Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2014

R. Moskowitz, Ed.
Verizon
R. Hummen
COMSYS, RWTH Aachen
March 04, 2014

HIP Diet EXchange (DEX)
draft-moskowitz-hip-dex-01

Abstract

This document specifies the Host Identity Protocol Diet EXchange (HIP DEX), a variant of the HIP Base EXchange (HIP BEX) [rfc5201-bis]. The HIP DEX protocol design aims at reducing the overhead of the employed cryptographic primitives by omitting public-key signatures and hash functions. In doing so, the main goal is to still deliver similar security properties to HIP BEX.

The HIP DEX protocol is primarily targeted at computation or memory-constrained sensor devices. Like HIP BEX, it is expected to be used together with another suitable security protocol such as the Encapsulated Security Payload (ESP) [rfc5202-bis] for the protection of upper layer protocols. HIP DEX can also be used as a keying mechanism for a MAC layer security protocol as is supported by IEEE 802.15.4 [IEEE.802-15-4.2011].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 5, 2014.

Copyright Notice

Table of Contents

1. Introduction

This memo specifies the Host Identity Protocol Diet EXchange (HIP DEX). HIP DEX builds on the HIP Base EXchange (HIP BEX) [rfc5201-bis]. Notably, HIP DEX preserves the protocol semantics as well as the general packet structure of HIP BEX. It is therefore recommended that [rfc5201-bis] is well-understood before reading this document.

The main differences between HIP BEX and HIP DEX are:

1. Minimum collection of cryptographic primitives.

    * Static Elliptic Curve Diffie-Hellman key pairs for encryption of the session key.

* AES-CTR for symmetric encryption and AES-CMAC for MACing functions.

* A simple fold function for HIT generation.

2. Forfeit of Perfect Forward Secrecy with the dropping of ephemeral Diffie-Hellman.

3. Forfeit of digital signatures with the removal of a hash function. Reliance on ECDH derived key used in HIP_MAC to prove ownership of the private key.

4. Diffie-Hellman derived key ONLY used to protect the HIP packets. A separate secret exchange within the HIP packets creates the session key(s).

5. Optional retransmission strategy tailored to handle the potentially extensive processing time of the cryptographic operations and the lossy nature of the communication links in sensor networks.

In this document, we focus on the protocol specifications related to these differences. Where differences are not call out explicitly, HIP DEX is the same as specified in [rfc5201-bis].

1.1. The HIP Diet EXchange (DEX)

The HIP Diet EXchange is a two-party cryptographic protocol used to establish a secure communication context between hosts. The first party is called the Initiator and the second party the Responder. The four-packet exchange helps to make HIP DoS resilient. The protocol exchanges Static Elliptic Curve Diffie-Hellman keys in the 2nd and 3rd packets and transmits keying material for the session key and authenticates the parties in the 3rd and 4th packets. Additionally, the Responder starts a puzzle exchange in the 2nd packet, with the Initiator completing it in the 3rd packet before the Responder performs computationally expensive operations or stores any state from the exchange. Thus, DEX is operationally very similar to BEX. The model is also fairly equivalent to 802.11-2007 [IEEE.802-11.2007] Master Key and Pair-wise Transient Key, but handled in a single exchange.

HIP DEX does not have the option to encrypt the Host Identity of the Initiator in the 3rd packet. The Responder's Host Identity is also not protected. Thus, there is no attempt at anonymity as in HIP BEX.

Data packets start to flow after the 4th packet. Similarly to HIP BEX, DEX does not have an explicit transition to connected state for

the Responder. When the Responder starts receiving protected datagrams, indicating that the Initiator received the R2 packet, it shifts to connected state. As such the Initiator should take care to NOT send the first data packet until some delta time after it received the R2 packet. This is to provide time for the Responder to process any aggressively retransmitted I2 packets.

An existing HIP association can be updated with the update mechanism defined [rfc5201-bis]. Likewise, the association can be torn down with the defined closing mechanism for HIP BEX if it is no longer needed. HIP DEX thereby omits the HIP_SIGNATURE parameters of the original HIP BEX specification.

Finally, HIP DEX is designed as an end-to-end authentication and key establishment protocol. As such, it can be used in combination with Encapsulated Security Payload (ESP) [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 DEX is not a replacement for IKEv2.

1.2. 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 Diet 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.

FOLD (x, K) denotes dividing x into n fragments of K length and XORing them together. The last fragment is padded to K bit by appending 0 bits.

Ltrunc (M(x), K) denotes the lowest order K bits of the result of the mac function M on the input x.

2.3. Definitions

HIP Diet Exchange (DEX): The ECDH-based HIP handshake for establishing a new HIP association.

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

Host Identity Tag (HIT): A shorthand for the HI in IPv6 format. It is generated by folding the HI (c.f. Section 3).

HIT Suite: A HIT Suite groups all algorithms that are required to generate and use an HI and its HIT. In particular, these algorithms are: 1) ECDH and 2) FOLD.

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

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

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

Responder's HIT Hash Algorithm (RHASH): In HIP DEX, RHASH is redefined as CMAC. Note that CMAC is a message authentication code and not a cryptographic hash function. As such, RHASH has different security properties in DEX and is not used for HIT generation.

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

CKDF: CMAC-based Key Derivation Function.

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 [rfc4423-bis] for more details on the difference between an identity and the corresponding identifier.

HIP DEX implementations MUST support the Elliptic Curve Diffie-Hellman (ECDH) [RFC6090] key exchange for generating the HI as defined in Section 5.2.3. No additional algorithms are supported at this time.

A compressed encoding of the HI, the Host Identity Tag (HIT), is used in the handshake to represent the Host Identity. The DEX Host Identity Tag (HIT) is different from the BEX HIT in two ways:

- The HIT suite ID MUST only be a DEX HIT ID (see Section 5.2.1).
- The DEX HIT is not generated via a cryptographic hash. Rather, it is a compression of the Host Identity.

Due to the latter property, an attacker may be able to find a collision with a HIT that is in use. Hence, policy decisions such as access control MUST NOT be based on the HIT. Instead, the HI of a host SHOULD be considered.

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 DEX Host Identity Tag is a 128-bit value - a compression of the HI prepended with a specific prefix. There are two advantages of using a Host Identity Tag over the actual Host Identity public key in protocols. Firstly, its fixed length makes for easier protocol coding and also better manages the packet size cost of this technology. Secondly, it presents a consistent format to the protocol whatever underlying identity technology and Host Identifier size is used.

