Host Identity Protocol
Internet-Draft
Intended status: Experimental
Expires: April 30, 2012

T. Heer, Ed.
R. Hummen
K. Wehrle
RWTH Aachen University,
Communication and Distributed
Systems Group
M. Komu
Aalto University
October 28, 2011

End-Host Authentication for HIP Middleboxes
draft-heer-hip-middle-auth-04

Abstract

The Host Identity Protocol [RFC5201] is a signaling protocol for secure communication, mobility, and multihoming that introduces a cryptographic namespace. This document specifies an extension for HIP that enables middleboxes to unambiguously verify the identities of hosts that communicate across them. This extension allows middleboxes to verify the liveness and freshness of a HIP association and, thus, to secure access control in middleboxes.

Requirements Language

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

Notation

Status of this Memo

Table of Contents

1. Introduction 4
  1.1. Authentication and Replay Attacks 5
2. Protocol Overview 6
  2.1. Signed Middlebox Nonces 6
  2.2. Identity Verification by Middleboxes 9
  2.3. Failure Signaling 15
  2.4. Fragmentation 16
  2.5. HIP Parameters 16
3. Security Services for the HIP Control Channel 18
  3.1. Adversary model and Security Services 18
4. Security Services for the HIP Payload Channel 19
  4.1. Access Control 20
  4.2. Resource allocation 21
5. Security Considerations 21
6. IANA Considerations 22
7. Acknowledgments 22
8. Changelog 22
  8.1. Version 4 22
  8.2. Version 3 22
9. Normative References 22
Authors' Addresses 23

1. Introduction

The Host Identity Protocol (HIP) introduces a new cryptographic namespace, based on public keys, in order to secure Internet communication. This namespace allows hosts to securely address and authenticate their peers. HIP was designed to be middlebox-friendly and to allow middleboxes to inspect HIP control traffic. Examples of such middleboxes are firewalls and Network Address Translators (NATs).

In this context, one can distinguish HIP-aware middleboxes, which are designed to process HIP packets, and other middleboxes, which are unaware of HIP. This document addresses only HIP-aware middleboxes while the behavior of HIP in combination with HIP-unaware middleboxes is specified in [RFC5770]. Moreover, the scope of this document is restricted to middleboxes that use HIP in order to provide Authentication, Authorization, and Accounting (AAA)-related services and, thus, need to authenticate the communicating peers that send traffic over the middlebox. The class of middleboxes this document focuses on does not require the end-host to explicitly register to the middlebox. HIP behavior for interacting and registering to such middleboxes is specified in [RFC5203]. Thus, we focus on middleboxes that build their state based on packets they forward (path-coupled signaling).

An example of such a middlebox is a firewall that only allows traffic from certain hosts to traverse. We assume that access control is performed based on Host Identities (HIs). Such an authenticating middlebox needs to observe the HIP Base EXchange (BEX) or a HIP mobility update [RFC5206] and check the Host Identifiers (HIs) in the packets.

Along the lines of [RFC5207], an authentication solution for middleboxes must have some vital properties. For one, the middlebox must be able to unambiguously identify one or both of the communicating peers. Additionally, the solution must not allow for new attacks against the middlebox. This document specifies a HIP extension that allows middleboxes to participate in the HIP handshake and the HIP update process in order to allow these middleboxes to reliably verify the identities of the communicating peers. To this end, this HIP extension defines how middleboxes can interact with end-hosts in order to verify their identities.

Verifying public-key (PK) signatures is costly in terms of CPU cycles. Thus, in addition to authentication capabilities, it is also necessary to provide middleboxes with a way of defending against resource-exhaustion attacks that target PK signature verification. This document defines how middleboxes can utilize the HIP puzzle mechanism defined in [RFC5201] to slow down resource-exhaustion attacks.

The presented authentication extension only targets the HIP control channel. Additional security considerations and possible security services for the HIP payload channel are discussed in Section 4.

1.1. Authentication and Replay Attacks

Middleboxes may need to verify the HIs in the HIP base exchange messages to perform access control based on Host Identities. However, passive verification of HIs in the messages is not sufficient to ensure the identity of an end-host because of a possible replay attack against which the basic HIP protocol as specified in [RFC5201] does not provide adequate protection.

To illustrate the need for additional security measures for HIP-aware middleboxes, we briefly outline the replay attack: Assume that the legitimate owner of Host Identity Tag (HIT) X establishes a HIP association with the legitimate owner of HIT Y at some point in time and an attacker A overhears the base exchange and records it.

Assume that a middlebox M checks HIP HIs in order to restrict traffic passing through the box. At some later point in time, Attacker A collaborates with another attacker B. They replay the very same BEX packets to the middlebox M on the communication path. Note that it is not required that the middlebox M was on the communication path between X and Y when the BEX was recorded.

The middlebox has no way to distinguish legitimate hosts X and Y from the attackers A and B as it can only overhear the BEX passively and it cannot can distinguish the replayed BEX from a the genuine handshake. As the attackers overheard the SPI numbers, they can traverse the middlebox with "fake" ESP packets with valid SPI numbers, and hence, send data across M without proper authentication. Since the middleboxes do not know the integrity and encryption keys for ESP, they cannot distinguish valid ESP packets from forged ones. Hence, collaborating attackers can use any replayed BEX to falsely authenticate to the middlebox and thus impersonate any host. This is problematic in cases in which the middlebox needs to know the identity of the peers that communicate across it. Examples for such cases are AAA-related services, such as access control, logging of activities, and accounting for traffic volume or connection duration.

This attack scenario is not addressed by the current HIP specifications. Therefore, this document specifies a HIP extension that allows middleboxes to defend against this attack.

2. Protocol Overview

This section gives an overview of the interaction between hosts and authenticating middleboxes. This document describes a framework that middleboxes can use to implement authentication of end-hosts and leaves its further use to other documents and to middlebox implementors.

2.1. Signed Middlebox Nonces

The described attack scenario shows the necessity for unambiguous end-host identity verification by middleboxes. However, this authentication cannot be purely end-to end: a) Relying on nonces generated by the end-hosts is not possible because middleboxes cannot verify the freshness of these nonces. b) Introducing time-stamps restricts the attack to a certain time frame but requires global time synchronization and therefore should be avoided.

The following sections specify how HIP hosts can prove their identity by performing a challenge-response protocol between the middlebox and the end-hosts. As a challenge, the middlebox adds information (e.g. self-generated nonces) to HIP control packets which the end-hosts sign with public-key (PK) signatures and echo back.

The challenge-response mechanism is similar to the ECHO_REQUEST/ ECHO_RESPONSE mechanism employed already by HIP end-hosts (see [RFC5201]). It assumes that the end-hosts exchange at least two HIP packets with each other. The middlebox adds a CHALLENGE_REQUEST parameter to the first HIP control packet. Similar to the ECHO_REQUEST parameter in the original HIP protocol, this parameter contains an opaque data field that must be echoed by its receiver. The receiver echoes the opaque data field in a CHALLENGE_RESPONSE parameter. The CHALLENGE_RESPONSE parameter must be covered by the packet signature, thereby proving that the receiver is in possession of the private key that corresponds to the HI.

The middlebox can either verify the identity of the initiator, the responder, or both peers, depending on the purpose of the middlebox. The choice of which authentication is required left to middlebox implementers.

2.1.1. CHALLENGE_REQUEST

Middleboxes MAY add CHALLENGE_REQUEST parameters to the R1 and I2 packets and to any UPDATE packet. This parameter contains an opaque data block of variable size, which the middlebox uses to carry arbitrary data (e.g., a nonce). The HIP packets that carry middlebox challenges may contain multiple CHALLENGE_REQUEST parameters, since all middleboxes on the path may add these parameters. A middlebox MUST append its own CHALLENGE_REQUEST parameter behind already existing CHALLENGE_REQUEST parameters in the HIP packet. In order to avoid packet fragmentation, the MBs should restrict the size of the variable data field in the CHALLENGE_REQUEST parameter. The total length of the packets SHOULD not exceed 1280 bytes to avoid IPv6 fragmentation [RFC2460].

The middleboxes add the CHALLENGE_REQUEST parameter to the unprotected part of a HIP message. Thus, it does not corrupt any HMAC or public-key signatures that protect the HIP packet. However, the middlebox MUST recompute the IP and HIP header checksums as defined in [RFC5201] and the UDP headers of UDP encapsulated HIP packets as defined in [RFC5770].

A HIP end-host that receives a HIP control packet containing one or more CHALLENGE_REQUEST parameters must copy the contents of each parameter without modification to a single CHALLENGE_RESPONSE parameter. This end-host MUST send the CHALLENGE_RESPONSE parameter within the signed part of its reply. Note that middleboxes MAY also add ECHO_REQUEST_UNSIGNED parameters as specified in [RFC5201] if the receiver of the parameter is not required to sign the contents of the ECHO_REQUEST.

Middleboxes can delay state creation by utilizing the CHALLENGE_REQUEST and CHALLENGE_RESPONSE parameters by hiding encrypted or otherwise protected information about previous authentication steps in the opaque data field.

2.1.2. CHALLENGE_RESPONSE

When a middlebox injects an opaque blob of data with a CHALLENGE_REQUEST parameter, it expects to receive the same data without modification as part of a CHALLENGE_RESPONSE parameter in a subsequent packet. Hence, the opaque data MUST be copied as it is from the corresponding CHALLENGE_REQUEST parameter. In the case of multiple CHALLENGE_REQUEST parameters, their order MUST be preserved within the corresponding CHALLENGE_RESPONSE parameter.

The CHALLENGE_REQUEST and CHALLENGE_RESPONSE parameters MAY be used for any purpose, in particular when a middlebox has to carry state information in a HIP packet to receive it in the next response packet. The CHALLENGE_RESPONSE MUST be covered by the HIP_SIGNATURE.

The CHALLENGE_RESPONSE parameter is non-critical. Depending on its local policy, a middlebox can react differently on a missing CHALLENGE_RESPONSE parameter. Possible actions range from degraded or restricted service, such as bandwidth limitation, up to refusing connections and reporting access violations.

