suite. This extra padding added to the set of parameters to satisfy the cipher block alignment rules is not counted in HIP TLV length fields, and this extra padding should be removed by the cipher suite upon decryption.

Note that the length of the cipher suite output may be smaller or larger than the length of the set of parameters to be encrypted, since the encryption process may compress the data or add additional padding to the data.

Once this encryption process is completed, the Encrypted data field is ready for inclusion in the parameter. If necessary, additional Padding for 8-byte alignment is then added according to the rules of Section 5.2.1.

5.2.19. NOTIFICATION

The NOTIFICATION parameter is used to transmit informational data, such as error conditions and state transitions, to a HIP peer. A NOTIFICATION parameter may appear in NOTIFY packets. The use of the NOTIFICATION parameter in other packet types is for further study.

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Reserved             |      Notify Message Type      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               /
    /                   Notification Data                           /
    /                                               +---------------+
    /                                               |     Padding   |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	832
Length	length in octets, excluding Type, Length, and Padding
Reserved	zero when sent, ignored when received
Notify Message Type	specifies the type of notification
Notification Data	informational or error data transmitted in addition to the Notify Message Type. Values for this field are type specific (see below).
	multiple of 8 bytes.

Notification information can be error messages specifying why an HIP Security Association could not be established. It can also be status data that a HIP implementation wishes to communicate with a peer process. The table below lists the notification messages and their Notification Message Types. HIP packets MAY contain multiple NOTIFICATION parameters if several problems exist or several independent pieces of information must be transmitted.

To avoid certain types of attacks, a Responder SHOULD avoid sending a NOTIFICATION to any host with which it has not successfully verified a puzzle solution.

Notify Message Types in the range 0-16383 are intended for reporting errors and in the range 16384-65535 for other status information. An implementation that receives a NOTIFY packet with a Notify Message Type that indicates an error in response to a request packet (e.g., I1, I2, UPDATE) SHOULD assume that the corresponding request has failed entirely. Unrecognized error types MUST be ignored except that they SHOULD be logged.

As currently defined, Notify Message Type values 1-10 are used for informing about errors in packet structures, values 11-20 for informing about problems in parameters.

Notification Data in NOTIFICATION parameters with status Notify Message Types MUST be ignored if not recognized.

Notify Message Types - Errors Value

UNSUPPORTED_CRITICAL_PARAMETER_TYPE 1

Sent if the parameter type has the "critical" bit set and the parameter type is not recognized. Notification Data contains the two-octet parameter type.

INVALID_SYNTAX 7

Indicates that the HIP message received was invalid because some type, length, or value was out of range or because the request was otherwise malformed. To avoid a denial- of-service attack using forged messages, this status may only be returned for packets whose HIP_MAC (if present) and SIGNATURE have been verified. This status MUST be sent in response to any error not covered by one of the other status types, and SHOULD NOT contain details to avoid leaking information to someone probing a node. To aid debugging, more detailed error information SHOULD be written to a console or log.

NO_DH_PROPOSAL_CHOSEN 14

None of the proposed group IDs was acceptable.

INVALID_DH_CHOSEN 15

The DH Group ID field does not correspond to one offered by the Responder.

NO_HIP_PROPOSAL_CHOSEN 16

None of the proposed HIT Suites or HIP Encryption Algorithms was acceptable.

INVALID_HIP_CIPHER_CHOSEN 17

The HIP_CIPHER Crypto ID does not correspond to one offered by the Responder.

UNSUPPORTED_HIT_SUITE 20

Sent in response to an I1 or R1 packet for which the HIT suite is not supported.

AUTHENTICATION_FAILED 24

Sent in response to a HIP signature failure, except when the signature verification fails in a NOTIFY message.

CHECKSUM_FAILED 26

Sent in response to a HIP checksum failure.

HIP_MAC_FAILED 28

Sent in response to a HIP HMAC failure.

ENCRYPTION_FAILED 32

The Responder could not successfully decrypt the ENCRYPTED parameter.

INVALID_HIT 40

Sent in response to a failure to validate the peer's HIT from the corresponding HI.

BLOCKED_BY_POLICY 42

The Responder is unwilling to set up an association for some policy reason (e.g., received HIT is NULL and policy does not allow opportunistic mode).

RESPONDER_BUSY_PLEASE_RETRY 44

The Responder is unwilling to set up an association as it is suffering under some kind of overload and has chosen to shed load by rejecting the Initiator's request. The Initiator may retry; however, the Initiator MUST find another (different) puzzle solution for any such retries. Note that the Initiator may need to obtain a new puzzle with a new I1/R1 exchange.

Notify Message Types - Status	Value
I2_ACKNOWLEDGEMENT	16384

The Responder has an I2 packet from the Initiator but had to queue the I2 packet for processing. The puzzle was correctly solved and the Responder is willing to set up an association but currently has a number of I2 packets in the processing queue. The R2 packet is sent after the I2 packet was processed.

5.2.20. ECHO_REQUEST_SIGNED

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Opaque data (variable length)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	897
Length	length of the opaque data in octets
Opaque data	opaque data, supposed to be meaningful only to the node that sends ECHO_REQUEST_SIGNED and receives a corresponding ECHO_RESPONSE_SIGNED or ECHO_RESPONSE_UNSIGNED.

The ECHO_REQUEST_SIGNED parameter contains an opaque blob of data that the sender wants to get echoed back in the corresponding reply packet.

The ECHO_REQUEST_SIGNED and corresponding echo response parameters MAY be used for any purpose where a node wants to carry some state in a request packet and get it back in a response packet. The ECHO_REQUEST_SIGNED is covered by the HIP_MAC and SIGNATURE. A HIP packet can contain only one ECHO_REQUEST_SIGNED parameter and MAY contain multiple ECHO_REQUEST_UNSIGNED parameter. The ECHO_REQUEST_SIGNED parameter MUST be responded to with an ECHO_RESPONSE_SIGNED.

5.2.21. ECHO_REQUEST_UNSIGNED

```
 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              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Opaque data (variable length)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	63661
Length	length of the opaque data in octets
Opaque data	opaque data, supposed to be meaningful only to the node that sends ECHO_REQUEST_UNSIGNED and receives a corresponding ECHO_RESPONSE_UNSIGNED.

The ECHO_REQUEST_UNSIGNED parameter contains an opaque blob of data that the sender wants to get echoed back in the corresponding reply

packet.

The ECHO_REQUEST_UNSIGNED and corresponding echo response parameters MAY be used for any purpose where a node wants to carry some state in a request packet and get it back in a response packet. The ECHO_REQUEST_UNSIGNED is not covered by the HIP_MAC and SIGNATURE. A HIP packet can contain one or more ECHO_REQUEST_UNSIGNED parameters. It is possible that middleboxes add ECHO_REQUEST_UNSIGNED parameters in HIP packets passing by. The creator of the ECHO_REQUEST_UNSIGNED (end-host or middlebox) has to create the Opaque field so that it can later identify and remove the corresponding ECHO_RESPONSE_UNSIGNED parameter.

The ECHO_REQUEST_UNSIGNED parameter MUST be responded to with an ECHO_RESPONSE_UNSIGNED parameter.

5.2.22. ECHO_RESPONSE_SIGNED

```
  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              |             Length            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                 Opaque data (variable length)                 |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	961
Length	length of the opaque data in octets
Opaque data	opaque data, copied unmodified from the ECHO_REQUEST_SIGNED or ECHO_REQUEST_UNSIGNED parameter that triggered this response.

The ECHO_RESPONSE_SIGNED parameter contains an opaque blob of data that the sender of the ECHO_REQUEST_SIGNED wants to get echoed back. The opaque data is copied unmodified from the ECHO_REQUEST_SIGNED parameter.

The ECHO_REQUEST_SIGNED and ECHO_RESPONSE_SIGNED parameters MAY be used for any purpose where a node wants to carry some state in a request packet and get it back in a response packet. The ECHO_RESPONSE_SIGNED is covered by the HIP_MAC and SIGNATURE.

5.2.23. ECHO_RESPONSE_UNSIGNED

```
     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              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                 Opaque data (variable length)                 |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	63425
Length	length of the opaque data in octets
Opaque data	opaque data, copied unmodified from the ECHO_REQUEST_SIGNED or ECHO_REQUEST_UNSIGNED parameter that triggered this response.

The ECHO_RESPONSE_UNSIGNED parameter contains an opaque blob of data that the sender of the ECHO_REQUEST_SIGNED or ECHO_REQUEST_UNSIGNED wants to get echoed back. The opaque data is copied unmodified from the corresponding echo request parameter.

The echo request and ECHO_RESPONSE_UNSIGNED parameters MAY be used for any purpose where a node wants to carry some state in a request packet and get it back in a response packet. The ECHO_RESPONSE_UNSIGNED is not covered by the HIP_MAC and SIGNATURE.

5.3. HIP Packets

There are eight basic HIP packets (see Table 10). Four are for the HIP base exchange, one is for updating, one is for sending notifications, and two are for closing a HIP association.

Packet type	Packet name
1	I1 - the HIP Initiator Packet
2	R1 - the HIP Responder Packet
3	I2 - the Second HIP Initiator Packet
4	R2 - the Second HIP Responder Packet
16	UPDATE - the HIP Update Packet
17	NOTIFY - the HIP Notify Packet
18	CLOSE - the HIP Association Closing Packet
19	CLOSE_ACK - the HIP Closing Acknowledgment Packet

Table 10: HIP packets and packet type values

Packets consist of the fixed header as described in Section 5.1, followed by the parameters. The parameter part, in turn, consists of zero or more TLV-coded parameters.

In addition to the base packets, other packet types may be defined later in separate specifications. For example, support for mobility and multi-homing is not included in this specification.

See Notation (Section 2.2) for the notation used in the operations.

In the future, an optional upper-layer payload MAY follow the HIP header. The Next Header field in the header indicates if there is additional data following the HIP header. The HIP packet, however, MUST NOT be fragmented. This limits the size of the possible additional data in the packet.

5.3.1. I1 - the HIP Initiator Packet

The HIP header values for the I1 packet:

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

```
IP ( HIP ( DH_GROUP_LIST ) )
```

The I1 packet contains the fixed HIP header and the Initiator's DH_GROUP_LIST.

Valid control bits: none

The Initiator receives the Responder's HIT either from a DNS lookup of the Responder's FQDN (see 5205-bis), from some other repository, or from a local table. If the Initiator does not know the Responder's HIT, it may attempt to use opportunistic mode by using NULL (all zeros) as the Responder's HIT. See also "HIP Opportunistic Mode" (Section 4.1.8).

Since the I1 packet is so easy to spoof even if it were signed, no attempt is made to add to its generation or processing cost.

The Initiator includes a DH_GROUP_LIST parameter in the I1 packet to inform the Responder of its preferred DH Group IDs. Note that the DH_GROUP_LIST in the I1 packet is not protected by a signature.

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

5.3.2. 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 ( [ R1_COUNTER, ]
           PUZZLE,
           DIFFIE_HELLMAN,
           HIP_CIPHER,
           HOST_ID,
           HIT_SUITE_LIST,
           DH_GROUP_LIST,
           [ ECHO_REQUEST_SIGNED, ]
           TRANSPORT_FORMAT_LIST,
           HIP_SIGNATURE_2 )
           <, ECHO_REQUEST_UNSIGNED >i)
```

Valid control bits: A

If the Responder's HI is an anonymous one, the A control MUST be set.

The Initiator's HIT MUST match the one received in the I1 packet if the R1 is a response to an I1. If the Responder has multiple HIs, the Responder's HIT used MUST match Initiator's request. If the Initiator used opportunistic mode, the Responder may select freely among its HIs. See also "HIP Opportunistic Mode" (Section 4.1.8).

The R1 packet generation counter is used to determine the currently valid generation of puzzles. The value is increased periodically, and it is RECOMMENDED that it is increased at least as often as solutions to old puzzles are no longer accepted.

The Puzzle contains a Random #I and the difficulty #K. The difficulty #K indicates the number of lower-order bits, in the puzzle hash result, that must be zeros; see Section 4.1.2. The Random #I is not covered by the signature and must be zeroed during the signature calculation, allowing the sender to select and set the #I into a precomputed R1 packet just prior sending it to the peer.

The Responder selects the Diffie-Hellman public value based on the Initiator's preference expressed in the DH_GROUP_LIST parameter in the I1 packet. The Responder sends back its own preference based on which it chose the DH public value as DH_GROUP_LIST. This allows the Initiator to determine whether its own DH_GROUP_LIST in the sent I1 packet was manipulated by an attacker.

