Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 26, 2012

R. Moskowitz
Verizon

May 25, 2012

HIP Diet EXchange (DEX)
draft-moskowitz-hip-rg-dex-06

Abstract

This document specifies the details of the Host Identity Protocol Diet EXchange (HIP DEX). HIP DEX is a variant of the HIP Base EXchange (HIP BEX) [RFC5201-bis] specifically designed to use as few crypto primitives as possible yet still deliver the same class of security features as HIP BEX.

The design goal of HIP DEX is to be usable by sensor devices that are memory and processor constrained. Like HIP BEX it is expected to be used together with another suitable security protocol, such as the Encapsulated Security Payload (ESP). HIP DEX can also be used directly 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 November 26, 2012.

Table of Contents

1. Introduction

This memo specifies the details of the Host Identity Protocol Diet EXchange (HIP DEX). HIP DEX uses the smallest possible set of established cryptographic primitives, in such a manner that does not change our understanding of their behaviour, yet in a different formulation to achieve assertions normally met with different primitives.

HIP DEX builds on the HIP Base Exchange (HIP BEX) [RFC5201-bis], and only the differences between BEX and DEX are documented here.

There are a few key differences between BEX and DEX.

Minimum collection of cryptographic primitives.

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

Static Elliptic Curve Diffie-Hellman key pairs used to encrypt the session key.

A simple truncation function for HIT generation.

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

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

Provide a Password Authentication within the exchange. This may be supported by BEX as well, but not defined there.

Operate in an aggressive retransmission manner to deal with the high packet loss nature of sensor networks.

1.1. The HIP Diet EXchange (DEX)

The HIP diet exchange is a two-party cryptographic protocol used to establish communications context between hosts. The first party is called the Initiator and the second party the Responder. The four-packet design helps to make HIP DoS resilient. The protocol exchanges Static Diffie-Hellman keys in the 2nd and 3rd packets, transmits session secrets in the 3rd and 4th packets, and authenticates the parties also 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 stores any state from the exchange.

Thus DEX is operationally similar to BEX. The model is 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 of encrypting 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 BEX.

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

This is learned when the Responder starts receiving protected datagrams, indicating that the Initiator received the R2 packet. As such the Intitator 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 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) [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 replacement for IKE.

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 4 gives an overview of the HIP base exchange protocol. Section 6 define the 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 (M(x), K) denotes the lowest order K bits of the result of the mac function M on the input x.

3. The DEX Host Identifier Tag (HIT) and Its Representations

The DEX Host Identity Tag (HIT) is distinguished in two ways from the BEX HIT:

The HIT SUITE ID Section 5.1.2 is ONLY a DEX ID.

The HIT DEX HIT is not generated via a cryptographic hash. Rather it is a truncation of the Elliptic Curve Host Identity.

3.1. Host Identity Tag (HIT)

The DEX Host Identity Tag is a 128-bit value -- a truncation of the Host Identifier appended with a 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 is used.

BEX uses RFC 4843-bis [RFC4843-bis] specified 128-bit hash-based identifiers, called Overlay Routable Cryptographic Hash Identifiers (ORCHIDs). Their prefix, allocated from the IPv6 address block, is defined in [RFC4843-bis].

In DEX, a cryptographic hash is NOT used to form the HIT. Rather the

HI is truncated to 96 bits.

3.2. Generating a HIT from an HI

The DEX HIT is not an ORCHID, as there is no hash function in DEX. Since a HI that is an ECDH key is directly computed from a random number it is already collision resistant. The DEX HIT is the left-truncated 96 bits of the HI. This 96 bit value is used in place of the hash in the ORCHID. The HIT suite (see Section 9) is used for the four bits of the Orchid Generation Algorithm (OGA) field in the ORCHID. The same IPv6 prefix used in BEX is used for DEX.

4. Protocol Overview

The following material is an overview of the differences between the BEX and DEX implementations of the HIP protocol. It is expected that [RFC5201-bis] is well understood first.