When sending a HIP control packet, an end-host may face the problem that not all opaque values of the received CHALLENGE_REQUEST parameters fit into the CHALLENGE_RESPONSE parameter due to HIP control packet size restrictions. In this case, the host should send several packets. The first packet contains a CHALLENGE_RESPONSE parameter that includes the received opaque values of the CHALLENGE_REQUEST parameters starting from the last occurrence in the packet. Further packets contain the remaining values in the reverse order of the inclusion in the received packet. This way, the middleboxes closest to the sender will already have authenticated the identity of the peers and can let further control packets pass through.

2.1.3. Middlebox Puzzles

Since PK operations are costly in terms of CPU cycles, a middlebox has to defend itself against resource-exhaustion attacks when verifying signatures in HIP packets. The HIP base protocol [RFC5201] specifies a puzzle mechanism to protect the Responder from I2 floods that require numerous public-key operations. However, middleboxes cannot utilize this mechanism because they cannot verify the freshness of the puzzle solution in the BEX packets. This section specifies how middleboxes can utilize the puzzle mechanism to add their own puzzles to R1, I2, and any UPDATE packets. This allows middleboxes to shelter against Denial of Service (DoS) attacks on PK verification.

The puzzle mechanism for middleboxes utilizes the CHALLENGE_REQUEST and CHALLENGE_RESPONSE parameters. The CHALLENGE_REQUEST parameter contains fields for setting the difficulty and the expiration date of the puzzle. In contrast to the PUZZLE parameter in the HIP base specifications, there is no dedicated puzzle seed field. Instead, the hash of the opaque data field in the CHALLENGE_REQUEST parameter serves as puzzle seed. The hash is generated by applying the SHA-1 algorithm to the opaque data field. The destination end-host of the HIP control packet MUST solve the puzzle and provide the solution in the CHALLENGE_RESPONSE parameter. The middlebox can set the puzzle difficulty by adjusting the K value in the CHALLENGE_REQUEST packet. The semantics of this field equal the semantics of the PUZZLE parameter. Setting K to 0 signifies that no puzzle solution is required.

In case of multiple CHALLENGE_RESPONSE parameters, the responder derives the puzzle seed from the concatenation of the opaque data of all CHALLENGE_REQUEST parameters in the received control packet in the reverse order of their inclusion. Furthermore, he MUST compute the solution based on the highest difficulty value K in the received CHALLENGE_REQUEST parameters. This selection of K satisfies the security requirements of each middlebox while preventing the the receiver from computing multiple puzzle solutions. The responder MUST meet the lowest time boundaries of the received CHALLENGE_REQUEST parameters. Otherwise, there exists one on-path middlebox that will not approve the solution.

When approaching the IPv6 packet fragmentation threshold, end-hosts should split the CHALLENGE_RESPONSE parameter in case of multiple CHALLENGE_REQUEST parameters. Hence, end-hosts SHOULD compute the puzzle solution after the overall packet size of the response packet has been determined. Hence, only the opaque values of the CHALLENGE_REQUEST parameters that are included in the respective CHALLENGE_RESPONSE parameter MUST be used during the puzzle seed generation.

Since a puzzle increases the delay and computational cost for establishing or updating a HIP association, a middlebox SHOULD only increase K when it is under attack. Moreover, middleboxes SHOULD distinguish attack directions. If the majority of the CPU load is caused by verifying HIP control messages that arrive from a certain interface, middleboxes MAY increase K for HIP control packets that leave the interface. The middlebox chooses the difficultly of the puzzle according to its load and local policies.

2.1.4. CHALLENGE_RESPONSE Verification

When a middlebox has added a CHALLENGE_REQUEST parameter to a control packet and receives a control packet that contains a CHALLENGE_RESPONSE parameter, it first checks if its opaque data has been echoed back correctly. To this end, it traverses the Opaque values included in the CHALLENGE_RESPONSE parameter.

If the opaque data has been echoed back correctly by the end-host, the middlebox verifies the provided puzzle solution. It, therefore, hashes the Opaque values as contained in the CHALLENGE_RESPONSE parameter and verifies the signaled solution. In case of a successful verification, the middlebox MAY check further security mechanisms such as the PK signature and process the packet according to its function.

2.2. Identity Verification by Middleboxes

This section describes how middleboxes can influence the BEX and the HIP update process in order to verify the identity of the HIP end-hosts.

2.2.1. Identity Verification During BEX

Middleboxes MAY add CHALLENGE_REQUEST parameters to R1 and I2 packets in order to verify the identities of the participating end-hosts. Middleboxes can choose either to authenticate the Initiator, the Responder, or both. Middleboxes MUST NOT add CHALLENGE_REQUEST parameters to I1 messages because this would expose the Responder to DoS attacks. Thus, middleboxes MUST let unauthenticated and minimal I1 packets traverse. Minimal means that the I1 packet MUST NOT contain more than the minimal set of parameters specified by HIP standards or internet drafts. In particular, the I1 packet MUST NOT contain any attached payload. Figure 1 illustrates the authentication process during the BEX.

```
    Main path:

    Initiator               Middlebox                        Responder
                       .-----------------.
     I1                |                 | I1
    -----------------> |                 |---------------------------->
                       |                 |
     R1, + CQ1         | Add CQ          | R1
    <----------------- |                 |<----------------------------
                       |                 |
     I2, {CR1}         | Verify CR1      | I2, {CR1} + CQ2
    -----------------> | Add CQ2         |---------------------------->
                       |                 |
                       |                 |
     R2, {CR2}         | Verify CR2      | R2, {CR2}
    <----------------- |                 |<------------------------------
                       '-----------------'

    CQ: Middlebox challenge reQuest
    CR: Middlebox challenge Response
    {}: Signature with sender's HI as key
```

Middlebox authentication of a HIP base exchange.

Figure 1

2.2.2. Identity Verification During Mobility Updates

HIP rekeying, mobility and multihoming UPDATE mechanisms for non-NATted environments are described in [RFC5206]. This section describes how middleboxes process UPDATE messages in non-NATted environments and leave NATted environments for future revisions of the draft.

The middleboxes can apply middlebox challenges to mobility related HIP control messages in the case where both end-hosts are single-homed. The middlebox challenges can be applied both ways as the UPDATE process consists of three packets (U1, U2, U3) which all traverse through the same middlebox as shown in Figure 2.

In cases, in which fewer packets are used for updating an association, the following rule applies.

RESPONSE RULE:

A HIP host, receiving a CHALLENGE_REQUEST MUST reply with a CHALLENGE_RESPONSE in its next UPDATE packet. If no further UPDATE packets are necessary to complete the update procedure, an additional UPDATE packet containing the CHALLENGE_RESPONSE MUST be sent.

```
 Initiator                      Middlebox                    Responder
                                 .-------.
  U1                             |       |  U1 + CQ1
 ------------------------------> |       | -------------------------->
                                 |       |
  U2, {CR1} + CQ2                |       |  U2, {CR1}
 <------------------------------ |OK     | <--------------------------
                                 |       |
  U3, {CR2}                      |       |  U3, {CR2}
 ------------------------------> |     OK| -------------------------->
                                 '-------'
 CQ: Middlebox challenge reQuest
 CR: Middlebox challenge Response
 {}: Signature with sender's HI as key
```

Middlebox authentication of a HIP mobility update over a single path.

Figure 2

Middlebox 1 in Figure 2 can verify the identity of the Responder by checking its PK signature and the presence of the CHALLENGE_RESPONSE in the U2 packet. If necessary, the middlebox MAY add an CHALLENGE_REQUEST for the Initiator of the update. The middlebox can verify the Initiator's identity by verifying its signature and the CHALLENGE_RESPONSE in the U3 packet.

2.2.3. Identity Verification for Multihomed Mobility Updates

Multihomed hosts may use multiple communication paths during an HIP mobility update. Depending on whether the middlebox is located on the communication path between the preferred locators of the hosts or not, the middlebox forwards different packets and, thus, needs to interact differently with the updates. Figure 3 I) and II) illustrates an update with Middlebox 1 on the path between the Initiator’s and the Responder’s preferred locators and with Middlebox 2 on an alternative path. Middlebox 2 is not located on the path between the preferred locators of the HIP end-hosts does not receive the U1 message. Therefore, it will not recognize any CHALLENGE_RESPONSE (CR1) in the second UPDATE packet. Thus, if a middlebox encounters non-matching or missing CHALLENGE_RESPONSE parameter in an initial update packet, the middlebox SHOULD ignore it.

Complying to the RESPONSE RULE stated in Section Section 2.2.2, the RESPONDER generates an additional fourth update packet on receiving the CHALLENGE_REQUEST. The update process for a middlebox on the preferred communication path (Middlebox 1) and a middlebox off the preferred communication path (Middlebox 2) is depicted in Figure 3.

```
   I)  Main path:

    Initiator                     Middlebox 1              Responder
                                  .------.
     U1                           |      |  U1 + CQ1
    ----------------------------> |      | --------------------------->
                                  |      |
     U2, {CR1} + CQ2              |      |  U2, {CR1}
    <---------------------------- |OK    | <---------------------------
                                  |      |
     U3, {CR2}                    |      |  U3, {CR2}
    ----------------------------> |    OK| --------------------------->
                                  ’------’

   II) Alternative path:

    Initiator                     Middlebox 2              Responder

     U1 (bypasses Middlebox 2)
    ------------------------------------------------------------------>
                                  .------.
     U2, {CR1} + CQ3              |      |  U2, {CR1}
    <---------------------------- | wrong| <---------------------------
                                  |      |
     U3’, {CR3}                   |      |  U3’, {CR3} + CQ4
    ----------------------------> |OK    | --------------------------->
                                  |      |
     U4, {CR4}                    |      |  U4,  {CR4}
    <---------------------------- |    OK| <---------------------------
                                  ’------’
    CQ: Middlebox challenge reQuest
    CR: Middlebox challenge Response
    {}: Signature with sender’s HI as key
```

Middlebox authentication of a HIP mobility update over different paths.