The Diffie-Hellman public value is ephemeral, and values SHOULD NOT be reused across different HIP sessions. Once the Responder has received a valid response to an R1 packet, that Diffie-Hellman value SHOULD be deprecated. It it is possible that the Responder has sent the same Diffie-Hellman value to different hosts simultaneously in corresponding R1 packets and those responses should also be accepted. However, as a defense against I1 packet storms, an implementation MAY propose, and re-use unless avoidable, the same Diffie-Hellman value for a period of time, for example, 15 minutes. By using a small number of different puzzles for a given Diffie-Hellman value, the R1 packets can be precomputed and delivered as quickly as I1 packets arrive. A scavenger process should clean up unused Diffie-Hellman values and puzzles.

Re-using Diffie-Hellman public values opens up the potential security risk of more than one Initiator ending up with the same keying material (due to faulty random number generators). Also, more than one Initiator using the same Responder public key half may lead to potentially easier cryptographic attacks and to imperfect forward security.

However, these risks involved in re-using the same public value are statistical; that is, the authors are not aware of any mechanism that would allow manipulation of the protocol so that the risk of the re-use of any given Responder Diffie-Hellman public key would differ from the base probability. Consequently, it is RECOMMENDED that Responders avoid re-using the same DH key with multiple Initiators, but because the risk is considered statistical and not known to be manipulable, the implementations MAY re-use a key in order to ease resource-constrained implementations and to increase the probability of successful communication with legitimate clients even under an I1 packet storm. In particular, when it is too expensive to generate enough precomputed R1 packets to supply each potential Initiator with a different DH key, the Responder MAY send the same DH key to several Initiators, thereby creating the possibility of multiple legitimate Initiators ending up using the same Responder-side public key. However, as soon as the Responder knows that it will use a particular DH key, it SHOULD stop offering it. This design is aimed to allow resource-constrained Responders to offer services under I1 packet storms and to simultaneously make the probability of DH key re-use both statistical and as low as possible.

If the Responder uses the same DH keypair for multiple handshakes. It must take care to avoid small subgroup attacks [RFC2785]. To avoid these attacks, when receiving the I2 message, the Responder SHOULD validate the Initiators DH public key as described in [RFC2785] Section 3.1. In case the validation fails, the Responder MUST NOT generate a DH shared key and MUST silently abort the HIP BEX.

The HIP_CIPHER contains the encryption algorithms supported by the Responder to encrypt the contents of the ENCRYPTED parameter, in the order of preference. All implementations MUST support AES [RFC3602].

The HIT_SUITE_LIST parameter is an ordered list of the Responder's preferred and supported HIT Suites. The list allows the Initiator to determine whether its own source HIT matches any suite supported by the Responder.

The ECHO_REQUEST_SIGNED and ECHO_REQUEST_UNSIGNED parameters contain data that the sender wants to receive unmodified in the corresponding response packet in the ECHO_RESPONSE_SIGNED or ECHO_RESPONSE_UNSIGNED parameter. The R1 packet may contain zero or more ECHO_REQUEST_UNSIGNED parameters as described in Section Section 5.2.21.

The TRANSPORT_FORMAT_LIST parameter is an ordered list of the Responder's preferred and supported transport format types. The list allows the Initiator and the Responder to agree on a common type for

payload protection. This parameter is described in Section 5.2.11.

The signature is calculated over the whole HIP packet as described in Section 5.2.15. This allows the Responder to use precomputed R1s. The Initiator SHOULD validate this signature. It MUST check that the Responder's HI matches with the one expected, if any.

5.3.3. I2 - the Second HIP Initiator Packet

The HIP header values for the I2 packet:

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

  IP ( HIP ( [R1_COUNTER,]
             SOLUTION,
             DIFFIE_HELLMAN,
             HIP_CIPHER,
             ENCRYPTED { HOST_ID } or HOST_ID,
             [ ECHO_RESPONSE_SIGNED ,]
             TRANSPORT_FORMAT_LIST,
             HIP_MAC,
             HIP_SIGNATURE
             <, ECHO_RESPONSE_UNSIGNED>i ) )
```

Valid control bits: A

The HITs used MUST match the ones used in the R1.

If the Initiator's HI is an anonymous one, the A control MUST be set.

If present in the I1 packet, the Initiator MUST include an unmodified copy of the R1_COUNTER parameter received in the corresponding R1 packet into the I2 packet.

The Solution contains the Random #I from R1 and the computed #J. The low-order #K bits of the RHASH(I | ... | J) MUST be zero.

The Diffie-Hellman value is ephemeral. If precomputed, a scavenger process should clean up unused Diffie-Hellman values. The Responder MAY re-use Diffie-Hellman values under some conditions as specified in Section 5.3.2.

The HIP_CIPHER contains the single encryption transform selected by the Initiator, that it uses to encrypt the ENCRYPTED parameters. The chosen cipher MUST correspond to one of the ciphers offered by the

Responder in the R1. All implementations MUST support AES [RFC3602].

The Initiator's HI MAY be encrypted using the HIP_CIPHER encryption algorithm. The keying material is derived from the Diffie-Hellman exchanged as defined in Section 6.5.

The ECHO_RESPONSE_SIGNED and ECHO_RESPONSE_UNSIGNED contain the unmodified Opaque data copied from the corresponding echo request parameter(s).

The TRANSPORT_FORMAT_LIST contains the single transport format type selected by the Initiator. The chosen type MUST correspond to one of the types offered by the Responder in the R1. Currently, the only transport format defined is the ESP transport format ([I-D.ietf-hip-rfc5202-bis]).

The HMAC value in the HIP_MAC parameter is calculated over the whole HIP packet, excluding any parameters after the HIP_MAC, as described in Section 6.4.1. The Responder MUST validate the HIP_MAC.

The signature is calculated over the whole HIP packet, excluding any parameters after the HIP_SIGNATURE, as described in Section 5.2.14. The Responder MUST validate this signature. The Responder uses the HI in the packet or a HI acquired by some other means for verifying the signature.

5.3.4. R2 - the Second HIP Responder Packet

The HIP header values for the R2 packet:

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

IP ( HIP ( HIP_MAC_2, HIP_SIGNATURE ) )
```

Valid control bits: none

The HIP_MAC_2 is calculated over the whole HIP packet, with Responder's HOST_ID parameter concatenated with the HIP packet. The HOST_ID parameter is removed after the HMAC calculation. The procedure is described in Section 6.4.1.

The signature is calculated over the whole HIP packet.

The Initiator MUST validate both the HIP_MAC and the signature.

5.3.5. UPDATE - the HIP Update Packet

Support for the UPDATE packet is MANDATORY.

The HIP header values for the UPDATE packet:

```
Header:
  Packet Type = 16
  SRC HIT = Sender's HIT
  DST HIT = Recipient's HIT

IP ( HIP ( [SEQ, ACK, ] HIP_MAC, HIP_SIGNATURE ) )
```

Valid control bits: None

The UPDATE packet contains mandatory HIP_MAC and HIP_SIGNATURE parameters, and other optional parameters.

The UPDATE packet contains zero or one SEQ parameter. The presence of a SEQ parameter indicates that the receiver MUST acknowledge the the UPDATE. An UPDATE that does not contain a SEQ but only an ACK parameter is simply an acknowledgment of a previous UPDATE and itself MUST NOT be acknowledged by a separate ACK. Such UPDATE packets containing only an ACK parameter do not require processing in relative order to other UPDATE packets. An UPDATE packet without either a SEQ or an ACK parameter is invalid; such unacknowledged updates MUST instead use a NOTIFY packet.

An UPDATE packet contains zero or one ACK parameters. The ACK parameter echoes the SEQ sequence number of the UPDATE packet being ACKed. A host MAY choose to acknowledge more than one UPDATE packet at a time; e.g., the ACK may contain the last two SEQ values received, for resilience against ACK loss. ACK values are not cumulative; each received unique SEQ value requires at least one corresponding ACK value in reply. Received ACKs that are redundant are ignored. Hosts MUST implement the processing of ACKs with multiple SEQ numbers even if they do not implement sending ACKs with multiple SEQ numbers.

The UPDATE packet may contain both a SEQ and an ACK parameter. In this case, the ACK is being piggybacked on an outgoing UPDATE. In general, UPDATEs carrying SEQ SHOULD be ACKed upon completion of the processing of the UPDATE. A host MAY choose to hold the UPDATE carrying ACK for a short period of time to allow for the possibility of piggybacking the ACK parameter, in a manner similar to TCP delayed acknowledgments.

A sender MAY choose to forgo reliable transmission of a particular

UPDATE (e.g., it becomes overcome by events). The semantics are such that the receiver MUST acknowledge the UPDATE, but the sender MAY choose to not care about receiving the ACK.

UPDATEs MAY be retransmitted without incrementing SEQ. If the same subset of parameters is included in multiple UPDATEs with different SEQs, the host MUST ensure that the receiver's processing of the parameters multiple times will not result in a protocol error.

5.3.6. NOTIFY - the HIP Notify Packet

Implementing the NOTIFY packet is optional. The NOTIFY packet MAY be used to provide information to a peer. Typically, NOTIFY is used to indicate some type of protocol error or negotiation failure. NOTIFY packets are unacknowledged. The receiver can handle the packet only as informational, and SHOULD NOT change its HIP state (see Section 4.4.2) based purely on a received NOTIFY packet.

The HIP header values for the NOTIFY packet:

```
Header:
  Packet Type = 17
  SRC HIT = Sender's HIT
  DST HIT = Recipient's HIT, or zero if unknown

IP ( HIP (<NOTIFICATION>i, [HOST_ID, ] HIP_SIGNATURE) )
```

Valid control bits: None

The NOTIFY packet is used to carry one or more NOTIFICATION parameters.

5.3.7. CLOSE - the HIP Association Closing Packet

The HIP header values for the CLOSE packet:

```
Header:
  Packet Type = 18
  SRC HIT = Sender's HIT
  DST HIT = Recipient's HIT

IP ( HIP ( ECHO_REQUEST_SIGNED, HIP_MAC, HIP_SIGNATURE ) )
```

Valid control bits: none

The sender MUST include an ECHO_REQUEST_SIGNED used to validate CLOSE_ACK received in response, and both a HIP_MAC and a signature

(calculated over the whole HIP packet).

The receiver peer MUST reply with a CLOSE_ACK containing an ECHO_RESPONSE_SIGNED corresponding to the received ECHO_REQUEST_SIGNED.

5.3.8. CLOSE_ACK - the HIP Closing Acknowledgment Packet

The HIP header values for the CLOSE_ACK packet:

```
  Header:
    Packet Type = 19
    SRC HIT = Sender's HIT
    DST HIT = Recipient's HIT

  IP ( HIP ( ECHO_RESPONSE_SIGNED, HIP_MAC, HIP_SIGNATURE ) )
```

Valid control bits: none

The sender MUST include both an HMAC and signature (calculated over the whole HIP packet).

The receiver peer MUST validate the ECHO_RESPONSE_SIGNED and validate both the HIP_MAC and the signature if the receiver has state for a HIP association.

5.4. ICMP Messages

When a HIP implementation detects a problem with an incoming packet, and it either cannot determine the identity of the sender of the packet or does not have any existing HIP association with the sender of the packet, it MAY respond with an ICMP packet. Any such replies MUST be rate-limited as described in [RFC4443]. In most cases, the ICMP packet has the Parameter Problem type (12 for ICMPv4, 4 for ICMPv6), with the Pointer field pointing to the field that caused the ICMP message to be generated.

5.4.1. Invalid Version

If a HIP implementation receives a HIP packet that has an unrecognized HIP version number, it SHOULD respond, rate-limited, with an ICMP packet with type Parameter Problem, with the Pointer pointing to the Version/RES. byte in the HIP header.

5.4.2. Other Problems with the HIP Header and Packet Structure

If a HIP implementation receives a HIP packet that has other unrecoverable problems in the header or packet format, it MAY

respond, rate-limited, with an ICMP packet with type Parameter Problem, the Pointer pointing to the field that failed to pass the format checks. However, an implementation MUST NOT send an ICMP message if the checksum fails; instead, it MUST silently drop the packet.

5.4.3. Invalid Puzzle Solution

If a HIP implementation receives an I2 packet that has an invalid puzzle solution, the behavior depends on the underlying version of IP. If IPv6 is used, the implementation SHOULD respond with an ICMP packet with type Parameter Problem, the Pointer pointing to the beginning of the Puzzle solution #J field in the SOLUTION payload in the HIP message.

If IPv4 is used, the implementation MAY respond with an ICMP packet with the type Parameter Problem, copying enough of bytes from the I2 message so that the SOLUTION parameter fits into the ICMP message, the Pointer pointing to the beginning of the Puzzle solution #J field, as in the IPv6 case. Note, however, that the resulting ICMPv4 message exceeds the typical ICMPv4 message size as defined in [RFC0792].