The structure of the HIT is based on RFC 4843-bis [rfc4843-bis], called Overlay Routable Cryptographic Hash Identifiers (ORCHIDs), 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 compressed representation of HI. Third, a 96-bit hashed representation of the Host Identity. In contrast to BEX HITs, DEX HITs are NOT generated by means of a cryptographic hash. Instead, the HI is compressed to 96 bits.

3.2. Generating a HIT from an HI

The DEX HIT does not follow the exact semantics of an ORCHID as there is no hash function in DEX. However, its structure is strongly aligned with the ORCHID design. The same IPv6 prefix used in BEX is used for DEX. The DEX HIT suite (see Section 9) is used for the four bits of the Orchid Generation Algorithm (OGA) field in the ORCHID. The hash representation in an ORCHID is replaced with a FOLD(HI,96).

4. Protocol Overview

This section gives a simplified overview of the HIP DEX protocol operation and does not contain all the details of the packet formats or the packet processing steps. Section 5 and Section 6 describe these aspects in more detail and are normative in case of any conflicts with this section. Importantly, the information given in this section focuses on the differences between the BEX and DEX specifications of the HIP protocol.

4.1. Creating a HIP Association

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

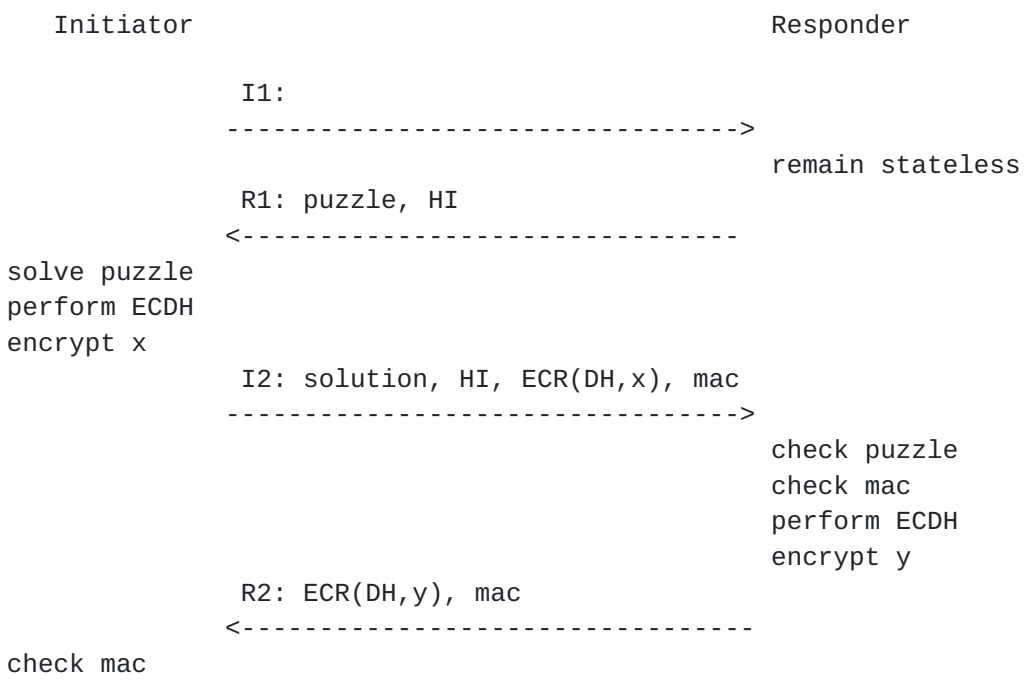
The HIP Diet EXchange serves to manage the establishment of state between an Initiator and a Responder. The first packet, I1, initiates the handshake, and the last three packets, R1, I2, and R2, constitute an authenticated Diffie-Hellman key exchange for the Master Key SA generation. This Master Key SA is used by the Initiator and the Responder to wrap secret keying material. Based on the exchanged keying material, the peers then derive a Pair-wise Key SA. If cryptographic keys are needed, e.g., for ESP, these keys are expected to be drawn from the wrapped keying material.

The second packet, R1, starts the actual 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. The R1 also contains lists of cryptographic algorithms supported by the Responder. Based on these lists, the Initiator can continue, abort, or restart the handshake with a different selection of cryptographic algorithms.

In the I2 packet, the Initiator must display the solution to the received puzzle. Without a correct solution, the I2 packet is discarded. The I2 also contains a key wrap parameter that carries a secret keying material of the Initiator. This keying material is only half the final session key. The packet is authenticated by the sender (Initiator) via a MAC.

The R2 packet finalizes the handshake. The R2 contains a key wrap parameter that carries the rest of the keying material of the Responder. The packet is authenticated by the sender (Initiator) via a MAC.

The HIP DEX handshake is illustrated below. The terms "ECR(DH,x)" and "ECR(DH,y)" refer to the wrapped random values that each constitute half the final session key. The packets contain other parameters not shown in this figure.



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 mechanism enables a Responder to immediately reject an I2 packet if it does not contain a valid puzzle solution. To verify the puzzle solution, the Responder only has to compute a single CMAC function. After a successful puzzle verification, the Responder can securely create session-specific state and perform CPU-intensive operations such as a Diffie-Hellman key generation. By varying the difficulty of the puzzle, the Responder can frustrate CPU or memory targeted DoS attacks. Under normal conditions, the puzzle difficulty SHOULD be zero, thus effectively disabling the puzzle-based DoS protection mechanism. Otherwise, computation resources at the Initiator would be churned unnecessarily.

Conceptually, the puzzle mechanism is the same in HIP DEX as in HIP BEX. The only difference is that DEX uses the CMAC function instead of a hash function for solving and verifying a cryptographic puzzle. Implementations SHOULD include sufficient randomness to the algorithm so that algorithmic complexity attacks become impossible [CRO03]. See Appendix A in [rfc5201-bis] for one possible implementation.

The signaling information of the puzzle exchange is the same as in HIP BEX. Hence, this document refers to 4.1.2. in [rfc5201-bis] for detailed information about the puzzle exchange.

4.1.2. HIP State Machine

The HIP DEX state machine has the same states as the BEX state machine (see 4.4. in [rfc5201-bis]). However, HIP DEX features an retransmission strategy with an optional packet receipt for the I2. The goal of this packet receipt is reducing premature I2 retransmissions in sensor networks with low computation resources and high packet loss [hummen2013tailoring]. As a result, there are minor changes to the transitioning steps between specific states. The following section documents these differences in the HIP DEX state machine compared to HIP BEX.