Figure 3

2.2.4. Identity Signaling During Updates

As middleboxes have to verify rapidly and forward HIP packets, they need to be supplied with all information necessary to do so. If end-hosts hand over communication to a new communication path, middleboxes need to be able to learn their Host Identifiers (HIs) from the UPDATE packets. Therefore, all packets that contain a CHALLENGE_RESPONSE parameter MUST contain the HOST_ID parameter.

2.2.5. Closing of Connections

The connection tear down as defined in [RFC5201] consists of two consecutive messages. This lack of a third message restricts middleboxes to authenticating the Responder of a CLOSE packet. However, verifying the legitimacy of the Responder suffices in most network scenarios, as CLOSE packets from unauthentic Initiators will be dropped by the Responder due to an invalid HMAC parameter. As a result, on-path middleboxes will not see CLOSE_ACK packets for rejected CLOSE packets. CLOSE_ACK packets can be authenticated by the middleboxes by adding a CHALLENGE_REQUEST parameter to the corresponding CLOSE packet as described above. Hence, middleboxes do not falsely tear down connections on illegitimate (forged) CLOSE packets.

If local policies still require a middlebox to authenticate the CLOSE messages of both peers, the tear down operation needs to be extended following the RESPONSE RULE in Section 2.2.2. Hence, the responder side CLOSE_ACK packet MUST be followed by an initiator side CLOSE_ACK if the received CLOSE_ACK packet contains a CHALLENGE_REQUEST parameter.

Middleboxes should have learned the identities of the peers during the BEX or an UPDATE prior to the CLOSE exchange. Hence, end-hosts are not required to include their identities in the CLOSE exchange. If a middlebox has not learned the identities of the peers when inspecting a CLOSE packet, it MUST forward the packet. In order to prevent misuse of the CLOSE exchange as a side channel for disallowed communication, middleboxes SHOULD rate limit unauthenticated CLOSE exchanges.

```
   I) Regular CLOSE authentication:

    Initiator                    Middlebox                   Responder
                                  .------.
     CLOSE                        |      |  CLOSE + CQ1
    ----------------------------> |      | --------------------------->
                                  |      |
     CLOSE_ACK, {CR1}             |      |  CLOSE_ACK, {CR1}
    <---------------------------- |OK    | <---------------------------
                                  |      |
                                  '------'

   II) Extended CLOSE authentication:

    Initiator                    Middlebox                   Responder
                                  .------.
     CLOSE                        |      |  CLOSE + CQ1
    ----------------------------> |      | --------------------------->
                                  |      |
     CLOSE_ACK, {CR1} + CQ2       |      |  CLOSE_ACK, {CR1}
    <---------------------------- |OK    | <---------------------------
                                  |      |
     CLOSE_ACK, {CR2}             |      |  CLOSE_ACK, {CR2}
    ----------------------------> |    OK| --------------------------->
                                  '------'
    CQ: Middlebox challenge reQuest
    CR: Middlebox challenge Response
    {}: Signature with sender's HI as key
```

Middlebox authentication of a HIP close with authentication of (I) the Responder and (II) both peers.

Figure 4

2.3. Failure Signaling

Middleboxes SHOULD inform the sender of a BEX packet or update packet if it does not satisfy the requirements of the middlebox. Reasons for non-satisfactory packets are missing HOST_ID or CHALLENGE_RESPONSE parameters. Other reasons may be middlebox policies regarding, for example, insufficient client capabilities or or insufficient credentials delivered in a HIP CERT parameter [RFC6253]. Options for expressing such shortcomings are ICMP packets if no HIP association is established and HIP_NOTIFY packets in case of an already established HIP association. Defining this signaling mechanism is future work.

2.4. Fragmentation

Analogously to the specification in [RFC5201], HIP aware middleboxes SHOULD support IP-level fragmentation and reassembly for IPv6 and MUST support IP-level fragmentation and reassembly for IPv4. However, when adding CHALLENGE_REQUEST parameters, a middlebox SHOULD keep the total packet size below 1280 bytes to avoid packet fragmentation in IPv6.

2.5. HIP Parameters

This HIP extension specifies four new HIP parameters that allow middleboxes to authenticate HIP end-hosts and to protect against DoS attacks.

2.5.1. CHALLENGE_REQUEST

A middlebox MAY append the CHALLENGE_REQUEST parameter to R1, I2, and UPDATE packets. The structure of the CHALLENGE_REQUEST parameter is depicted in the following figure. The semantics of the K and Lifetime fields are identical to the fields defined in the PUZZLE parameter in [RFC5201]. The opaque data field serves as nonce and puzzle seed value. To generate the seed corresponding to the 8-byte value I in [RFC5201], the receiver of the puzzle applies Ltrunc as defined in [RFC5201] to the received opaque data and truncates the result to 8 bytes. Note that the opaque data field must provide sufficient randomness to serve as puzzle seed.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | K, 1 byte     |    Lifetime   |                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               /
   /                                                               /
   /                     Opaque, (variable length)                 /
   /                                            +-+-+-+-+-+-+-+-+-+|
   /                                            |      Padding     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	65334
Length	Variable
K	K is the number of verified bits
Lifetime	Challenge lifetime 2^(value-32) seconds
Opaque	Opaque data that serves as nonce and as basis for the puzzle. The puzzle value I is generated by hashing the opaque data field with the hash function SHA-1 and truncating it to 8-byte length.

2.5.2. CHALLENGE_RESPONSE

The CHALLENGE_RESPONSE parameter is the response to one or more CHALLENGE_REQUEST parameters. The receiver of a CHALLENGE_REQUEST parameter SHOULD reply with a CHALLENGE_RESPONSE. Otherwise, the middlebox that added the CHALLENGE_REQUEST parameter MAY decide to degrade or deny its service. The Opaque fields of the received CHALLENGE_REQUEST parameters must be copied to the CHALLENGE_RESPONSE parameter in the reverse order of reception without any modification. As the number of opaque fields may be variable, it is encoded in the CHALLENGE_RESPONSE parameter. Furthermore, the length of each Opaque value is variable and is included in the parameter. The Opaque values are appended behind the last Opaque length field. Instead of copying the Opaque field of each CHALLENGE_REQUEST parameter, the input for the puzzle generation procedure may be reused. If the puzzle difficulty in the received CHALLENGE_REQUEST parameters is set to any other value except 0, an appropriate puzzle solution (adhering to the SOLUTION specifications in [RFC5201]) must be provided in the CHALLENGE_RESPONSE parameter. The CHALLENGE_RESPONSE parameter is non-critical and covered by the SIGNATURE. The structure of the CHALLENGE_RESPONSE parameter is depicted below:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | K, 1 byte     |    Lifetime   |      No. opaque values        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                  Puzzle solution #J, 8 bytes                  /
   /                                                               /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Opaque length        |         Opaque length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                  Opaque, (variable length)                    /
   /                                   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                   |                          /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                          /
   /                  Opaque, (variable length)                    /
   /                         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                         |             Padding                 |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	322
Length	Variable
K	K is the number of verified bits
Lifetime	Challenge lifetime 2^(value-32) seconds
No. opaque values	Number of included opaque values
Puzzle solution	Random number
Opaque length	Length of an included Opaque field
Opaque	Copied unmodified from the received CHALLENGE_REQUEST parameters

3. Security Services for the HIP Control Channel

In this section, we define the adversary model that the security analysis in the later sections will be based on.

3.1. Adversary model and Security Services

For discussing the security properties of the proposed HIP extension we first define an attacker model. We assume a Dolev-Yao threat model in which an adversary can eavesdrop on all traffic regardless of its source and destination. The adversary can inject arbitrary packets with any source and destination addresses. Consequently, an adversary can also replay previously eavesdropped messages. However, the adversary cannot subvert the cryptographic ciphers and hash function, nor can it compromise one of the communicating nodes.

Even in the face of this strong attacker, the proposed HIP extension enables middleboxes to verify the identity of the communicating HIP peers. It ensures that both peers are involved in the communication and that the HIP BEX or update packets are fresh, i.e. not replayed. It enables the middlebox to verify the source and destination (in terms of HIs) of the HIP association and the integrity of RSA and DSA signed HIP packets.

4. Security Services for the HIP Payload Channel

The presented extension for HIP authentication by middleboxes only covers the HIP control channel, i.e., the HIP control messages. Depending on the binding between the HIP control and payload channel, certain security properties for the payload channel can be derived from the strong cryptographic authentication of the end-hosts. Assuming that there is a secure binding between packets belonging to a payload stream and the control stream, the same security properties as in Section 3 apply to the payload stream.

ESP [RFC5202] is currently the default payload encapsulation format for HIP. A limitation of ESP is that it does not provide a secure binding between the HIP control channel and the ESP traffic on a per-packet basis. Hence, the achievable level of security for the payload channel is lower compared to the HIP control channel.

This section discusses security properties of an ESP payload channel bound to a HIP control channel. Depending on the assumed adversary model, certain security services are possible. We briefly describe two application scenarios and how they benefit from the resulting security services. For the payload channel, HIP in combination with the middlebox authentication scheme offers the following security services:

Attribute binding: Middleboxes can extract certain payload channel attributes (e.g. locators and SPIs) from the control channel. These attributes can be used to enforce certain restrictions on the payload channel, e.g., to exhibit the same attributes as the control channel. The attributes can either be stated explicitly in the HIP control packets or can be derived from the IP or UDP packets carrying the HIP control messages.

Host involvement: Middleboxes can verify whether a certain host is involved in the establishment of a HIP association and, thus, involved in the establishment of the payload channel.

Based on these security services we construct two use cases that illustrate the use of HIP authentication by middleboxes: access control and resource allocation as described in the following sections.

4.1. Access Control

Middleboxes can manage resources based on HIs. As an example, let us assume that a middlebox only forwards HIP payload packets after a successful HIP BEX or HIP update. The middlebox uses the parameters in the control channel (specifically IP addresses and SPIs) to filter the payload traffic. The middlebox only forwards traffic from and to specific authenticated hosts and drops other traffic.