5.4.4. Non-Existing HIP Association

If a HIP implementation receives a CLOSE or UPDATE packet, or any other packet whose handling requires an existing association, that has either a Receiver or Sender HIT that does not match with any existing HIP association, the implementation MAY respond, rate-limited, with an ICMP packet with the type Parameter Problem. The Pointer of the ICMP Parameter Problem packet is set pointing to the beginning of the first HIT that does not match.

A host MUST NOT reply with such an ICMP if it receives any of the following messages: I1, R2, I2, R2, and NOTIFY packet. When introducing new packet types, a specification SHOULD define the appropriate rules for sending or not sending this kind of ICMP reply.

6. Packet Processing

Each host is assumed to have a single HIP protocol implementation that manages the host's HIP associations and handles requests for new ones. Each HIP association is governed by a conceptual state machine, with states defined above in Section 4.4. The HIP implementation can simultaneously maintain HIP associations with more than one host. Furthermore, the HIP implementation may have more than one active HIP association with another host; in this case, HIP associations are distinguished by their respective HITs. It is not

possible to have more than one HIP association between any given pair of HITs. Consequently, the only way for two hosts to have more than one parallel association is to use different HITs, at least at one end.

The processing of packets depends on the state of the HIP association(s) with respect to the authenticated or apparent originator of the packet. A HIP implementation determines whether it has an active association with the originator of the packet based on the HITs. In the case of user data carried in a specific transport format, the transport format document specifies how the incoming packets are matched with the active associations.

6.1. Processing Outgoing Application Data

In a HIP host, an application can send application-level data using an identifier specified via the underlying API. The API can be a backwards-compatible API (see [RFC5338]), using identifiers that look similar to IP addresses, or a completely new API, providing enhanced services related to Host Identities. Depending on the HIP implementation, the identifier provided to the application may be different; for example, it can be a HIT or an IP address.

The exact format and method for transferring the user data from the source HIP host to the destination HIP host is defined in the corresponding transport format document. The actual data is transferred in the network using the appropriate source and destination IP addresses.

In this document, conceptual processing rules are defined only for the base case where both hosts have only single usable IP addresses; the multi-address multi-homing case is specified separately.

The following conceptual algorithm describes the steps that are required for handling outgoing datagrams destined to a HIT.

1. If the datagram has a specified source address, it MUST be a HIT. If it is not, the implementation MAY replace the source address with a HIT. Otherwise, it MUST drop the packet.

2. If the datagram has an unspecified source address, the implementation MUST choose a suitable source HIT for the datagram. Selecting the source HIT is subject to local policy.

3. If there is no active HIP association with the given <source, destination> HIT pair, one MUST be created by running the base exchange. While waiting for the base exchange to complete, the implementation SHOULD queue at least one packet per HIP

association to be formed, and it MAY queue more than one.

4. Once there is an active HIP association for the given <source, destination> HIT pair, the outgoing datagram is passed to transport handling. The possible transport formats are defined in separate documents, of which the ESP transport format for HIP is mandatory for all HIP implementations.

5. Before sending the packet, the HITs in the datagram are replaced with suitable IP addresses. For IPv6, the rules defined in [RFC3484] SHOULD be followed. Note that this HIT-to-IP-address conversion step MAY also be performed at some other point in the stack, e.g., before wrapping the packet into the output format.

6.2. Processing Incoming Application Data

The following conceptual algorithm describes the incoming datagram handling when HITs are used at the receiving host as application-level identifiers. More detailed steps for processing packets are defined in corresponding transport format documents.

1. The incoming datagram is mapped to an existing HIP association, typically using some information from the packet. For example, such mapping may be based on the ESP Security Parameter Index (SPI).

2. The specific transport format is unwrapped, in a way depending on the transport format, yielding a packet that looks like a standard (unencrypted) IP packet. If possible, this step SHOULD also verify that the packet was indeed (once) sent by the remote HIP host, as identified by the HIP association.

   Depending on the used transport mode, the verification method can vary. While the HI (as well as HIT) is used as the higher-layer identifier, the verification method has to verify that the data packet was sent by the correct node identity and that the actual identity maps to this particular HIT. When using ESP transport format [I-D.ietf-hip-rfc5202-bis], the verification is done using the SPI value in the data packet to find the corresponding SA with associated HIT and key, and decrypting the packet with that associated key.

3. The IP addresses in the datagram are replaced with the HITs associated with the HIP association. Note that this IP-address-to-HIT conversion step MAY also be performed at some other point in the stack.

4. The datagram is delivered to the upper layer (e.g., UDP or TCP). When demultiplexing the datagram, the right upper-layer socket is selected based on the HITs.

6.3. Solving the Puzzle

This subsection describes the details for solving the puzzle.

In the R1 packet, the values #I and #K are sent in network byte order. Similarly, in the I2 packet, the values #I and #J are sent in network byte order. The hash is created by concatenating, in network byte order, the following data, in the following order and using the RHASH algorithm:

n-bit random value #I (where n is RHASH_len), in network byte order, as appearing in the R1 and I2 packets.

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

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

n-bit random value #J (where n is RHASH_len), in network byte order, as appearing in the I2 packet.

In a valid response puzzle, the #K low-order bits of the resulting RHASH digest MUST be zero.

Notes:

i) The length of the data to be hashed is variable depending on the output length of the Responder's hash function RHASH.

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

iii) The order of the Initiator's and Responder's HITs are different in the R1 and I2 packets; see Section 5.1. Care must be taken to copy the values in the right order to the hash input.

iv) For a puzzle #I, there may exist multiple valid puzzle solutions #J.

The following procedure describes the processing steps involved, assuming that the Responder chooses to precompute the R1 packets:

```
Precomputation by the Responder:
   Sets up the puzzle difficulty #K.
   Creates a signed R1 and caches it.

Responder:
   Selects a suitable cached R1.
   Generates a random number #I.
   Sends #I and #K in an R1.
   Saves #I and #K for a Delta time.

Initiator:
   Generates repeated attempts to solve the puzzle until a matching
   #J is found:
   Ltrunc( RHASH( #I | HIT-I | HIT-R | #J ), #K ) == 0
   Sends #I and #J in an I2.


Responder:
   Verifies that the received #I is a saved one.
   Finds the right #K based on #I.
   Computes V := Ltrunc( RHASH( #I | HIT-I | HIT-R | #J ), #K )
   Rejects if V != 0
   Accept if V == 0
```

6.4. HIP_MAC and SIGNATURE Calculation and Verification

The following subsections define the actions for processing HIP_MAC, HIP_MAC_2, HIP_SIGNATURE and HIP_SIGNATURE_2 parameters. The HIP_MAC_2 parameter is contained in the R2 packet. The HIP_SIGNATURE_2 parameter is contained in the R1 packet. The HIP_SIGNATURE and HIP_MAC parameter are contained in other HIP control packets.

6.4.1. HMAC Calculation

The HMAC uses RHASH as underlying hash function. The type of RHASH depends on the HIT Suite of the Responder. Hence, HMAC-SHA-256 [RFC4868] is used for HIT Suite RSA/DSA/SHA-256, HMAC-SHA-1 [RFC2404] is used for HIT Suite ECDSA_LOW/SHA-1, and HMAC-SHA-384 [RFC4868] for HIT Suite ECDSA/SHA-384.

The following process applies both to the HIP_MAC and HIP_MAC_2 parameters. When processing HIP_MAC_2, the difference is that the HIP_MAC calculation includes a pseudo HOST_ID field containing the Responder's information as sent in the R1 packet earlier.

Both the Initiator and the Responder should take some care when verifying or calculating the HIP_MAC_2. Specifically, the Initiator

has to preserve the HOST_ID exactly as it was received in the R1 packet until it receives the HIP_MAC_2 in the R2 packet.

The scope of the calculation for HIP_MAC is:

HMAC: { HIP header | [Parameters] }

where Parameters include all HIP parameters of the packet that is being calculated with Type values ranging from 1 to (HIP_MAC's Type value - 1) and exclude parameters with Type values greater or equal to HIP_MAC's Type value.

During HIP_MAC calculation, the following applies:

- In the HIP header, the Checksum field is set to zero.
- In the HIP header, the Header Length field value is calculated to the beginning of the HIP_MAC parameter.

Parameter order is described in Section 5.2.1.

The scope of the calculation for HIP_MAC_2 is:

HIP_MAC_2: { HIP header | [Parameters] | HOST_ID }

where Parameters include all HIP parameters for the packet that is being calculated with Type values from 1 to (HIP_MAC_2's Type value - 1) and exclude parameters with Type values greater or equal to HIP_MAC_2's Type value.

During HIP_MAC_2 calculation, the following applies:

- In the HIP header, the Checksum field is set to zero.
- In the HIP header, the Header Length field value is calculated to the beginning of the HIP_MAC_2 parameter and increased by the length of the concatenated HOST_ID parameter length (including type and length fields).
- HOST_ID parameter is exactly in the form it was received in the R1 packet from the Responder.

Parameter order is described in Section 5.2.1, except that the HOST_ID parameter in this calculation is added to the end.

The HIP_MAC parameter is defined in Section 5.2.12 and the HIP_MAC_2 parameter in Section 5.2.13. The HMAC calculation and verification process (the process applies both to HIP_MAC and HIP_MAC_2 except

where HIP_MAC_2 is mentioned separately) is as follows:

Packet sender:

1. Create the HIP packet, without the HIP_MAC, HIP_SIGNATURE, HIP_SIGNATURE_2, or any other parameter with greater Type value than the HIP_MAC parameter has.

2. In case of HIP_MAC_2 calculation, add a HOST_ID (Responder) parameter to the end of the packet.

3. Calculate the Header Length field in the HIP header including the added HOST_ID parameter in case of HIP_MAC_2.

4. Compute the HMAC using either HIP-gl or HIP-lg integrity key retrieved from KEYMAT as defined in Section 6.5.

5. In case of HIP_MAC_2, remove the HOST_ID parameter from the packet.

6. Add the HIP_MAC parameter to the packet and any parameter with greater Type value than the HIP_MAC's (HIP_MAC_2's) that may follow, including possible HIP_SIGNATURE or HIP_SIGNATURE_2 parameters

7. Recalculate the Length field in the HIP header.

Packet receiver:

1. Verify the HIP header Length field.

2. Remove the HIP_MAC or HIP_MAC_2 parameter, as well as all other parameters that follow it with greater Type value including possible HIP_SIGNATURE or HIP_SIGNATURE_2 fields, saving the contents if they are needed later.

3. In case of HIP_MAC_2, build and add a HOST_ID parameter (with Responder information) to the packet. The HOST_ID parameter should be identical to the one previously received from the Responder.

4. Recalculate the HIP packet length in the HIP header and clear the Checksum field (set it to all zeros). In case of HIP_MAC_2, the length is calculated with the added HOST_ID parameter.

5. Compute the HMAC using either HIP-gl or HIP-lg integrity key as defined in Section 6.5 and verify it against the received HMAC.

6. Set Checksum and Header Length field in the HIP header to original values. Note that the checksum and length fields contain incorrect values after this step.

7. In case of HIP_MAC_2, remove the HOST_ID parameter from the packet before further processing.

6.4.2. Signature Calculation

The following process applies both to the HIP_SIGNATURE and HIP_SIGNATURE_2 parameters. When processing the HIP_SIGNATURE_2, the only difference is that instead of the HIP_SIGNATURE parameter, the HIP_SIGNATURE_2 parameter is used, and the Initiator's HIT and PUZZLE Opaque and Random #I fields are cleared (set to all zeros) before computing the signature. The HIP_SIGNATURE parameter is defined in Section 5.2.14 and the HIP_SIGNATURE_2 parameter in Section 5.2.15.

The scope of the calculation for HIP_SIGNATURE and HIP_SIGNATURE_2 is:

HIP_SIGNATURE: { HIP header | [Parameters] }

where Parameters include all HIP parameters for the packet that is being calculated with Type values from 1 to (HIP_SIGNATURE's Type value - 1).

During signature calculation, the following applies:

- In the HIP header, the Checksum field is set to zero.

- In the HIP header, the Header Length field value is calculated to the beginning of the HIP_SIGNATURE parameter.

The parameter order is described in Section 5.2.1.

HIP_SIGNATURE_2: { HIP header | [Parameters] }

where Parameters include all HIP parameters for the packet that is being calculated with Type values ranging from 1 to (HIP_SIGNATURE_2's Type value - 1).

During signature calculation, the following apply:

- In the HIP header, the Initiator's HIT field and Checksum fields are set to zero.

- In the HIP header, the Header Length field value is calculated to the beginning of the HIP_SIGNATURE_2 parameter.

- PUZZLE parameter's Opaque and Random #I fields are set to zero.

Parameter order is described in Section 5.2.1.