4.1.2.1. HIP DEX Retransmission Mechanism

For the retransmission of I1 and I2 packets, the Initiator adopts the retransmission strategy of HIP BEX (see Section 4.4.3. in [rfc5201-bis]). This strategy is based on a timeout value that is set to the worst-case anticipated round-trip time (RTT). For each received I1 or I2, the Responder sends an R1 or R2, respectively. This design trait enables the Responder to remain stateless until the reception of the I2. The Initiator stops retransmitting I1 or I2 packets after the reception of the corresponding R1 or R2. If the Initiator did not receive an R1 after I1_RETRIES_MAX tries, it stops I1 retransmissions. Likewise, it stops retransmitting I2 packets after I2_RETRIES_MAX unsuccessful tries.

The Responder SHOULD NOT perform operations related to the Diffie-Hellman key exchange or the keying material wrapped in the ENCRYPTED_KEY parameters for retransmitted I2 packets. Instead, it SHOULD re-use the previously established state to re-create the R2.

The potentially high processing time of an I2 packet at the Responder may cause retransmissions if the time required for I2 transmission and processing exceeds the RTT-based retransmission timeout. Thus, the Initiator should also take the processing time of I2 packets into account. To this end, the Responder MAY optionally notify the Initiator about the anticipated delay if the I2 incurs a considerable processing overhead. The Responder MAY therefore send a NOTIFY packet to the Initiator before it commences the ECDH operation. The NOTIFY packet serves as an acknowledgement for the I2 and consists of a NOTIFICATION parameter with Notify Message Type I2_ACKNOWLEDGEMENT (see Section 5.2.19. in [rfc5201-bis]). The NOTIFICATION parameter contains the anticipated remaining processing time for the I2 packet in milliseconds as Notification Data . This processing time can,

e.g., be estimated by measuring the computation time of the ECDH key derivation operation at Responder boot-up. After the I2 processing has finished, the Responder sends the regular R2.

When the Initiator receives the NOTIFY packet, it resets the I2 retransmission timer to the processing time indicated by the Responder in the NOTIFICATION parameter. If the indicated processing time is shorter than the RTT-based timeout, the Initiator MUST set the retransmission timer to the RTT-based timeout. Additionally, the Initiator MUST NOT set a higher retransmission timeout than allowed by a local policy. Hence, I2 retransmissions are never triggered in shorter succession than without this optional retransmission extension. Moreover, there is a defined upper bound to which unauthenticated NOTIFY messages can delay the handshake in case of lost R2 packets.

4.1.2.2. HIP State Processes

HIP DEX clarifies or introduces the following new transitions.

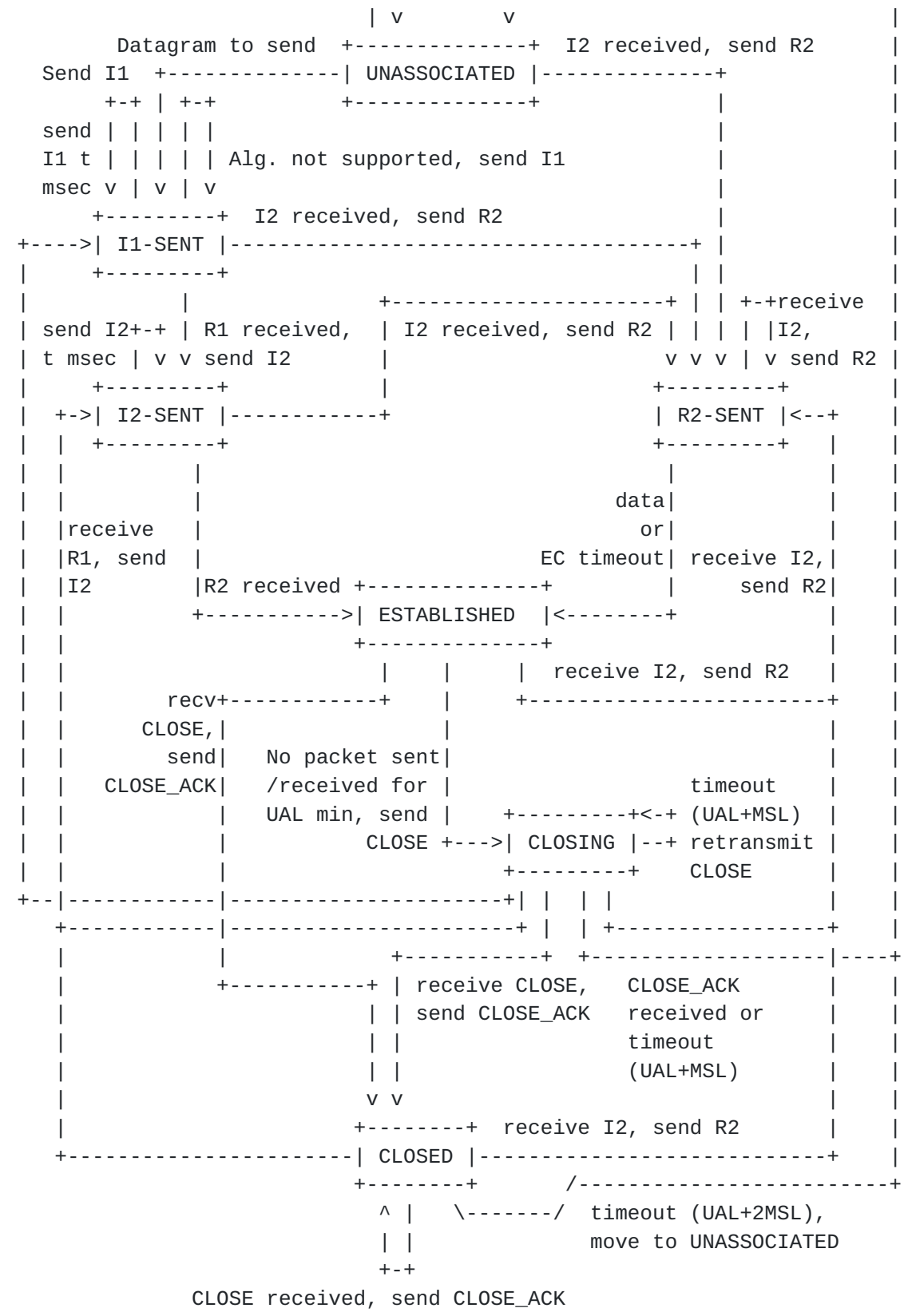
System behavior in state I2-SENT, Table 1.

