HIP Diet EXchange (DEX)

Abstract

This document specifies the Host Identity Protocol Diet EXchange
(HIP DEX), a variant of the Host Identity Protocol Version 2 (HIPv2)
and specifically developed for use on low end processors. The HIP
DEX protocol design aims at reducing the overhead of the employed
cryptographic primitives by omitting public-key signatures and
cryptographic hash functions.

The HIP DEX protocol is primarily designed for computation or
memory-constrained sensor/actuator devices. Like HIPv2, it is
expected to be used together with a suitable security protocol such
as the Encapsulated Security Payload (ESP) for the protection of
upper layer protocol data. Unlike HIPv2, HIP DEX does not support
Forward Secrecy (FS), and MUST only be used on devices where FS is
prohibitively expensive. In addition, HIP DEX can also be used as a
keying mechanism for security primitives at the MAC layer, e.g., for
IEEE 802.15.4 networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies the Host Identity Protocol Diet EXchange (HIP DEX), specifically developed for use on low end processors that cannot support the cryptographic requirements of HIP BEX. HIP DEX is derived from the Base EXchange (BEX) of the Host Identity Protocol Version 2 (HIPv2) [RFC7401]. HIP DEX preserves the protocol semantics as well as the general packet structure of HIPv2. Hence, it is recommended that [RFC7401] is well-understood before reading this document.

The main differences between HIP BEX and HIP DEX are:

1. HIP DEX uses a different set of cryptographic primitives compared to HIP BEX with the goal to reduce the protocol overhead:

   *Peer authentication and key agreement in HIP DEX are based on static Elliptic Curve Diffie-Hellman (ECDH) key pairs. This replaces the use of public-key signatures and ephemeral Diffie-Hellman key pairs in HIPv2.

   *HIP DEX uses AES-CTR for symmetric-key encryption of HIP payloads and AES-CMAC as its MACing function. In contrast, HIPv2 currently supports AES-CBC for encryption and HMAC-SHA-1, HMAC-SHA-256, or HMAC-SHA-384 for MACing.

   *HIP DEX defines a simple fold function to efficiently generate HITs, whereas the HIT generation of HIPv2 is based on SHA-1, SHA-256, or SHA-384.

2. HIP DEX forfeits the HIPv2 Forward Secrecy property due to the removal of the ephemeral Diffie-Hellman key agreement. As this weakens the security properties of HIP DEX, it MUST be used only with constrained devices where this is prohibitively expensive as further explained in [Section 1.2](#).

3. HIP DEX forfeits the use of digital signatures with the removal of a hash function. Peer authentication with HIP DEX, therefore, is based on the use of the ECDH derived key in the CMAC-based HIP_MAC parameter.

4. The forfeiture of the use of digital signatures leaves the R1 packet open to a MITM attack. Such an attack is managed in the R2 packet validation and is yet another DOS attack mitigated through the HIP state machine. This state machine mitigation is augmented by HIT,HI ACL controls.

5. The forfeiture of a cryptographic hash leaves the HIT generated by a fold function vulnerable to pre-image attacks. This MUST be mitigated through a HIT,HI pairing as in an ACL control mechanism Section 7.1.

6. With HIP DEX, the ECDH derived key is only used to protect HIP packets. Separate session key(s) are used to protect the transmission of upper layer protocol data. These session key(s) are established via a new secret exchange during the handshake.

7. HIP DEX introduces a new, optional retransmission strategy that is specifically designed to handle potentially extensive processing times of the employed cryptographic operations on computationally constrained devices.

By eliminating the need for public-key signatures and the ephemeral DH key agreement, HIP DEX reduces the computational, energy, transmission, and memory requirements for public-key cryptography (see [LN08]) in the HIPv2 protocol design. This makes HIP DEX especially suitable for constrained devices as defined in [RFC7228].

Cryptographic hashing was eliminated due to the memory/code space or gate cost for a hash. This is based on actual implementation efforts on 8-bit cpu sensors with 16KB memory and 64KB flash for code.

This document focuses on the protocol specifications related to differences between HIP BEX and HIP DEX. Where differences are not called out explicitly, the protocol specification of HIP DEX is the same as defined in [RFC7401].

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 DEX Denial of Service (DoS) resilient. The Initiator and the Responder exchange their static Elliptic Curve Diffie-Hellman (ECDH) keys in the R1 and I2 handshake packet. The parties then authenticate each other in the I2 and R2 handshake packets based on the ECDH-derived keying material. The Initiator and the Responder additionally transmit keying material for the session key in these last two handshake packets (I2 and R2). This is to prevent overuse of the static ECDH-derived keying material. Moreover, the Responder starts a puzzle exchange in

the R1 packet and the Initiator completes this exchange in the I2
packet before the Responder performs computationally expensive
operations or stores any state from the exchange. Given this
handshake structure, HIP DEX operationally is very similar to HIP
BEX. Moreover, the employed model is also fairly equivalent to
802.11-2007 [IEEE.802-11.2016] 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 I2 packet. The Responder's Host Identity also is
not protected. Thus, contrary to HIPv2, HIP DEX does not provide for
end-point anonymity and any signaling (i.e., HOST_ID parameter
contained with an ENCRYPTED parameter) that indicates such anonymity
should be ignored.

As in [RFC7401], data packets start to flow after the R2 packet. The
I2 and R2 packets may carry a data payload in the future. The
details of this may be defined later.

An existing HIP association can be updated with the update mechanism
defined in [RFC7401]. Likewise, the association can be torn down
with the defined closing mechanism for HIPv2 if it is no longer
needed. Standard HIPv2 uses a HIP_SIGNATURE to authenticate the
association close operation, but since DEX does not provide for
signatures, the usual per-message MAC suffices.

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) [RFC7402] as well as with other
end-to-end security protocols. In addition, HIP DEX can also be used
as a keying mechanism for security primitives at the MAC layer,
e.g., for IEEE 802.15.4 networks [IEEE.802-15-4.2015]. It is worth
mentioning that the HIP DEX base protocol does not cover all the
fine-grained policy control found in Internet Key Exchange Version 2
(IKEv2) [RFC7296] that allows IKEv2 to support complex gateway
policies. Thus, HIP DEX is not a replacement for IKEv2.

1.2. Applicability

HIP DEX achieves its lightweight nature in large part due to the
intentional removal of Forward Secrecy (FS) from the key exchange.
Current mechanisms to achieve FS use an authenticated ephemeral
Diffie-Hellman exchange (e.g., SIGMA or PAKE). HIP DEX targets usage
on devices where even the most lightweight ECDH exchange is
prohibitively expensive for recurring (ephemeral) use. For example,
experience with the 8-bit 8051-based ZWAVE ZW0500 microprocessor has
shown that EC25519 keypair generation exceeds 10 seconds and
consumes significant energy (i.e., battery resources). Even the ECDH
multiplication for the HIP DEX static-static key exchange takes 8-9

seconds, again with measurable energy consumption. The ECDH multiplication resource consumption via a static EC25519 keypair is tolerable as a one-time event during provisioning, but would render the protocol unsuitable for use on these devices if it was required to be a recurring part of the protocol. Further, for devices constrained in this manner, a FS-enabled protocol's cost will likely provide little gain. Since the resulting "FS" key, likely produced during device deployment, would typically end up being used for the remainder of the device's lifetime. Since this key (or the information needed to regenerate it) persists for the device's lifetime, the key step of 'throw away old keys' in achieving forward secrecy does not occur, thus the forward secrecy would not be obtained in practice.

With such a usage pattern, the inherent benefit of ephemeral keys is not realized. The security properties of such usage are very similar to those of using a statically provisioned symmetric pre-shared key, in that there remains a single PSK in static storage that is susceptible to exfiltration/compromise, and compromise of that key in effect compromises the entire protocol for that node. HIP DEX achieves marginally better security properties by computing the effective long-term PSK from a DH exchange, so that the provisioning service is not required to be part of the risk surface due to also possessing the PSK.

If the device is not able to generate the ECDH keypair, the provisioning service can generate and install the ECDH keypair provided it wipes knowledge of the private key. Typically, the provisioning service will make the public key (HI) and PSK available for the deployment step.

Due to the substantially reduced security guarantees of HIP DEX compared to HIP BEX, HIP DEX MUST only be used when at least one of the two endpoints is a class 0 or 1 constrained device defined in Section 3 of [RFC7228]). HIP DEX MUST NOT be used when both endpoints are class 2 devices or unconstrained.

It is inevitable that both HIP BEX and DEX will be available on some systems, most noticeably sensor gateways. HIP DEX MUST NOT be used between systems capable of HIP BEX. This may be controlled by limiting the use of DEX to an "internal" interface, or for such systems to first offer a BEX HIT in an I1 and only if that fails to try a DEX HIT. Note that such a downgrade (from BEX to DEX) offer approach is open to attack, requiring additional mitigation (e.g. ACL controls).

1.3. Memo Structure

The rest of this memo is structured as follows. Section 2 defines
the central keywords, notation, and terms used throughout this
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, 9,
and 10 discuss policy, interoperability between HIPv2 vs DEX,
security, and IANA considerations, respectively. Appendix A defines
a two factor authentication scheme and Appendix B highlights some
discussions with the IESG.

2. Terms, Notation and Definitions

2.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in
BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all
capitals, as shown here.

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 and Y.

FOLD (X, K) denotes the partitioning of X into n K-bit segments and
   the iterative folding of these segments via XOR. I.e., X = x_1,
   x_2, ..., x_n, where x_i is of length K and the last segment x_n

is padded to length K by appending 0 bits. FOLD then is computed
as FOLD(X, K) = t_n, where t_i = t_i-1 XOR x_i and t_1 = x_1.

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

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.

2.3. Definitions

CKDF: CMAC-based Key Derivation Function.

CMAC: The Cipher-based Message Authentication Code with the 128-bit
Advanced Encryption Standard (AES) defined in [NIST.SP.800-38B].

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

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

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

HI (Host Identity): 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 (see Section 3) in the appropriate I2 or R2
packet.

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

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

KEYMAT: Keying material. That is, the bit string(s) used as
cryptographic keys.