The signature calculation and verification process (the process applies both to HIP_SIGNATURE and HIP_SIGNATURE_2 except in the case where HIP_SIGNATURE_2 is separately mentioned) is as follows:

Packet sender:

1. Create the HIP packet without the HIP_SIGNATURE parameter or any other parameters that follow the HIP_SIGNATURE parameter.

2. Calculate the Length field and zero the Checksum field in the HIP header. In case of HIP_SIGNATURE_2, set Initiator's HIT field in the HIP header as well as PUZZLE parameter's Opaque and Random #I fields to zero.

3. Compute the signature using the private key corresponding to the Host Identifier (public key).

4. Add the HIP_SIGNATURE parameter to the packet.

5. Add any parameters that follow the HIP_SIGNATURE parameter.

6. Recalculate the Length field in the HIP header, and calculate the Checksum field.

Packet receiver:

1. Verify the HIP header Length field and checksum.

2. Save the contents of the HIP_SIGNATURE parameter and any other parameters following the HIP_SIGNATURE parameter and remove them from the packet.

3. Recalculate the HIP packet Length in the HIP header and clear the Checksum field (set it to all zeros). In case of HIP_SIGNATURE_2, set Initiator's HIT field in the HIP header as well as PUZZLE parameter's Opaque and Random #I fields to zero.

4. Compute the signature and verify it against the received signature using the packet sender's Host Identity (public key).

5. Restore the original packet by adding removed parameters (in step 2) and resetting the values that were set to zero (in step 3).

The verification can use either the HI received from a HIP packet,

the HI from a DNS query, if the FQDN has been received in the HOST_ID packet or one received by some other means.

6.5. HIP KEYMAT Generation

HIP keying material is derived from the Diffie-Hellman session key, Kij, produced during the HIP base exchange (see Section 4.1.3). 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 into the key derivation function defined by the DH Group ID. Currently the only key derivation function defied in this document is the Hash-based Key Derivation Function (HKDF) [RFC5869] using the RHASH hash function. Other documents may define new DH Group IDs and corresponding key distribution functions.

In the following we provide the details for deriving the keying material using HKDF.

where

```
info    = sort(HIT-I | HIT-R)
salt    =  #I | #J
```

Sort(HIT-I | HIT-R) is defined as the network byte order concatenation of the two HITs, with the smaller HIT preceding the larger HIT, resulting from the numeric comparison of the two HITs interpreted as positive (unsigned) 128-bit integers in network byte order. The #I and #J values are from the puzzle and its solution that were exchanged in R1 and I2 messages when this HIP association was set up. Both hosts have to store #I and #J values for the HIP association for future use.

The initial keys are drawn sequentially in the order that is determined by the numeric comparison of the two HITs, with comparison method described in the previous paragraph. HOST_g denotes the host with the greater HIT value, and HOST_l the host with the lower HIT value.

The drawing order for the four initial keys is as follows:

HIP-gl encryption key for HOST_g's ENCRYPTED parameter

HIP-gl integrity (HMAC) key for HOST_g's outgoing HIP packets

HIP-lg encryption key for HOST_l's ENCRYPTED parameter

HIP-lg integrity (HMAC) key for HOST_l's outgoing HIP packets

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

AES	128 or 256 bits
SHA-1	160 bits
SHA-256	256 bits
SHA-384	384 bits
NULL	0 bits

If other key sizes are used, they MUST be treated as different encryption algorithms and defined separately.

6.6. Initiation of a HIP Base Exchange

An implementation may originate a HIP base exchange to another host based on a local policy decision, usually triggered by an application datagram, in much the same way that an IPsec IKE key exchange can dynamically create a Security Association. Alternatively, a system may initiate a HIP exchange if it has rebooted or timed out, or otherwise lost its HIP state, as described in Section 4.5.4.

The implementation prepares an I1 packet and sends it to the IP address that corresponds to the peer host. The IP address of the peer host may be obtained via conventional mechanisms, such as DNS lookup. The I1 packet contents are specified in Section 5.3.1. The selection of which source or destination Host Identity to use, if a Initiator or Responder has more than one to choose from, is typically a policy decision.

The following steps define the conceptual processing rules for initiating a HIP base exchange:

1. The Initiator receives one or more of the Responder's HITs and one or more addresses either from a DNS lookup of the Responder's FQDN, from some other repository, or from a local database. If the Initiator does not know the Responder's HIT, it may attempt opportunistic mode by using NULL (all zeros) as the Responder's HIT (see also "HIP Opportunistic Mode" (Section 4.1.8)). If the Initiator can choose from multiple Responder HITs, it selects a

HIT for which the Initiator supports the HIT Suite.

2. The Initiator sends an I1 packet to one of the Responder's addresses. The selection of which address to use is a local policy decision.

3. The Initiator includes the DH_GROUP_LIST in the I1 packet. The selection and order of DH Group IDs in the DH_GROUP_LIST MUST be stored by the Initiator because this list is needed for later R1 processing. In most cases, the preferences regarding the DH Groups will be static, so no per-association storage is necessary.

4. Upon sending an I1 packet, the sender transitions to state I1-SENT, starts a timer for which the timeout value SHOULD be larger than the worst-case anticipated RTT. The sender SHOULD also increment the timeout counter associated with the I1.

5. Upon timeout, the sender SHOULD retransmit the I1 packet and restart the timer, up to a maximum of I1_RETRIES_MAX tries.

6.6.1. Sending Multiple I1 Packets in Parallel

For the sake of minimizing the session establishment latency, an implementation MAY send the same I1 packet to more than one of the Responder's addresses. However, it MUST NOT send to more than three (3) Responder addresses in parallel. Furthermore, upon timeout, the implementation MUST refrain from sending the same I1 packet to multiple addresses. That is, if it retries to initialize the connection after a timeout, it MUST NOT send the I1 packet to more than one destination address. These limitations are placed in order to avoid congestion of the network, and potential DoS attacks that might occur, e.g., because someone's claim to have hundreds or thousands of addresses could generate a huge number of I1 packets from the Initiator.

As the Responder is not guaranteed to distinguish the duplicate I1 packets it receives at several of its addresses (because it avoids storing states when it answers back an R1 packet), the Initiator may receive several duplicate R1 packets.

The Initiator SHOULD then select the initial preferred destination address using the source address of the selected received R1, and use the preferred address as a source address for the I2 packet. Processing rules for received R1s are discussed in Section 6.8.

6.6.2. Processing Incoming ICMP Protocol Unreachable Messages

A host may receive an ICMP 'Destination Protocol Unreachable' message as a response to sending a HIP I1 packet. Such a packet may be an indication that the peer does not support HIP, or it may be an attempt to launch an attack by making the Initiator believe that the Responder does not support HIP.

When a system receives an ICMP 'Destination Protocol Unreachable' message while it is waiting for an R1 packet, it MUST NOT terminate waiting. It MAY continue as if it had not received the ICMP message, and send a few more I1 packets. Alternatively, it MAY take the ICMP message as a hint that the peer most probably does not support HIP, and return to state UNASSOCIATED earlier than otherwise. However, at minimum, it MUST continue waiting for an R1 packet for a reasonable time before returning to UNASSOCIATED.

6.7. Processing Incoming I1 Packets

An implementation SHOULD reply to an I1 with an R1 packet, unless the implementation is unable or unwilling to set up a HIP association. If the implementation is unable to set up a HIP association, the host SHOULD send an ICMP Destination Protocol Unreachable, Administratively Prohibited, message to the I1 packet source IP address. If the implementation is unwilling to set up a HIP association, the host MAY ignore the I1 packet. This latter case may occur during a DoS attack such as an I1 packet flood.

The implementation SHOULD be able to handle a storm of received I1 packets, discarding those with common content that arrive within a small time delta.

A spoofed I1 packet can result in an R1 attack on a system. An R1 packet sender MUST have a mechanism to rate-limit R1 packets sent to an address.

It is RECOMMENDED that the HIP state machine does not transition upon sending an R1 packet.

The following steps define the conceptual processing rules for responding to an I1 packet:

1. The Responder MUST check that the Responder's HIT in the received I1 packet is either one of its own HITs or NULL. Otherwise it must drop the packet.

2. If the Responder is in ESTABLISHED state, the Responder MAY respond to this with an R1 packet, prepare to drop an existing

HIP security association with the peer, and stay at ESTABLISHED state.

3. If the Responder is in I1-SENT state, it MUST make a comparison between the sender's HIT and its own (i.e., the receiver's) HIT. If the sender's HIT is greater than its own HIT, it should drop the I1 packet and stay at I1-SENT. If the sender's HIT is smaller than its own HIT, it SHOULD send the R1 packet and stay at I1-SENT. The HIT comparison is performed as defined in Section 6.5.

4. If the implementation chooses to respond to the I1 packet with an R1 packet, it creates a new R1 or selects a precomputed R1 according to the format described in Section 5.3.2. It creates or chooses an R1 that contains its most preferred DH public value that is also contained in the DH_GROUP_LIST in the I1 packet. If no suitable DH Group ID was contained in the DH_GROUP_LIST in the I1 packet, it sends an R1 with any suitable DH public key.

5. If the received Responder's HIT in the I1 is NULL, the Responder selects a HIT with a the same HIT Suite as the Initiator's HIT. If this HIT Suite is not supported by the Responder, it SHOULD select a REQUIRED HIT Suite from Section 5.2.10, which is currently RSA/DSA/SHA-256. Other than that, selecting the HIT is a local policy matter.

6. The responder expresses its supported HIP transport formats in the TRANSPORT_FORMAT_LIST as described in Section 5.2.11. The Responder MUST at least provide one payload transport format type.

7. The Responder sends the R1 packet to the source IP address of the I1 packet.

6.7.1. R1 Management

All compliant implementations MUST be able to produce R1 packets. An R1 packet MAY be precomputed. An R1 packet MAY be reused for time Delta T, which is implementation dependent, and SHOULD be deprecated and not used once a valid response I2 packet has been received from an Initiator. During an I1 message storm, an R1 packet MAY be re-used beyond this limit. R1 information MUST NOT be discarded until Delta S after T. Time S is the delay needed for the last I2 packet to arrive back to the Responder.

Implementations that support multiple DH groups MAY pre-compute R1 packets for each supported group so that incoming I1 packets with different DH Group IDs in the DH_GROUP_LIST can be served quickly.

An implementation MAY keep state about received I1 packets and match the received I2 packets against the state, as discussed in Section 4.1.1.

6.7.2. Handling Malformed Messages

If an implementation receives a malformed I1 packet, it SHOULD NOT respond with a NOTIFY message, as such practice could open up a potential denial-of-service threat. Instead, it MAY respond with an ICMP packet, as defined in Section 5.4.

6.8. Processing Incoming R1 Packets

A system receiving an R1 packet MUST first check to see if it has sent an I1 packet to the originator of the R1 packet (i.e., it is in state I1-SENT). If so, it SHOULD process the R1 as described below, send an I2 packet, and transition to state I2-SENT, setting a timer to protect the I2 packet. If the system is in state I2-SENT, it MAY respond to the R1 packet if the R1 packet has a larger R1 generation counter; if so, it should drop its state due to processing the previous R1 packet and start over from state I1-SENT. If the system is in any other state with respect to that host, the system SHOULD silently drop the R1 packet.

When sending multiple I1 packets, an Initiator SHOULD wait for a small amount of time after the first R1 reception to allow possibly multiple R1 packets to arrive, and it SHOULD respond to an R1 packet among the set with the largest R1 generation counter.

The following steps define the conceptual processing rules for responding to an R1 packet:

1. A system receiving an R1 MUST first check to see if it has sent an I1 packet to the originator of the R1 packet (i.e., it has a HIP association that is in state I1-SENT and that is associated with the HITs in the R1). Unless the I1 packet was sent in opportunistic mode (see Section 4.1.8), the IP addresses in the received R1 packet SHOULD be ignored by the R1 processing and, when looking up the right HIP association, the received R1 packet SHOULD be matched against the associations using only the HITs. If a match exists, the system should process the R1 packet as described below.

2. Otherwise, if the system is in any other state than I1-SENT or I2-SENT with respect to the HITs included in the R1 packet, it SHOULD silently drop the R1 packet and remain in the current state.

3. If the HIP association state is I1-SENT or I2-SENT, the received Initiator's HIT MUST correspond to the HIT used in the original I1. Also, the Responder's HIT MUST correspond to the one used in the I1, unless the I1 packet contained a NULL HIT.

4. The system SHOULD validate the R1 signature before applying further packet processing, according to Section 5.2.15.

5. If the HIP association state is I1-SENT, and multiple valid R1 packets are present, the system MUST select from among the R1 packets with the largest R1 generation counter.