Trigger	Action
Receive NOTIFY, process	Set I2 retransmission timer according to NOTIFY payload and stay at I2-SENT
Timeout	Increment timeout counter If counter is less than I2_RETRIES_MAX, send I2, reset timer to RTT and stay at I2-SENT If counter is greater than I2_RETRIES_MAX, go to E-FAILED

Table 1: I2-SENT - Waiting to finish the HIP Diet EXchange

4.1.2.3. Simplified HIP State Diagram

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


Datagram to send
Send I1
send I1 t msec
UNASSOCIATED
I2 received, send R2
Alg. not supported, send I1
I1-SENT
I2 received, send R2
send I2+t msec
R1 received, send I2
I2-SENT
receive R1, send I2
R2 received
ESTABLISHED
data or EC timeout
receive I2, send R2
receive I2, send R2
recv CLOSE, send CLOSE_ACK
No packet sent /received for UAL min, send CLOSE
CLOSING
timeout (UAL+MSL) retransmit CLOSE
receive CLOSE, send CLOSE_ACK
CLOSE_ACK received or timeout (UAL+MSL)
CLOSED
timeout (UAL+2MSL), move to UNASSOCIATED
CLOSE received, send CLOSE_ACK

4.1.3. HIP DEX Security Associations

HIP DEX establishes two Security Associations (SA), one for the Diffie-Hellman derived key, or Master Key, and one for the session key, or Pair-wise Key.

4.1.3.1. Master Key SA

The Master Key SA is used to authenticate HIP packets and to encrypt selected HIP parameters in HIP DEX packet exchanges. Since only little data is protected by this SA, it can be long-lived with no need for rekeying.

The Master Key SA contains the following elements:

- Source HIT
- Destination HIT
- HIP_Encrypt Key
- HIP_MAC Key

The HIP_Encrypt and HIP_MAC keys are extracted from the Diffie-Hellman derived key as described in Section 6.2. Their length is determined by the HIP_CIPHER.

4.1.3.2. Pair-wise Key SA

The Pair-wise Key SA is used to authenticate and to encrypt user data. It is refreshed (or rekeyed) using an UPDATE packet exchange. The Pair-wise Key SA elements are defined by the data transform (e.g. ESP_TRANSFORM [rfc5202-bis]).

The keys for the Pair-wise Key SA are derived based on the wrapped keying material exchanged in the ENCRYPTED_KEY parameter (see Section 5.2.5) of the I2 and R2 packets. Specifically, the exchanged keying material of the two peers is concatenated. This concatenation forms the input to a Key Derivation Function (KDF). If the data transform does not specify its own KDF, then the key derivation function defined in Section 6.2 is used. Even though this input is randomly distributed, a KDF Extract phase may be needed to get the proper length for the input to the KDF Expand phase.

4.1.4. User Data Considerations

The User Data Considerations in Section 4.5. of [rfc5201-bis] also apply to HIP DEX. There is only one difference between HIP BEX and

HIP DEX. Loss of state due to system reboot may be a critical performance issue for constrained sensor devices. Thus, implementors MAY choose to use non-volatile, secure storage for HIP states in order for them to survive a system reboot. This will limit state loss during reboots to only those situations with an SA timeout.

5. Packet Formats

5.1. Payload Format

HIP DEX employs the same fixed HIP header and payload structure as HIP BEX. As such, the specifications in Section 5.1 of [rfc5201-bis] also apply to HIP DEX.

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.

HIP DEX reuses the HIP parameters of HIP BEX defined in Section 5.2. of [rfc5201-bis] where possible. Still, HIP DEX further restricts and/or extends the following existing parameter types:

- HIT_SUITE_LIST is limited to HIT suite ECDH/FOLD.
- DH_GROUP_LIST and HOST_ID are restricted to ECC-based suites.
- HIP_CIPHER is restricted to NULL-ENCRYPT and AES-128-CTR.
- RHASH and RHASH_len are redefined to CMAC for PUZZLE, SOLUTION, HIP_MAC (see Section 6 and Section 6.1).

In addition, HIP DEX introduces the following new parameter:

TLV	Type	Length	Data
ENCRYPTED_KEY	643	variable	Encrypted container for key generation exchange

5.2.1. HIT_SUITE_LIST

The HIT_SUITE_LIST parameter contains a list of the supported HIT suite IDs of the Responder. The HIT suites in DEX are limited to:

HIT suite	ID
ECDH/FOLD	8

Since the HIT of the Initiator is a DEX HIT, the Responder MUST only respond with a DEX HIT suite ID.

5.2.2. DH_GROUP_LIST

The DH_GROUP_LIST parameter contains the list of supported DH Group IDs of a host. The following ECC curves are supported in HIP DEX:

Group	KDF	Value
NIST P-256 [RFC5903]	CKDF	7
NIST P-384 [RFC5903]	CKDF	8
NIST P-521 [RFC5903]	CKDF	9
SECP160R1 [SECG]	CKDF	10

The ECDH groups 7 - 9 are defined in [RFC5903] and [RFC6090]. ECDH group 10 is covered in [rfc5201-bis].

5.2.3. HOST_ID

The HI Algorithms in DEX are limited to:

Algorithm profiles	Values
ECDH	1

ECC-based Host Identities are serialized as described in Section 5.2.9. of [rfc5201-bis]. The supported curves for the HI in HIP DEX are defined in Section 5.2.2.

