Network Working Group J. Ylitalo
Internet-Draft V. Torvinen
Expires: November 30, 2004 Ericsson Research Nomadiclab
E. Nordmark
Sun Microsystems, Inc.
June 2004

Weak Identifier Multihoming Protocol Framework (WIMP-F)
draft-ylitalo-multi6-wimp-01

Status of this Memo

Copyright Notice

Abstract

Weak Identifier Multihoming Protocol Framework (WIMP-F) is a wedge layer 3.5 framework to be applied with different kind of routable application layer identifiers (AIDs) and layer 3.5 context identifiers (CIDs) presented in Group-F. WIMP-F consists of context establishment and re-addressing exchanges that are protected with one-way hash chains and a technique called as secret splitting. The hash chain protects a host from re-direction attacks, but not

directly from an CID or AID theft. The ownerships can be provided in variable ways presented in other Multi6 drafts.

Table of Contents

1. Introduction

The reason to name this draft as WIMP-F is that it relies on the same one-way hash chain procedures as the initial version of WIMP. However, this draft does not define new application layer identifiers (AIDs). The WIMP-F exchanges are designed to support any kind of 128-bits AIDs and layer 3.5 context identifiers (CIDs). By default, the ownership of AIDs can be proved by the reachability test implemented by the context establishment exchange. A separate ephemeral CID layer 3.5 namespace can be used to protect host from CID theft.

WIMP-F offers a context establishment and locator update exchanges for other Group-F identifiers. It protects hosts from re-direction attacks by supporting one-way hash chains. Other Group-F protocols applying WIMP-F are responsible for defining the properties of AIDs and CIDs

WIMP-F defines a new service at the IP layer rather than a new layer in the stack. It assumes that AIDs are routable from their nature, and there is one-to-many binding between the AIDs and the locators. In other words, a single AID is bound to a IP layer locator set, but it still has a locator role also itself. In addition, the AID may also have cryptographical properties. If the AIDs are not cryptographical from their nature, it may be possible to apply the same framework also with IPv4 hosts. Therefore, the message structures are designed to support both IPv6 and IPv4 hosts.

WIMP-F can be used to gradually update a legacy TCP connection to Multi6 enabled connection depending on the Group-F design. Furthermore, if WIMP-F protocol finds out that the responding party has already a state for context identifier -pair, the initiating party may change its context identifier or prove the identifier ownership using some strong authentication mechanism, depending on the type of context identifier.

2. Notational Conventions

- 'I' is an initiator. The party that initiates an exchange is called an initiator.

- 'R' is a responder.

- Upper Layer Protocol (ULP). A protocol layer immediately above IP. Examples are transport protocols such as TCP and UDP, control protocols such as ICMP, routing protocols such as OSPF, and internet or lower-layer protocols being "tunneled" over (i.e., encapsulated in) IP such as IPX, AppleTalk, or IP itself.

- Application Identifier (AID). AID is a 128-bit routable IP locator which has been selected for communication with a peer to be used by the upper layer protocol. This is used for pseudo-header checksum computation and connection identification in the ULP.

- Context Identifier (CID). CID-pair is used by the wedge layer to establish and identify the context during context establishment and address update exchanges. A 128-bit CID may be the same as the corresponding AID, or they may be separated from each other.

- Context Identifier Tag (CIDT). CIDT together with a locator-pair are included in every payload packet to identify a context. E.g. in NOID, the flowid, and in HIP, the SPI value. CIDT is conceptually distinguished from the any specific field, so that it can be used with any payload extension header.

- 'Ls' is a locator set consisting of L1, ... Ln.

- 'Hk' is k:th hash value in the hash chain. 'H0' is the first revealed value, i.e. the "anchor" of the hash chain. Note that the parameter k defines the revealing order, not the computation order.

3. Cryptographic techniques used in WIMP-F

3.1 One-Way hash chain

One-Way hash chain [7] is cryptographically generated list of data entities with some interesting characteristics. It is practically impossible to calculate or otherwise figure out the next value in the chain even when you know the previous value. However, it is very easy to verify whether some given value is the next value or not.