6. The system MUST check that the Initiator HIT Suite is contained in the HIT_SUITE_LIST parameter in the R1 packet (i.e., the Initiator's HIT Suite is supported by the Responder). If the HIT Suite is supported by the Responder, the system proceeds normally. Otherwise, the system MAY stay in state I1-sent and restart the BEX by sending a new I1 packet with an Initiator HIT that is supported by the Responder and hence is contained in the HIT_SUITE_LIST in the R1 packet. The system MAY abort the BEX if no suitable source HIT is available. The system SHOULD wait for an acceptable time span to allow further R1 packets with higher R1 generation counters or different HIT and HIT Suites to arrive before restarting or aborting the BEX.

7. The system MUST check that the DH Group ID in the DIFFIE_HELLMAN parameter in the R1 matches the first DH Suite ID in the Responder's DH_GROUP_LIST in the R1 packet that was also contained in the Initiator's DH_GROUP_LIST in the I1 packet. If the DH Group ID of the DIFFIE_HELLMAN parameter does not express the Responder's best choice, the Initiator can conclude that the DH_GROUP_LIST in the I1 packet was adversely modified. In such case, the Initiator MAY send a new I1 packet, however, it SHOULD NOT change its preference in the DH_GROUP_LIST in the new I1 packet. Alternatively, the Initiator MAY abort the HIP base exchange.

8. If the HIP association state is I2-SENT, the system MAY re-enter state I1-SENT and process the received R1 packet if it has a larger R1 generation counter than the R1 packet responded to previously.

9. The R1 packet may have the A bit set -- in this case, the system MAY choose to refuse it by dropping the R1 packet and returning to state UNASSOCIATED. The system SHOULD consider dropping the R1 packet only if it used a NULL HIT in I1 packet. If the A bit is set, the Responder's HIT is anonymous and SHOULD NOT be stored permanently.

10. The system SHOULD attempt to validate the HIT against the received Host Identity by using the received Host Identity to construct a HIT and verify that it matches the Sender's HIT.

11. The system MUST store the received R1 generation counter for future reference.

12. The system attempts to solve the puzzle in the R1 packet. The system MUST terminate the search after exceeding the remaining lifetime of the puzzle. If the puzzle is not successfully solved, the implementation MAY either resend the I1 packet within the retry bounds or abandon the HIP base exchange.

13. The system computes standard Diffie-Hellman keying material according to the public value and Group ID provided in the DIFFIE_HELLMAN parameter. The Diffie-Hellman keying material Kij is used for key extraction as specified in Section 6.5.

14. The system selects the HIP_CIPHER ID from the choices presented in the R1 packet and uses the selected values subsequently when generating and using encryption keys, and when sending the I2 packet. If the proposed alternatives are not acceptable to the system, it may either resend an I1 within the retry bounds or abandon the HIP base exchange.

15. The system chooses one suitable transport format from the TRANSPORT_FORMAT_LIST and includes the respective transport format parameter in the subsequent I2 packet.

16. The system initializes the remaining variables in the associated state, including Update ID counters.

17. The system prepares and sends an I2 packet, as described in Section 5.3.3.

18. The system SHOULD start a timer whose timeout value SHOULD be larger than the worst-case anticipated RTT, and MUST increment a timeout counter associated with the I2 packet. The sender SHOULD retransmit the I2 packet upon a timeout and restart the timer, up to a maximum of I2_RETRIES_MAX tries.

19. If the system is in state I1-SENT, it SHALL transition to state I2-SENT. If the system is in any other state, it remains in the current state.

6.8.1. Handling of Malformed Messages

If an implementation receives a malformed R1 message, it MUST silently drop the packet. Sending a NOTIFY or ICMP would not help, as the sender of the R1 packet typically doesn't have any state. An implementation SHOULD wait for some more time for a possibly well-formed R1, after which it MAY try again by sending a new I1 packet.

6.9. Processing Incoming I2 Packets

Upon receipt of an I2 packet, the system MAY perform initial checks to determine whether the I2 packet corresponds to a recent R1 packet that has been sent out, if the Responder keeps such state. For example, the sender could check whether the I2 packet is from an address or HIT for which the Responder has recently received an I1. The R1 packet may have had Opaque data included that was echoed back in the I2 packet. If the I2 packet is considered to be suspect, it MAY be silently discarded by the system.

Otherwise, the HIP implementation SHOULD process the I2 packet. This includes validation of the puzzle solution, generating the Diffie-Hellman key, possibly decrypting the Initiator's Host Identity, verifying the signature, creating state, and finally sending an R2 packet.

The following steps define the conceptual processing rules for responding to an I2 packet:

1. The system MAY perform checks to verify that the I2 packet corresponds to a recently sent R1 packet. Such checks are implementation dependent. See Appendix A for a description of an example implementation.

2. The system MUST check that the Responder's HIT corresponds to one of its own HITs and MUST drop the packet otherwise.

3. The system MUST further check that the Initiator's HIT Suite is supported. The Responder SHOULD silently drop I2 packets with unsupported Initiator HITs.

4. If the system's state machine is in the R2-SENT state, the system MAY check if the newly received I2 packet is similar to the one that triggered moving to R2-SENT. If so, it MAY retransmit a previously sent R2 packet, reset the R2-SENT timer, and the state machine stays in R2-SENT.

5. If the system's state machine is in the I2-SENT state, the system makes a comparison between its local and sender's HITs

(similarly as in Section 6.5). If the local HIT is smaller than the sender's HIT, it should drop the I2 packet, use the peer Diffie-Hellman key and nonce #I from the R1 packet received earlier, and get the local Diffie-Hellman key and nonce #J from the I2 packet sent to the peer earlier. Otherwise, the system should process the received I2 packet and drop any previously derived Diffie-Hellman keying material Kij it might have formed upon sending the I2 packet previously. The peer Diffie-Hellman key and the nonce #J are taken from the just arrived I2 packet. The local Diffie-Hellman key and the nonce I are the ones that were sent earlier in the R1 packet.

6. If the system's state machine is in the I1-SENT state, and the HITs in the I2 packet match those used in the previously sent I1 packet, the system uses this received I2 packet as the basis for the HIP association it was trying to form, and stops retransmitting I1 packets (provided that the I2 packet passes the additional checks below).

7. If the system's state machine is in any other state than R2-SENT, the system SHOULD check that the echoed R1 generation counter in the I2 packet is within the acceptable range if the counter is included. Implementations MUST accept puzzles from the current generation and MAY accept puzzles from earlier generations. If the generation counter in the newly received I2 packet is outside the accepted range, the I2 packet is stale (and perhaps replayed) and SHOULD be dropped.

8. The system MUST validate the solution to the puzzle by computing the hash described in Section 5.3.3 using the same RHASH algorithm.

9. The I2 packet MUST have a single value in the HIP_CIPHER parameter, which MUST match one of the values offered to the Initiator in the R1 packet.

10. The system must derive Diffie-Hellman keying material Kij based on the public value and Group ID in the DIFFIE_HELLMAN parameter. This key is used to derive the HIP association keys, as described in Section 6.5. If the Diffie-Hellman Group ID is unsupported, the I2 packet is silently dropped.

11. The encrypted HOST_ID is decrypted by the Initiator's encryption key defined in Section 6.5. If the decrypted data is not a HOST_ID parameter, the I2 packet is silently dropped.

12. The implementation SHOULD also verify that the Initiator's HIT in the I2 packet corresponds to the Host Identity sent in the I2

packet. (Note: some middleboxes may not able to make this verification.)

13. The system MUST process the TRANSPORT_FORMAT_LIST parameter. Other documents specifying transport formats (e.g. [I-D.ietf-hip-rfc5202-bis]) contain specifications for handling any specific transport selected.

14. The system MUST verify the HIP_MAC according to the procedures in Section 5.2.12.

15. The system MUST verify the HIP_SIGNATURE according to Section 5.2.14 and Section 5.3.3.

16. If the checks above are valid, then the system proceeds with further I2 processing; otherwise, it discards the I2 and its state machine remains in the same state.

17. The I2 packet may have the A bit set -- in this case, the system MAY choose to refuse it by dropping the I2 and the state machine returns to state UNASSOCIATED. If the A bit is set, the Initiator's HIT is anonymous and should not be stored permanently.

18. The system initializes the remaining variables in the associated state, including Update ID counters.

19. Upon successful processing of an I2 message when the system's state machine is in state UNASSOCIATED, I1-SENT, I2-SENT, or R2-SENT, an R2 packet is sent and the system's state machine transitions to state R2-SENT.

20. Upon successful processing of an I2 packet when the system's state machine is in state ESTABLISHED, the old HIP association is dropped and a new one is installed, an R2 packet is sent, and the system's state machine transitions to R2-SENT.

21. Upon the system's state machine transitioning to R2-SENT, the system starts a timer. The state machine transitions to ESTABLISHED if some data has been received on the incoming HIP association, or an UPDATE packet has been received (or some other packet that indicates that the peer system's state machine has moved to ESTABLISHED). If the timer expires (allowing for maximal amount of retransmissions of I2 packets), the state machine transitions to ESTABLISHED.

6.9.1. Handling of Malformed Messages

If an implementation receives a malformed I2 message, the behavior SHOULD depend on how many checks the message has already passed. If the puzzle solution in the message has already been checked, the implementation SHOULD report the error by responding with a NOTIFY packet. Otherwise, the implementation MAY respond with an ICMP message as defined in Section 5.4.

6.10. Processing of Incoming R2 Packets

An R2 packet received in states UNASSOCIATED, I1-SENT, or ESTABLISHED results in the R2 packet being dropped and the state machine staying in the same state. If an R2 packet is received in state I2-SENT, it MUST be processed.

The following steps define the conceptual processing rules for an incoming R2 packet:

1. If the system is in any other state than I2-SENT, the R2 packet is silently dropped.

2. The system MUST verify that the HITs in use correspond to the HITs that were received in the R1 packet that caused the transition to the I1-SENT state.

3. The system MUST verify the HIP_MAC_2 according to the procedures in Section 5.2.13.

4. The system MUST verify the HIP signature according to the procedures in Section 5.2.14.

5. If any of the checks above fail, there is a high probability of an ongoing man-in-the-middle or other security attack. The system SHOULD act accordingly, based on its local policy.

6. Upon successful processing of the R2 packet, the state machine transitions to state ESTABLISHED.

6.11. Sending UPDATE Packets

A host sends an UPDATE packet when it intends to update some information related to a HIP association. There are a number of possible scenarios when this can occur, e.g., mobility management and rekeying of an existing ESP Security Association. The following paragraphs define the conceptual rules for sending an UPDATE packet to the peer. Additional steps can be defined in other documents where the UPDATE packet is used.

The sequence of UPDATE messages is indicated by their SEQ parameter. Before sending an UPDATE message, the system first determines whether there are any outstanding UPDATE messages that may conflict with the new UPDATE message under consideration. When multiple UPDATEs are outstanding (not yet acknowledged), the sender must assume that such UPDATEs may be processed in an arbitrary order by the receiver. Therefore, any new UPDATEs that depend on a previous outstanding UPDATE being successfully received and acknowledged MUST be postponed until reception of the necessary ACK(s) occurs. One way to prevent any conflicts is to only allow one outstanding UPDATE at a time. However, allowing multiple UPDATEs may improve the performance of mobility and multihoming protocols.

The following steps define the conceptual processing rules for sending UPDATE packets.

1. The first UPDATE packet is sent with Update ID of zero. Otherwise, the system increments its own Update ID value by one before continuing the steps below.

2. The system creates an UPDATE packet that contains a SEQ parameter with the current value of Update ID. The UPDATE packet MAY also include zero or more ACKs of the peer's Update ID(s) from previously received UPDATE SEQ parameter(s)

3. The system sends the created UPDATE packet and starts an UPDATE timer. The default value for the timer is 2 * RTT estimate. If multiple UPDATEs are outstanding, multiple timers are in effect.

4. If the UPDATE timer expires, the UPDATE is resent. The UPDATE can be resent UPDATE_RETRY_MAX times. The UPDATE timer SHOULD be exponentially backed off for subsequent retransmissions. If no acknowledgment is received from the peer after UPDATE_RETRY_MAX times, the HIP association is considered to be broken and the state machine SHOULD move from state ESTABLISHED to state CLOSING as depicted in Section 4.4.4. The UPDATE timer is cancelled upon receiving an ACK from the peer that acknowledges receipt of the UPDATE.

6.12. Receiving UPDATE Packets

When a system receives an UPDATE packet, its processing depends on the state of the HIP association and the presence and values of the SEQ and ACK parameters. Typically, an UPDATE message also carries optional parameters whose handling is defined in separate documents.