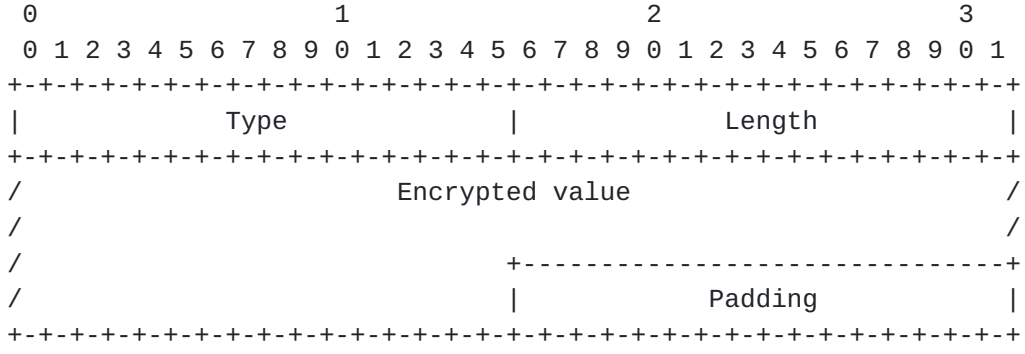
5.2.4. HIP_CIPHER

The HIP ciphers in DEX are limited to:

Suite ID	Value	
RESERVED	0	
NULL-ENCRYPT	1	([RFC2410])
AES-128-CTR	5	([RFC3686])

Mandatory implementation: AES-128-CTR. NULL-ENCRYPTION [RFC2410] is included for testing purposes.

5.2.5. ENCRYPTED_KEY



Type	643
Length	length in octets, excluding Type, Length, and Padding
Encrypted value	The value is encrypted using an encryption algorithm as defined in the HIP_CIPHER parameter.

The ENCRYPTED_KEY parameter encapsulates a random value that is later used in the session key creation process. The Puzzle value I (see [rfc5201-bis]) is used as the initialization vector (IV) for the encryption process. The AES-CTR counter is a 16 bit value that is initialized to zero with the first use.

Once this encryption process is completed, the "encrypted value" 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 TLV Format in [rfc5201-bis].

5.3. HIP Packets

DEX uses the same eight basic HIP packets as in BEX (see [rfc5201-bis]). Four are for the HIP handshake (I1, R1, I2, and R2), one is for updating (UPDATE), one is for sending notifications (NOTIFY), and two are for closing a HIP association (CLOSE and CLOSE_ACK). There are some differences regarding the included HIP parameters in the exchange packets of BEX and DEX. This section covers these differences for the DEX packets. Packets not discussed here, follow the structure defined in [rfc5201-bis].

An important difference between packets in HIP BEX and HIP DEX is that the DIFFIE_HELLMAN and the HIP_SIGNATURE parameters are not included in DEX. Thus, the R1 packet is completely unprotected and can be spoofed. As a result, negotiation parameters contained in the R1 packet have to be re-included in later, protected packets in order to detect and prevent potential downgrading attacks. Moreover, the I2, R2, UPDATE, NOTIFY, CLOSE, and CLOSE_ACK packets are not covered by a signature and purely rely on the HIP_MAC parameter for packet authentication. The processing of these packets is changed accordingly.

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.

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 ) )

  Minimum size = 48 bytes
```

Valid control bits: none

The I1 packet contains the fixed HIP header and the Initiator's DH_GROUP_LIST. The Initiator's HIT Suite ID MUST be of a DEX type.

The Initiator receives the Responder's HIT either from a DNS lookup of the Responder's FQDN, from some other repository, or from a local table. The Responder's HIT MUST be a DEX HIT. 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" [rfc5201-bis].

The Initiator includes a DH_GROUP_LIST parameter in the I1 to inform the Responder of its preferred DH Group IDs. Only ECDH groups may be included in this list. The group ID corresponding to the current Initiator HIT MUST be listed first in this DH_GROUP_LIST parameter to notify the Responder about the DH group that must be used to successfully complete the handshake.

Since this packet is so easy to spoof even if it were protected, no attempt is made to add to its generation or processing cost. As a result, the DH_GROUP_LIST in the I1 packet is not protected.

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,
             DH_GROUP_LIST,
             HIP_CIPHER,
             HOST_ID,
             HIT_SUITE_LIST,
             TRANSPORT_FORMAT_LIST,
             [ <, ECHO_REQUEST_UNSIGNED >i ])

  Minimum size = 120 bytes
```

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 MUST match the Initiator's request. If the Initiator used opportunistic mode, the Responder may select among its HIs as described below. See also "HIP Opportunistic Mode" [rfc5201-bis].

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 value #I and the puzzle difficulty K. The difficulty K indicates the number of lower-order bits, in the puzzle CMAC result, that MUST be zeros (see [rfc5201-bis]). Responders SHOULD set K to zero by default and only increase the puzzle difficulty to protect against a DoS attack targeting the DEX handshake. A puzzle difficulty of zero effectively disables the puzzle mechanism and is strongly encouraged to conserve energy resources as well as to prevent unnecessary handshake delay in case of a resource-constrained Initiator during normal operation.

The HIP_CIPHER contains the encryption algorithms supported by the Responder to protect the key exchange, in the order of preference. All implementations MUST support the AES-CTR [RFC3686].

The used Initiator HIT already determines the DH group that must be used to successfully conclude the handshake. However, while the Initiator HIT conveys that ECDH keys must be used for the HOST_ID, it does not provide information about the employed DH group. Still, the Responder can determine the DH Group ID based on the DH_GROUP_LIST parameter in the I1. If the I1 contains a Responder HIT, the Responder verifies that its Responder HIT matches the required DH group of the Initiator. The Responder then selects the corresponding HOST_ID for inclusion in the R1. If the Responder HIT is NULL (i.e., during an opportunistic handshake) the Responder chooses its HOST_ID according to the Initiator's preference expressed in the DH_GROUP_LIST parameter in the I1. The Responder sends back its own preference based on which it chose the HOST_ID as DH_GROUP_LIST. This allows the Initiator to determine whether its own DH_GROUP_LIST in the I1 was manipulated by an attacker. There is a further risk that the Responder's DH_GROUP_LIST was manipulated by an attacker, as R1 cannot be authenticated in DEX as it can in BEX. Thus, it is repeated in R2 allowing for a final check at that point.

The HIT_SUITE_LIST parameter is an ordered list of the Responder's supported and preferred HIT Suites. It enables a Responder to notify the Initiator about other available HIT suites than the one used in the current handshake. Based on the received HIT_SUITE_LIST, the Initiator MAY decide to abort the current handshake and initiate a new handshake based on a different mutually supported HIT suite. This mechanism can, e.g., be used to move from an initial HIP DEX handshake to a HIP BEX handshake for peers supporting both protocol variants.

The TRANSPORT_FORMAT_LIST parameter is an ordered list of the Responder's supported and preferred transport format types. The list

allows the Initiator and the Responder to agree on a common type for payload protection.