The feasibility of subverting the function of the middlebox depends on the assumed adversary model.

4.1.1. Adversary model and Security Services

If we assume a Dolev-Yao threat model, attribute binding is not helpful to aid packet filtering for access control. An attacker can send packets from any IP address and can read packets destined to any IP address. Without per packet verification by the middlebox, such an attacker can inject arbitrary forged packets into the HIP payload channel and make them traverse the middlebox. The attacker can also read the packets from the HIP payload channel, and hence, communicate across the middlebox. However, the forged packets are disclosed by inconsistencies in the ESP sequence numbers, which makes the attack visible to the middlebox as well as the HIP end hosts. Moreover, attackers can only inject packets into an already established HIP payload channel. Opening a new payload channel and replaying a closing of the channel are not possible.

An attacker that is not able to send IP packets from an arbitrary source address and receive IP packets addressed to any destination, cannot use the ESP channel to send fake ESP packets when the middleboxes bind HIs and SPI numbers to addresses. By fixing the set of source and destination IP addresses, the opportunity to successfully inject packets into the payload channel is limited to hosts that can send packets from the same source address as the legitimate HIP hosts. Moreover, an attacker can only receive injected packets if it is on the communication path towards the legitimate HIP peer. Attackers cannot open new HIP payload channels and thus have no influence on the bound payload stream parameters.

Finally, attackers cannot close HIP associations of legitimate peers.

4.2. Resource allocation

When using HIs to limit the resources (e.g. bandwidth) allocated for a certain host, the HIs can be used to authenticate the hosts in a similar fashion to the access control illustrated above. Regarding authentication, both use cases share the same strengths and weaknesses. However, the implications for the targeted scenarios differ. Therefore, we restrict the following discussion to these differences.

4.2.1. Adversary Model and Security Services

When assuming an Dolev-Yao threat model, an attacker is able to use resources allocated for the payload channel of another host by injecting packets into this channel. Also, the attacker cannot open a new payload channel with another host nor can it close an existing one.

When binding the IP addresses of the HIP payload channel to the IP addresses used in the HIP control channel and assuming an attacker is unable to receive IP packets addressed to the IP address of an authenticated host, the attacker cannot utilize the resources allocated to authenticated host. However, the attacker can still inject packets and waste resources, yet without having any benefit other than causing disturbance to the other host. Specifically, it cannot increase the share of resources allocated to itself. Hence, this measure takes incentive from selfish users that try to benefit by mounting a DoS attack. Defense against purely malicious attackers that aim at creating disturbance without immediate benefit is difficult to achieve and out of scope of this document.

5. Security Considerations

This HIP extension specifies how HIP-aware middleboxes interact with the handshake and mobility-signaling of the Host Identity Protocol. The scope is restricted to the authentication of end-hosts and excludes the issue of stronger authentication of ESP traffic at the middlebox.

Providing middleboxes with a way of adding puzzles to the HIP control packets may cause both HIP peers, including the Responder, to spend CPU time on solving these puzzles. Thus, it is advised that HIP implementations for servers employ mechanisms to prevent middlebox puzzles from being used as DoS attacks. Under high CPU load, servers can rate limit or assign lower priority to packets containing middlebox puzzles.

6. IANA Considerations

This document specifies two new HIP parameter types. The preliminary parameter type numbers are 322 and 65334.

7. Acknowledgments

Thanks to Thomas Jansen, Shaohui Li, and Janne Lindqvist for the fruitful discussions on this topic. Many thanks to Julien Laganier, Stefan Goetz, Ari Keranen, Samu Varjonen, and Kate Harrison for commenting and helping to improve the quality of this document.

8. Changelog

8.1. Version 4

- Some clarifications.

- Add new way to compute single solution for multiple CHALLENGE_REQUEST parameters.

- Modify parameter layout for CHALLENGE_RESPONSE parameter.

- Add middlebox authentication for the CLOSE exchange.

- Updated outdated references.

8.2. Version 3

- Some editorial changes.

- Added text about space issues in response packets with too many CHALLENGE_RESPONSE parameters in Section Section 2.1.2

9. Normative References

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

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

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

[RFC5202] Jokela, P., Moskowitz, R., and P. Nikander, "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)", RFC 5202, April 2008.

[RFC5203] Laganier, J., Koponen, T., and L. Eggert, "Host Identity Protocol (HIP) Registration Extension", RFC 5203, April 2008.

[RFC5206] Nikander, P., Henderson, T., Vogt, C., and J. Arkko, "End-Host Mobility and Multihoming with the Host Identity Protocol", RFC 5206, April 2008.

[RFC5207] Stiemerling, M., Quittek, J., and L. Eggert, "NAT and Firewall Traversal Issues of Host Identity Protocol (HIP) Communication", RFC 5207, April 2008.

[RFC5770] Komu, M., Henderson, T., Tschofenig, H., Melen, J., and A. Keranen, "Basic Host Identity Protocol (HIP) Extensions for Traversal of Network Address Translators", RFC 5770, April 2010.

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

Authors’ Addresses

Tobias Heer (editor)
RWTH Aachen University, Communication and Distributed Systems Group
Ahornstrasse 55
Aachen 52062
Germany

Email: heer@cs.rwth-aachen.de
URI: http://www.comsys.rwth-aachen.de/team/tobias-heer/

Rene Hummen
RWTH Aachen University, Communication and Distributed Systems Group
Ahornstrasse 55
Aachen 52062
Germany

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

Klaus Wehrle
RWTH Aachen University, Communication and Distributed Systems Group
Ahornstrasse 55
Aachen 52062
Germany

Email: heer@cs.rwth-aachen.de
URI: http://www.comsys.rwth-aachen.de/team/klaus/

Miika Komu
Aalto University, Department of Computer Science and Engineering
Konemiehentie 2
Espoo
Finland

Phone: +358947027117
Fax: +358947025014
Email: miika@iki.fi
URI: http://www.hiit.fi/

HIP Research Group
Internet Draft
Intended status: Experimental

Expires: April 2012

Pascal Urien
Telecom ParisTech
Gyu Myoung Lee
Telecom SudParis
Guy Pujolle
LIP6
October 2011

HIP support for RFIDs
draft-irtf-hiprg-rfid-04

Abstract

This document describes an architecture based on the Host Identity Protocol (HIP), for active RFIDs, i.e. Radio Frequency Identifiers including tamper resistant computing resources, as specified for example in the ISO 14443 or 15693 standards. HIP-RFIDs never expose their identity in clear text, but hide this value (typically an EPC-Code) by a particular equation that can be only solved by a dedicated entity, referred as the portal. HIP exchanges occur between HIP-RFIDs and portals; they are transported by IP packets, through the Internet cloud.

Requirements Language

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

Status of this Memo

Table of Contents

Abstract.. 1
Requirements Language... 1
Status of this Memo... 1
Copyright Notice.. 2
Table of Contents... 3
1 Overview.. 5
   1.1 Motivation... 5
   1.2 Passive and active RFIDs................................. 5
   1.3 About the Internet of Things (IoT)....................... 6
   1.4 HIP-RFIDs.. 6
   1.5 Main differences between HIP-RFID and HIP................ 7
2. Basic Exchange... 8
   2.1 I1-T... 9
   2.2 R1-T... 9
   2.3 I2-T... 9
   2.4 R2-T.. 10
   2.5 HIT format.. 10
   2.6 State Machine... 11
       2.6.1 Unassociated. 11
       2.6.2 I1-Sent 11
       2.6.3 R1-Sent 11
       2.6.4 I2-Sent 11
       2.6.5 R2-Sent 11
       2.6.6 Established 11
3. Formats... 12
   3.1 Payload... 12
   3.2 Packet types.. 13
   3.3 Summary of HIP parameters............................... 14
   3.4 R-T... 14
   3.5 HIP-T-Transform... 15
   3.6 F-T... 15
   3.7 MAC-T... 16
   3.8 ESP-Transform... 16
   3.9 ESP-Info.. 16
4. BEX Example... 17
   4.1 Generic example... 17
       4.1.1 I1-T 17
       4.1.2 R1-T 17
       4.1.3 I2-T 18
       4.1.4 R2-T 19
   4.2 HIP-T Transform 0x0001, HMAC............................ 19
       4.2.1 I1-T 19
       4.2.2 R1-T 19
       4.2.3 I2-T 20
5. HIP-T-Transforms Definition................................. 20
   5.1 Type 0x0001, HMAC....................................... 20
       5.1.1 Suite-ID 20
       5.1.2 F-T computing (f function) 20
       5.1.3 K-Auth-Key computing (g function) 21

```
      5.1.4 MAC-T computing ..................................... 21
   5.2 Type 0x0002, Keys-Tree.................................... 21
      5.2.1 Suite-ID ................................................ 21
      5.2.2 F-T computing (f function) .......................... 21
      5.2.3 K-Auth-Key computing (g function) .................. 22
      5.2.4 MAC-T computing ..................................... 22
6. Security Considerations.......................................... 22
7. IANA Considerations.............................................. 23
8 References........................................................ 23
   8.1 Normative references........................................ 23
   8.2 Informative references...................................... 24
9 Annex I........................................................... 24
   9.1 Binary Interface with HIP RFIDs............................. 24
   9.3 Exchanged data.............................................. 25
   9.3 Javacard code sample........................................ 26
Author’s Addresses.................................................. 31
```

1 Overview

1.1 Motivation

RFIDs are electronic devices, associated to things or computers, which transmit their identifier (usually a serial number) via radio links. The Host Identity Protocol [HIP] is a security protocol based on the use of cryptographic identifiers, and specified for IP-based networks [HIP].

The first motivation for designing HIP support for RFIDs is to enforce a strong privacy for the Internet of Things, e.g. identity is protected by cryptographic procedures compatible with RFID computing resources. As an illustration, EPC codes or IP addresses are today transmitted in the clear.