4.1. Creating a HIP Association

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

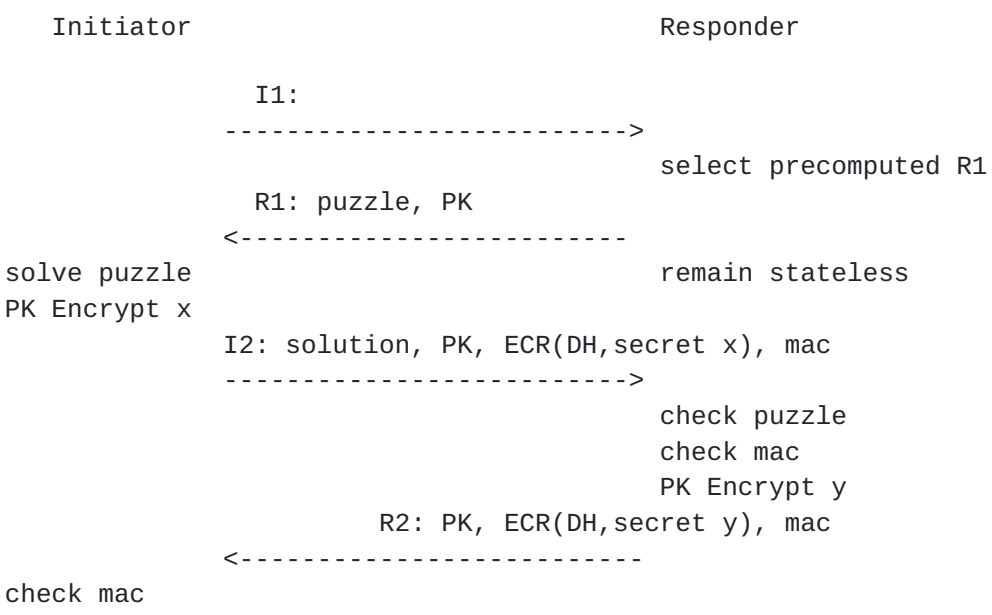
The HIP Diet EXchange serves to manage the establishment of state between an Initiator and a Responder. The first packet, I1, initiates the exchange, and the last three packets, R1, I2, and R2, constitute an authenticated secret key wrapped by a Diffie-Hellman derived key for session key generation. The HIP association keys are drawn from this keying material. If other cryptographic keys are needed, e.g., to be used with ESP, they are expected to be drawn from the same 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 base exchange 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 message is discarded. The I2 also contains a key wrap parameter that carries the key for the Responder. This key is only half the final session key. The packet is authenticated by the sender (Initiator).

The R2 packet finalizes the base exchange. The R2 contains a key wrap parameter that carries the rest of the key for the Initiator. The packet is authenticated by the sender (Initiator).

The base exchange is illustrated below. The term "key" refers to the Host Identity public key, "secret" refers to a random value encrypted by a public key, and "sig" represents a signature using such a 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 I2. 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 CPU cycles in solving the puzzle.

DEX uses the CMAC function instead of a hash function as in BEX.

The puzzle mechanism has been explicitly designed to give space for various implementation options. It allows a Responder implementation to completely delay session-specific state creation until a valid I2 is received. In such a case, a correctly formatted I2 can be rejected only once the Responder has checked its validity by computing one CMAC function. On the other hand, the design also allows a Responder implementation to keep state about received I1s, and match the received I2s against the state, thereby allowing the implementation to avoid the computational cost of the CMAC function.

The drawback of this latter approach is the requirement of creating state. Finally, it also allows an implementation to use other combinations of the space-saving and computation-saving mechanisms.

Generally speaking, the puzzle mechanism works in DEX the same as in BEX. There are some implementation differences, using CMAC rather than a hash.

See Appendix A for one possible implementation. Implementations SHOULD include sufficient randomness to the algorithm so that algorithmic complexity attacks become impossible [CRO03].

4.1.2. Puzzle Exchange

The Responder starts the puzzle exchange when it receives an I1. The Responder supplies a random number I, and requires the Initiator to find a number J. To select a proper J, the Initiator must create the concatenation of the HITs of the parties and J, and feed this concatenation using I as the key into the CMAC 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 CMAC target of zeros. The Initiator SHOULD give up after exceeding the puzzle lifetime in the PUZZLE parameter ([RFC5201-bis]). The Responder needs to re-create the concatenation of the HITs and the provided J, and compute the CMAC using I once to prove that the Initiator did 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 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 an ECHO_REQUEST_UNSIGNED parameter ([RFC5201-bis]), the Responder can include some data in R1 that the Initiator must copy unmodified in the corresponding I2 packet. The Responder can generate the Opaque data in various ways; e.g., using some secret, the sent I, and possibly other related data. Using the same secret, the received I (from the I2), and the other related data (if any), the Receiver can verify that it has itself sent the I to the Initiator. The Responder MUST periodically change such a used secret.

It is RECOMMENDED that the Responder generates a new puzzle and a new R1 once every few minutes. Furthermore, it is RECOMMENDED that the Responder remembers an old puzzle at least 2*Lifetime seconds after

the puzzle has been deprecated. These time values allow a slower Initiator to solve the puzzle while limiting the usability that an old, solved puzzle has to an attacker.