The ECHO_REQUEST_UNSIGNED parameters contain data that the sender wants to receive unmodified in the corresponding response packet in the ECHO_RESPONSE_UNSIGNED parameter. The R1 packet may contain zero or more ECHO_REQUEST_UNSIGNED parameters.

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,
             HIP_CIPHER,
             ENCRYPTED_KEY,
             HOST_ID,
             TRANSPORT_FORMAT_LIST,
             HIP_MAC,
             [<, ECHO_RESPONSE_UNSIGNED>i )] )

  Minimum size = 180 bytes
```

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 bit 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 HIP_CIPHER contains the single encryption transform selected by the Initiator, that it uses to encrypt the ENCRYPTED and ENCRYPTED_KEY parameters. The chosen cipher MUST correspond to one of the ciphers offered by the Responder in the R1. All implementations MUST support the AES-CTR transform [RFC3686].

The HOST_ID parameter contains the Initiator HI corresponding to the Initiator HIT.

The ENCRYPTED_KEY contains an Initiator generated random value that MUST be uniformly distributed. This random value is encrypted with the Master Key SA using the HIP_CIPHER encryption algorithm.

The ECHO_RESPONSE_UNSIGNED contains the unmodified Opaque data copied from the corresponding echo request parameter(s). This parameter can also be used for two-factor password authentication as shown in Appendix B.

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 [rfc5202-bis].

The MAC is calculated over the whole HIP envelope, excluding any parameters after the HIP_MAC, as described in Section 6.1. The Responder MUST validate the HIP_MAC.

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 ( DH_GROUP_LIST,
             HIP_CIPHER,
             ENCRYPTED_KEY,
             HIT_SUITE_LIST,
             TRANSPORT_FORMAT_LIST,
             HIP_MAC)

  Minimum size = 108 bytes
```

Valid control bits: none

The HITs used MUST match the ones used in the I2.

The Responder repeats the DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST parameters in R2. These parameters MUST be the same as included in R1. The parameter are re-included here because R2 is MACed and thus cannot be altered by an attacker. For verification purposes, the Initiator re-evaluates the selected suites

and compares the results against the chosen ones. If the re-evaluated suites do not match the chosen ones, the Initiator acts based on its local policy.

The ENCRYPTED_KEY contains an Responder generated random value that MUST be uniformly distributed. This random value is encrypted with the Master Key SA using the HIP_CIPHER encryption algorithm.

The MAC is calculated over the whole HIP envelope, excluding any parameters after the HIP_MAC, as described in Section 6.1. The Initiator MUST validate the HIP_MAC.

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 reply 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. The problem cases specified in Section 5.4. of [rfc5201-bis] also apply to HIP DEX.

6. Packet Processing

Due to the adopted protocol semantics and the inherited general packet structure, packet processing in HIP DEX only differs from HIP BEX in very few places. Here, we focus on these differences and refer to Section 6 in [rfc5201-bis] otherwise.

The processing of outgoing and incoming application data remains the same as in HIP BEX (see Sections 6.1 and 6.2 in [rfc5201-bis]).

Puzzle solving and verification is very similar to HIP BEX with the exception that RHASH is replaced with CMAC in HIP DEX. Furthermore, R1 packets are not pre-computed in HIP DEX. The remaining procedure in Section 6.3 of [rfc5201-bis] is unchanged.

6.1. HIP_MAC Calculation and Verification

The following subsections define the actions for processing the HIP_MAC parameter.

6.1.1. CMAC Calculation

The HIP_MAC calculation uses RHASH, i.e., CMAC, as the underlying cryptographic function. The scope of the calculation for HIP_MAC is:

CMAC: { 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 [rfc5201-bis].

The CMAC calculation and verification process is as follows:

Packet sender:

1. Create the HIP packet, without the HIP_MAC or any other parameter with greater Type value than the HIP_MAC parameter has.
2. Calculate the Header Length field in the HIP header.
3. Compute the CMAC using either HIP-gl or HIP-lg integrity key retrieved from KEYMAT as defined in Section 6.2.
4. Add the HIP_MAC parameter to the packet and any parameter with greater Type value than the HIP_MAC's that may follow.
5. Recalculate the Length field in the HIP header.

Packet receiver:

1. Verify the HIP header Length field.
2. Remove the HIP_MAC parameter, as well as all other parameters that follow it with greater Type value, saving the contents if they will be needed later.
3. Recalculate the HIP packet length in the HIP header and clear the Checksum field (set it to all zeros).
4. Compute the CMAC using either HIP-gl or HIP-lg integrity key as defined in Section 6.2 and verify it against the received CMAC.

5. Set Checksum and Header Length fields in the HIP header to original values.

6.2. HIP DEX KEYMAT Generation

The HIP DEX KEYMAT process is used to derive the keys for Master Key SA as well as for the Pair-wise Key SA. The keys for the Master Key SA are based from the Diffie-Hellman derived key, Kij, produced during the HIP Diet EXchange. The Initiator generates Kij during the creation of the I2 packet and the Responder generates Kij once it receives the I2 packet. Hence, I2, R2, UPDATE, CLOSE, and CLOSE_ACK packets can contain authenticated and/or encrypted information.

The keys of the Pair-wise Key SA are not directly used in the HIP DEX handshake. Instead, these keys are made available as payload protection keys. Some payload protection mechanisms have their own Key Derivation Function, and if so this mechanism SHOULD be used. Otherwise, the HIP DEX KEYMAT process MUST be used.

The HIP DEX KEYMAT process consists of two components, CKDF-Extract and CKDF-Expand. The Extract function COMPRESSES a non-uniformly distributed key, as is the output of a Diffie-Hellman key derivation, to EXTRACT the key entropy into a fixed length output. The Expand function takes either the output of the Extract function or directly uses a uniformly distributed key and EXPANDS the length of the key, repeatedly distributing the key entropy, to produce the keys needed.

The key derivation for Master Key SA employs both the Extract and Expand phases, while the Pair-wise Key SA MAY need the Extract and Expand phases if the key is longer than 128 bits. Otherwise, it only needs the Expand phase.