RHASH_len: The natural output length of the RHASH Algorithm in
bits.

Nonce #I: Nonce #I refers to the corresponding field in the PUZZLE
parameter (see section 5.2.4 in [RFC7401]. It is also referred to
as "random value #I" in this document.

OGA (Orchid Generation Algorithm):
Hash function used in generating
   the ORCHID.

ORCHID (Overlay Routable Cryptographic Hash Identifiers): IPv6
   addresses intended to be used as endpoint identifiers at
   applications and Application Programming Interfaces (APIs) and
   not as identifiers for network location at the IP layer.

Puzzle difficulty K: The Initiator has to compute a solution for
   the puzzle. The level of computational difficulty is denoted by
   the #K field in the puzzle parameter (see section 5.2.4 in
   [RFC7401].

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

RHASH (Responder's HIT Hash Algorithm): In HIP DEX, RHASH is
   redefined as CMAC. Still, note that CMAC is a message
   authentication code (MAC) and not a cryptographic hash function.
   Thus, a mapping from CMAC(x,y) to RHASH(z) must be defined where
   RHASH is used. Moreover, RHASH has different security properties
   in HIP DEX and is not used for HIT generation.

Security Association (SA): An SA is a simplex "connection" that
   affords security services to the traffic carried by it. HIP DEX
   has two forms of SAs, a Master Key SA for the actual HIP traffic,
   and a Pair-wise Key SA for use by a data transport service.

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

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

A compressed encoding of the HI, the Host Identity Tag (HIT), is
used in the handshake packets to represent the HI. 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 5.2.4).

*The DEX HIT is not generated via a cryptographic hash. Rather, it
   is a compression of the HI.

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 use an unverified HIT as input. The full HI
of a host SHOULD be considered, and the HIT MAY be used as a hint
for locating the full HI (see Section 7.1).

Carrying HIs or HITs in the header of user data packets would
increase the overhead of packets. Thus, it is not expected that
these parameters are carried in every packet, but other methods are
used to map the data packets to the corresponding HIs. In some
cases, this allows use of HIP DEX 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 DEX association. When other user data packet formats are used,
the corresponding extensions need to define a replacement for the
ESP_TRANSFORM [RFC7402] parameter along with associated semantics,
but this procedure is outside the scope of this document.

3.1. Host Identity Tag (HIT)

With HIP DEX, the HIT is a 128-bit value - a compression of the HI
prepended with a specific prefix. There are two advantages of using
this compressed encoding over the actual variable-sized public key
in protocols. First, the fixed length of the HIT keeps packet sizes
manageable and eases protocol coding. Second, it presents a
consistent format for the protocol, independent of the underlying
identity technology in use.

The structure of the HIT is based on RFC 7343 [RFC7343], 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 the HI. Third, the 96-bit
compressed representation of the HI. In contrast to HIPv2, HIP DEX
employs HITs that are NOT generated by means of a cryptographic
hash. Instead, the HI is compressed to 96 bits as defined in the
following section.

3.2. Generating a HIT from an HI

The HIT does not follow the exact semantics of an ORCHID as there is
no hash function in HIP DEX. Still, its structure is strongly
aligned with the ORCHID design. The same IPv6 prefix used in HIPv2
is used for HIP DEX. The HIP DEX HIT suite (see Section 10) 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
FOLD(HI,96).

3.2.1. Why Introduce FOLD

HIP DEX by design lacks a cryptographic hash function. The
generation of the HIT is one of the few places in the protocol where
this presents a challenge. CMAC was first considered for this
purpose, but to use CMAC for HIT generation would require using a
static key, either ZERO or some published value. NIST does not
consider CMAC an approved cryptographic hash as:

> It is straightforward to demonstrate that CMAC is not collision-
> resistant for any choice of a published key.

Since collision-resistance is not possible with the tools at hand,
any reasonable function (e.g. FOLD) that takes the full value of the
HI into generating the HIT can be used, provided that collision
detection is part of the HIP-DEX deployment design. This is achieved
here through either an ACL or some other lookup process that
externally binds the HIT and HI.

HIT collisions have always been a statistical possibility in BEX and
thus the HI has always been a part of the R1 and I2 packets for HI
validation.

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 HIPv2 and HIP
DEX protocol specifications.

4.1. Creating a HIP Association

By definition, the system initiating a HIP Diet EXchange is the
Initiator, and the peer is the Responder. This distinction is
typically 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 exchange, and the last three packets, R1, I2, and R2,
constitute an authenticated Diffie-Hellman [DH76] key exchange for
the Master Key Security Association (SA) generation. This Master Key
SA is used by the Initiator and the Responder to wrap secret keying

material in the I2 and R2 packets. Based on the exchanged keying
material, the peers then derive a Pair-wise Key SA if cryptographic
keys are needed, e.g., for ESP-based protection of user data.

The Initiator first sends a trigger packet, I1, to the Responder.
This packet contains the HIT of the Initiator and the HIT of the
Responder, if it is known. Moreover, the I1 packet initializes the
negotiation of the Diffie-Hellman group that is used for generating
the Master Key SA by including a list of Diffie-Hellman Group IDs
supported by the Initiator.

The second packet, R1, starts the actual authenticated Diffie-
Hellman key 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 knowledge of the Initiator, current load, or other
factors. In addition, the R1 contains the Responder's Diffie-Hellman
parameter and lists of cryptographic algorithms supported by the
Responder. Based on these lists, the Initiator can continue, abort,
or restart the handshake with a different selection of cryptographic
algorithms.

Unlike in HIP BEX, the R1 packet in DEX is not signed. Thus the
Initiator MUST compare the content of R1 with that it later gets in
R2 to ensure there was no MITM attack on R1.

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 nonce and key wrap parameter that
carries secret keying material of the Initiator. This keying
material is only half of the final session (pair-wise) key. The
packet is authenticated by the sender (Initiator) via a MAC.

The R2 packet acknowledges the receipt of the I2 packet and
completes the handshake. The R2 echos the nonce from I2 and contains
a key wrap parameter that carries the rest of the keying material of
the Responder. The packet is authenticated by the sender (Responder)
via a MAC. The R2 repeats the lists from R1 for signed validation to
defend them against a MITM attack.

The HIP DEX handshake is illustrated below. The terms "ENC(DH,x)"
and "ENC(DH,y)" refer to the random values x and y that are wrapped
based on the Master Key SA (indicated by ENC and DH). Note that x
and y each constitute half of the final session key material. The
packets also contain other parameters that are not shown in this
figure.

```
   Initiator                                          Responder

            I1: DH List
          --------------------------------------->

                                                 remain stateless

            R1: puzzle, (DH, Suite, Trans) Lists,
                HI
            <-------------------------------------

solve puzzle
perform ECDH
encrypt x

            I2: solution, HI, ENC(DH,x), Trans List,
                I_Nonce, mac
          --------------------------------------->

                                                 check puzzle
                                                 perform ECDH
                                                 check MAC
                                                 decrypt x
                                                 encrypt y

            R2: (DH, Suite, Trans) Lists, ENC(DH,y),
                I_Nonce, mac
            <-------------------------------------

check MAC
validate lists in R1
decrypt y
```
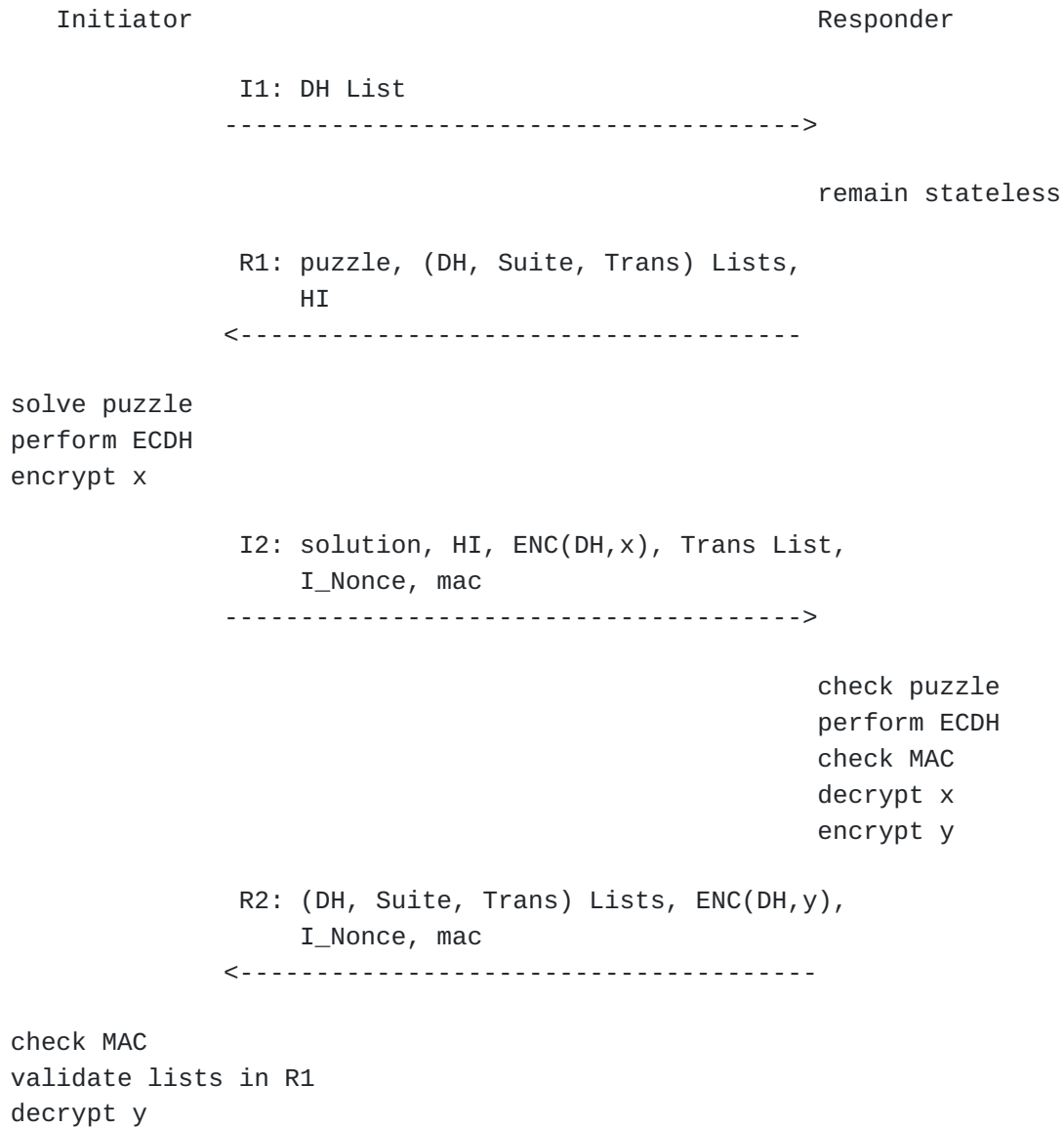
Figure 1: Figure 1: High-level overview of the HIP Diet EXchange

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

operation. 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 network conditions, the puzzle difficulty SHOULD be zero, thus effectively reverting the puzzle mechanism to a cookie-based DoS protection mechanism. Without setting the puzzle difficulty to zero under normal network conditions, potentially scarce computation resources at the Initiator would be churned unnecessarily.