The second motivation is to define an identity layer for RFIDs logically independent from the transport facilities, which may optionally support IP stacks.

In other words, we believe that the Internet of Things will be Identity oriented; RFIDs will act as electronic ID for objects to which they are linked. In this context, privacy is a major challenge.

1.2 Passive and active RFIDs

An RFID is a slice of silicon whose area is about 1 mm2 for components used as cheap electronic RFIDs, and around 25 mm2 for chips like contact-less smart cards inserted in passports and mobile phones.

RFIDs are divided into two classes, the first includes devices that embed CPU and memory (RAM, ROM, E2PROM) such as contact-less smart cards, and the second comprises electronic chips based on cabled logic circuits.

There are multiple standards relative to RFIDs. The ISO 14443 standard introduces components dealing with the 13.56 MHz frequency that embed a CPU and consume about 10mW; data throughput is about 100 Kbits/s and the maximum working distance (from the reader) is around 10cm.

The ISO 15693 standard also uses the same 13.56 MHz frequency, but enables working distances as high as one meter, with a data throughput of a few Kbits/s.

The ISO 18000 standard defines parameters for air interface communications associated with frequency such as 135 KHz, 13.56 MHz, 2.45 GHz, 5.8 GHz, 860 to 960 MHz and 433 MHz. The ISO 18000-6 standard uses the 860-960 MHz range and is the basis for the Class-1

Generation-2 UHF RFID, introduced by the EPCglobal [EPCGLOBAL] consortium.

1.3 About the Internet of Things (IoT)

The term "Internet of Thing (IoT)" was invented by the MIT Auto-ID Center, in 2001, and refers to an architecture that comprises four levels,

- Passive RFIDs, such as Class-1 Generation-2 UHF RFIDs, introduced by the EPC Global consortium and operating in the 860-960 MHz range.

- Readers plugged to a local (computing) system, which read the Electronic Product Code [EPC].

- A local system, offering IP connectivity, which collects information pointed by the EPC thanks to a protocol called Object Naming Service (ONS)

- EPCIS (EPC Information Services) servers, which process incoming ONS requests and returns PML (Physical Markup Language) files [PML], e.g. XML documents that carry meaningful information linked to RFIDs.

1.4 HIP-RFIDs

```
             PORTAL                    READER              RFID

   +-----------------------+
   !                       !                           +-----------+
   !               +-----+ !                           ! +-------+ !
   !  +---------+  + HIP + !<=========================>! +  HIP  + !
   !  + IDENTITY+  +-----+ !   +-------------------+   ! +-------+ !
   !  + SOLVER  +    [HEP] !<=>! [HEP]             !   !     |     !
   !  +---------+  +-----+ !   ! +------+-------+  !   ! +-------+ !
   !               +     + !   ! +      + RFID  +  !   ! + RFID  + !
   !    EPC-Code   + IP  + !<=>! +  IP  + Radio +  !<=>! + Radio + !
   !               +     + !   ! +      + Ptcol +  !   ! + Ptcol + !
   !               +-----+ !   ! +------+-------+  !   ! +-------+ !
   !                       !   !                   !   !           !
   +----------+------------+   +-------------------+   +-----------+
              !
              V
        TO EPC GLOBAL
           SERVICES
```

Figure 1. HIP-RFID Architecture

This document suggests embedding a modified version of a HIP-enabled stack in active RFIDs, named HIP-RFIDs. It assumes that such devices would not support an IP stack, but should be rather identity oriented, i.e. will use readers' IP resources in order to unveil their EPC-Code only to trusted entities (called portals in the architecture shown by Figure 1). Privacy, e.g. identity protection seems a key prerequisite [SEC] before the effective massive deployment of these devices.

The HIP-RFID architecture includes three functional entities: HIP RFIDs, RFID readers, and portals, and defines a new HIP encapsulation protocol (HEP):

- HIP RFIDs. HIP, as defined in [HIP], is transported by IP packets. HIP-RFIDs support a modified version of this protocol but do not require end-to-end IP transport.

- RFID readers. These provide IP connectivity and communicate with RFIDs through radio links either defined by EPC Global or ISO standards. The IP layer transports HIP messages between RFIDs and other HIP entities. According to HIP, an SPI (Security Parameter Index) associated to an IPsec tunnel MAY be used by the IP host (e.g. a reader) in order to route HIP packets to/from the right software identity.

- HEP, HIP Encapsulation Protocol. HIP messages MAY be encapsulated by protocols such as UDP or TCP in order to facilitate HIP transport in existing software and networking architectures. The HEP does not modify the content of an HIP packet. This class of protocol is not specified by this document.

- PORTAL entity. This device manages a set of readers; it is a HIP entity that includes a full IP stack. Communications between portal and RFIDs logically work as peer to peer HIP exchanges. RFID identifier (HIT) is hidden and appears as a pseudo random value; within the portal a software block called the IDENTITY SOLVER resolves an equation f, whose solution is an EPC Code. The portal accesses EPCIS services; when required privacy may be enforced by legacy protocol such as SSL or IPsec.

- The portal maintains a table linking HIT and EPC-Code. It acts as a router for that purpose it MUST provide an identity resolution mechanism, i.e. a relation between HIT and EPC-Code.

1.5 Main differences between HIP-RFID and HIP

In HIP [HIP], the HIT (Host Identifier Tag) is a fixed value obtained from the hash of an RSA public key. This parameter is therefore linked to a unique identity, and can be used for traceability purposes; in other words HIP does not natively include privacy features.

In [BLIND], it is proposed to hide the HIT with a random number thanks to a hash function, i.e.

B-HIT = sha1(HIT || N), with N a random value and || the concatenation operation.

The case in which only one HIT (either initiator or responder) is blinded looks similar to the HIP-RFID protocol described in this draft working with a particular transform (HMAC Transform, 0x0001).

2. Basic Exchange

The HIP-RFID base exchange (T-BEX) is derived from the "classical" base exchange (BEX), introduced in [HIP]. It is a four way handshake illustrated by Figure 2.

```
   RFID             READER                                   PORTAL
   --+--            --+--                                    ---+---
     !     START      !                                         !
     !<---------------!                                         !
     !                !                                         !
     !  I1-T                                                    !
     !  HIT-I  HIT-R                                            !
     ! ---------------------------------------------------->    !
     !                                                          !
     !                                                          !
     !  R1-T                                                    !
     !  HIT-I  HIT-R  R-T(r1) HIP-T-Transforms                  !
     !  [*ESP-Transforms]                                       !
     ! <----------------------------------------------------    !
     !                                                          !
     !                                                          !
     !  I2-T                                                    !
     !  HIT-I HIT-R HIP-T-Transform [*ESP-Transform] R-T(r2)    !
     !  F-T=f(r1, r2, EPC-Code) [*ESP-Info] MAC-T               !
     ! ---------------------------------------------------->    !
     !                                                          !
     !                                                          !
     !  R2-T                                                    !
     !  HIT-I HIT-R  [*ESP-Info]  MAC-T                         !
     ! <----------------------------------------------------    !
     !                                                          !
     !                                                          !
     !                  Optional ESP Dialog                     !
     ! <--------------------------------------------------->    !
     !                                                          !
     !                                                          !
```

Figure 2. HIP-RFIDs Base Exchange (T-BEX), *means optional attributes

A HEP layer MAY be used to transport HIP messages in a non-IP context, but this optional facility is out of scope for this document.

2.1 I1-T

When a reader detects an RFID, it realizes all low level operations in order to set up a radio communication link. Finally the reader delivers a START message that triggers the RFID.

The HIP-RFID sends the I1-T packet (I suffix meaning initiator), in which HIT-I is a pseudorandom value internally generated by the HIP-RFID.

If the RFID doesn’t known the portal HIT it sets the HIT-R value to zero; in that case the reader MAY modify this field in order to identify the appropriate entity.

The I1-T message is not MACed.

2.2 R1-T

The portal produces the R1-T (R suffix meaning responder) packet, which includes a nonce r1 and optional parameters. These fields indicate a list of supported authentication schemes (HIP-T-TRANSFORMs) and a list of ESP-TRANSFORMs, i.e. secure channels that could be opened between portal and RFIDs.

This message includes the following fields:
- HIT-I, a random number which identifies a RFID
- HIT-R, the portal HIP, either a null or fixed value.
- HIT-T-TRANSFORMs, a list of authentication schemes
- ESP-T-TRANSFORMs, an optional list of ESP secure channels

The R1-T message is not MACed.

2.3 I2-T

The HIP-RFID builds the I2-T message, which contains

- The selected HIP-T-TRANSFORM (the current authentication scheme).
- An optional ESP-TRANSFORM (a class of secure channel between RFID and portal).
- A nonce r2, included in the R-T attribute.
- An equation f(r1, r2, EPC-Code), whose solution, according to the selected HIP-T-TRANSFORM, unveils the EPC-Code value.
- An optional ESP-Info attribute that gives information about the secure (ESP) channel, and which includes the SPI-I value.
- A keyed MAC (MAC-T), which works with a KI-Auth-key deduced from r1, r2 and the hidden EPC-Code value.

KI-Auth-key = g(r1, r2, EPC-Code)

The keyed MAC is by default computed over the complete I2-T message, the content of MAC-T resulting from this calculation is initially set to a null value. Particular HIP-T-TRANSFORMs MAY work with different rules (see section 6).

The portal and the RFID shares secret keys. The meaning of these keys are dependent upon the f equation.

In some cases the EPC-Code is the only shared key. The portal knows a list of EPC-Code and tries all solutions for solving f, according to brute force techniques. As an illustration a hash function may be used for f:

f= sha1(r1 || r2 || EPC-Code), where || is the concatenation operation.

In other cases a set of keys is shared between portal and RFIDs. For example a binary tree of HMAC procedure MAY be used, each HMAC beeing associated to a particular key. A binary tree of depth n may identify 2**n RFIDs, each of them stores n keys (ki:j). The f function is a list of n values such as

HMAC(r1 || r2, ki:j)

Where ki:j is a secret key, and j the bit value (either 0 or 1) at the rank i (ranging between 0 and n-1) for the EPC-Code (or the RFID index).