The CKDF-Extract function is the following operation, where | denotes the concatenation.

```
CKDF-Extract(I, BigK, info) -> PRK
```

where

I	I from the PUZZLE parameter
BigK	Diffie-Hellman derived key or concatenation of the keying material exchanged in the ENCRYPTED_KEY parameters in the same order as the HITs for the info input (i.e., if HIT-I < HIT-R, then the keying material of the Initiator precedes the keying material of the Responder)
info	sort(HIT-I | HIT-R) | "CKDF-Extract"
PRK	a pseudorandom key (of RHASH_len/8 octets)
|	denotes the concatenation

The pseudorandom key PRK is calculated as follows:

```
PRK     = CMAC(I, BigK | info)
```

The CKDF-Expand function is the following operation:

```
CKDF-Expand(PRK, info, L) -> OKM

  PRK       a pseudorandom key of at least RHASH_len/8 octets
            (usually, the output from the extract step or
            concatenation of the keying material exchanged
            in the ENCRYPTED_KEY parameters in the same order
            as the HITs for the info input, i.e., if
            HIT-I < HIT-R, then the keying material of the
            Initiator precedes the keying material of the
            Responder)
  info      sort(HIT-I | HIT-R) | "CKDF-Expand"
  L         length of output keying material in octets
            (<= 255*RHASH_len/8)
  |         denotes the concatenation
```

The output keying material OKM is calculated as follows:

```
N       =  ceil(L/RHASH_len/8)
T       =  T(1) | T(2) | T(3) | ... | T(N)
OKM     =  first L octets of T
```

where

```
    T(0) = empty string (zero length)
    T(1) = CMAC(PRK, T(0) | info | 0x01)
    T(2) = CMAC(PRK, T(1) | info | 0x02)
    T(3) = CMAC(PRK, T(2) | info | 0x03)
    ...
```

(where the constant concatenated to the end of each T(n) is a single octet.)

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 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 initial keys:

1. HIP-gl encryption key for HOST_g's outgoing HIP packets

2. HIP-gl integrity (CMAC) key for HOST_g's outgoing HIP packets

3. HIP-lg encryption key for HOST_l's outgoing HIP packets

4. HIP-lg integrity (CMAC) 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

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

6.3. Initiation of a HIP Diet EXchange

The initiation of a HIP DEX handshake proceeds as described in Section 6.6. of [rfc5201-bis].

6.4. Processing Incoming I1 Packets

I1 packets in HIP DEX are handled identically to HIP BEX (see Section 6.7. in [rfc5201-bis]). The only differences are that the Responder SHOULD select a DEX HIT in the R1 response. Moreover, as R1 packets are neither covered by a signature nor incur the overhead of generating an ephemeral Diffie-Hellman key-pair, pre-computation of an R1 is only marginally beneficial, but would incur additional memory resources. Hence, the R1 pre-computation is omitted in HIP DEX.

6.5. Processing Incoming R1 Packets

R1 packets in HIP DEX are handled identically to HIP BEX with the following differences (see Section 6.8. in [rfc5201-bis]). Only step 4 is omitted in HIP DEX as there is no HIP_SIGNATURE in the R1 packet.

6.6. 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. To this end, 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, verifying the MAC, extracting the ENCRYPTED_KEY, 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 in [rfc5201-bis] 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 MUST check if the newly received I2 packet is similar to the one that triggered moving to R2-SENT. If so, it MUST retransmit a previously sent R2 packet and the state machine stays in R2-SENT.

5. If the system's state machine is in the I2-SENT state, the system MUST make a comparison between its local and sender's HITs (similarly as in Section 6.2). If the local HIT is smaller than the sender's HIT, it should drop the I2 packet, use the peer Diffie-Hellman key, ENCRYPTED_KEY keying material and nonce #I from the R1 packet received earlier, and get the local Diffie-Hellman key, ENCRYPTED_KEY keying material, 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 and ENCRYPTED_KEY keying material it might have generated upon sending the I2 packet previously. The peer Diffie-Hellman key, ENCRYPTED_KEY, and the nonce #J are taken from the just arrived I2 packet. The local Diffie-Hellman key, ENCRYPTED_KEY keying material, 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 as described in Section 6.

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 HOST_ID parameter. This key is used to derive the keys of the Master Key SA as described in Section 6.2. If the Diffie-Hellman Group ID is unsupported, the I2 packet is silently dropped.

11. 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.)

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

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

14. 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.

15. 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.

16. The system MUST extract the keying material from the ENCRYPTED_KEY parameter. This keying material is a partial input to the key derivation process for the Pair-wise Key SA (see Section 6.2).

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

18. 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.

19. 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.

20. 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.

6.7. Processing 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 according to the procedures in Section 6.1.

4. The system MUST re-evaluate the DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST parameters in the R2 and compare the results against the chosen suites.

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. The system MUST extract the keying material from the ENCRYPTED_KEY parameter. This keying material is a partial input to the key derivation process for the Pair-wise Key SA (see Section 6.2).

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

6.8. Processing UPDATE, NOTIFY, CLOSE, and CLOSE_ACK Packets

UPDATE, NOTIFY, CLOSE, and CLOSE_ACK packets are handled similarly in HIP DEX as in HIP BEX (see Sections 6.11. - 6.15. in [rfc5201-bis]). The only difference is the that the HIP_SIGNATURE is never present and, therefore, is not needed be processed.

6.9. Handling State Loss

Implementors MAY choose to use non-volatile, secure storage for HIP states in order for them to survive a system reboot. If no secure storage capabilities are available, the system SHOULD delete the corresponding HIP state, including the keying material. 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 exchanges that each host must support. All HIP DEX implementations SHOULD provide for an ACL of Initiator's HI to Responder's HI. This ACL SHOULD also include preferred transform and local lifetimes. Wildcards SHOULD also be supported for this ACL.

The value of the puzzle difficulty #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.

8. Security Considerations

HIP DEX replaces the SIGMA-based authenticated Diffie-Hellman key exchange of HIP BEX with a exchange of random keying material that is encrypted by a Diffie-Hellman derived key. Both the Initiator and Responder contribute to this keying material.

- The strength of the keys for the Pair-wise Key SA is based on the quality of the random keying material generated by the Initiator and Responder. Since the Initiator is commonly a sensor there is a natural concern about the quality of its random number generator.

- DEX lacks Perfect Forward Secrecy (PFS). If the Initiator's HI is compromised, ALL HIP connections protected with that HI are compromised.

- The puzzle mechanism using CMAC may need further study that it does present the desired level of difficulty.

- The DEX HIT generation MAY present new attack opportunities; further study is needed.

The R1 packet is unprotected and offers an attacker new resource attacks against the Initiator. This is mitigated by only processing a received R1 when the Initiator has previously sent a corresponding I1.