Conceptually, the puzzle mechanism in HIP DEX is the same as in HIPv2. Hence, this document refers to Sections 4.1.1 and 4.1.2 in [RFC7401] for more detailed information about the employed mechanism. Notably, the only differences between the puzzle mechanism in HIP DEX and HIPv2 are that HIP DEX does not employ pre-computation of R1 packets and uses CMAC instead of a hash function for solving and verifying a puzzle. The implications of these changes on the puzzle implementation are discussed in Section 6.1.

4.1.2. HIP State Machine

The HIP DEX state machine has the same states as the HIPv2 state machine (see Section 4.4. in [RFC7401]); this is for easier comparison between the two Exchanges. However, HIP DEX features a retransmission strategy with an optional reception acknowledgement for the I2 packet. The goal of this additional acknowledgement is to reduce premature I2 retransmissions in case of devices with low computation resources [HWZ13]. As a result, there are minor changes regarding the transitions in the HIP DEX state machine. The following section documents these differences compared to HIPv2.

4.1.2.1. HIP DEX Retransmission Mechanism

For the retransmission of I1 and I2 packets, the Initiator adopts the retransmission strategy of HIPv2 (see Section 4.4.3. in [RFC7401]). This strategy is based on a timeout that is set to a value larger than the worst-case anticipated round-trip time (RTT). For each received I1 or I2 packet, the Responder sends an R1 or R2 packet, respectively. This design trait to always send an R1 after an I1 enables the Responder to remain stateless until the reception and successful processing of the I2 packet. 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 packet after I1_RETRIES_MAX tries, it stops I1 retransmissions. Likewise, it stops retransmitting the I2 packet after I2_RETRIES_MAX unsuccessful tries.

For repeatedly received I2 packets, the Responder SHOULD NOT perform operations related to the Diffie-Hellman key exchange or the keying material wrapped in the ENCRYPTED_KEY parameters. Instead, it SHOULD re-use the previously established state to re-create the corresponding R2 packet in order to prevent unnecessary computation overhead.

The potentially high processing time of an I2 packet at a (resource-constrained) Responder may cause premature 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 the I2 packet at the Responder into account for retransmission purposes. To this end, the Responder MAY notify the Initiator about the anticipated delay once the puzzle solution was successfully verified that the remaining I2 packet processing will incur a high processing delay. The Responder MAY therefore send a NOTIFY packet (see Section 5.3.6. in [RFC7401]) to the Initiator before the Responder commences the ECDH operation. The NOTIFY packet serves as an acknowledgement for the I2 packet and consists of a NOTIFICATION parameter with Notify Message Type I2_ACKNOWLEDGEMENT (see Section 5.2.19. in [RFC7401]). The NOTIFICATION parameter contains the anticipated remaining processing time for the I2 packet in milliseconds as two-octet Notification Data. This processing time can, e.g., be estimated by measuring the computation time of the ECDH key derivation operation during the Responder start-up procedure. After the I2 processing has finished, the Responder sends the regular R2 packet.

When the Initiator receives the NOTIFY packet, it sets the I2 retransmission timeout to the I2 processing time indicated in the NOTIFICATION parameter plus half the RTT-based timeout value. In doing so, the Initiator MUST NOT set the retransmission timeout to a higher value than allowed by a local policy. This is to prevent unauthenticated NOTIFY packets from maliciously delaying the handshake beyond a well-defined upper bound in case of a lost R2 packet. At the same time, this extended retransmission timeout enables the Initiator to defer I2 retransmissions until the point in time when the Responder should have completed its I2 packet processing and the network should have delivered the R2 packet according to the employed worst-case estimates.

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 to value in I2_ACKNOWLEDGEMENT Notification Data plus 1/2 RTT-based timeout value and stay at I2-SENT
Timeout	Increment trial counter
	If counter is less than I2_RETRIES_MAX, send I2, reset timer to RTT-based timeout, 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.

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

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 the HIP DEX packet exchanges. Since only a small amount of data is protected by this SA, it can be long-lived with no need for rekeying. At the latest, the system MUST initiate rekeying when its incoming ESP sequence counter is going to overflow, and the system MUST NOT replace its keying material until the rekeying packet exchange successfully completes as described in Section 6.8 in [RFC7402].

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 6.3. 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 [RFC7402]).

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, the key derivation function defined in Section 6.3 is used. Even though the concatenated 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 [RFC7401] also apply
to HIP DEX. There is only one difference between HIPv2 and HIP DEX.
Loss of state due to system reboot may be a critical performance
issue for resource-constrained devices. Thus, implementors MAY
choose to use non-volatile, secure storage for HIP states in order
for them to survive a system reboot as discussed in Section 6.11.
Using non-volatile storage 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
HIPv2. As such, the specifications in Section 5.1 of [RFC7401] 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 HIPv2 defined in Section 5.2.
of [RFC7401] where possible. Still, HIP DEX further restricts and/or
extends the following existing parameter types:

  *DH_GROUP_LIST and HOST_ID are restricted to ECC-based suites.

  *HIP_CIPHER is restricted to AES-128-CTR.

  *HIT_SUITE_LIST is limited to the HIT suite ECDH/FOLD.

  *PUZZLE, SOLUTION, and HIP_MAC parameter processing is altered to
   support CMAC in RHASH and RHASH_len (see Section 6.1 and Section
   6.2).

In addition, HIP DEX introduces the following new parameters:

TLV	Type	Length	Data
ENCRYPTED_KEY	TBD1 (suggested value 643)	variable	Encrypted container for the session key exchange
I_NONCE	TBD6 (suggested value 644)	variable	Nonce from Initator for Master Key

Table 2

5.2.1. DH_GROUP_LIST

The DH_GROUP_LIST parameter contains the list of supported DH Group
IDs of a host. It is defined in Section 5.2.6 of [RFC7401]. With HIP
DEX, the DH Group IDs are restricted to:

```
Group                      KDF          Value

Curve25519 [RFC7748]       CKDF         TBD7 (suggested value 12)
Curve448   [RFC7748]       CKDF         TBD8 (suggested value 13)
```

The ECDH groups with values TBD7 and TBD8 are defined in [RFC7748].
These curves have cofactors of 8 and 4 (respectively).

It is not known if Curve448 Diffie Hellman can meet the performance
requirements on 8-bit cpus. It is included for "completeness". An
implementor should ensure they can get the needed performance for
their target platform before committing to support this group.

5.2.2. HIP_CIPHER

The HIP_CIPHER parameter contains the list of supported cipher
algorithms to be used for encrypting the contents of the ENCRYPTED
and ENCRYPTED_KEY parameters. The HIP_CIPHER parameter is defined in
Section 5.2.8 of [RFC7401]. With HIP DEX, the Suite IDs are limited
to:

```
Suite ID          Value

RESERVED          0
AES-128-CTR       TBD4 (suggested: 5)     ([RFC3686])
```

Mandatory implementation: AES-128-CTR.

The counter for AES-128-CTR MUST have a length of 128 bits. The
puzzle value #I and the puzzle solution #J (see Section 4.1.2 in
[RFC7401]) are used to construct the initialization vector (IV) as
FOLD(I | J, 112) which are the high-order bits of the CTR counter. A
16 bit value as a block counter, which is initialized to zero on

first use, is appended to the IV in order to guarantee that a non-repeating nonce is fed to the AES-CTR encryption algorithm.

This counter is incremented as it is used for all encrypted HIP parameters. That is a single AES-129-CTR counter associated with the Master Key SA.

5.2.3. HOST_ID

The HOST_ID parameter conveys the Host Identity (HI) along with optional information about a host. The HOST_ID parameter is defined in Section 5.2.9 of [RFC7401].

HIP DEX uses the public portion of a host's static ECDH key-pair as the HI. Correspondingly, HIP DEX limits the HI algorithms to the following new profile:

Algorithm profiles   Value

ECDH   TBD5 (suggested: 11)   [RFC6090]   (REQUIRED)

For hosts that implement ECDH as the algorithm, the following curves are required:

Group   Value

Curve25519   5 [RFC7748]
Curve448   6 [RFC7748]

HIP DEX HIs are serialized equally to the ECC-based HIs in HIPv2 (see Section 5.2.9. of [RFC7401]). The Group ID of the HIP DEX HI is encoded in the "ECC curve" field of the HOST_ID parameter. The supported DH Group IDs are defined in Section 5.2.1.
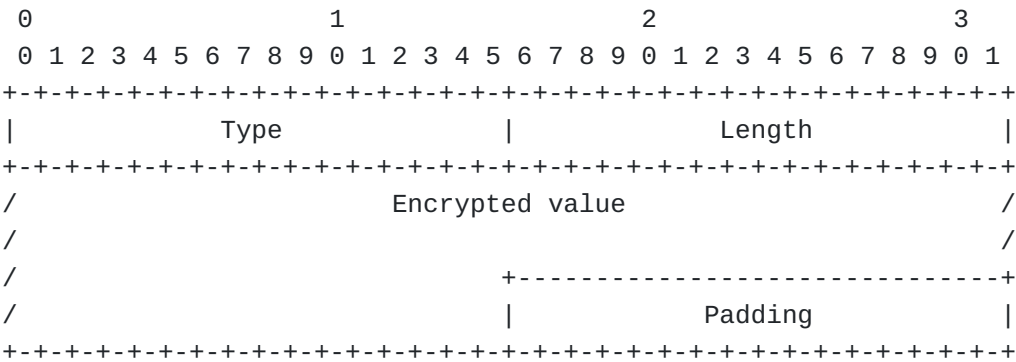
5.2.4. HIT_SUITE_LIST

The HIT_SUITE_LIST parameter contains a list of the supported HIT suite IDs of the Responder. Based on the HIT_SUITE_LIST, the Initiator can determine which source HIT Suite IDs are supported by the Responder. The HIT_SUITE_LIST parameter is defined in Section 5.2.10 of [RFC7401].