4.1.3. HIP State Machine

The HIP protocol itself has little state. In HIP DEX, as in BEX, 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 HIP DEX state machine has the same states as the BEX state machine. However, there is an optional aggressive transmission feature to provide better performance in sensor networks with high packet loss. The following section documents the few differences in the DEX state machine.

4.1.3.1. HIP Aggressive Transmission Mechanism

HIP DEX may be used on networks with high packet loss. DEX deals with this by using an aggressive transmission practice for I1 and I2 packets. The Initiator SHOULD continually send I1 and I2 packets at some short interval t msec, based on local policy. The transmission stops on receipt of the corresponding R1 or R2 packet, which acts as an acknowledgment receipt.

Since the Responder is stateless until it receives an I2, it does not need any special behaviour on sending R1 other than to send one whenever it receives an I1. The Responder sends an R2 after receipt every I2. The Responder does need to know that R2 was received by the Initiator. Like in BEX, the Responder can learn this when it starts receiving datagrams.

4.1.3.2. 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 exchange failed

Table 1: HIP States

4.1.3.3. HIP State Processes

System behavior in state I1-SENT, Table 2.

Trigger	Action
t msec	Send I1 and stay at I1-SENT

Table 2: I1-SENT - Initiating HIP

System behavior in state I2-SENT, Table 3.

Trigger	Action
t msec	Send I2 and stay at I2-SENT

Table 3: I2-SENT - Waiting to finish HIP

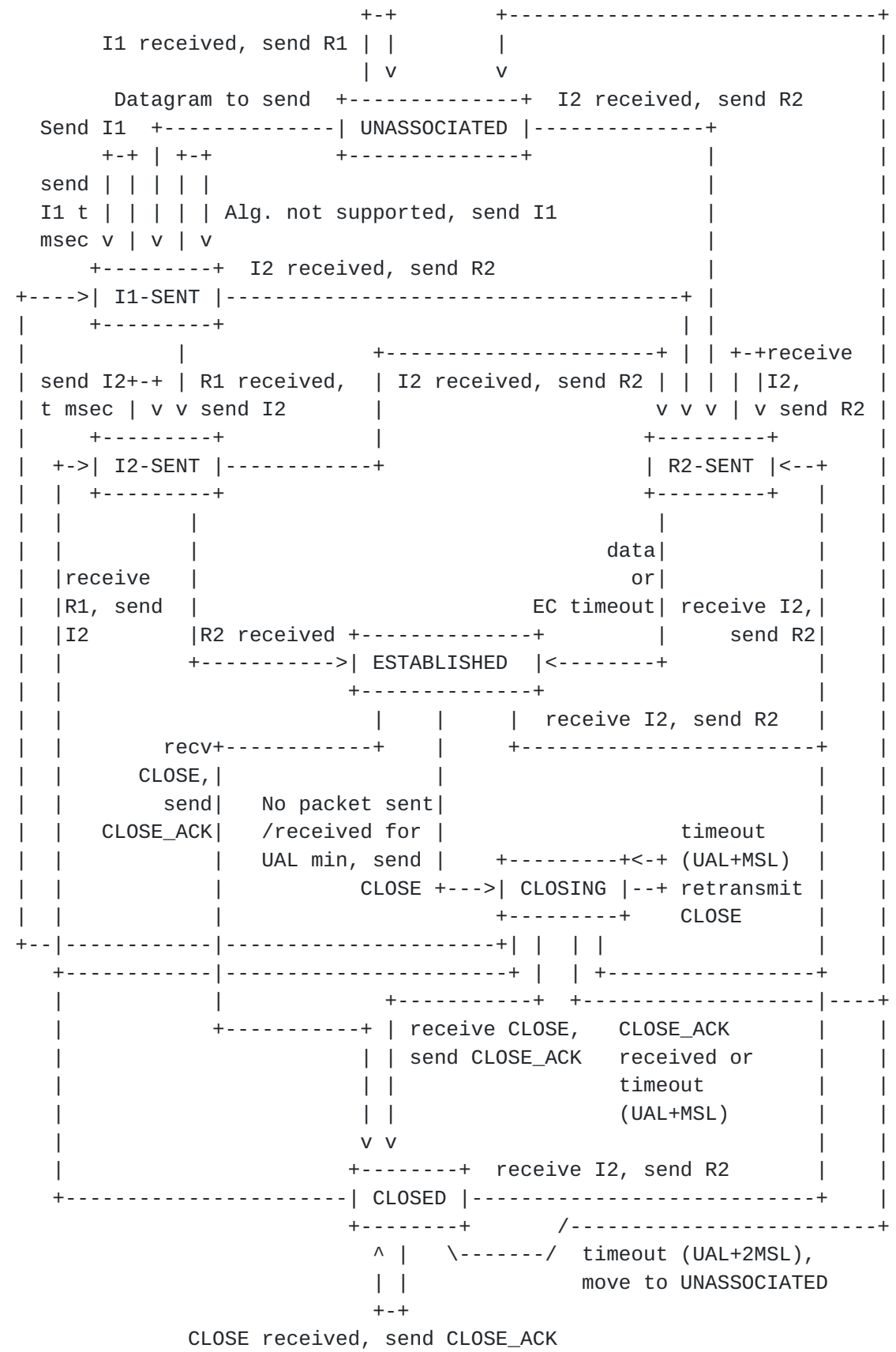
System behavior in state R2-SENT, Table 4.