For each association, a host stores the peer's next expected in-sequence Update ID ("peer Update ID"). Initially, this value is

zero. Update ID comparisons of "less than" and "greater than" are performed with respect to a circular sequence number space. Hence, a wrap around after 2^32 updates has to be expected and MUST be handled accordingly.

The sender MAY send multiple outstanding UPDATE messages. These messages are processed in the order in which they are received at the receiver (i.e., no re-sequencing is performed). When processing UPDATEs out-of-order, the receiver MUST keep track of which UPDATEs were previously processed, so that duplicates or retransmissions are ACKed and not reprocessed. A receiver MAY choose to define a receive window of Update IDs that it is willing to process at any given time, and discard received UPDATEs falling outside of that window.

The following steps define the conceptual processing rules for receiving UPDATE packets.

1. If there is no corresponding HIP association, the implementation MAY reply with an ICMP Parameter Problem, as specified in Section 5.4.4.

2. If the association is in the ESTABLISHED state and the SEQ (but not ACK) parameter is present, the UPDATE is processed and replied to as described in Section 6.12.1.

3. If the association is in the ESTABLISHED state and the ACK (but not SEQ) parameter is present, the UPDATE is processed as described in Section 6.12.2.

4. If the association is in the ESTABLISHED state and there is both an ACK and SEQ in the UPDATE, the ACK is first processed as described in Section 6.12.2, and then the rest of the UPDATE is processed as described in Section 6.12.1.

6.12.1. Handling a SEQ Parameter in a Received UPDATE Message

The following steps define the conceptual processing rules for handling a SEQ parameter in a received UPDATE packet.

1. If the Update ID in the received SEQ is not the next in the sequence of Update IDs and is greater than the receiver's window for new UPDATEs, the packet MUST be dropped.

2. If the Update ID in the received SEQ corresponds to an UPDATE that has recently been processed, the packet is treated as a retransmission. The HIP_MAC verification (next step) MUST NOT be skipped. (A byte-by-byte comparison of the received and a stored packet would be acceptable, though.) It is recommended that a

host caches UPDATE packets sent with ACKs to avoid the cost of generating a new ACK packet to respond to a replayed UPDATE. The system MUST acknowledge, again, such (apparent) UPDATE message retransmissions but SHOULD also consider rate-limiting such retransmission responses to guard against replay attacks.

3. The system MUST verify the HIP_MAC in the UPDATE packet. If the verification fails, the packet MUST be dropped.

4. The system MAY verify the SIGNATURE in the UPDATE packet. If the verification fails, the packet SHOULD be dropped and an error message logged.

5. If a new SEQ parameter is being processed, the parameters in the UPDATE are then processed. The system MUST record the Update ID in the received SEQ parameter, for replay protection.

6. An UPDATE acknowledgment packet with ACK parameter is prepared and sent to the peer. This ACK parameter MAY be included in a separate UPDATE or piggybacked in an UPDATE with SEQ parameter, as described in Section 5.3.5. The ACK parameter MAY acknowledge more than one of the peer's Update IDs.

6.12.2. Handling an ACK Parameter in a Received UPDATE Packet

The following steps define the conceptual processing rules for handling an ACK parameter in a received UPDATE packet.

1. The sequence number reported in the ACK must match with an UPDATE packet sent earlier that has not already been acknowledged. If no match is found or if the ACK does not acknowledge a new UPDATE, the packet MUST either be dropped if no SEQ parameter is present, or the processing steps in Section 6.12.1 are followed.

2. The system MUST verify the HIP_MAC in the UPDATE packet. If the verification fails, the packet MUST be dropped.

3. The system MAY verify the SIGNATURE in the UPDATE packet. If the verification fails, the packet SHOULD be dropped and an error message logged.

4. The corresponding UPDATE timer is stopped (see Section 6.11) so that the now acknowledged UPDATE is no longer retransmitted. If multiple UPDATEs are acknowledged, multiple timers are stopped.

6.13. Processing of NOTIFY Packets

Processing of NOTIFY packets is OPTIONAL. If processed, any errors in a received NOTIFICATION parameter SHOULD be logged. Received errors MUST be considered only as informational, and the receiver SHOULD NOT change its HIP state (see Section 4.4.2) purely based on the received NOTIFY message.

6.14. Processing CLOSE Packets

When the host receives a CLOSE message, it responds with a CLOSE_ACK message and moves to CLOSED state. (The authenticity of the CLOSE message is verified using both HIP_MAC and SIGNATURE). This processing applies whether or not the HIP association state is CLOSING in order to handle simultaneous CLOSE messages from both ends that cross in flight.

The HIP association is not discarded before the host moves to the UNASSOCIATED state.

Once the closing process has started, any new need to send data packets triggers creating and establishing of a new HIP association, starting with sending of an I1 packet.

If there is no corresponding HIP association, the CLOSE packet is dropped.

6.15. Processing CLOSE_ACK Packets

When a host receives a CLOSE_ACK message, it verifies that it is in CLOSING or CLOSED state and that the CLOSE_ACK was in response to the CLOSE. A host can map CLOSE_ACK messages to CLOSE messages by comparing the value of ECHO_REQUEST_SIGNED (in the CLOSE packet) to the value of ECHO_RESPONSE_SIGNED (in the CLOSE_ACK packet).

The CLOSE_ACK contains the HIP_MAC and the SIGNATURE parameters for verification. The state is discarded when the state changes to UNASSOCIATED and, after that, the host MAY respond with an ICMP Parameter Problem to an incoming CLOSE message (see Section 5.4.4).

6.16. Handling State Loss

In the case of a system crash and unanticipated state loss, the system SHOULD delete the corresponding HIP state, including the keying material. That is, the state SHOULD NOT be stored in long-term storage. If the implementation does drop the state (as RECOMMENDED), it MUST also drop the peer's R1 generation counter value, unless a local policy explicitly defines that the value of

that particular host is stored. An implementation MUST NOT store a peer's R1 generation counters by default, but storing R1 generation counter values, if done, MUST be configured by explicit HITs.

7. HIP Policies

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

Initiators MAY use a different HI for different Responders to provide basic privacy. Whether such private HIs are used repeatedly with the same Responder and how long these HIs are used is decided by local policy and depends on the privacy requirements of the Initiator.

The value of #K used in the HIP R1 must be chosen with care. Too high numbers of #K will exclude clients with weak CPUs because these devices cannot solve the puzzle within reasonable time. #K should only be raised if a Responder is under high load, i.e., it cannot process all incoming HIP handshakes any more. If a responder is not under high load, K SHOULD be 0.

Responders that only respond to selected Initiators require an ACL, representing for which hosts they accept HIP base exchanges, and the preferred transform and local lifetimes. Wildcarding SHOULD be supported for such ACLs, and also for Responders that offer public or anonymous services.

8. Security Considerations

HIP is designed to provide secure authentication of hosts. HIP also attempts to limit the exposure of the host to various denial-of-service and man-in-the-middle (MitM) attacks. In doing so, 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.

Denial-of-service attacks often take advantage of asymmetries in the cost of an starting an association. One example of such asymmetry is the need of a Responder to store local state while a malicious Initiator can stay stateless. HIP makes no attempt to increase the cost of the start of state at the Initiator, but makes an effort to reduce the cost for the Responder. This is accomplished by having the Responder start the 3-way exchange instead of the Initiator, making the HIP protocol 4 packets long. In doing this, the first

packet from the Responder, R1, becomes a 'stock' packet that the Responder MAY use many times, until some Initiator has provided a valid response to such an R1 packet. During an I1 packet storm, the host may reuse the same DH value also even if some Initiator has provided a valid response using that particular DH value. However, such behavior is discouraged and should be avoided. Using the same Diffie-Hellman values and random puzzle #I value has some risks. 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 can spoof the I1 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 packet. The defense against this attack is to simply ignore any R1 packet where a corresponding I1 packet was not sent (as defined in Section 6.8 step 1).

The R1 packet is considerably larger than the I1 packet. This asymmetry can be exploited in an reflection attack. A malicious attacker could spoof the IP address of a victim and send a flood of I1 messages to a powerful Responder. For each small I1 packet, the Responder would send a larger R1 packet to the victim. The difference in packet sizes can further amplify a flooding attack against the victim. To avoid such reflection attacks, the Responder SHOULD rate limit the sending of R1 packets in general or SHOULD rate limit the sending of R1 packets to a specific IP address.

Floods of forged I2 packets form a second kind of DoS attack. Once the attacking Initiator has solved the puzzle, it can send packets with spoofed IP source addresses with either an invalid HIP signature or invalid encrypted HIP payload (in the ENCRYPTED parameter). 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 with the same puzzle solution, the Responder would discard any I2 packets that contain the given solution. This will shut down the attack. The attacker would have to request another R1 packet and use that to launch a new attack. The Responder could increase the value of #K while under attack. Keeping a list of solutions from malformed packets requires that the Responder keeps state for these malformed I2 packets. This state has to be kept until the R1 counter is increased. As malformed packets are generally filtered by their checksum before signature verification, only solutions in packets that are forged to pass the checksum and puzzle are put to the blacklist. In addition, a valid puzzle is required before a new list entry is created. Hence, attackers that intend to flood the blacklist must solve puzzles first.

A third form of DoS attack is emulating the restart of state after a reboot of one of the peers. A restarting host would send an I1 packet to the peers, which would respond with an R1 packet even if it were in the ESTABLISHED state. If the I1 packet were spoofed, the resulting R1 packet would be received unexpectedly by the spoofed host and would be dropped, as in the first case above.

A fourth form of DoS attack is emulating closing of the HIP association. HIP relies on timers and a CLOSE/CLOSE_ACK handshake to explicitly signal the end of a HIP association. Because both CLOSE and CLOSE_ACK messages contain a HIP_MAC, an outsider cannot close a connection. The presence of an additional SIGNATURE allows middleboxes to inspect these messages and discard the associated state (for e.g., firewalling, SPI-based NATing, etc.). However, the optional behavior of replying to CLOSE with an ICMP Parameter Problem packet (as described in Section 5.4.4) might allow an attacker spoofing the source IP address to send CLOSE messages to launch reflection attacks.

A fifth form of DoS attack is replaying R1s to cause the Initiator to solve stale puzzles and become out of synchronization with the Responder. The R1 generation counter is a monotonically increasing counter designed to protect against this attack, as described in Section 4.1.4.

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 the HI (after having got the I2 HIP packet) and verify that the HI indeed can be trusted.

The HIP Opportunistic Mode concept has been introduced in this document, but this document does not specify what the semantics of such a connection setup are for applications. There are certain concerns with opportunistic mode, as discussed in Section 4.1.8.

NOTIFY messages are used only for informational purposes and they are unacknowledged. A HIP implementation cannot rely solely on the information received in a NOTIFY message because the packet may have been replayed. An implementation SHOULD NOT change any state information purely based on a received NOTIFY message.

Since not all hosts will ever support HIP, ICMP 'Destination Protocol Unreachable' messages are to be expected and may be used for 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 from 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. Normally, if an I1 message received by a Responder was a bogus one sent by an attacker, the Responder may receive an ICMP message from the IP address the R1 message was sent to. However, a sophisticated attacker can try to take advantage of such a behavior and try to break up the HIP base exchange by sending such an ICMP message to the Responder before the Initiator has a chance to send a valid I2 message. Hence, the Responder SHOULD NOT act on such an ICMP message. Especially, it SHOULD NOT remove any minimal state created when it sent the R1 HIP packet (if it did create one), but wait for either a valid I2 HIP packet or the natural timeout (that is, if R1 packets are tracked at all). Likewise, the Initiator SHOULD ignore any ICMP message while waiting for an R2 HIP packet, and SHOULD delete any pending state only after a natural timeout.

9. IANA Considerations

IANA has reserved protocol number 139 for the Host Identity Protocol.

This document defines a new 128-bit value under the CGA Message Type namespace [RFC3972], 0xF0EF F02F BFF4 3D0F E793 0C3C 6E61 74EA, to be used for HIT generation as specified in ORCHID [I-D.ietf-hip-rfc4843-bis].

This document uses HIP version number 2 for the four-bit Version field in a HIP protocol packet defined in [RFC5201].

This document also creates a set of new namespaces. These are described below.

Packet Type

The 7-bit Packet Type field in a HIP protocol packet describes the type of a HIP protocol message. It is defined in Section 5.1. The current values are defined in Sections 5.3.1 through 5.3.8.

New values are assigned through IETF Review or IESG Approval [RFC5226].

HIT Suite

The four-bit HIT Suite ID uses the OGA field in the ORCHID to express the type of the HIT. This document defines two HIT Suites (see Appendix E).

The HIT Suite ID is also carried in the four higher-order bits of the ID field in the HIT_SUITE_LIST parameter. The four lower-order bits are reserved for future extensions of the HIT Suite ID space beyond 16 values.