The following new HIT Suite ID is defined for HIP DEX, and the relationship between the four-bit ID value used in the OGA ID field and the eight-bit encoding within the HIT_SUITE_LIST ID field is clarified:

```
HIT Suite        Four-bit ID              Eight-bit encoding

ECDH/FOLD        TBD2 (suggestion: 4)    TBD3 (suggestion: 0x40)
```

Note that the dedicated HIP DEX HIT Suite ID in the OGA ID field
allows the peers to distinguish a HIP DEX handshake from a HIPv2
handshake. The Responder MUST respond with a HIP DEX HIT suite ID
when the HIT of the Initiator is a HIP DEX HIT.

5.2.5. ENCRYPTED_KEY

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Type             |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       Encrypted value                         /
/                                                               /
/                               +-------------------------------+
/                               |            Padding            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
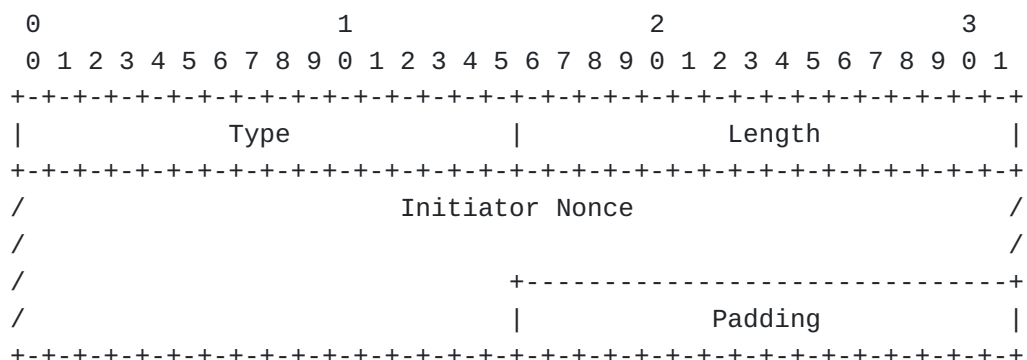
```
Type            TBD1 (suggested value 643)
Length          length in octets, excluding Type, Length, and
                Padding
Encrypted       The value is encrypted using an encryption algorithm
value           as defined in the HIP_CIPHER parameter.
```

The ENCRYPTED_KEY parameter encapsulates a random value that is
later used in the session key creation process (see Section 6.3).
This random value MUST have a length of at least 64 bits. The
HIP_CIPHER is used for the encryption.

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 [RFC7401].

5.2.6. I_NONCE

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       Initiator Nonce                         /
/                                                               /
/                               +-------------------------------+
/                               |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type   TBD6 (suggested value 644)
Length   length in octets, excluding Type, Length, and
                Padding
Initiator   Nonce provided by the Initiator for use in the
Nonce   Master Key

The I_NONCE parameter encapsulates a random value that is later used
in the Master key creation process (see Section 6.3). This random
value MUST have a length of 2 x RHASH_len. This parameter is sent to
the Responder in I2 which echos it back to the Initiator in R2.

If necessary, additional Padding for 8-byte alignment is added
according to the rules of TLV Format in [RFC7401].

5.3. HIP Packets

HIP DEX uses the same eight basic HIP packets as HIPv2 (see Section
5.3 of [RFC7401]). Four of them are for the HIP handshake (I1, R1,
I2, and R2), one is for updating an association (UPDATE), one is for
sending notifications (NOTIFY), and two are for closing the
association (CLOSE and CLOSE_ACK). There are some differences
regarding the HIP parameters that are included in the handshake
packets concerning HIP BEX and HIP DEX. This section covers these
differences for the DEX packets. Packets not discussed here, follow
the structure defined in [RFC7401].

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 HIP 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 ) )
```

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 HIP DEX
type as defined in Section 5.2.4.

Regarding the Responder's HIT, the Initiator may receive this HIT
either from a DNS lookup of the Responder's FQDN (see [RFC8005]),
from some other repository, or from a local table. The Responder's
HIT also MUST be of a HIP DEX type. 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 Section 4.1.8 of
[RFC7401] for detailed information about the "HIP Opportunistic
Mode".

As the Initiator's and the Responder's HITs are compressions of the
employed HIs, they determine the DH Group ID that must be used in
order to successfully conclude the triggered handshake. HITs,
however, only include the OGA ID identifying the HI algorithm. They
do not include information about the specific group ID of the HI. To
inform the Responder about its employed and its otherwise supported
DH Group IDs, the Initiator therefore includes the DH_GROUP_LIST
parameter in the I1 packet. This parameter MUST include the DH group
ID that corresponds to the currently employed Initiator HIT as the
first list element. With HIP DEX, the DH_GROUP_LIST parameter MUST
only include ECDH groups defined in Section 5.2.1.

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 ])
```

Valid control bits: none

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 Section 4.1.8 of [RFC7401] for
detailed information about the "HIP Opportunistic Mode".

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 [RFC7401]). Responders
SHOULD set K to zero by default and only increase the puzzle
difficulty to protect against a DoS attack targeting the HIP DEX
handshake. A puzzle difficulty of zero effectively turns the puzzle
mechanism into a return-routability test and is strongly encouraged
during normal operation in order to conserve energy resources as
well as to prevent unnecessary handshake delay in case of a
resource-constrained Initiator. Please also refer to Section 7 for
further recommendations on choosing puzzle difficulty.

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 DH_GROUP_LIST parameter contains the Responder's order of preference based on the Responder's choice the ECDH key contained in the HOST_ID parameter (see below). This allows the Initiator to begin to determine whether its own DH_GROUP_LIST in the I1 packet was manipulated by an attacker. There is a further risk that the Responder's DH_GROUP_LIST was manipulated by an attacker, as the R1 packet cannot be authenticated in HIP DEX. Thus, this parameter is repeated in the R2 packet to allow for a final, cryptographically secured validation.

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 with 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 HOST_ID parameter depends on the received DH_GROUP_LIST parameter and the Responder HIT in the I1 packet. Specifically, if the I1 contains a Responder HIT, the Responder verifies that this HIT matches the preferred DH group based on the received DH_GROUP_LIST parameter included in the I1. In case of a positive result, the Responder selects the corresponding HOST_ID for inclusion in the R1 packet. Likewise, if the Responder HIT in the I1 packet is NULL (i.e., during an opportunistic handshake), the Responder chooses its HOST_ID according to the Initiator's employed DH group as indicated in the received DH_GROUP_LIST parameter and sets the source HIT in the R1 packet accordingly. If the Responder however does not support the DH group required by the Initiator or if the Responder HIT in the I1 packet does not match the required DH group, the Responder selects the mutually preferred and supported DH group based on the DH_GROUP_LIST parameter in the I1 packet. The Responder then includes the corresponding ECDH key in the HOST_ID parameter. This parameter also indicates the selected DH group. Moreover, the Responder sets the source HIT in the R1 packet based on the destination HIT from the I1 packet. Based on the deviating DH group ID in the HOST_ID parameter, the Initiator then MUST abort the current handshake and SHOULD initiate a new handshake with the mutually supported DH group as far as local policies (see Section 7) permit.

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 different format types are DEFAULT, ESP (Mandatory to Implement) and ESP-TCP (Experimental, as explained in Section 3.1 in [RFC6261]).

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,
           I_NONCE,
           HIP_MAC
           [<, ECHO_RESPONSE_UNSIGNED>i )] )
```

Valid control bits: none

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

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

The Solution contains the Random #I from the R1 packet and the computed #J value. 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 parameter 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 parameter(s) contain 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 A.

The TRANSPORT_FORMAT_LIST parameter 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 packet. The different format types are DEFAULT, ESP and ESP-TCP as explained in Section 3.1 in [RFC6261].

The I_NONCE parameter contains the nonce, supplied by the Initiator for the Master Key generation as shown in Section 6.3. This is echoed back to the Initiator in the R2 packet.

The MAC is calculated over the whole HIP envelope, excluding any parameters after the HIP_MAC parameter as described in Section 6.2. The Responder MUST validate the HIP_MAC parameter.

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,
           I_NONCE,
           HIP_MAC )
```

Valid control bits: none

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

The Responder repeats the DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and TRANSPORT_FORMAT_LIST parameters in the R2 packet. These parameters MUST be the same as included in the R1 packet. The parameter are re-included here because the R2 packet 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 parameter 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 I_NONCE parameter contains the nonce, supplied by the Initiator for the Master Key generation as shown in Section 6.3. The Responder is echoing the value back to the Initiator to show it used the Initiator provided nonce.

The MAC is calculated over the whole HIP envelope, excluding any parameters after the HIP_MAC, as described in Section 6.2. The Initiator MUST validate the HIP_MAC parameter.

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) and Code of 0. The Pointer field pointing to the field that caused the ICMP message to be generated, for example to the first 8 bytes of a UDP payload for "SPI is Unknown". The problem cases specified in Section 5.4. of [RFC7401] also apply to HIP DEX.