The chain is created by recursively computing a hash function over a result of the same function. The initial argument for the first hash value computation is typically a large random number. The last generated value of the chain is called as the "anchor" or "root" value. The hash values are revealed in the reverse order starting from the anchor value.

```
Hn = Hash(random number)
Hn-1 = Hash(Hn)
...
H0 = anchor = Hash(H1)

CHAIN:  H0,..,Hn
```

This technique is usually based on an assumption that only an authentic end-point knows the correct successor values of the chain. In the bootstrapping process, the end-point computes a new hash chain and reveals the anchor value of the chain to its peer. It is important to notice that each hash chain MUST be used only once.

3.2 One-Way hash chain and message authentication

Hashed Message Authentication Codes [2] are typically used to authenticate messages with a symmetric key. This requires, naturally, that all communicating peers share a secret.

One-Way hash chain values can be used as keys in the delayed message authentication (see TESLA [6]). This is different from the shared secret scheme, because anybody who is able to receive the subsequent messages is able to verify that the messages belong together. The one-way hash chain value (key) used in the message authentication is not revealed before the next message. In other words, the peer is able verify the message only after it has received the next message. It is good to notice that a host must receive a confirmation message before sending the next message to avoid delay attacks. This procedure can be used to verify that all subsequent messages come from the same entity than the first message.

A Man-in-the-Middle (MitM) attacker is not able to (unnoticeably) modify the messages after the first message in the exchange. However, an attacker may spoof the first message and use its own hash chain. The protocol is based on opportunistic principle where the peers trust that the initial message comes from the authentic host.

```
A -> B: Msg1(A), HMAC(H1(A), Msg1(A))
A <- B: Msg1(B), HMAC(H1(B), Msg1(B))
A -> B: Msg2(A), H1(A), HMAC(H2(A), Msg2(A))
A <- B: Msg2(B), H1(B), HMAC(H2(B), Msg2(B))
...
A -> B: Msgn(A), Hn-1(A), HMAC(Hn(A), Msgn(A))
A <- B: Msgn(B), Hn-1(A), HMAC(Hn(B), Msgn(B))
A -> B: Hn(A)
A <- B: Hn(B)
```

3.3 Chained bootstrapping

In the basic model, each one-way hash chain is an independent entity. This is not a problem if the anchor of the chain can be authenticated, and if the hash chain is known to be long enough to have values to every message in the communication session. However, it is possible to use short hash chains to avoid extra computation.

Basically, the bootstrapping message can be authenticated using public keys. In the WIMP-F approach, the peers do not authenticate the bootstrapping message with a signature, but they rely on the delayed message authentication. Two independently created one-way hash chains can be linked together with HMAC computation. The last value of the first one-way hash chain is used to authenticate the first value of the second chain:

```
...
A -> B: Msgn, H0new, Hn-1, HMAC(Hn, Msgn || H0new)
A <- B: (conf)
A -> B: Msgn+1, Hn, HMAC(H1new, Msgn+1)
A <- B: (conf)
A -> B: Msgn+2, H1new, HMAC(H2new, Msgn+2)
...
```

3.4 Secret splitting

In secret splitting [4][5], a secret is divided into several pieces. Any piece alone does not give enough information for an attacker to create the original secret. The only way to create the secret is to posses all the pieces. The splitting is done by generating random-bit strings, the same length as the original secret. The key splitting and secret reconstruction is done in the following way:

Constructing key pieces:

```
  K_1 = nonce1
  ...
  K_k-1 = noncek-1
  K_k = SECRET XOR K_1 ... XOR K_k-1
```

Combining the secret:

```
  SECRET = K_1 XOR ... XOR K_k
```

Secret splitting is useful technique for storing secrets in two physical places, or for sending a secret to the other end-point using two or more parallel communication paths.

4. Protocol overview

The framework consists of AIDs, CIDs, CIDTs, and IP layer locators. Each of them having a special role from the conceptual point of view. In addition, the framework consists of mechanisms and policies. The policies, like address selection policy, are out of this draft scope. The mechanism are used to establish a context and prove the ownership

of the AID and the context. If AIDs are also used for wedge layer CIDs, there is no reason to separate these two mechanisms from each other, and AID ownership can be also used to prove the context ownership.