2.4 R2-T

The fourth and last R2-T packet is optional. It includes

- A keyed MAC (MAC-T) computed with the KI-Auth-key deduced from r1, r2 and the hidden EPC-Code value.

KI-Auth-key = g(r1, r2, EPC-Code)

- An optional ESP-Info attribute that gives information about the secure (ESP) channel, and which includes the SPI-R value.

The R2-T packet is mandatory when an ESP channel has been previously negotiated. ESP channel is required if the portal intends to perform read or write operations with the RFIDs.

2.5 HIT format

HIT-R MAY be a fixed value embedded in the RFID during the manufacturing process or a null value if no specific portal is required.

HIT-I MAY comprise an optional header given coded according to various hierarchical rules and MUST include a trailer, which is a true random number.

2.6 State Machine

The state machine is similar to the one described in [RFC 5201]. No retry operations are performed, because the communication with the RFID may be lost at any time. Furthermore RFIDs are generally not equipped with timers.

2.6.1 Unassociated.

The state machine starts.

2.6.2 I1-Sent

The RFID has been reset by the reader, and has sent the I1-T message.

2.6.3 R1-Sent

The responder has received the I1-T message and has sent the R1-T packet.

2.6.4 I2-Sent

The RFID has received the R1-T packet, and has sent the I2-T message.

2.6.5 R2-Sent

The responder has received the I2-T message and has sent the optional R2-T packet.

2.6.6 Established

The RFID has received the R2-T message. A secure channel is established.

3. Formats

3.1 Payload

The payload format is imported from the [HIP] specification.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | Next Header   | Header Length |0| Packet Type |  VER. | RES.|1|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Checksum             |           Controls            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                Sender’s Host Identity RFID (HIT)              |
    |                                                               |
    |                                                               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                Receiver’s Host Identity RFID (HIT)            |
    |                                                               |
    |                                                               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    /                        HIP Parameters                         /
    /                                                               /
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header : normal value is decimal 59, IPPROTO_NONE.

Header Length: the length of the HIP Header and HIP parameters in 8 bytes units, excluding the first 8 bytes

Packet Type: Detailed in section 4.2

VER: 0001

RES: 000

Checksum: This checksum covers the source and destination addresses in the IP header.

HIP-RFIDs always deliver HIP packets with the null value for the checksum field. The reader MUST compute the checksum.

HIP-RFIDs do not check the checksum of received packets.

Controls: this field is reserved for future use (RFU)

Sender’s Host Identity RFID: 16 bytes HIT

Receiver’s Host Identity RFID: 16 bytes HIT

HIP Parameters: a list of attributes encoded in the TLV format

3.2 Packet types

Packet type	Packet name
0x40	I1-T - The HIP-RFID Initiator Packet
0x41	R1-T - The HIP-RFID Responder Packet
0x42	I2-T - The Second HIP-RFID Initiator Packet
0x43	R2-T - The Second HIP-RFID Responder Packet

3.3 Summary of HIP parameters

TLV	Type	Length	Data
R-T	0x400	variable	Random value r1 or r2
HIP-T-TRANSFORM	0x402	variable	HIP-RFID transform(s)
F-T	0x404	variable	f function value
MAC-T	0x406	variable	Keyed MAC
ESP-Transform	0x408	variable	ESP transform(s)
ESP-Info	0x40A	variable	ESP parameter(s)

3.4 R-T

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Padding-Length          |              value            /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/         value                 |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	0x400
Length	total length in bytes
Value	random value
Padding-Length	padding length in bytes
Padding	padding bytes

3.5 HIP-T-Transform

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Padding-Length          |            Suite-ID#1         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+     Length-of-Suite-ID#1      |              value            +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/           value               |            Suite-ID#2         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	0x402
Length	Total length
Padding-Length	Number of padding bytes
Suite-ID	Defines the HIP Cipher Suite to be used
Length-of-Suite-ID	Defines the length of optional data
Padding	Padding bytes

3.6 F-T

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Padding-Length        |            value              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |            Padding            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	0x404
Length	total length, in bytes
Padding-Length	padding length in bytes
Value	the f value with a variable length
Padding	padding bytes

3.7 MAC-T

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Padding-Length        |               MAC             /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                               |             Padding           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	0x406
Length	total length, in bytes
Padding-Length	padding length, in bytes
Value	Keyed MAC value
Padding	padding bytes

A MAC procedure works with the K-Auth-Key and is computed over the whole HIP message according to the following rules

- The checksum field of the HIP header is set to a null value.

- The MAC field of the MAC-T attribute is set to a null value

3.8 ESP-Transform

Details of the attribute will be specified by another document.

3.9 ESP-Info

Details of the attribute will be specified by another document.

4. BEX Example

4.1 Generic example

4.1.1 I1-T

```
Next Header:                    0x3B
Header Length:                  0x4
Packet Type:                    0x40
Version:                        0x1
Reserved:                       0x1
Control:                        0x0
Checksum:                       0x0000
Sender’s HIT (RFID) :           0x0123456789ABCDEF
                                  0123456789ABCDEF
Receiver’s HIT (Portal) :       0x0000000000000000
                                  0000000000000000
```

The checksum is computed by portal and reader according to rules specified in [HIP]; it covers the source and destination IP addresses.

4.1.2 R1-T

```
Next Header:                    0x3B
Header Length:                  0xB
Packet Type:                    0x41
Version:                        0x1
Reserved:                       0x1
Control:                        0x0
Checksum:                       0xabcd
Sender’s HIT (Portal)           0xA5A5A5A5A5A5A5A5
                                  5A5A5A5A5A5A5A5A
Receiver’s HIT (RFID)           0x0123456789ABCDEF
                                  0123456789ABCDEF
R-T                             0x040000280002rrrr
                                  rrrrrrrrrrrrrrrr
                                  rrrrrrrrrrrrrrrr
                                  rrrrrrrrrrrrrrrr
                                  rrrrrrrrrrrrpppp
HIP-T-Transforms                0x0402001000020001
                                  000000020000pppp
```

r1 is a 128 bits value
Transforms 1, 2 are supported by the reader.

4.1.3 I2-T

```
       Next Header:                    0x3B
       Header Length:                  0x14
       Packet Type:                    0x42
       Version:                        0x1
       Reserved:                       0x1
       Control:                        0x0
       Checksum:                       0x0000
       Sender’s HIT (RFID) :           0x0123456789ABCDEF
                                         0123456789ABCDEF
       Sender’s HIT (Portal) :         0xA5A5A5A5A5A5A5A5
                                         5A5A5A5A5A5A5A5A
       HIP-T-Transform                 0x0402001000060001
                                         0000pppppppppppp
       R-T                             0x040000280002rrrr
                                         rrrrrrrrrrrrrrrr
                                         rrrrrrrrrrrrrrrr
                                         rrrrrrrrrrrrrrrr
                                         rrrrrrrrrrrrpppp
       F-T                             0x040400280002ffff
                                         ffffffffffffffff
                                         ffffffffffffffff
                                         ffffffffffffffff
                                         ffffffffffffpppp
       MAC-T                           0x040600040006ssss
                                         ssssssssssssssss
                                         ssssssssssssssss
                                         sssspppppppppppp
```

The RFID selects the HIP-Transform number one. It produces an r2 nonce and computes a f value. It appends a 20 bytes keyed MAC.

4.1.4 R2-T

Next Header:	0x3B
Header Length:	0x08
Packet Type:	0x40
Version:	0x1
Reserved:	0x1
Control:	0x0
Checksum:	0xabcd
Sender’s HIT (RFID) :	0x0123456789ABCDEF 0123456789ABCDEF
Sender’s HIT (Portal) :	0xA5A5A5A5A5A5A5A5 5A5A5A5A5A5A5A5A
MAC-T	0x040600040006ssss ssssssssssssssss ssssssssssssssss sssspppppppppppp

Reader ends the BEX-T.

4.2 HIP-T Transform 0x0001, HMAC

EPC = 0123456789abcdefcdab

4.2.1 I1-T

```
<< 3B 04 40 11 00 00 00 00 6A 68 2E 53 51 6B 51 6F
   2F 58 CE 60 25 42 1A E6 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00

HEAD 3b04401100000000
sHIT 6a682e53516b516f2f58ce6025421ae6
dHIT 00000000000000000000000000000000
```

4.2.2 R1-T

```
>> 3B 0A 41 11 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 6A 68 2E 53 51 6B 51 6F
   2F 58 CE 60 25 42 1A E6 04 00 00 20 00 06 27 6D
   03 4D DD 2D 52 79 3B 17 2C B9 5B CD 02 97 E2 DF
   61 15 00 00 00 00 00 00 04 02 00 10 00 06 00 02
   00 00 00 00 00 00 00 00

HEAD 3b0a411100000000
sHIT 00000000000000000000000000000000
dHIT 6a682e53516b516f2f58ce6025421ae6

ATT 0400 20 bytes  276d034ddd2d52793b172cb95bcd0297e2df6115
ATT 0402 04 bytes  00020000
```

4.2.3 I2-T

```
<< 3B 13 40 11 00 00 00 00 6A 68 2E 53 51 6B 51 6F
   2F 58 CE 60 25 42 1A E6 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 04 02 00 10 00 06 00 01
   00 00 00 00 00 00 00 00 04 00 00 20 00 06 C5 95
   8B 23 6B 9B 0E AA 7A BB 25 F2 7D 24 C5 04 6E 89
   19 9E 00 00 00 00 00 00 04 04 00 20 00 06 80 1D
   BC 55 C5 F3 97 89 F8 3C 6C BA 14 50 18 7D 83 83
   3C AF 00 00 00 00 00 00 04 06 00 20 00 06 2A 23
   68 93 2B F7 3A BE C4 6B DD B8 3F 1B 3F 7F 9D ED
   8B 83 00 00 00 00 00 00

HEAD 3b13401100000000
sHIT 6a682e53516b516f2f58ce6025421ae6
dHIT 00000000000000000000000000000000

ATT 0402 04 bytes  00010000
ATT 0400 20 bytes  c5958b236b9b0eaa7abb25f27d24c5046e89199e
ATT 0404 20 bytes  801dbc55c5f39789f83c6cba1450187d83833caf
ATT 0406 20 bytes  2a2368932bf73abec46bddb83f1b3f7f9ded8b83
```

5. HIP-T-Transforms Definition

5.1 Type 0x0001, HMAC

5.1.1 Suite-ID

```
Suite-ID:            0x0001
Length-of-Suite-ID: 0x0000
```

5.1.2 F-T computing (f function)

The F-T function produces a 20 bytes result, according to the relation:

K = HMAC-SHA1(r1 | r2, EPC-Code)

Y = f(r1, r2, EPC-Code) = HMAC-SHA1(K, CT1 | "Type 0001 key")

Where:

- SHA1 is the SHA1 digest function

- EPC-Code is the RFID identity

- HMAC-SHA1 is the keyed MAC algorithm based on the SHA1 digest procedure.