9. IANA Considerations

HIP DEX introduces the following new HIP HIT suite:

Index	Hash function	Signature algorithm family	Description
5	FOLD	ECDH	ECDH HI folded to 96 bits

Table 2: HIT Suites

In addition, this document specified a new HIP Parameter Type defined in Section 5.2.5.

Moreover, a new HIP Cipher ID is defined in Section 5.2.4.

10. Acknowledgments

The drive to put HIP on a cryptographic 'Diet' came out of a number of discussions with sensor vendors at IEEE 802.15 meetings. David McGrew was very helpful in crafting this document.

11. Changelog

This section summarizes the changes made from draft-moskowitz-hip-rg-dex-05, which was the first stable version of the draft. Note that the draft was renamed after draft-moskowitz-hip-rg-dex-06.

11.1. Changes in draft-moskowitz-hip-rg-dex-06

- A major change in the ENCRYPT parameter to use AES-CTR rather than AES-CBC.

11.2. Changes in draft-moskowitz-hip-dex-00

- Draft name change. HIPRG ended in IRTF, HIP DEX is now individual submission.
- Added the change section.
- Added a Definitions section.
- Changed I2 and R2 packets to reflect use of AES-CTR for ENCRYPTED_KEY parameter.
- Cleaned up KEYMAT Generation text.
- Added Appendix with C code for the ECDH shared secret generation on an 8 bit processor.

11.3. Changes in draft-moskowitz-hip-dex-01

- Numerous editorial changes.
- New retransmission strategy.
- New HIT generation mechanism.
- Modified layout of ENCRYPTED_KEY parameter.

- Clarify to use puzzle difficulty of zero under normal network conditions.
- Align inclusion directive of R1_COUNTER with HIPv2 (from SHOULD to MUST).
- Align inclusion of TRANSPORT_FORMAT_LIST with HIPv2 (added to R1 and I2).
- HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST must now be echoed in R2 packet.
- Added new author.

12. References

12.1. Normative References

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

[RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, November 1998.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, January 2004.

[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.

[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.

[rfc4843-bis] Laganier, J. and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)", draft-ietf-hip-rfc4843-bis-01 (work in progress), March 2011.

[rfc5201-bis] Moskowitz, R., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", draft-ietf-hip-rfc5201-bis-09 (work in progress), July 2012.

[rfc5202-bis]
Jokela, P., Moskowitz, R., Nikander, P., and J. Melen, "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)", draft-ietf-hip-rfc5202-bis-00 (work in progress), September 2010.

12.2. Informative References

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

[IEEE.802-11.2007]
"Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11, June 2007, <http://standards.ieee.org/getieee802/download/802.11-2007.pdf>.

[IEEE.802-15-4.2011]
"Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)", IEEE Standard 802.15.4, September 2011, <http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>.

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

[hummen2013tailoring]
Hummen, R., Wirtz, H., Ziegeldorf, J., Hiller, J., and K. Wehrle, "Tailoring End-to-End IP Security Protocols to the Internet of Things", in Proceedings of IEEE International Conference on Network Protocols (ICNP 2013), October 2013.

[rfc4423-bis]
Moskowitz, R., "Host Identity Protocol Architecture", draft-ietf-hip-rfc4423-bis-04 (work in progress), July 2012.

Appendix A. C code for ECDH point multiplication on an 8 bit processor

The following code illustrates the process to generate the shared secret key from ECDH exchange.

```
// The following code is taken from the
//      Relic library (http://code.google.com/p/relic-toolkit/)
// This code includes the function for point multiplication over
//      prime fields


/**
 * The following function multiplies a prime elliptic point by an
 *     integer using the binary method.
 *
 * @param[out] r            - the result.
 * @param[in] p             - the point to multiply.
 * @param[in] k             - the integer.
 */

/**
 * ep_t is library specific representation for an elliptic curve
 *    point
 * bn_t is library specific representation for a multiple precision
 *    integer structure
 */

void ep_mul_basic(ep_t r, ep_t p, bn_t k) {
    int i, l;
    ep_t t;

    ep_null(t);

    if (bn_is_zero(k)) {
//If k is 0 then assign the prime elliptic curve point to a point
//  at the infinity
        ep_set_infty(r);
        return;
    }

    TRY {
//ep_new allocates memory for the point and may throw ERR_NO_MEMORY
//  incase of insufficient memory
        ep_new(t);
//bn_bits returns the number of bits of a multiple precision
//  integer     l = bn_bits(k);
//bn_test_bit returns 0 if the bit at the given location is 0 and
//  returns non zero otherwise
        if (bn_test_bit(k, l - 1)) {
```

```
            ep_copy(t, p);
        } else {
            ep_set_infty(t);
        }

        for (i = l - 2; i >= 0; i--) {
  //ep_dbl doubles a prime elliptic curve point
            ep_dbl(t, t);
            if (bn_test_bit(k, i)) {
  //ep_add adds two prime elliptic curve points
                ep_add(t, t, p);
            }
        }

        ep_copy(r, t);
  //ep_norm converts a point to affine coordinates
        ep_norm(r, r);
    }
    CATCH_ANY {
        THROW(ERR_CAUGHT);
    }
    FINALLY {
        ep_free(t);
    }
}
```

Appendix B. Password-based two-factor authentication during the HIP DEX handshake

HIP DEX allows to identify authorized connections based on a two-factor authentication mechanism. With two-factor authentication, devices that are authorized to communicate with each other are required to be pre-provisioned with a shared (group) key. The Initiator uses this pre-provisioned key to encrypt the ECHO_RESPONSE_UNSIGNED in the I2 packet. Upon reception of the I2, the Responder verifies that its challenge in the ECHO_REQUEST_UNSIGNED parameter in the R1 packet has been encrypted with the correct key. If verified successfully, the Responder proceeds with the handshake. Otherwise, it silently drops the I2 packet.

Note that there is no explicit signaling in the HIP DEX handshake for this behavior. Thus, knowledge of two-factor authentication must be configured externally prior to the handshake.

Authors' Addresses

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

EMail: robert.moskowitz@verizon.com

Rene Hummen
Chair of Communication and Distributed Systems, RWTH Aachen
Ahornstrasse 55
Aachen 52074
Germany

EMail: hummen@comsys.rwth-aachen.de
URI: http://www.comsys.rwth-aachen.de/team/rene-hummen/