WIMP-F supports, by default, locators without any cryptographical properties. The weak ownership of routable AID is provided by the implicit reachability during the context establishment. Otherwise, this draft leaves open the security properties of the routable AIDs and other mechanisms used to prove the ownership of the AID. It is good to notice that proving the ownership of AID is an essential mechanism in the final stand-alone Multi6 solution. However, routable AIDs may have ephemeral suffixes making the guessing of AIDs difficult for an attacker. However, in some cases it is not possible to have such a randomness. The ownership of an AID could be e.g. based on reverse DNS or public key based cryptography.

The IP layer locators are included in the address fields in the IP packet headers. Further, each wedge 3.5 layer implementation must establish a state, i.e. a context, for AID-locator bindings. The context must be identified with some context identifier (CID) during context establishment exchange. However, to avoid overhead in packet sizes, it is possible to compress the CID and include a smaller CID tag (CIDT) in every payload packet. CIDT is used together with a locator-pair, in the IP header, to identify a context at the other end-point.

The WIMP-F context establishment exchange is based on an opportunistic principle. That is, a host does not care who its peer is, as long as the peer is the same during the communication context lifetime. The trust between the peers is established using one-way hash chains during the initial exchange. The context owner is the owner of the hash chain. Basically, the CID is just an index that refers to a correct context. However, if an attacker is able to guess or otherwise figure out the initiator's CID, the host becomes vulnerable for context identifier theft. That is, an attacker tries to establish a state using the identifier of the initiator. However, it is possible to use ephemeral or cryptographically generated AIDs as CIDs. In the latter case, a public key signature field must be added (TBD) to the WIMP-F packet structures. In this case, the responder should verify the ownership of the AID only after an AID collision happens. Ephemeral port numbers helps to mitigate AID ownership problem. However, when AIDs are used for CIDs, such an extra randomness will disappear.

To protect from redirection attacks the presented protocol relies on the ability to verify that the entity requesting redirection indeed holds the successor values of a hash chain and is able to combine a

divided secret that is sent via parallel paths. WIMP-F offers context establishment and re-addressing exchanges. The exchanges are based on UDP, by default. The former exchange establishes a state for both communication end-points. The re-addressing exchange is used to implement reachability test and to update the locators belonging to the communicating parties.

4.1 Wedge layer

```
+-----------------------------------+
|        Transport Protocols        |
+-----------------------------------+
| AH | ESP | Frag/reass | Dest opts |
+-----------------------------------+
|            Wedge layer            |
+-----------------------------------+
|               IP                  |
------------------------------------+
```

Figure 1: Protocol stack

The proposal uses a wedge layer between IP and the ULPs as shown in Figure 1, in order to provide ULP independence. Conceptually the wedge layer behaves as if it is an extension header, which would be ordered immediately after any hop-by-hop options in the packet. Layering AH and ESP above the wedge layer means that IPsec can be made to be unaware of locator changes the same way that transport protocols can be unaware. Thus the IPsec security associations remain stable even though the locators are changing. Layering the fragmentation header above the wedge makes reassembly robust in the case that there is broken multi-path routing which results in using different paths, hence potentially different source locators, for different fragments.

4.2 Translation between AIDs and Locators

Applications and upper layer protocols use AIDs which the wedge layer will map to/from different locators. The wedge layer maintains state, called host-pair context, in order to perform this mapping. The mapping is performed consistently at the sender and the receiver, thus from the perspective of the upper layer protocols packets appear to be sent using AIDs from end to end, even though the packets travel through the network containing locators in the IP address fields, and even though those locators may be even rewritten in flight.

```
   +--------------------+           +--------------------+
   | Initiator          |           | Responder          |
   |                    |           |                    |
   |  ULP               |           |  ULP               |
   |   | src AID(I)     |           |   ^                |
   |   | dst AID(R)     |           |   | src AID(I)     |
   |   v                |           |   | dst AID(R)     |
   |  WEDGE             |           |  WEDGE             |
   |   | src L1(I)      |           |   ^                |
   |   | dst L1(R)      |           |   | src L1(I)      |
   |   v                |           |   | dst L1(R)      |
   |   IP               |           |   IP               |
   +--------------------+           +--------------------+
       |                              ^
       +- cloud with some routers ----+
```

Figure 2: Mapping identifiers to locators.