Trigger	Action
Receive duplicate I2	Send R2 and stay at R2-SENT

Table 4: R2-SENT - Waiting to finish HIP

4.1.3.4. 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.


I1 received, send R1
Datagram to send
Send I1
send I1 t msec
UNASSOCIATED
I2 received, send R2
Alg. not supported, send I1
I1-SENT
send I2 t msec
R1 received, send I2
I2 received, send R2
I2-SENT
receive R1, send I2
R2 received
ESTABLISHED
R2-SENT
receive I2, send R2
data or EC timeout
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)
receive I2, send R2
CLOSED
timeout (UAL+2MSL), move to UNASSOCIATED
CLOSE received, send CLOSE_ACK

4.1.4. HIP DEX Security Associations

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

4.1.4.1. Master Key SA

The Master Key SA is used to secure DEX parameters and authenticate HIP packets. Since so little data will be protected by this SA it can be very longed lived.

The Master Key SA contains the following elements.

Source HIT

Destination HIT

HIP_Encrypt Key

HIP_MAC Key

Both keys are extracted from the Diffie-Hellman derived key via Section 6.3. Their length is determined by HIP_CIPHER.

4.1.4.2. Pair-wise Key SA

The Pair-wise Key SA is used to secure and authenticate user data. It is refreshed (or rekeyed) using the UPDATE packet exchange.

The Pair-wise Key SA elements are defined by the data transform (e.g. ESP_TRANSFORM [rfc5202-bis]).

The secrets in ENCRYPTED_KEY from I2 and R2 are concatenated to form the input to a Key Derivation Function (KDF). If the data transform does not have its own KDF, then Section 6.3 is used. Even though this input is randomly distributed, a KDF Extract phase may be needed to get the proper length for input to the KDF Expand phase.

4.1.5. User Data Considerations

There is only one difference in User Data Considerations between BEX and DEX. Loss of state due to system reboot may be a critical performance issue. Thus implementors MAY choose to use non-volatile, secure storage for HIP state so that it survives system reboot. This will limit state loss during reboots to only those situtations that there is an SA timeout.

5. Packet Formats

5.1. HIP Parameters

The HIP Parameters are used to carry the public key associated with the sender's HIT, together with related security and other information. They consist of parameters, ordered according to their numeric type number and encoded in TLV format.

The following new parameter types are currently defined for DEX, in addition to those defined for BEX. Also listed are BEX parameters that have additional values for DEX.

For the BEX parameters, DIFFIE_HELLMAN, DH_GROUP_LIST, and HOST_ID, only the ECC values are valid in DEX.

TLV	Type	Length	Data
ENCRYPTED_KEY	643	variable	Encrypted container for key generation exchange
HIP_MAC_3	61507	variable	CMAC-based message authentication code
HIP_CIPHER	579	variable	List of HIP encryption algorithms
HIT_SUITE_LIST	715	variable	Ordered list of the HIT suites supported by the Responder

5.1.1. HIP_CIPHER

The HIP ciphers in DEX are limited to:

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

The HIP_CIPHER parameter is limited to NULL or AES-CTR.