6. Packet Processing

Due to the adopted protocol semantics and the inherited general packet structure, the packet processing in HIP DEX only differs from HIPv2 in very few places. Here, we focus on these differences and refer to Section 6 in [RFC7401] otherwise.

The processing of outgoing and incoming application data remains the same as in HIP BEX (see Sections 6.1 and 6.2 in [RFC7401]).

6.1. Solving the Puzzle

The procedures for solving and verifying a puzzle in HIP DEX are strongly based on the corresponding procedures in HIPv2. The only

exceptions are that HIP DEX does not use pre-computation of R1
packets and that RHASH is set to CMAC. As a result, the pre-
computation step in Section 6.3 of [RFC7401] is skipped in HIP DEX.

Moreover, the Initiator solves a puzzle by computing:
   Ltrunc(CMAC(I, HIT-I | HIT-R | J), K) == 0

Similarly, the Responder verifies a puzzle by computing:
   V := Ltrunc(CMAC(I, HIT-I | HIT-R | J), K)

Apart from these modifications, the procedures defined in Section
6.3 of [RFC7401] also apply for HIP DEX.

6.2. HIP_MAC Calculation and Verification

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

6.2.1. 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.

The parameter order is described in Section 5.2.1 of [RFC7401].

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](#). HIP-gl refers to host with greater HIT value and HIP-lg refers to the host with smaller HIT value.

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 6.3](#) and verify it against the received CMAC.

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

6.3. HIP DEX KEYMAT Generation

The HIP DEX KEYMAT process is used to derive the keys for the Master Key SA as well as for the Pair-wise Key SA. The keys for the Master Key SA are based on the Diffie-Hellman derived key, Kij, which is produced during the HIP DEX handshake. The Initiator generates Kij during the creation of the I2 packet and the Responder generates Kij once it receives the I2 packet. This is why the I2 packet can already contain authenticated and/or encrypted information.

The keys derived for the Pair-wise Key SA are not used during the HIP DEX handshake. Instead, these keys are made available as payload protection keys (e.g., for IPsec's ESP).

The HIP DEX KEYMAT process is similar to the Hash-based Key Derivation Function (HKDF) defined in [[RFC5869](#)], but uses CMAC in place of a cryptographic hash. DEX KEYMAT follows the CMAC usage guidance for a KDF construct provided in [[NIST.SP.800-56C](#)], [[NIST.SP.800-108](#)] and [[KeyDerivation](#)]. This CMAC Key Derivation Function (CKDF) consists of two components, CKDF-Extract and CKDF-Expand. The CKDF-Extract function compresses a non-uniformly distributed key, such as the output of a Diffie-Hellman key

derivation, to extract the key entropy into a fixed length output.
The CKDF-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 the Master Key SA employs always both the
Extract and Expand phases. The Pair-wise Key SA needs only the
Extract phase when the key is smaller or equal to 128 bits, but
otherwise requires also the Expand phase.

The CKDF-Extract function is the following operation:

CKDF-Extract(I, IKM, info) -> PRK

Inputs:
  I   Random #I, provided by the Responder, from the PUZZLE
            parameter
  Kij   Diffie-Hellman derived key
  IKM   IKMm for Master Key SA Input keying material
            or
            IKMp for Pair-wise Key SA Input keying material

  IKMm   Kij | I_NONCE
  IKMp   Kij | I_NONCE | (concatenated random values of the
                 ENCRYPTED_KEY parameters in the same order as
                 the HITs with sort(HIT-I | HIT-R))

  info   sort(HIT-I | HIT-R) | "CKDF-Extract"
             Where the input text: "CKDF-Extract"
             Is the hex string: 0x434b44462d45787472616374

 Output:
  PRK   a pseudorandom key (of RHASH_len/8 octets)

The pseudorandom key PRK is calculated as follows:

   PRK   = CMAC(I, IKM | info)

  The CKDF-Expand function is the following operation:

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


   Inputs:
     PRK        a pseudorandom key of at least RHASH_len/8 octets
                (either the output from the extract step or the
                concatenation of the random values of the
                ENCRYPTED_KEY parameters in the same order as the
                HITs with sort(HIT-I | HIT-R) in case of no extract)
     info       sort(HIT-I | HIT-R) | "CKDF-Expand"
                Where the input text: "CKDF-Expand"
                Is the hex string: 0x434b44462d457870616e64
     L          length of output keying material in octets
                (<= 255*RHASH_len/8)

   Output:
     OKM         output keying material (of L octets)

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 for the Master Key SA are drawn sequentially in the
   order that is determined by the numeric comparison of the two HITs,
   with the 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 regarding the algorithm defined in the HIP_CIPHER. For the mandatory algorithms, the following size applies:

AES 128 bits

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

6.4. Initiation of a HIP Diet EXchange

The initiation of a HIP DEX handshake proceeds as described in Section 6.6 of [RFC7401]. The I1 packet contents are specified in Section 5.3.1.

6.5. Processing Incoming I1 Packets

I1 packets in HIP DEX are handled almost identical to HIPv2 (see Section 6.7 of [RFC7401]). The main differences are that the Responder SHOULD select a HIP DEX HIT Suite 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 at the Responder. Hence, the R1 pre-computation SHOULD be omitted in HIP DEX.

Correspondingly, the modified conceptual processing rules for responding to an I1 packet are as follows:

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

2. If the Responder is in ESTABLISHED state, the Responder MAY respond to this with an R1 packet, prepare to drop an existing HIP security association with the peer, and stay at ESTABLISHED state.

3. If the Responder is in I1-SENT state, it MUST make a comparison between the sender's HIT and its own (i.e., the receiver's)

HIT. If the sender's HIT is greater than its own HIT, it should
drop the I1 packet and stay at I1-SENT. If the sender's HIT is
smaller than its own HIT, it SHOULD send the R1 packet and stay
at I1-SENT. The HIT comparison is performed as defined in
Section 6.3.

4. If the implementation chooses to respond to the I1 packet with
   an R1 packet, it creates a new R1 according to the format
   described in Section 5.3.2. It chooses the HI based on the
   destination HIT and the DH_GROUP_LIST in the I1 packet. If the
   implementation does not support the DH group required by the
   Initiator or if the destination HIT in the I1 packet does not
   match the required DH group, it selects the mutually preferred
   and supported DH group based on the DH_GROUP_LIST parameter in
   the I1 packet. The implementation includes the corresponding
   ECDH public key in the HOST_ID parameter. If no suitable DH
   Group ID was contained in the DH_GROUP_LIST in the I1 packet,
   it sends an R1 packet with any suitable ECDH public key.

5. If the received Responder's HIT in the I1 packet is not NULL,
   the Responder's HIT in the R1 packet MUST match the destination
   HIT in the I1 packet. Otherwise, the Responder MUST select a
   HIT with the same HIT Suite as the Initiator's HIT. If this HIT
   Suite is not supported by the Responder, it SHOULD select a
   REQUIRED HIT Suite from Section 5.2.10 of [RFC7401], which is
   currently RSA/DSA/SHA-256. Other than that, selecting the HIT
   is a local policy matter.

6. The Responder expresses its supported HIP transport formats in
   the TRANSPORT_FORMAT_LIST as described in Section 5.2.11 of
   [RFC7401]. The Responder MUST provide at least one payload
   transport format type.

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

Note that only steps 4 and 5 have been changed with regard to the
processing rules of HIPv2. The considerations about R1 management
(except pre-computation) and malformed I1 packets in Sections 6.7.1
and 6.7.2 of [RFC7401] likewise apply to HIP DEX.

6.6. Processing Incoming R1 Packets

R1 packets in HIP DEX are handled identically to HIPv2 (see Section
6.8 in [RFC7401]) with the following exceptions: HIP DEX uses ECDH
public keys as HIs and does not employ signatures.

As R1 is not signed and no proof is possible in the authenticity of
its contents, all processing of the R1 is provisional until verified
by the R2 processing.

The modified conceptual processing rules for responding to an R1 packet are as follows:

1. A system receiving an R1 MUST first check to see if it has sent an I1 packet to the originator of the R1 packet (i.e., it has a HIP association that is in state I1-SENT and that is associated with the HITs in the R1). Unless the I1 packet was sent in opportunistic mode (see Section 4.1.8 of [RFC7401]), the IP addresses in the received R1 packet SHOULD be ignored by the R1 processing and, when looking up the correct HIP association, the received R1 packet SHOULD be matched against the associations using only the HITs. If a match exists, the system processes the R1 packet as described below.

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

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

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

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

6. The system MUST check that the DH Group ID in the HOST_ID parameter in the R1 matches the first DH Group ID in the Responder's DH_GROUP_LIST in the R1 packet, and also that this Group ID corresponds to a value that was included in the Initiator's DH_GROUP_LIST in the I1 packet. If the DH Group ID of the HOST_ID parameter does not express the Responder's best

choice, the Initiator can conclude that the DH_GROUP_LIST in the I1 or R1 packet was adversely modified. In such a case, the Initiator MAY send a new I1 packet; however, it SHOULD NOT change its preference in the DH_GROUP_LIST in the new I1 packet. Alternatively, the Initiator MAY abort the HIP DEX handshake. Moreover, if the DH Group ID indicated in the HOST_ID parameter does not match the DH Group ID of the HI employed by the Initiator, the system SHOULD wait for an acceptable time span to allow further R1 packets with different DH Group IDs to arrive before restarting or aborting the HIP DEX handshake. When restarting the handshake, the Initiator MUST consult local policies (see Section 7) regarding the use of another, mutually supported DH group for the subsequent handshake with the Responder.

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

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

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

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

11. The system computes standard Diffie-Hellman keying material according to the public value and Group ID provided in the HOST_ID parameter. The Diffie-Hellman keying material Kij is used for key extraction as specified in Section 6.3.

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

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

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

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

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

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

Note that step 4 from the original processing rules of HIPv2 (signature verification) has been removed in the above processing rules for HIP DEX. Moreover, step 7 of the original processing rules has been adapted in step 6 above to account for the fact that HIP DEX uses ECDH public keys as HIs. The considerations about malformed R1 packets in Sections 6.8.1 of [RFC7401] also apply to HIP DEX.