- CT1 is a 32 bits string, whose value is equal to 0x00000001

- r1 and r2 are the two random values exchanged by the BEX

5.1.3 K-Auth-Key computing (g function)

The K-Auth-Key is computing according to the relation:

K = HMAC-SHA1(r1 | r2, EPC-Code)

Y = HMAC-SHA1(K, CT2 | "Type 0001 key")

Where:

- SHA1 is the SHA1 digest function

- EPC-Code is the RFID identity

- HMAC-SHA1 is the keyed MAC algorithm based on the SHA1 digest procedure.

- CT2 is a 32 bits string, whose value is equal to 0x00000002

- r1 and r2 are the two random values exchanged by the BEX

5.1.4 MAC-T computing

The HMAC-SHA1 function is used with the K-Auth-Key secret value:

MAC-T(HIT-T packet) = HMAC-SHA1(K-Auth-Key, HIP-T packet)

5.2 Type 0x0002, Keys-Tree

5.2.1 Suite-ID

```
Suite-ID:           0x0002
Length-of-Suite-ID: 0x0006
Value1: an index, a two bytes number, identifying a HASH function
(H), which produces h bytes.
Value2: n, the depth of the tree, a two bytes number.
Value3: p, the maximum number of child nodes, for each node, a two
bytes number.
```

The maximum elements of a keys-tree is therefore p**n

5.2.2 F-T computing (f function)

The F-T function produces a list of Hi, 1<= i <= n, of nh bytes results, according to the relation:

Y = f(r1, r2, EPC-Code) = H1 | H2 | Hi | Hn

With
Hi = HMAC-SHA1(r1 | r2, Ki:j)

Where:

- H is digest function producing t bytes

- Ki:j is a set of pn secret keys.

Each EPC-Code is associated with an index, whose value is written as:

RFID-Index = an p**(n-1) + an-1 p**(n-2) + a1

Each ai digit(ai p**(i-1))whose value ranges between 0 and p-1, is associated with a key Ki:j (i.e. the tree is made with pn keys, but only n values are stored in a given RFID), with j=ai

- HMAC-H is the keyed MAC algorithm based on the H digest procedure.

- r1 and r2 are the two random values exchanged by the BEX.

5.2.3 K-Auth-Key computing (g function)

The K-Auth-Key is computing according to the relation:

K-Auth-Key = HMAC-H(r1 | r2, RFID-Index)

Where:

- H is a digest function producing t bytes

- HMAC-H is the keyed MAC algorithm based on the H digest procedure.

- RFID INDEX is the RFID index.

- r1 and r2 are the two random values exchanged by the BEX.

5.2.4 MAC-T computing

The HMAC-H function is used with the K-Auth-Key secret value:

MAC-T(HIT-T packet) = HMAC-H(K-Auth-Key, HIP-T packet)

6. Security Considerations

In this section we only discuss the case where no ESP channel is negotiated, i.e. a three ways handshake is performed thanks to the I1-T, R1-T and I2-T packets.

The HIP-RFID infrastructure comprises a set readers establishing sessions with a PORTAL. The exchanged packets MUST be protected by secure tunnels such as IPSEC or any appropriate means. Readers feed RFIDs and consequently deliver information about their position. Without security association between readers and PORTALs rogue devices can inject malicious packets such as I1-T and I2-T whose goal is to forward a fake f equation that could not be solved by the IDENTITY-SOLVER entity. This class of attack targets a Denial of Service (DoS) threat; computing resources will be consumed by the PORTAL that will stop its solving process after a given timeout.

Malicious RFIDs can also perform DoS attacks. However upon detection, they could be discarded by their associated reader.

The I1-T packet includes no security feature. It may be forged by any entity.

The R1-T packet includes no security feature. It may be forged by any entity. A rogue portal SHOULD NOT expect to retrieve the HIP-RFID identity thanks to cryptographic weaknesses of the f equation. Nerveless hardware or software implementation of the HIP-RFID protocol MUST be aware that the R1-T packet MUST be carefully parsed and checked.

The I2-T packet includes a pseudo unique value r2, the f equation and is MACed. The MAC field proves this packet integrity and optionally the whole dialog integrity (dealing with I1-T, R1-T and I2-T). Although HIP-T-TRANSFORMs detailed in this document only deal with I2-T integrity, other transforms MAY use different schemes.

The two main classes of the f(r1,r2,EPC-Code) equation are bijections (such as cipher algorithms) and surjections (such as digest procedures). In the first case the solution (EPC-Code) is unique; its correctness is checked via the keyed MAC. In the second case there are multiples solutions, with very low probability of collisions; the correctness of the highly probable solution is checked by the keyed MAC.

7. IANA Considerations

None.

8 References

8.1 Normative references

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

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

8.2 Informative references

[EPC] Brock, D.L, The Electronic Product Code (EPC), A Naming Scheme for Physical Objects, MIT AUTO-ID CENTER, 2001.

[PML] Brock, D.L - The Physical Markup Language, MIT AUTO-ID CENTER, 2001.

[EPCGLOBAL] EPCglobal, EPC Radio Frequency Identity Protocols Class 1 1516 Generation 2 UHF RFID Protocol for Communications at 860 MHz-960 MHz Version 1517 1.0.9, EPCglobal Standard, January 2005.

[NIST-800-108] NIST Special Publication 800-108, Recommendation for Key Derivation Using Pseudorandom Functions.

[SEC] S. Weis, S. Sarma, R. Rivest and D. Engels. "Security and privacy aspects of low-cost radio frequency identification systems" In D. Hutter, G. Muller, W. Stephan and M. Ullman, editors, International Conference on Security in Pervasive Computing - SPC 2003, volume 2802 of Lecture Notes in computer Science, pages 454-469. Springer-Verlag, 2003.

[HIP-TAG-EXP] Pascal Urien, Simon Elrharbi, Dorice Nyamy, Herve Chabanne, Thomas Icart, Francois Lecocq, Cyrille Pepin, Khalifa Toumi, Mathieu Bouet, Guy Pujolle, Patrice Krzanik, Jean-Ferdinand Susini, "HIP-Tags architecture implementation for the Internet of Things", AH-ICI 2009. First Asian Himalayas International Conference on Internet, 3-5 Nov. 2009.

[BLIND] Dacheng Zhang, Miika Komu, "An Extension of HIP Base Exchange to Support Identity Privacy", draft-zhang-hip-privacy-protection-00, work in progress, March 2010.

9 Annex I

This annex provides a sample code, for NFC RFIDs working at 13.56 Mhz and implementing a Java Virtual Machine.

9.1 Binary Interface with HIP RFIDs

According to the ISO 7816 standards, embedded RFID applications are identified by an AID attribute (Application IDentifier) whose size ranges between 5 and 16 bytes.

Commands exchanged between RFIDs and readers are named APDUs and are associated with a short prefix, whose size is usually 5 bytes referred as CLA, INS, P1, P2, P3.

In our sample we choose an arbitrary value for the AID (11223344556601, in hexadecimal representation) and a unique command CLA=00, INS=C2, P1=00, P2=00. The P3 byte is set to null in order to trig the RFID (which resets its state machine and returns the I1 packet, or a non null value when it pushes the R1 packet.

9.3 Exchanged data

The reader selects the embedded HIP-RFID application.

```
>> 00 A4 04 00 07 11 22 33 44 55 66 01
<< 90 00
```

The reader trigs the first packet I1-T.

```
>> 00 C2 00 00 00
```

The RFID delivers the R1-T packet.

```
<< 3B 04 40 11 00 00 00 00 A3 12 9D 5E 28 16 67 4F FC 4F A8 08 4E 30
55 E8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 90 00
```

The reader forwards the R1-T packet to the HIP RFID.

```
>> 00 C2 00 00 58 3B 0A 41 11 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 A3 12 9D 5E 28 16 67 4F FC 4F A8 08 4E 30 55 E8
04 00 00 20 00 06 68 46 95 15 02 10 32 C2 B7 8D 13 E7 53 F6 25 0F 09
AD 7A BD 00 00 00 00 00 00 04 02 00 10 00 06 00 01 00 00 00 00 00 00
00 00
```

The RFID produces the I2-T packet.

```
<< 3B 13 40 11 00 00 00 00 A3 12 9D 5E 28 16 67 4F FC 4F A8 08 4E 30
55 E8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 02 00 10 00
06 00 01 00 00 00 00 00 00 00 00 04 00 00 20 00 06 71 3A DD 19 C4 CB
59 D4 AF D0 2B FD F9 7C 2F 8A D1 23 32 E0 00 00 00 00 00 00 04 04 00
20 00 06 70 DA C1 F7 0B CA 63 15 57 CB D7 AA 66 A9 FD 36 B4 1F DB E3
00 00 00 00 00 00 04 06 00 20 00 06 A6 A7 00 67 5D FD A9 2F 3E 5C 00
D6 B0 8A 55 A2 99 D8 86 79 00 00 00 00 00 00 90 00
```