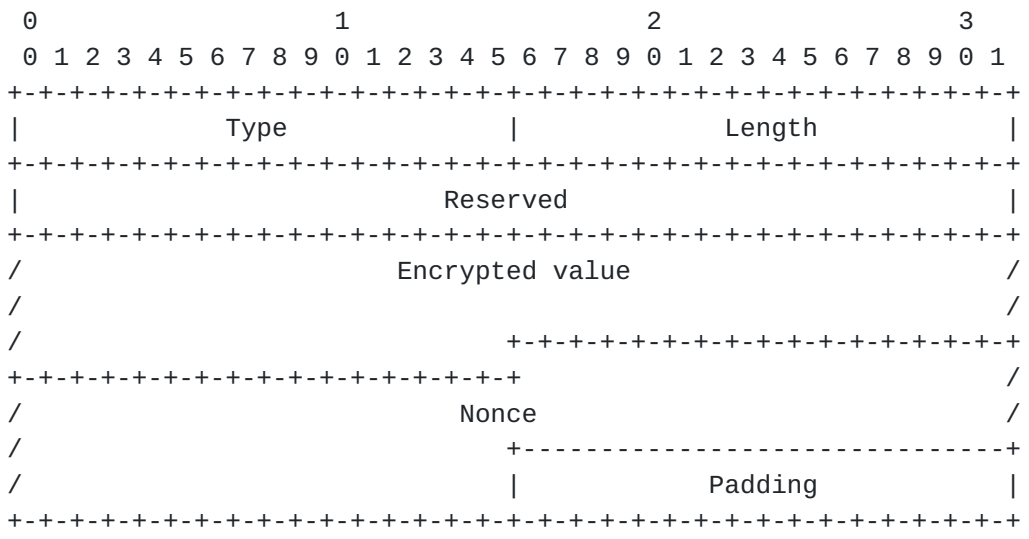
5.1.2. HIT_SUITE_LIST

The HIT suites in DEX are limited to:

HIT suite	ID
ECDH/DEX	8

The HIT_SUITE_LIST parameter contains a list of the supported HIT suite IDs of the Responder. Since the HIT of the Initiator is a DEX HIT, the Responder MUST only respond with a DEX HIT suite ID. Currently, only one such suite ID has been defined.

5.1.3. 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.
Nonce	Nonce included in encrypted text.

The ENCRYPTED parameter encapsulates a value and a nonce. The value is typically a random number used in a key creation process and the nonce is known to the receiver to validate successful decryption.

Some encryption algorithms require an IV (initialization vector). The IV MUST be known to the receiver through some source other than within the Encrypted_key block. For example the Puzzle value, I, can be used as an IV.

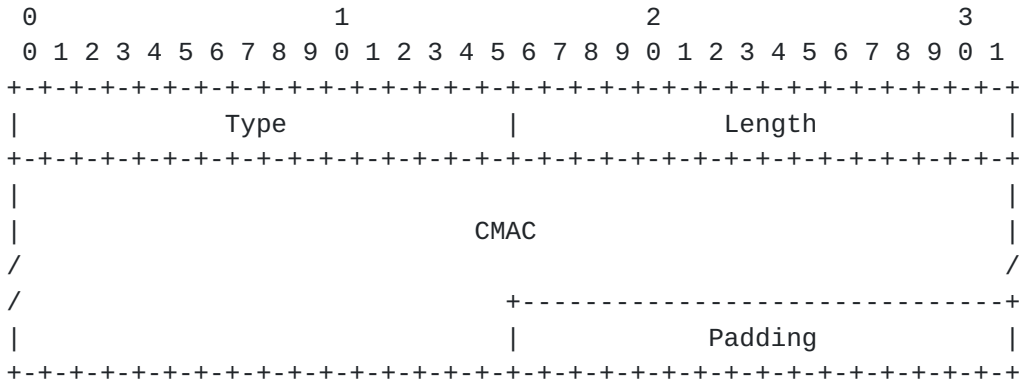
Text on CTR use here.

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 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 value and nonce 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_key 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].

5.1.4. HIP_MAC_3



Type	61507
Length	length in octets, excluding Type, Length, and Padding
CMAC	CMAC computed over the HIP packet, excluding the HIP_MAC parameter itself. 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 CMAC is calculated. The size of the CMAC is the natural size of the AES block depending on the AES key size.

The CMAC calculation and verification process is presented in Section 6.2.1.

5.2. HIP Packets

DEX uses the same eight basic HIP packets (see [RFC5201-bis]) as in BEX. Four are for the HIP exchange, one is for updating, one is for sending notifications, and two are for closing a HIP association. There are some differences in the HIP parameters in the exchange packets between BEX and DEX. This section will cover the DEX packets.

An important difference between BEX and DEX HIP packets is that there is no HIP_SIGNATURE parameter available in DEX. Thus R1 is completely unprotected and can be spoofed. The I2, R2, UPDATE, NOTIFY, CLOSE, and CLOSE_ACK parameters only have a HIP_MAC_3 parameter for packet authentication. The processing of these packets are 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. The HIP packet, however, MUST NOT be fragmented. This limits the size of the possible

additional data in the packet.