6.7. Processing Incoming I2 Packets

The processing of I2 packets follows similar rules as HIPv2 (see Section 6.9 of [RFC7401]). The main differences to HIPv2 are that HIP DEX introduces a new session key exchange via the ENCRYPTED_KEY parameter as well as an I2 reception acknowledgement for retransmission purposes. Moreover, with HIP DEX the Initiator is responsible for triggering retransmissions, whereas the Responder merely replies to received I2 packets.

The modified HIP DEX conceptual processing rules for responding to an I2 packet are:

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 [RFC7401] 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. The system MUST validate the Initiator's HI per Section 9.2.

5. If the system's state machine is in the R2-SENT state, the system MUST check to see 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 reset the R2-SENT timer. The system SHOULD re-use the previously established state to re-create the corresponding R2 packet in order to prevent unnecessary computation overhead.

6. 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 6.3). 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 processes the received I2 packet and drops 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.

7. 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).

8. If the system's state machine is in any state other 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.

9. The system MUST validate the solution to the puzzle as described in Section 6.1.

10. 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.

11. The system MUST derive Diffie-Hellman keying material Kij based on the public value and Group ID in the HOST_ID parameter. This keying material is used to derive the keys of the Master Key SA as described in Section 6.3. If the Diffie-Hellman Group ID is unsupported, the I2 packet is silently dropped. If the processing time for the derivation of the Diffie-Hellman keying material Kij is likely to cause premature I2 retransmissions by the Initiator, the system MAY send a NOTIFY packet before starting the key derivation process. The NOTIFY packet contains a NOTIFICATION parameter with Notify Message Type I2_ACKNOWLEDGEMENT. The NOTIFICATION parameter indicates the anticipated remaining processing time for the I2 packet in milliseconds as two-octet Notification Data.

12. The implementation SHOULD also verify that the Initiator's HIT in the I2 packet corresponds to the Host Identity sent in the I2 packet. (Note: some middleboxes may not be able to make this verification.)

13. The system MUST process the TRANSPORT_FORMAT_LIST parameter. Other documents specifying transport formats (e.g., [RFC7402]) contain specifications for handling any specific transport selected.

14. The system MUST verify the HIP_MAC according to the procedures in Section 6.2.

15. 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.

16. The system MUST decrypt 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 6.3).

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

18. Upon successful processing of an I2 packet when the system's state machine is in state UNASSOCIATED, I1-SENT, I2-SENT, or R2-SENT, an R2 packet is sent as described in Section 5.3.4 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 as described in Section 5.3.4, 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 a maximal amount of retransmissions of I2
    packets), the state machine transitions to ESTABLISHED.

Note that steps 11 (encrypted HOST_ID) and 15 (signature
verification) from the original processing rules of HIPv2 have been
removed in the above processing rules for HIP DEX. Moreover, step 10
of the HIPv2 processing rules has been adapted to account for
optional extension of the retransmission mechanism. Step 16 has been
added to the processing rules in this document. The considerations
about malformed I2 packets in Sections 6.9.1 of [RFC7401] also apply
to HIP DEX.

6.8. Processing Incoming R2 Packets

R2 packets in HIP DEX are handled identically to HIPv2 (see Section
6.10 of [RFC7401]) with the following exceptions: HIP DEX introduces
a new session key exchange via the ENCRYPTED_KEY parameter and does
not employ signatures.

The modified conceptual processing rules for responding to an R2
packet are as follows:

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 I2-SENT state.

3. The system MUST verify the HIP_MAC according to the procedures
   in Section 6.2.

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

5. The system MUST validate the Responder's HI per Section 9.2.

6. 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 (e.g. aborting with logging),
   based on its local policy.

7. The system MUST decrypt 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 6.3).

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

Note that step 4 (signature verification) from the original
processing rules of HIPv2 has been replaced with a negotiation re-
evaluation in the above processing rules for HIP DEX. Moreover, step
6 has been added to the processing rules.

6.9. Processing Incoming NOTIFY Packets

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

If a NOTIFY packet is received in state I2-SENT, this packet is an
I2 reception acknowledgement of the optional retransmission
mechanism extension and SHOULD be processed. The following steps
define the conceptual processing rules for such incoming NOTIFY
packets in state I2-SENT:

1. 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 I2-SENT state. If this check fails, the
   NOTIFY packet MUST be dropped silently.

2. If the NOTIFY packet contains a NOTIFICATION parameter with
   Notify Message Type I2_ACKNOWLEDGEMENT, the system SHOULD set
   the I2 retransmission timer to the I2 processing time indicated
   in the NOTIFICATION parameter plus half the RTT-based timeout
   value. The system MUST NOT set the retransmission timeout to a
   higher value than allowed by a local policy. Moreover, the
   system SHOULD reset the I2 retransmission timer to the RTT-
   based timeout value after the next I2 retransmission.

6.10. Processing UPDATE, CLOSE, and CLOSE_ACK Packets

UPDATE, CLOSE, and CLOSE_ACK packets are handled similarly in HIP
DEX as in HIPv2 (see Sections 6.11, 6.12, 6.14, and 6.15 of
[RFC7401]). The only difference is the that the HIP_SIGNATURE is
never present and, therefore, is not required to be processed by the
receiving party.

[RFC7402] specifies the rekeying of an existing HIP SA using the UPDATE message. This rekeying procedure can also be used with HIP DEX. However, where rekeying involves a new Diffie-Hellman key exchange, HIP DEX peers MUST establish a new HIP association in order to create a new Pair-wise Key SA due to the use of static ECDH key-pairs with HIP DEX.

6.11. 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.

Storing of the R1 generation counter values and ENCRYPTED_KEY counter (Section 5.2.5) 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. The value of puzzle difficulty K used in the HIP R1 must be chosen with care. Values for the K that are too high will exclude clients with weak CPUs because these devices cannot solve the puzzle within a reasonable amount of time. The K value 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.

All HIP DEX implementations SHOULD provide for an Access Control List (ACL), representing for which hosts they accept HIP diet exchanges, and the preferred transport format and local lifetimes. Wildcarding SHOULD be supported for such ACLs.

7.1. HIT/HI ACL

Both the Initiator and Responder SHOULD implement an ACL. Minimally, these ACLs will be a list of trusted HIT/HIs. They may also contain the password used in the password-based two-factor authentication (Appendix A) and preferred HIT Suite.

ACL processing is applied to all HIP packets. A HIP peer MAY reject any packet where the Receiver's HIT is not in the ACL. The HI (in the R1, I2, and optionally NOTIFY packets) MUST be validated as well, when present in the ACL. This is the defense against collision and second-image attacks on the HIT generation.

Devices with no input mechanism (e.g. sensors) SHOULD accept R1 packets from unknown HITs. These R1 packets SHOULD contain the start of the password-based two-factor authentication . If the R2 for this HIT indicates success, then the device may add this HIT/HI to its ACL for future use.

Devices unable to manage an ACL (e.g. sensors) are subject to MITM attacks, even with the use of the password authentication (password theft by attacker). As long as the other peer (e.g. sensor controller) does use an ACL, the attack can be recognized there and addressed. This is often seen where the sensor does not appear as properly operating with the controller, as the attacker cannot impersonate information in the ACL.

8. Interoperability between HIP DEX and HIPv2

HIP DEX and HIPv2 both use the same protocol number and packet formats. Hence, an implementation that either supports HIP DEX or HIPv2 has to be able to detect the dialect that the peer is speaking. This section outlines how a HIP DEX implementation can achieve such detection for the two relevant cases where:

1. the Initiator supports HIP DEX and the Responder supports HIP BEX,

2. the Initiator supports HIP BEX and the Responder supports HIP DEX.

In the first case, the HIP DEX implementation (Initiator) inspects the Responder's HIT prior to sending the I1 packet. If the OGA ID field of this HIT does not indicate the HIP DEX HIT Suite ID, the HIP DEX implementation cancels the handshake. If the Responder is unknown prior to sending the I1 packet (i.e., opportunistic mode), the HIP DEX implementation performs the above check on reception of the R1 packet and cancels the handshake in case of a negative result. In both failure scenarios, the implementation should report an error to the user via appropriate means.

In the second case, the HIP DEX implementation (Responder) inspects the Initiator's HIT on reception of an I1 packet. If the OGA ID field of this HIT does not indicate the HIP DEX HIT Suite ID, the HIP DEX implementation cancels the handshake and sends an ICMP packet with type Parameter Problem, with the Pointer pointing to the source HIT, to the Initiator. As an off-path adversary could also send such an ICMP packet with the aim to prevent the HIP DEX handshake from completing, the Initiator SHOULD NOT react to an ICMP message before retransmission counter reaches I1_RETRIES_MAX in its state machine (see Table 3 in [RFC7401]).

9. Security Considerations

HIP DEX closely resembles HIPv2. As such, the security considerations discussed in Section 8 of [RFC7401] similarly apply to HIP DEX. HIP DEX, however, replaces the SIGMA-based authenticated Diffie-Hellman key exchange of HIPv2 with an exchange of random keying material that is encrypted with a Diffie-Hellman derived key. Both the Initiator and Responder contribute to this keying material. As a result, the following additional security considerations apply to HIP DEX:

  *The strength of the keys for both the Master and Pair-wise Key
   SAs is based on the quality of the random keying material
   generated by the Initiator and the Responder. As either peer may
   be a sensor or an actuator device, there is a natural concern
   about the quality of its random number generator. Thus at least a
   CSPRNG SHOULD be used.

  *HIP DEX lacks the Forward Secrecy (FS) property of HIPv2.
   Consequently, if an HI is compromised, all previous HIP
   connections protected with that HI are compromised as explained
   in Section 1.

  *The HIP DEX HIT generation may present new attack opportunities.
   Hence, HIP DEX HITs MUST NOT be used as the only means to
   identify a peer in an ACL. Instead, the use of the peer's HI is
   recommended as explained in Section 3.