9.3 Javacard code sample

```
   package hiprfid;

   // Author Pascal Urien

   import  javacard.framework.*;
   import  javacard.security.* ;


   public class rfid extends Applet
   {
    final static byte  SELECT          = (byte)0xA4 ;
    final static byte  INS-HIP         = (byte)0xC2 ;

    final static short R-T             = (short)0x400 ;
    final static short HIP-T-TRANSFORM = (short)0x402 ;
    final static short F-T             = (short)0x404 ;
    final static short Signature-T     = (short)0x406 ;
    final static short ESP-Transform   = (short)0x408 ;
    final static short ESP-Info        = (short)0x40A ;

    final static short ALIGN = 8;
    final static short len-r2 =(short)20;
    final byte[] algo1 = {(byte)0x00,(byte)0x01,(byte)0x00,(byte)0x00 };

    final byte[] ct1 =  {
    (byte)0x00,(byte)0x00,(byte)0x00,(byte)0x01,
    (byte)’T’,(byte)’y’, (byte)’p’,(byte)’e’,
    (byte)’ ’,(byte)’0’,(byte)’0’,(byte)’0’,(byte)’1’,
    (byte)’ ’,(byte)’k’,(byte)’e’,(byte)’y’ };

    final byte[] ct2 =  {
    (byte)0x00,(byte)0x00,(byte)0x00,(byte)0x02,
    (byte)’T’,(byte)’y’,(byte)’p’,(byte)’e’,
    (byte)’ ’,(byte)’0’,(byte)’0’,(byte)’0’,(byte)’1’,
    (byte)’ ’,(byte)’k’,(byte)’e’,(byte)’y’ };

    MessageDigest sha1=null ;
    RandomData rnd=null;
    byte[] DB =null;
    final static short DBSIZE=(short)200;
    final static short off-myHIT = (short)0   ;
    final static short off-rHIT =  (short)16  ;
    final static short off-R1   =  (short)32  ;
    final static short off-R2   =  (short)64  ;
    final static short off-kaut =  (short)96  ;
    final static short off-k    =  (short)128 ;
    final static short off-FT   =  (short)160 ;
```

```
final byte[] HEADER= {
 (byte)0x3b,(byte)0x04,(byte)0x40,(byte)0x11,
 (byte)0x00,(byte)0x00,(byte)0x00,(byte)0x00 };


 final byte[] MyEPCCODE = {
 (byte)0x01,(byte)0x23,(byte)0x45,(byte)0x67,(byte)0x89,
 (byte)0xab,(byte)0xcd,(byte)0xef,(byte)0xcd,(byte)0xab };

 public void init(){
 try { sha1=MessageDigest.getInstance(MessageDigest.ALG-SHA,false);}
 catch (CryptoException e){sha1=null;}

 try { rnd = RandomData.getInstance(RandomData.ALG-SECURE-RANDOM);}
 catch (CryptoException e){rnd=null;}

 DB = JCSystem.makeTransientByteArray(DBSIZE,
                                      JCSystem.CLEAR-ON-DESELECT);

}

public short GetAttOffset(byte[] pkt, short off, short len,short att)
{ boolean more=true;
  short type=(short)0;
  short tl=(short)0;

  if (len <= (short)40) return (short)-1 ;

  while (more)
  { type = Util.getShort(pkt,off)           ;
    tl   = Util.getShort(pkt,(short)(off+2));
    if (type == att) return off    ;
    off =(short)(off+tl) ;
    if (off >= (short)(off+len))more=false;
  }

 return -1;
}


public static short GetPadLength(short size)
{
 if ( (short)(size % ALIGN) == (short)0) return (short)0;
 return (short)(ALIGN - size % ALIGN );
 }


public static short Set_Att(short att, byte[] ref-att, short off-att,
                            short len-att, byte[] pkt, short off)
{
  short tl = (short) (len-att + 6) ;
```

```
    short tp = GetPadLength(tl)      ;

    tl= (short) (tp+tl);

    Util.setShort(pkt,off,att)  ;
    Util.setShort(pkt,(short)(off+2),tl);
    Util.setShort(pkt,(short)(off+4),tp);

    if (ref_att != null)
    Util.arrayCopy(ref-att,off-att,pkt,(short)(off+6),len-att);
    else
    Util.arrayFillNonAtomic(pkt,(short)(off+6),len-att,(byte)0);

    if (tp != (short)0)
    Util.arrayFillNonAtomic(pkt,(short)(off+6+len-att),tp,(byte)0);

    return tl ;
   }


  public void process(APDU apdu) throws ISOException
  {
   short len=(short)0, readCount=(short)0;
   short off=(short)0,pad=(short)0,len-r1=(short)0;
   short size=(short)0;

   byte[] buffer = apdu.getBuffer() ; // CLA INS P1 P2 P3

   byte cla = buffer[ISO7816.OFFSET_CLA];
   byte ins = buffer[ISO7816.OFFSET_INS];
   byte P1  = buffer[ISO7816.OFFSET_P1] ;
   byte P2  = buffer[ISO7816.OFFSET_P2] ;
   byte P3  = buffer[ISO7816.OFFSET_LC] ;

   switch (ins)
   {
    case SELECT:
    size = apdu.setIncomingAndReceive();
    return;

   case INS_HIP:

    if (P3 == (byte)0)
    {
     rnd.generateData(DB,off_myHIT,(short)16);
     Util.arrayCopy(HEADER,(short)0,buffer,(short)0,(short)8);
     Util.arrayCopy(DB,off-myHIT,buffer,(short)8,(short)16)  ;
     Util.arrayFillNonAtomic(DB,(short)24,(short)16,(byte)0) ;
     apdu.setOutgoingAndSend((short)0,(short)40)             ;
     break;
    }
    else
    {
    size = apdu.setIncomingAndReceive();
    len  = Util.makeShort((byte)0,buffer[6]);
    len  = (short)(len << 3);
    len  = (short)(len+(short)8)   ;

    if (len != size) ISOException.throwIt(ISO7816.SW-DATA-INVALID) ;
    size = (short)(len-(short)40);

    // HEADER 00...08
    // HIT-S  08...24
    // HIT-D  24...40

    Util.arrayCopy(buffer,(short)13,DB,off_rHIT,(short)16);
    off= GetAttOffset(buffer,(short)45,size,R-T);
    if (off==(short)-1) ISOException.throwIt(ISO7816.SW-DATA-INVALID) ;
    len = Util.getShort(buffer,(short)(off+2));
    pad = Util.getShort(buffer,(short)(off+4));
    len = (short)(len-pad-6);

    len-r1=len;
    Util.arrayCopy(buffer,(short)(off+6),DB,off-R1,len);
    off= GetAttOffset(buffer,(short)45,size,HIP-T-TRANSFORM)      ;

    if (off==(short)-1) ISOException.throwIt(ISO7816.SW-DATA-INVALID) ;
    len = Util.getShort(buffer,(short)(off+2));
    pad = Util.getShort(buffer,(short)(off+4));
    len = (short)(len-pad-6);

    // algo=Util.getShort(buffer,(short)(off+6)
    rnd.generateData(DB,(short)(off-R1+len-r1),len-r2); // r1 || r2

    Util.arrayCopy(MyEPCCODE,(short)0,buffer,
                             (short)0,(short)MyEPCCODE.length);

    hmac(DB,off_R1,(short)(len-r1 + len-r2),
         buffer,(short)0,(short)MyEPCCODE.length,
         sha1,
         DB,off-k);

    Util.arrayCopy(ct1,(short)0,buffer,(short)0,(short)ct1.length);

    hmac(DB,off_k,(short)20,
         buffer,(short)0,(short)ct1.length,
         sha1,
         DB, off-FT);

    Util.arrayCopy(ct2,(short)0,buffer,(short)0,(short)ct2.length);
```

```
  hmac(DB,off-k,(short)20,
       buffer,(short)0,(short)ct2.length,
       sha1,
       DB, off-kaut);

  Util.arrayCopy(HEADER,(short)0,buffer,
                (short)0,(short)HEADER.length);

  Util.arrayCopy(DB,off-myHIT, buffer, (short)8,(short)16);
  Util.arrayCopy(DB, off-rHIT, buffer,(short)24,(short)16);

  off=(short)40;
  len = Set-Att(HIP-T-TRANSFORM,algo1,
                (short)0,(short)algo1.length,buffer,off);
  off = (short)(off+len);
  len = Set-Att(R-T,DB,(short)(off-R1+len-r1),len-r2,buffer,off);
  off = (short)(off+len);
  len = Set-Att(F-T,DB,off-FT,(short)20,buffer,off);
  off = (short)(off+len);
  len = Set-Att(Signature-T,null,(short)0,(short)20,buffer,off);
  size= (short)(off+len);
  buffer[1] = (byte) (size >>3);

 hmac(DB,off-kaut,(short)20,
      buffer,(short)0,size,
      sha1,
      buffer,(short)(off+6));

apdu.setOutgoingAndSend((short)0,size);
  break;
}

 default:
 ISOException.throwIt(ISO7816.SW-INS-NOT-SUPPORTED);
  }

}

protected rfid(byte[] bArray,short bOffset,byte bLength)
{init();
 register();
}

public static void install( byte[] bArray, short bOffset, byte
bLength )
{
new rfid(bArray,bOffset,bLength);
}
```

```
   public boolean select()
   {
   return true;
   }

   public void deselect()
   {
   }
```

Author’s Addresses

Pascal Urien
Telecom ParisTech
23 avenue d’italie, 75013 Paris, France

Email: Pascal.Urien@telecom-paristech.fr

Gyu Myoung Lee
Telecom SudParis
9 rue Charles Fourier, 91011 Evry, France

Email: gm.lee@it-sudparis.eu

Guy Pujolle
Laboratoire d’informatique de Paris 6 (LIP6)
4 place Jussieu
75005 Paris France

Email: Guy.Pujolle@lip6.fr