5.2.1. 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 = 40 bytes
```

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

Valid control bits: none

The Initiator HIT MUST be a DEX HIT. The HIT Suite ID MUST be of a DEX type. Currently only ECDH/DEX is defined.

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].

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

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. Note that the DH_GROUP_LIST in the I1 packet is not protected by a MAC.

Implementations MUST be able to handle a storm of received I1 packets, discarding those with common content that arrive within a small time delta, but distinguishing this from arriving at a set time delta. This behaviour is the expected behaviour for an Initiator on a network with high packet loss. The HIP state machine calls out this behaviour in this case and the Initiator will stop sending I1 packets after it receives an R1 packet.

5.2.2. 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,
             HIP_CIPHER,
             HOST_ID,
             HIT_SUITE_LIST,
             DH_GROUP_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 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" [RFC5201-bis].

The R1 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 CMAC result, that MUST be zeros; see Section 4.1.2.

The Initiator HIT does not provide the HOST_ID key size. The Responder selects its HOST_ID based on 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.

In DEX, the Diffie-Hellman HOST_ID values are static. They are NOT discarded.

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-CBC [RFC3602].

The ECHO_REQUEST_UNSIGNED contains data that the sender wants to receive unmodified in the corresponding response packet in the ECHO_RESPONSE_UNSIGNED parameter.

5.2.3. 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,
             HOST_ID,
             ENCRYPTED_KEY {DH, secret-x|I},
             [ ENCRYPTED {DH, ENCRYPTED_KEY {passwd, challenge } },]
             HIP_MAC_3,
             [<, ECHO_RESPONSE_UNSIGNED>i )] )

  Minimum size = 180 bytes
```

Valid control bits: A

The HITs used MUST match the ones used previously.

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

The Initiator MAY 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 CMAC(S, | ... | J) MUST be zero.

In DEX, the Diffie-Hellman HOST_ID values are static. They are NOT discarded.

The HIP_CIPHER contains the single encryption transform selected by the Initiator, that will be used to protect the HI exchange. The

chosen transform MUST correspond to one offered by the Responder in the R1. All implementations MUST support the AES-CBC transform [RFC3602].

The ECHO_RESPONSE_UNSIGNED contain the unmodified Opaque data copied from the corresponding echo request parameter.

The ENCRYPTED_KEY contains an Initiator generated random secret x that MUST be uniformly distributed that is concatenated with I from the puzzle. The secret x's length matches the keysize of the selected encryption transform. I from the puzzle is used as the IV in the encryption transform. This acts as a nonce from the Responder to prove freshness of the secret wrapping from the Initiator. I in the ENCRYPTED block enables the Responder to validate a proper decryption of the block. The key for the encryption is the HIP_Encrypt key.

If the Initiator has prior knowledge that the Responder is expecting a password authenication, the Initiator encrypts the ECHO_REQUEST_UNSIGNED with the password, then wraps the ENCRYPTED parameter in the secret x. I from the puzzle is used as the nonce here as well. There is no signal within R1 for this behaviour. Knowledge of password authencation must be externally configured.

The MAC is calculated over the whole HIP envelope, excluding any parameters after the HIP_MAC_3, as described in Section 6.2.1. The Responder MUST validate the HIP_MAC_3.

5.2.4. 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,
             ENCRYPTED_KEY {DH, secret-y|I},
             HIP_MAC_3)