  *The R1 packet is unauthenticated and offers an adversary a new
   attack vector against the Initiator. This is mitigated by only
   processing a received R1 packet when the Initiator has previously
   sent a corresponding I1 packet. Moreover, the Responder repeats
   the DH_GROUP_LIST, HIP_CIPHER, HIT_SUITE_LIST, and
   TRANSPORT_FORMAT_LIST parameters in the R2 packet in order to
   enable the Initiator to verify that these parameters have not
   been modified by an attacker in the unprotected R1 packet as
   explained in Section 6.8.

  *Contrary to HIPv2, HIP DEX does not provide for end-point
   anonymity for the Initiator or Responder. Thus, any signaling
   that indicates such anonymity should be ignored as explained in
   Section 1.1.

  *It is critical to properly manage the ENCRYPTED_KEY counter
   (Section 5.2.5). If non-volatile store is used to maintain HIP
   state across system resets, then this counter MUST be part of the
   state store.

The optional retransmission extension of HIP DEX is based on a
NOTIFY packet that the Responder can use to inform the Initiator

about the reception of an I2 packet. The Responder, however, cannot
protect the authenticity of this packet as it did not yet set up the
Master Key SA. Hence, an eavesdropping adversary may send spoofed
reception acknowledgements for an overheard I2 packet and signal an
arbitrary I2 processing time to the Initiator. The adversary can,
e.g., indicate a lower I2 processing time than actually required by
the Responder in order to cause premature retransmissions. To
protect against this attack, the Initiator SHOULD set the NOTIFY-
based timeout value to the maximum indicated packet processing time
in case of conflicting NOTIFY packets. This allows the legitimate
Responder to extend the retransmission timeout to the intended
length. The adversary, however, can still arbitrarily delay the
protocol handshake beyond the Responder's actual I2 processing time.
To limit the extend of such a maliciously induced handshake delay,
this specification additionally requires the Initiator not to set
the NOTIFY-based timeout value higher than allowed by a local
policy.

Section 5.3.1 mentions that implementations need to be able to
handle storms of I1 packets. Contrary to HIPv2, R1 packets cannot be
pre-computed in HIP DEX and also the state machine does not include
an "R1_SENT" state (that would enable caching of R1 packets).
Therefore, an implementation has to cache information (e.g., at
least the HITs) from incoming I1 packets and rate control the
incoming I1 packets to avoid unnecessary packet processing during I1
packet storms.

9.1. Use of AES-CTR for HIP Parameter Encryption

AES-CTR is a basic cipher mode that does not accept an
initialization vector to allow for key re-use. In essence, it
stretches the initial key into a much longer keystream (akin to a
stream cipher) that is used like a one-time pad. Any reuse of that
keystream breaks the confidentiality of the protected data, so when
using AES-CTR, care must be taken to ensure that within a given
keystream the counter value is never reused, and that any given key
is only used to generate a single keystream. The integration of AES-
CTR into IPsec ESP (RFC 3686) used by HIP (and, thus, by HIP-DEX)
improves on the situation by partitioning the 128-bit counter space
into a 32-bit nonce, 64-bit IV, and 32-bits of counter. The counter
is incremented to provide a keystream for protecting a given packet,
the IV is chosen by the encryptor in a "manner that ensures
uniqueness", and the nonce persists for the lifetime of a given SA.
In particular, in this usage the nonce must be unpredictable, not
just single-use. In HIP-DEX, the properties of nonce uniqueness/
unpredictability and per-packet IV uniqueness are defined in Section
5.2.2.

9.2. Need to Validate Public Keys

With the curves specified here, there is a straightforward key
extraction attack, which is a very serious problem with the use of
static keys by HIP-DEX. Thus it is MANDATORY to validate the peer's
Public Key.

With the NIST curves, there are no internal length markers, so each
number representation occupies as many octets as implied by the
curve parameters. For P-256, this means that each of X and Y use 32
octets, padded on the left by zeros if necessary. For P-384, they
take 48 octets each.

For Curve25519 and Curve448, the contents of the public value are
the byte string inputs and outputs of the corresponding functions
defined in [RFC7748]: 32 bytes for EC25519 and 56 bytes for EC448.

The validation is done in Section 6.7, step 4 and Section 6.8, step
5.

10. IANA Considerations

The following changes to the "Host Identity Protocol (HIP)
Parameters" registries have been made:

ENCRYPTED_KEY "ENCRYPTED_KEY" with type number TBD1 (suggested:
    643) (see Section 5.2.5) in the "Parameter Types" subregistry of
    the "Host Identity Protocol (HIP) Parameters" registry.

DH_GROUP_LIST This document defines the new DH_GROUPS Curve25519
    with value TBD7 (suggested: 12) and Curve448 with value TBD8
    (suggested: 13) (see Section 5.2.1) in the "Group IDs"
    subregistry of the "Host Identity Protocol (HIP) Parameters"
    registry.

HIT Suite ID This document defines the new HIT Suite "ECDH/FOLD"
    without four-bit ID of TBD2 (suggested: 4) and eight-bit encoding
    of TBD3 (suggested: 0x40) (see Section 5.2.4) in the "HIT Suite
    ID" subregistry of the "Host Identity Protocol (HIP) Parameters"
    registry.

HIP Cipher ID This document defines the new HIP Cipher ID "AES-128-
    CTR" with type number TBD4 (suggested: 5) (see Section 5.2.2) in
    the "HIP Cipher ID" subregistry of the "Host Identity Protocol
    (HIP) Parameters" registry.

HI Algorithm This document defines the new HI Algorithm "ECDH" with
    type number TBD5 (suggested: 11) (see Section 5.2.3) in the "HI
    Algorithm" subregistry of the "Host Identity Protocol (HIP)
    Parameters" registry.

I_NONCE
>   "I_NONCE" with type number TBD6 (suggested: 644) (see
>   [Section 5.2.6](#)) in the "Parameter Types" subregistry of the "Host
>   Identity Protocol (HIP) Parameters" registry.

ECC Curve Label This document specifies a new algorithm-specific
>   subregistry named "ECDH Curve Label". The values for this
>   subregistry are defined in [Section 5.2.1](#). The complete list of
>   algorithms for the DH_GROUP_LIST parameter are listed in the
>   "Group IDs" subregistry of the "Host Identity Protocol (HIP)
>   Parameters" registry.

11. Acknowledgements

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. Special thanks to
Mohit Sethi in helping with the draft during IESG process.

Special thanks to Dr. Hugo Krawczyk for early guidance on the IRTF
CFRG list on how to safely use CMAC in a key derivation function.
And Dr. Lily Chen of NIST who spent time discussing CKDF at IEEE 802
and IETF meetings.

12. 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.

The draft was then renamed from draft-moskowitz-hip-dex to draft-
ietf-hip-dex.

12.1. Changes in draft-ietf-hip-dex-22

   *Apply editorial comment from Roman Danyliw

   *Clarify IKM content for Master SA and Pairwise SA in [Section 6.3](#)

   *Add behavior on BEX before DEX to [Section 1.2](#)

   *Added [[NIST.SP.800-56C](#)], [[NIST.SP.800-108](#)], and [[KeyDerivation](#)]
    as source guidance for CKDF to [Section 6.3](#)

   *Removed NIST curves from [Section 5.2.1](#) and [Section 5.2.3](#) as too
    slow for 8-bit cpus

12.2. Changes in draft-ietf-hip-dex-21

   *Clarified on security concerns of using AES-CTR in [Section 9.1](#)

*Edits for SECDIR comments

12.3. Changes in draft-ietf-hip-dex-20

*Clarified text on AES-CTR for HIP parameter encryption. This
includes [Section 9.1](#)

*Clarified text on R2 processing to validate content of R1.

*Clarified Applicability section.

*Expanded Fig 1.

*Clarified differences between BEX and DEX state machines.

*ESP transform is MTI and ESP-TCP is Experimental.

12.4. Changes in draft-ietf-hip-dex-19

*Replaced reference to RFC4493 for CMAC with NIST SP800-38B.

*Remove NIST P-521 from DH_GROUP_LIST.

*Remove NULL-ENCRYPT.

*Added reference to rfc8005 for HIT lookup in DNS.

*Remove setting Control bit: A.

*Many textual improvements per Benjamin Kaduk comments.

12.5. Changes in draft-ietf-hip-dex-18

*Changed Perfect Forward Secrecy to Forward Secrecy.

12.6. Changes in draft-ietf-hip-dex-17

*Added hex values for strings CKDF-Extract and CKDF-Expand.

*Replace Perfect Forward Secrecy with Forward Secrecy.

12.7. Changes in draft-ietf-hip-dex-16

*Remove old placeholder text.

*Remove SECP160R1. Experience has shown EC25519 performance equal
enough to not need it.

12.8. Changes in draft-ietf-hip-dex-15

*Added Public Key validation in I2 and R2 processing.

*Added ACL processing ([Section 7.1](#)).

   *Added IANA instructions for DH_GROUP_LIST.

12.9. Changes in draft-ietf-hip-dex-14

   *Changes to ([Section 5.4](#)) per Jeff Ahrenholz for Suresh Krishnan
    comment

12.10. Changes in draft-ietf-hip-dex-12 and 13

   *Nits from Jeff Ahrenholz (including some formatting issues)

12.11. Changes in draft-ietf-hip-dex-11 and 12

   *Included more precise references to the IANA subregistries

   *Addressed GEN-ART feedback from Francis Dupont

   *Added reasoning for FS in a separate section, and it is mentioned
    also in the abstract and intro.

   *Donald Eastlake's (secdir) nits addressed

   *Resolved IANA nits from Amanda Baber.

   *New sections: "Why introduce folding" ([Section 3.2.1](#)), "SECP160R1
    Considered Unsafe" (removed in ver 16), "Need to Validate Public
    Keys" ([Section 9.2](#)), and "I_NONCE" ([Section 5.2.6](#)) to address
    Eric Rescorla's concerns.

12.12. Changes in draft-ietf-hip-dex-11

   *Update IANA considerations as requested by Eric Envyncke

12.13. Changes in draft-ietf-hip-dex-10

   *Explanations on why the document includes so many SHOULDs

12.14. Changes in draft-ietf-hip-dex-09

   *Fixed values for

      -DH_GROUP_LIST

      -HIT_SUITE_LIST

    to match [[RFC7401](#)].