At the time being, the HIT Suite uses only four bits because these bits have to be carried in the HIT. Using more bits for the HIT Suite ID reduces the cryptographic strength of the HIT. HIT Suite IDs must be allocated carefully to avoid namespace exhaustion. Moreover, deprecated IDs should be reused after an appropriate time span. If 16 Suite IDs prove insufficient and more HIT Suite IDs are needed concurrently, more bits can be used for the HIT Suite ID by using one HIT Suite ID (0) to indicate that more bits should be used. The HIT_SUITE_LIST parameter already supports 8-bit HIT Suite IDs, should longer IDs be needed. Possible extensions of the HIT Suite ID space to accommodate eight bits and new HIT Suite IDs are defined through IETF Review or IESG Approval.

Parameter Type

The 16-bit Type field in a HIP parameter describes the type of the parameter. It is defined in Section 5.2.1. The current values are defined in Sections 5.2.3 through 5.2.23.

With the exception of the assigned Type codes, the Type codes 0 through 1023 and 61440 through 65535 are reserved for future base protocol extensions, and are assigned through IETF Review or IESG Approval.

The Type codes 32768 through 49151 are reserved for experimentation. Types SHOULD be selected in a random fashion from this range, thereby reducing the probability of collisions. A method employing genuine randomness (such as flipping a coin) SHOULD be used.

All other Type codes are assigned through First Come First Served, with Specification Required [RFC5226].

Group ID

The eight-bit Group ID values appear in the DIFFIE_HELLMAN parameter and the DH_GROUP_LIST parameter and are defined in Section 5.2.7. New values are assigned through IETF Review or IESG Approval.

HIP Cipher ID

The 16-bit Cipher ID values in a HIP_CIPHER parameter are defined in Section 5.2.8. New values either from the reserved or unassigned space are assigned through IETF Review or IESG Approval.

DI-Type

The four-bit DI-Type values in a HOST_ID parameter are defined in Section 5.2.9. New values are assigned through IETF Review or IESG Approval.

Notify Message Type

The 16-bit Notify Message Type values in a NOTIFICATION parameter are defined in Section 5.2.19.

Notify Message Type values 1-10 are used for informing about errors in packet structures, values 11-20 for informing about problems in parameters containing cryptographic related material, values 21-30 for informing about problems in authentication or packet integrity verification. Parameter numbers above 30 can be used for informing about other types of errors or events. Values 51-8191 are error types reserved to be allocated by IANA. Values 8192-16383 are error types for experimentation. Values 16385-40959 are status types to be allocated by IANA, and values 40960-65535 are status types for experimentation. New values in ranges 51-8191 and 16385-40959 are assigned through First Come First Served, with Specification Required.

10. Acknowledgments

The drive to create HIP came to being after attending the MALLOC meeting at the 43rd IETF meeting. 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 versions 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 valuable input at early stages of discussions about identifier handling and Keith Moore the impetus 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 Bob Moskowitz how to make the puzzle exchange expensive for the Initiator to respond, but easy for the Responder to validate. Bill Sommerfeld supplied the Birthday concept, which later evolved into the R1 generation counter, to simplify reboot management. Erik Nordmark supplied the CLOSE-mechanism for closing connections. Rodney Thayer and Hugh Daniels provided extensive feedback. In the early times of this document, John Gilmore kept Bob Moskowitz challenged to provide something of value.

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

In the usual IETF fashion, a large number of people have contributed to the actual text or ideas. The list of these people include Jeff Ahrenholz, Francis Dupont, Derek Fawcus, George Gross, Andrew McGregor, Julien Laganier, Miika Komu, Mika Kousa, Jan Melen, Henrik Petander, Michael Richardson, Rene Hummen, Tim Shepard, Jorma Wall, Xin Gu, and Jukka Ylitalo. Our apologies to anyone whose name is missing.

Once the HIP Working Group was founded in early 2004, a number of changes were introduced through the working group process. Most notably, the original document was split in two, one containing the base exchange and the other one defining how to use ESP. Some modifications to the protocol proposed by Aura, et al., [AUR03] were added at a later stage.

11. Changes from RFC 5201

This section summarizes the changes made from [RFC5201].

11.1. Changes from draft-ietf-hip-rfc5201-bis-11

- Specify that TRANSFORM_FORMAT_LIST is mandatory in R1 and I2; fix incorrect section reference.

11.2. Changes from draft-ietf-hip-rfc5201-bis-10

- Issue 39: Text clarifying R1 counter rollover and Initiator response to unexpected reset of the counter.

11.3. Changes from draft-ietf-hip-rfc5201-bis-09

- Editorial changes based on working group last call.

11.4. Changes from draft-ietf-hip-rfc5201-bis-08

- Issue 29: Use different RSA mode OEAP/PSS, elevate ECDSA to REQUIRED status

- Issue 35: limiting ECC cofactor to 1

- Changed text regarding issue 33 reusing DH values

- Fix tracker issue 32 on Domain Identifier normative text

11.5. Changes from draft-ietf-hip-rfc5201-bis-07

- Removed lingering references to SHA-1 as the mandatory hash algorithm (which was changed to SHA-256 in the -02 draft version).

- For parameter type number changes, changed "IETF Review" to "IETF Review or IESG Approval".

- Updated Appendix C checksum examples to conform to HIPv2 packets.

11.6. Changes from draft-ietf-hip-rfc5201-bis-06

- Made echoing the R1_COUNTER in the I2 mandatory if the R1 contains an R1_COUNTER. This required to make the R1 counter a critical parameter. Hence, the parameter type number of the R1_COUNTER changed from 128 to 129.

- Made KDF dependent on DH Group to enable negotiation of the KDF.

11.7. Changes from draft-ietf-hip-rfc5201-bis-05

- Changed type number of DH_GROUP_LIST from 2151 to 511 because it was in the number space that is reserved for the HIP transport mode negotiations.

- Added transport form type list parameter. Transport forms are now negotiated with this list instead of by their order in the HIP packet. This allows to remove the exception of the transport

format parameters that were ordered by their preference instead of by their type number. This should remove complexity from implementations.

- Clarify that in HIP signature processing, the restored checksum and length fields have been rendered invalid by the previous steps.

- Clarify behavior for when UPDATE does not contain SEQ or ACQ (disallow this).

- For namespace changes, changed "IETF Review" to "IETF Review or IESG Approval".

- Addressed IESG comment about ignoring packet IP addresses.

- Permit using Anonymous HI control in packets other than R1/I2.

- Fixed minor reference error (RFC2418, RFC2410).

- Deleted comment that NULL-ENCRYPTION SHOULD NOT be configurable via the UI.

- Editorial changes.

11.8. Changes from draft-ietf-hip-rfc5201-bis-04

- Clarifications of the Security Considerations section. One DoS defense mechanism was changed to be more effective and less prone to misuse.

- Minor clarifications of the state machine.

- Clarified text on HIP puzzle.

- Added names and references for figures.

- Extended the definitions section.

- Added a reference to the HIP Version 1 certificate document.

- Added Initiator, Responder, HIP association, and signed data to the definitions section.

- Changed parameter figure for PUZZLE and SOLUTION to use RHASH_len/8 instead of n-byte.

- Replaced occurrences of SHOULD not with SHOULD NOT.

- Changed text to reflect the fact that several ECHO_REQUEST_UNSIGNED parameters may be present in an R1 and several ECHO_RESPONSE parameters may be present in an I2.

- Added text on verifying the ECHO_RESPONSE_SIGNED parameter in CLOSE_ACK.

- Changed wording from HMAC to HIP_MAC in Section 5.3.8.

- Reflected fact that the UPDATE packet MAY include zero or more ACKs.

- Added BEX to Definitions section.

- Changed HIP_SIGNATURE algorithm field from 8 bit to 16 bit to achieve alignment with the HOST_ID parameters.

- Fixed the wrong figures of the SEQ and ACK parameters. SEQ always contains ONE update ID. ACK may acknowledge SEVERAL update IDs.

- Added wording that several NOTIFY parameters may be present in a HIP packet.

- Changed wording for the ECHO_RESPONSE_SIGNED parameter. Also lifted the restriction that only one ECHO_RESPONSE_UNSIGNED parameter MUST be present in each HIP control packet. This did contradict the definition of the ECHO_RESPONSE_UNSIGNED parameter.

- Changed IETF Consensus to IETF Review or IESG Approval in IANA section.

- Aligned use of I, J, and K. Now I is #I, J is #J and K is #K throughout the document.

- Updated references.

- Editorial changes.

11.9. Changes from draft-ietf-hip-rfc5201-bis-03

- Editorial changes to improve clarity and readability.

- Removed obsoleted (not applicable) attack from security consideration section.

- Added a requirement that hosts MUST support processing of ACK parameters with several SEQ numbers even when they do not support sending such parameters.

- Removed note on memory bound puzzles. The use of memory bound puzzles was reconsidered but no convincing arguments for inclusion in this document have been made on the list.

- Changed references to reference the new bis documents.

- Specified the ECC curves and the hashes used for these.

- Specified representation of ECC curves in the HI.

- Added text on the dependency between RHASH and HMAC.

- Rephrased part of the security considerations to make them clearer.

- Clarified the use of HITs in opportunistic mode.

- Clarified the difference between HIP_MAC and HIP_MAC_2 as well as between SIGNATURE and SIGNATURE_2.

- Changed NOTIFY name for value 44 from SERVER_BUSY_PLEASE_RETRY to RESPONDER_BUSY_PLEASE_RETRY.

- Mentioned that there are multiple valid puzzle solutions.

11.10. Changes from draft-ietf-hip-rfc5201-bis-02

- Added recommendation to not use puzzle #I twice for the same host to avoid identical key material.

- Revised state machine and added missing event handling.

- Added UNSUPPORTED_HIT_SUITE to NOTIFY to indicate unsupported HIT suites.

- Revised parameter type numbers (corresponding to IANA allocations) and added missing "free for experimentation" range to the description.

- Clarifying note on the use of the C bit in the parameter type numbers.

11.11. Changes from draft-ietf-hip-rfc5201-bis-01

- Changed RHASH-len to RHASH_len to avoid confusion in calculations (- could be minus)
- Added RHASH_len to list of abbreviations
- Fixed length of puzzle #I and #J to be 1*RHASH_len
- Changed RHASH-len to RHASH_len to avoid confusion in calculations (- could be minus)
- Added RHASH_len to list of abbreviations
- Fixed length of puzzle #I and #J to be 1*RHASH_len
- Included HIT_SUITEs.
- Added DH negotiation to I1 and R1.
- Added DH_LIST parameter.
- Added text for DH Group negotiation.
- Removed second DH public value from DH parameter.
- Added ECC to HI generation.
- Added Responder HIT selection to opportunistic mode.
- Added ECDSA HI text and references (not complete yet).
- Added separate section on aborting BEX.
- Added separate section on downgrade attack prevention.
- Added text about DH Group selection for use cases without I1.
- Removed type range allocation for parameters related to HIP transform types.
- New type range allocation for parameters that are only covered by a signature if a signature is present (Applies to DH_GROUP_LIST).
- Renamed HIP_TRANSFORM to HIP_CIPHER and removed hashes from it - hashes are determined by RHASH.

- The length of #I and #J for the puzzle now depends on RHASH.
- New keymat generation.
- Puzzle seed and solution now use RHASH and have variable length.
- Moved timing definitions closer to state machine.
- Simplified text regarding puzzle lifetime.
- Clarified the description of the use of #I in the puzzle
- Removed "Opportunistic mode" description from general definitions.
- More consistency across the old RFC5201 text. Aligned capitalization and abbreviations.
- Extended protocol overview to include restart option.
- Extended state machine to include restart option because of unsupported Algorithms.
- Replaced SHA-1 with SHA-256 for required implementation.
- Added OGA list parameter (715) for detecting the Responder's set of OGAs.
- Added Appendix on ORCHID use in HITs.
- Added truncated SHA-256 option for HITs.
- Added truncated SHA-1 option for HITs.
- Added text about new ORCHID structure to HIT overview.
- Moved Editor role to Robert Moskowitz.
- Added SHA-256 to puzzle parameter.
- Generalized LTRUNC to be hash-function agnostic.
- Added text about RHASH depending on OGA.

11.12. Changes from draft-ietf-hip-rfc5201-bis-00

- Added reasoning why BIS document is needed.

11.13. Contents of draft-ietf-hip-rfc5201-bis-00

- RFC5201 was submitted as draft-RFC.

12. References

12.1. Normative References

[FIPS.180-2.2002] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002, <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.

[I-D.ietf-hip-rfc4843-bis] Laganier, J. and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2)", draft-ietf-hip-rfc4843-bis-02 (work in progress), September 2012.