  Minimum size = 108 bytes
```

Valid control bits: none

The Responder repeats the DH_GROUP_LIST parameter in R2. This MUST be the same list as included in R1. The DH_GROUP_LIST parameter is repeated here because R2 is MACed and thus cannot be altered by an

attacker. This allows the Initiator to determine whether its own DH_GROUP_LIST in the I1 was manipulated by an attacker.

The ENCRYPTED contains an Responder generated random secret y that MUST be uniformly distributed that is concatenated with I from the puzzle. The secret y's length matches the keysize of the selected encryption transform. I from the puzzle is used as the IV in the encryption transform. This acts as a nonce from the Initiator to prove freshness of the secret wrapping from the Responder. I in the ENCRYPTED block enables the Responder to validate a proper decryption of the block. The key for the encryption is the HIP_Encrypt key.

The HIP_MAC_3 is calculated over the whole HIP envelope, with Responder's HOST_ID parameter concatenated with the HIP envelope. The HOST_ID parameter is removed after the CMAC calculation. The procedure is described in Section 6.2.1.

The Initiator MUST validate the HIP_MAC_3.

5.3. 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 [RFC2463]. In most cases, the ICMP packet will have 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.

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 4.1.3. 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.

6.1. Solving the Puzzle

This subsection describes the puzzle-solving details.

In R1, the values I and K are sent in network byte order. Similarly, in I2, the values I and J are sent in network byte order. The mac is created by concatenating, in network byte order, the following data, in the following order and using the CMAC algorithm with I as the key:

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

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

n-bit random value J (where n is CMAC-len), in network byte order, as appearing in I2.

In order to be a valid response puzzle, the K low-order bits of the resulting CMAC MUST be zero.

Notes:

i) All the data in the CMAC input MUST be in network byte order.

ii) The order of the Initiator's and Responder's HITs are different in the R1 and I2 packets; see [RFC5201-bis]. Care must be taken to copy the values in the right order to the CMAC input.

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 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(CMAC(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(CMAC(I, HIT-I | HIT-R | J), K)
Rejects if V != 0
Accept if V == 0

6.2. HIP_MAC Calculation and Verification

The following subsections define the actions for processing the HIP_MAC_3 parameter.

6.2.1. CMAC Calculation

Both the Initiator and the Responder should take some care when verifying or calculating the HIP_MAC_3. Specifically, the Responder should preserve other parameters than the HOST_ID when sending the R2. Also, the Initiator has to preserve the HOST_ID exactly as it was received in the R1 packet.

The scope of the calculation for HIP_MAC_3 is:

CMAC: { HIP header | [Parameters] }

where Parameters include all HIP parameters of the packet that is being calculated with Type values 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 HIP_MAC parameter is defined in Section 5.1.4. 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 6.3.

4. Add the HIP_MAC_3 parameter to the packet and any parameter with greater Type value than the HIP_MAC's (HIP_MAC_3'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_3 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 6.3 and verify it against the received CMAC.

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

6.3. HIP DEX KEYMAT Generation

The HIP DEX KEYMAT process is used for both the Diffie-Hellman Derived Master key and the Encrypted secrets Pair-wise key. The former uses both the Extract and Expand phases, while the later MAY need the Extract and Expand phases if the key is longer than 128 bits. Othewise it only needs the Expand phase.

The Diffie-Hellman Derived Master key is exchanged in R1 and I2 and used in I2, R2. UPDATE, NOTIFY, and ACK packets. The Encrypted secrets Pair-wise key is not used in HIP, but is available as the datagram protection key. Some datagram protection mechanisms have their own Key Derivation Function, and if so that SHOULD be used rather than the HIP DEX KEYMAT.

The KEYMAT has 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 all 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 CKDF-Extract function is following operation; the | operation denotes concatenation.

```
CKDF-Extract(DHK, info, L) -> CK
```

where

```
info    =  sort(HIT-I | HIT-R) | "CKDF-Extract"
BigK    =  Diffie-Hellman Derived or Session (x | y) Key
I       =  I from PUZZLE Parameter
```

The output CK is calculated as follows:

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

The CKDF-Expand function is following operation; the | operation denotes concatenation.

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

where

```
info    =  sort(HIT-I | HIT-R) | "CKDF-Expand"
CK      =  CK from CKDF-Extract or (x | y)
PRKlen  =  Length of PRK in octets
maclen  =  Length of CMAC in octets = 128/8 = 16
L        length of output keying material in octets
         (<= 255*macLen)

If PRKlen != macLen then PRK = CMAC(0^128, PRK)
```

The output OKM is calculated as follows:

```
N = ceil(L/macLen)
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(CK, T(0) | info | 0x01)
T(2) = CMAC(CK, T(1) | info | 0x02)
T(3) = CMAC(CK, 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.

x and y values are from the ENCRYPTED parameters from I2 and R2 respectively.

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:

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

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

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

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.

6.4. 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. An I1 in DEX is handled identically to BEX with the exception that in constructing the R1, the Responder SHOULD select a HIT that is constructed with the MUST algorithm, which is currently ECDH.

6.4.1. R1 Management

All compliant implementations MUST produce R1 packets. An R1 in DEX is handled identically to BEX.

6.5. Processing Incoming R1 Packets

A system receiving an R1 MUST first check to see if it has sent an I1 to the originator of the R1 (i.e., it is in state I1-SENT). An R1 in DEX is handled identically to BEX with the following differences.

If the system has been sending out a stream of I1 packets to work around high packet loss on a network, it stops sending the I1 packets AFTER successfully processing a R1 packet.

There is no HIP_SIGNATURE in the R1 packet. It is an unauthentication packet.

The following steps define the conceptual processing rules for responding to an R1 packet that are different than in BEX:

1. If the system is configured with an authentication password for the responder, it constructs the authentication response to include in the I2.