12.15. Changes in draft-ietf-hip-dex-05

*Clarified main differences between HIP BEX and HIP DEX in Section 1.

*Addressed MitM attack in Section 8.

*Minor editorial changes.

12.16. Changes in draft-ietf-hip-dex-04

*Added new paragraph on rekeying procedure with HIP DEX.

*Updated references.

*Editorial changes.

12.17. Changes in draft-ietf-hip-dex-03

*Added new section on HIP DEX/HIPv2 interoperability

*Added reference to RFC4493 for CMAC.

*Added reference to RFC5869 for CKDF.

*Added processing of NOTIFY message in I2-SENT of state diagram.

*Editorial changes.

12.18. Changes in draft-ietf-hip-dex-02

*Author address change.

12.19. Changes in draft-ietf-hip-dex-01

*Added the new ECDH groups of Curve25519 and Curve448 from RFC 7748.

12.20. Changes in draft-ietf-hip-dex-00

*The Internet Draft was adopted by the HIP WG.

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

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

12.22. 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.

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

    *Numerous editorial changes.

    *New retransmission strategy.

    *New HIT generation mechanism.

    *Modified layout of ENCRYPTED_KEY parameter.

    *Clarify 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.

12.24. Changes in draft-moskowitz-hip-dex-02

    *Introduced formal definition of FOLD function.

    *Clarified use of CMAC for puzzle computation in section "Solving
     the Puzzle".

    *Several editorial changes.

12.25. Changes in draft-moskowitz-hip-dex-03

    *Addressed HI crypto agility.

    *Clarified purpose of secret exchanged via ENCRYPTED_KEY
     parameter.

*Extended the IV in the ENCRYPTED_KEY parameter.

*Introduced forward-references to HIP DEX KEYMAT process and
 improved KEYMAT section.

*Replaced Appendix A on "C code for ECC point multiplication" with
 short discussion in introduction.

*Updated references.

*Further editorial changes.

12.26. Changes in draft-moskowitz-hip-dex-04

*Improved retransmission extension.

*Updated and strongly revised packet processing rules.

*Updated security considerations.

*Updated IANA considerations.

*Move the HI Algorithm for ECDH to a value of 11.

*Many editorial changes.

13. References

13.1. Normative References

[NIST.SP.800-38B]
           Dworkin, M., "Recommendation for block cipher modes of
           operation :: the CMAC mode for authentication", National
           Institute of Standards and Technology report, DOI
           10.6028/nist.sp.800-38b, 2016, <https://doi.org/10.6028/
           nist.sp.800-38b>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
           RFC2119, March 1997, <https://www.rfc-editor.org/info/
           rfc2119>.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES)
           Counter Mode With IPsec Encapsulating Security Payload
           (ESP)", RFC 3686, DOI 10.17487/RFC3686, January 2004,
           <https://www.rfc-editor.org/info/rfc3686>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet
           Control Message Protocol (ICMPv6) for the Internet
           Protocol Version 6 (IPv6) Specification", STD 89, RFC

4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC6261] Keranen, A., "Encrypted Signaling Transport Modes for the
           Host Identity Protocol", RFC 6261, DOI 10.17487/RFC6261,
           May 2011, <https://www.rfc-editor.org/info/rfc6261>.

[RFC7343] Laganier, J. and F. Dupont, "An IPv6 Prefix for Overlay
           Routable Cryptographic Hash Identifiers Version 2
           (ORCHIDv2)", RFC 7343, DOI 10.17487/RFC7343, September
           2014, <https://www.rfc-editor.org/info/rfc7343>.

[RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T.
           Henderson, "Host Identity Protocol Version 2 (HIPv2)",
           RFC 7401, DOI 10.17487/RFC7401, April 2015, <https://
           www.rfc-editor.org/info/rfc7401>.

[RFC7402] Jokela, P., Moskowitz, R., and J. Melen, "Using the
           Encapsulating Security Payload (ESP) Transport Format
           with the Host Identity Protocol (HIP)", RFC 7402, DOI
           10.17487/RFC7402, April 2015, <https://www.rfc-editor.org/info/rfc7402>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
           2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
           May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[DH76]   Diffie, W. and M. Hellman, "New directions in
           cryptography", IEEE Transactions on Information Theory
           Vol. 22, pp. 644-654, DOI 10.1109/tit.1976.1055638,
           November 1976, <https://doi.org/10.1109/tit.1976.1055638>.

[hip-rfc4423-bis] Moskowitz, R. and M. Komu, "Host Identity Protocol
           Architecture", Work in Progress, Internet-Draft, draft-
           ietf-hip-rfc4423-bis-20, 14 February 2019, <https://
           tools.ietf.org/html/draft-ietf-hip-rfc4423-bis-20>.

[HWZ13]   Hummen, R., Wirtz, H., Ziegeldorf, J., Hiller, J., and K.
           Wehrle, "Tailoring end-to-end IP security protocols to
           the Internet of Things", 2013 21st IEEE International
           Conference on Network Protocols (ICNP), DOI 10.1109/icnp.2013.6733571, October 2013, <https://doi.org/10.1109/
           icnp.2013.6733571>.

[IEEE.802-11.2016] "IEEE Standard for 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, DOI 10.1109/ieeestd.2016.7786995, n.d.,
<https://doi.org/10.1109/ieeestd.2016.7786995>.

[IEEE.802-15-4.2015]
Engineers, I. O. E. A. E., "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, December 2015, <http://
standards.ieee.org/getieee802/download/
802.15.4-2015.pdf>.

[KeyDerivation] Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H.,
and T. Rabin, "Randomness Extraction and Key Derivation
Using the CBC, Cascade and HMAC Modes", Advances in
Cryptology - CRYPTO 2004 pp. 494-510, DOI
10.1007/978-3-540-28628-8_30, 2004, <https://doi.org/
10.1007/978-3-540-28628-8_30>.

[LN08]   Liu, A. and P. Ning, "TinyECC: A Configurable Library for
Elliptic Curve Cryptography in Wireless Sensor Networks",
2008 International Conference on Information Processing
in Sensor Networks (ipsn 2008), DOI 10.1109/ipsn.2008.47,
April 2008, <https://doi.org/10.1109/ipsn.2008.47>.

[NIST.SP.800-108]
Chen, L., "Recommendation for key derivation using
pseudorandom functions (revised)", National Institute of
Standards and Technology report, DOI 10.6028/nist.sp.
800-108, 2009, <https://doi.org/10.6028/nist.sp.800-108>.

[NIST.SP.800-56C]
Chen, L., "Recommendation for key derivation through
extraction-then-expansion", National Institute of
Standards and Technology report, DOI 10.6028/nist.sp.
800-56c, 2011, <https://doi.org/10.6028/nist.sp.800-56c>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-
Expand Key Derivation Function (HKDF)", RFC 5869, DOI
10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/
info/rfc5869>.

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

          10.17487/RFC6090, February 2011, <https://www.rfc-
          editor.org/info/rfc6090>.

   [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
              Constrained-Node Networks", RFC 7228, DOI 10.17487/
              RFC7228, May 2014, <https://www.rfc-editor.org/info/
              rfc7228>.

   [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T.
              Kivinen, "Internet Key Exchange Protocol Version 2
              (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October
              2014, <https://www.rfc-editor.org/info/rfc7296>.

   [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves
              for Security", RFC 7748, DOI 10.17487/RFC7748, January
              2016, <https://www.rfc-editor.org/info/rfc7748>.

   [RFC8005] Laganier, J., "Host Identity Protocol (HIP) Domain Name
              System (DNS) Extension", RFC 8005, DOI 10.17487/RFC8005,
              October 2016, <https://www.rfc-editor.org/info/rfc8005>.

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

   HIP DEX allows identifying 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.

Appendix B. IESG Considerations

   During IESG review, a concern was raised on the number of SHOULDs in
   this document. Here is an analysis of the 57 SHOULDs in HIP DEX.

   46 of SHOULDs are also in [RFC7401]. Here are the sections with
   SHOULDs that match up with [RFC7401]:

Many of the other 11 SHOULDs are due to the nature of constrained
devices and in most cases the text points this out:

In Section 4.1.1, this is clearly adjusting for how the puzzle could
actually be an attack against a constrained device. Same situation
in Section 5.3.2.

Section 6, clearly states that:

    it should be noted that many of the packet processing rules are
    denoted here with "SHOULD" but may be updated to "MUST" when
    further implementation experience provides better guidance.

thus the SHOULD here is informative of future guidance.

The SHOULD in Section 6.3, clearly reflects new work with the new
Sponge Function KDFs:

The keys derived for the Pair-wise Key SA are not used during the
HIP DEX handshake. Instead, these keys are made available as payload
protection keys (e.g., for IPsec). 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 to derive the keys of the Pair-wise Key SA based on the
concatenation of the random values that are contained in the
exchanged ENCRYPTED_KEY parameters.

In Section 6.5, the reason why this is a SHOULD should be clear to any implementer. That is the HIT Suite list in I1 is a desire on what suite to use. The Responder may have 'different ideas' about the Suite to use (like what it supports). Thus it is best that the Responder selects a DEX HIT, but there are good reasons, in an implementation not to do so. The implementer should know this and will deal with it appropriately.

The SHOULDs in Section 6.7 and Section 6.9 are there for considerations for constrained systems. Some constrained systems need this approach, others may not.

The 2nd SHOULD in Section 7 follows the same as above. ACLs and constrained systems tend to go together.

Similarly in Section 8 the SHOULD is again is highlighting constrained system processing considerations.

Authors' Addresses

Robert Moskowitz (editor)
HTT Consulting
Oak Park, MI
United States of America

Email: rgm@htt-consult.com

Rene Hummen
Hirschmann Automation and Control
Stuttgarter Strasse 45-51
72654 Neckartenzlingen
Germany

Email: rene.hummen@belden.com

Miika Komu
Ericsson Research, Finland
Hirsalantie 11
FI-02420 Jorvas
Finland

Email: miika.komu@ericsson.com