[I-D.ietf-hip-rfc5202-bis] Jokela, P., Moskowitz, R., and J. Melen, "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)", draft-ietf-hip-rfc5202-bis-02 (work in progress), June 2013.

[NIST.800-131A.2011] National Institute of Standards and Technology, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths", NIST 800-131A, January 2011.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

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

[RFC2404] Madson, C. and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.

[RFC2410]	Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, November 1998.
[RFC2460]	Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
[RFC2536]	Eastlake, D., "DSA KEYs and SIGs in the Domain Name System (DNS)", RFC 2536, March 1999.
[RFC2785]	Zuccherato, R., "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME", RFC 2785, March 2000.
[RFC2898]	Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.
[RFC3110]	Eastlake, D., "RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)", RFC 3110, May 2001.
[RFC3447]	Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.
[RFC3484]	Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.
[RFC3526]	Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.
[RFC3602]	Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, September 2003.
[RFC3972]	Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
[RFC4034]	Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource

Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4754] Fu, D. and J. Solinas, "IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 4754, January 2007.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, May 2007.

[RFC5201] Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol", RFC 5201, April 2008.

[RFC5702] Jansen, J., "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5702, October 2009.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

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

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

12.2. Informative References

[AUR03] Aura, T., Nagarajan, A., and A. Gurtov, "Analysis of the HIP Base Exchange Protocol", in Proceedings of 10th Australasian Conference on Information

Security and Privacy, July 2003.

[CRO03] Crosby, SA. and DS. Wallach, "Denial of Service via Algorithmic Complexity Attacks", in Proceedings of Usenix Security Symposium 2003, Washington, DC., August 2003.

[DIF76] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory vol. IT-22, number 6, pages 644-654, Nov 1976.

[FIPS.197.2001] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[I-D.ietf-btns-c-api] Richardson, M., Williams, N., Komu, M., and S. Tarkoma, "C-Bindings for IPsec Application Programming Interfaces", draft-ietf-btns-c-api-04 (work in progress), March 2009.

[I-D.ietf-hip-rfc4423-bis] Moskowitz, R., "Host Identity Protocol Architecture", draft-ietf-hip-rfc4423-bis-05 (work in progress), September 2012.

[I-D.ietf-hip-rfc5204-bis] Laganier, J. and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension", draft-ietf-hip-rfc5204-bis-02 (work in progress), September 2012.

[I-D.ietf-hip-rfc5205-bis] Laganier, J., "Host Identity Protocol (HIP) Domain Name System (DNS) Extension", draft-ietf-hip-rfc5205-bis-02 (work in progress), September 2012.

[I-D.ietf-hip-rfc5206-bis] Henderson, T., Vogt, C., and J. Arkko, "Host Mobility with the Host Identity Protocol", draft-ietf-hip-rfc5206-bis-04 (work in progress), July 2012.

[KAU03] Kaufman, C., Perlman, R., and B. Sommerfeld, "DoS protection for UDP-based protocols", ACM Conference on Computer

and Communications Security , Oct 2003.

[KRA03] Krawczyk, H., "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols", in Proceedings of CRYPTO 2003, pages 400-425, August 2003.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, July 2004.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5338] Henderson, T., Nikander, P., and M. Komu, "Using the Host Identity Protocol with Legacy Applications", RFC 5338, September 2008.

[RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, June 2009.

[RFC5747] Wu, J., Cui, Y., Li, X., Xu, M., and C. Metz, "4over6 Transit Solution Using IP Encapsulation and MP-BGP Extensions", RFC 5747, March 2010.

[RFC6253] Heer, T. and S. Varjonen, "Host Identity Protocol Certificates", RFC 6253, May 2011.

[SECG] SECG, "Recommended Elliptic Curve Domain Parameters", SEC 2 , 2000, <http://www.secg.org/>.

Appendix A. Using Responder Puzzles

As mentioned in Section 4.1.1, the Responder may delay state creation and still reject most spoofed I2 packets by using a number of pre-calculated R1 packets and a local selection function. This appendix defines one possible implementation in detail. The purpose of this appendix is to give the implementors an idea on how to implement the mechanism. If the implementation is based on this appendix, it MAY contain some local modification that makes an attacker's task harder.

The Responder creates a secret value S, that it regenerates periodically. The Responder needs to remember the two latest values of S. Each time the S is regenerated, the R1 generation counter value is incremented by one.

The Responder generates a pre-signed R1 packet. The signature for pre-generated R1s must be recalculated when the Diffie-Hellman key is recomputed or when the R1_COUNTER value changes due to S value regeneration.

When the Initiator sends the I1 packet for initializing a connection, the Responder receives the HIT and IP address from the packet, and generates an #I value for the puzzle. The #I value is set to the pre-signed R1 packet.

```
#I value calculation:
#I = Ltrunc( RHASH ( S | HIT-I | HIT-R | IP-I | IP-R ), n)
where n = RHASH_len
```

The RHASH algorithm is the same that is used to generate the Responder's HIT value.

From an incoming I2 packet, the Responder receives the required information to validate the puzzle: HITs, IP addresses, and the information of the used S value from the R1_COUNTER. Using these values, the Responder can regenerate the #I, and verify it against the #I received in the I2 packet. If the #I values match, it can verify the solution using #I, #J, and difficulty #K. If the #I values do not match, the I2 is dropped.

```
puzzle_check:
V := Ltrunc( RHASH( I2.I | I2.hit_i | I2.hit_r | I2.J ), #K )
if V != 0, drop the packet
```

If the puzzle solution is correct, the #I and #J values are stored for later use. They are used as input material when keying material is generated.

Keeping state about failed puzzle solutions depends on the implementation. Although it is possible for the Responder not to keep any state information, it still may do so to protect itself against certain attacks (see Section 4.1.1).

Appendix B. Generating a Public Key Encoding from an HI

The following pseudo-code illustrates the process to generate a public key encoding from an HI for both RSA and DSA.

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

```
switch ( HI.algorithm )
{

case RSA:
buffer := encode_in_network_byte_order ( HI.RSA.e_len,
          ( HI.RSA.e_len > 255 ) ? 3 : 1 )
buffer += encode_in_network_byte_order ( HI.RSA.e, HI.RSA.e_len )
buffer += encode_in_network_byte_order ( HI.RSA.n, HI.RSA.n_len )
break;

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

}
```

Appendix C. Example Checksums for HIP Packets

The HIP checksum for HIP packets is specified in Section 5.1.1. Checksums for TCP and UDP packets running over HIP-enabled security associations are specified in Section 3.5. The examples below use [RFC3849] and [RFC5747] addresses, and HITs with the prefix of 2001:10 followed by zeros, followed by a decimal 1 or 2, respectively.

The following example is defined only for testing the checksum

calculation.

C.1. IPv6 HIP Example (I1 packet)

Source Address:	2001:d88::1	
Destination Address:	2001:d88::2	
Upper-Layer Packet Length:	48	0x30
Next Header:	139	0x8b
Payload Protocol:	59	0x3b
Header Length:	4	0x4
Packet Type:	1	0x1
Version:	2	0x2
Reserved:	1	0x1
Control:	0	0x0
Checksum:	6878	0x1ade
Sender's HIT :	2001:10::1	
Receiver's HIT:	2001:10::2	
DH_GROUP_LIST type:	511	0x1ff
DH_GROUP_LIST length:	3	0x3
DH_GROUP_LIST group IDs:	3,4,8	

C.2. IPv4 HIP Packet (I1 packet)

The IPv4 checksum value for the example I1 packet is shown below.

Source Address:	192.0.2.1	
Destination Address:	192.0.2.2	
Upper-Layer Packet Length:	48	0x30
Next Header:	139	0x8b
Payload Protocol:	59	0x3b
Header Length:	4	0x4
Packet Type:	1	0x1
Version:	2	0x2
Reserved:	1	0x1
Control:	0	0x0
Checksum:	61934	0xf1ee
Sender's HIT :	2001:10::1	
Receiver's HIT:	2001:10::2	
DH_GROUP_LIST type:	511	0x1ff
DH_GROUP_LIST length:	3	0x3
DH_GROUP_LIST group IDs:	3,4,8	

C.3. TCP Segment

Regardless of whether IPv6 or IPv4 is used, the TCP and UDP sockets use the IPv6 pseudo-header format [RFC2460], with the HITs used in place of the IPv6 addresses.

```
     Sender's HIT:                    2001:10::1
     Receiver's HIT:                  2001:10::2
     Upper-Layer Packet Length:       20               0x14
     Next Header:                     6                0x06
     Source port:                     65500            0xffdc
     Destination port:                22               0x0016
     Sequence number:                 1                0x00000001
     Acknowledgment number:           0                0x00000000
     Header length:                   20               0x14
     Flags:                           SYN              0x02
     Window size:                     65535            0xffff
     Checksum:                        28618            0x6fca
     Urgent pointer:                  0                0x0000


       0x0000:  6000 0000 0014 0640 2001 0010 0000 0000
       0x0010:  0000 0000 0000 0001 2001 0010 0000 0000
       0x0020:  0000 0000 0000 0002 ffdc 0016 0000 0001
       0x0030:  0000 0000 5002 ffff 6fca 0000
```

Appendix D. ECDH and ECDSA 160 Bit Groups

The ECDH and ECDSA 160-bit group SECP160R1 is rated at 80 bits symmetric strength. Once this was considered appropriate for one year of security. Today these groups should be used only when the host is not powerful enough (e.g., some embedded devices) and when security requirements are low (e.g., long-term confidentiality is not required).

Appendix E. HIT Suites and HIT Generation

The HIT as an ORCHID [I-D.ietf-hip-rfc4843-bis] consists of three parts: A 28-bit prefix, a 4-bit encoding of the ORCHID generation algorithm (OGA) and the representation of the public key. The OGA is an index pointing to the specific algorithm by which the public key and the 96-bit hashed encoding is generated. The OGA is protocol specific and is to be interpreted as defined below for all protocols that use the same context ID as HIP. HIP groups sets of valid combinations of signature and hash algorithms into HIT Suites. These HIT suites are addressed by an index, which is transmitted in the OGA field of the ORCHID.

The set of used HIT Suites will be extended to counter the progress in computation capabilities and vulnerabilities in the employed algorithms. The intended use of the HIT Suites is to introduce a new HIT Suite and phase out an old one before it becomes insecure. Since the 4-bit OGA field only permits 15 HIT Suites (the HIT Suite with ID 0 is reserved) to be used in parallel, phased-out HIT Suites must be

reused at some point. In such a case, there will be a rollover of the HIT Suite ID and the next newly introduced HIT Suite will start with a lower HIT Suite index than the previously introduced one. The rollover effectively deprecates the reused HIT Suite. For a smooth transition, the HIT Suite should be deprecated a considerable time before the HIT Suite index is reused.

Since the number of HIT Suites is tightly limited to 16, the HIT Suites must be assigned carefully. Hence, sets of suitable algorithms are grouped in a HIT Suite.

The HIT Suite of the Responder's HIT determines the RHASH and the hash function to be used for the HMAC in HIP control packets as well as the signature algorithm family used for generating the HI. The list of HIT Suites is defined in Table 11.

The following HIT Suites are defined for HIT generation. The input for each generation algorithm is the encoding of the HI as defined in Section 3.2. The output is 96 bits long and is directly used in the ORCHID.

Index	Hash function	HMAC	Signature algorithm family	Description
0				Reserved
1	SHA-256	HMAC-SHA-256	RSA, DSA	RSA or DSA HI hashed with SHA-256, truncated to 96 bits
2	SHA-384	HMAC-SHA-384	ECDSA	ECDSA HI hashed with SHA-384, truncated to 96 bits
3	SHA-1	HMAC-SHA-1	ECDSA_LOW	ECDSA_LOW HI hashed with SHA-1, truncated to 96 bits

Table 11: HIT Suites

The hash of the responder as defined in the HIT Suite determines the HMAC to be used for the HMAC parameter. The HMACs currently defined here are HMAC-SHA-256 [RFC4868], HMAC-SHA-384 [RFC4868], and HMAC-SHA-1 [RFC2404].

Authors' Addresses

Robert Moskowitz (editor)
Verizon Telcom and Business
1000 Bent Creek Blvd, Suite 200
Mechanicsburg, PA
USA

EMail: robert.moskowitz@verizonbusiness.com

Tobias Heer
Hirschmann Automation and Control
Stuttgarter Strasse 45-51
Neckartenzlingen 72654
Germany

EMail: tobias.heer@belden.com

Petri Jokela
Ericsson Research NomadicLab
JORVAS FIN-02420
FINLAND

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

Thomas R. Henderson
The Boeing Company
P.O. Box 3707
Seattle, WA
USA

EMail: thomas.r.henderson@boeing.com