2. The system prepares and sends an I2, as described in Section 5.2.3. The system MAY be configured to continually send this I2 until it receives and validates an R2.

6.6. Processing Incoming I2 Packets

Upon receipt of an I2, the system MAY perform initial checks to determine whether the I2 corresponds to a recent R1 that has been sent out, if the Responder keeps such state. An I2 in DEX is handled identically to BEX with the following differences.

The HIP implementation SHOULD process the I2. This includes validation of the puzzle solution, extracting the ENCRYPTED key for processing I2, decrypting the Initiator's Host Identity, verifying the mac, creating state, and finally sending an R2.

There is no HIP_SIGNATURE on this packet. Authentication is completely based on the HIP_MAC_3 parameter.

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

1. 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 6.3). If the local HIT is smaller than the sender's HIT, it should drop the I2 packet, and continue using the R1 received and I2 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 keying material it might have formed upon sending the I2 previously. The peer Diffie-Hellman key, ENCRYPTED keying material 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 earlier sent in the R1 packet.

2. The system MUST validate the solution to the puzzle by computing the mac described in Section 5.2.3 using the CMAC algorithm.

3. The system must extract the keying material from the ENCRYPTED parameter. This key is used to derive the HIP data keys.

4. 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. If the system has been sending a stream of R1 packets to the HIT in the I2 the system stops sending the R1s.

6.7. Processing Incoming R2 Packets

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

There is no HIP_SIGNATURE on this packet. Authentication is completely based on the HIP_MAC_3 parameter.

The conceptual processing rules for an incoming R2 packet in DEX are identical to BEX with the following differences.

1. The system checks the DH_GROUP_LIST as in R1 packet processing. If the list is different from R1's there may have been a DH downgrade attack against the unprotected R1 packet. If the DH_GROUP_LIST presents a better list than recieved in the R1 packet, the system may either resend I1 within the retry bounds or abandon the HIP exchange.

2. The system must extract the keying material from the ENCRYPTED parameter. This key is concatanated with that sent in the I2 packet to form the HIP data keys.

6.8. Sending UPDATE Packets

A host sends an UPDATE packet when it updates some information related to a HIP association. DEX UPDATE handling is the similar in DEX as in BEX. The key difference is the HIP_SIGNATURE is not present.

6.9. Handling State Loss

In the case of 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 on stable 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 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 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.

Many Initiators would want to use a different HI for different Responders. The implementations SHOULD provide for an ACL of Initiator's HIT to Responder's HIT. This ACL SHOULD also include preferred transform and local lifetimes.

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

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

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 so doing, HIP itself is subject to its own DoS and MitM attacks that potentially could be more damaging to a host's ability to conduct business as

usual.

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

The strength of the key is based on the quality of the secrets generated 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 extraction 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 the Initator only processing a received R1 when it has sent an I1. This is another DoS attack, but for battery powered Initiators, it could be a concern.

9. IANA Considerations

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

The following HIT suites are defined for DEX HIT generation.

Index	Hash function	Signature algorithm family	Description
5	LTRUNC	ECDH	ECDH HI truncated to 96 bits

Table 5: HIT Suites

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

11. References

11.1. Normative References

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2463] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 2463, December 1998.

[RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, September 2003.

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

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

[RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005.

[RFC4843-bis] Laganier, J. and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)", draft-ietf-hip-rfc4843-bis-00 (work in progress), August 2010.

[RFC5201-bis] Moskowitz, R., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", draft-ietf-hip-rfc5201-bis-08 (work

in progress), March 2012.

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

[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.

11.2. 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.

[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>.

[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>.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

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

[rfc4423-bis] Moskowitz, R., "Host Identity Protocol Architecture", draft-ietf-hip-rfc4423-bis-02 (work in progress), February 2011.

Appendix A. Using Responder Puzzles

As mentioned in Section 4.1.1, the Responder may delay state creation and still reject most spoofed I2s by using a number of pre-calculated R1s and a local selection function. This appendix defines one possible implementation in detail. The purpose of this appendix is to give the 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 and the Responder generates an R1 packet.

When the Initiator sends the I1 packet for initializing a connection, the Responder gets the HIT and IP address from the packet, and generates an I value for the puzzle.

```
I value calculation:
I = Ltrunc( CMAC ( S, HIT-I | HIT-R | IP-I | IP-R ), n)
where n = CMAC-len
```

From an incoming I2 packet, the Responder gets 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( CMAC( I2.I | 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 ECDH.

Author's Address

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

EMail: robert.moskowitz@verizon.com