The result of this consistent mapping is that there is no impact on the ULPs. In particular, there is no impact on pseudo-header checksums and connection identification.

4.3 Host-Pair Context

The context establishment exchange establishes a state for both communication end-points. The initiator creates a host-pair context based on IDs and the locator set. The responder establishes a state after receiving the second message from the initiator.

The context contains the following information:

- Context identifiers.

- Locator and locator status (e.g. if locator has been verified, and which locators are preferred for communication)

- Hash chain information (e.g. parameters needed in the construction of hash chains, last used local chains values, and last known peer chain values)

Every IP packet must contain information about the related host-pair context to find the the right one for the received packets. Basically, it is possible to add an extra extension header to each packet, containing a context identifier. This results into extra overhead in the payload packets. On the other hand, a Multi6 protocol may include the context identifier into the IPv6 header using, e.g., flowid field (NOID), or SPI value (HIP). Each Multi6 protocol defines own mechanism to map packets to Multi6 context.

Thus, WIMP-F context establishment exchange supports variable size CIDTs.

4.4 Generating one-way hash chains

The hash chains are needed by the initiator and the responder during the context establishment and re-addressing exchange. In addition, the initiator bootstraps a new hash chain during every re-addressing exchange.

WIMP-F sets the following requirements for the hash chains generation:

- Each hash chain MUST be bound to end-point identifiers, i.e., CID(I) and CID(R).

- Each hash chain MUST be bound to a local secret. Using the local secret the responder does not need to establish a state during the first round-trip in the context establishment exchange. In addition, the local secret, stored in a persistent memory system, solves the state loss problem. The responder SHOULD reuse the same local secret with multiple initiators to avoid establishing a state during the first round-trip.

- Each hash chain MUST be bound to a random string, i.e., a challenge. The challenge makes each of the hash chains statistically unique and protects the hosts from attackers that try to find out hash chain values using spoofed identifiers. The challenge is initially generated by the host that constructs the related hash chain.

Initiator:

```
secret(I/R)  = secret random number generated by I/R
challenge(I/R) = public random number generate by I/R
Hk(I/R)      = hash chain value
ID(I/R)      = end-point identifier

Hn(I) = SHA1(secret(I) || CID(I) || CID(R) || challenge(I))
...
H1(I) = SHA1(H2(I))
H0(I) = SHA1(H1(I))
```

Responder:

```
Hn(R) = SHA1(secret(R) || CID(R) || CID(I) || challenge(R))
...
H1(R) = SHA1(H2(R))
```

```
   H0(R) = SHA1(H1(R))
```

The default length of both of the hash chains should be n=10. Theoretically speaking, the minimum length of the hash chain is four (4) hash values. This will last for one context establishment exchange, and one re-addressing exchange for both directions. The re-addressing exchange includes a bootstrapping procedure where a new hash chain is created for the initiator. However, each unsuccessful re-addressing exchange attempt will require one more hash chain value. Failures in re-addressing exchange may be due to connection loss in some of the locators, or an active attack. A host should bootstrap a new hash chain at latest when it has only two hash values left.

4.5 Context establishment exchange

The context establishment exchange bootstraps hash chains and creates a state between two end-points. The protocol uses delayed authentication procedure (see Section 3.2). The responder remains stateless during the first round-trip. At the end of the exchange both parties have a uniquely identifiable host-pair context containing the peer's anchor value.

```
       Initiator                                       Responder

       Construct hash chain.
       Define CIDs and CIDT(I).

              INIT: CIDs, challenge(I), [challenge_old(R), Hn_old(R)],
               HMAC_INIT{H0(I), (CIDs || challenge(I) || CIDT(I) || Ls(I))}
              -------------------------->
                                         No host-pair context found.
                                         Generate challenge(R).
                                         Construct temporary hash chain.

               CC: CIDs, HMAC_INIT, challenge(R), H0(R), Ls(R),
                 HMAC_CC{H1(R), (CIDs || HMAC_INIT || challenge(R) || Ls(R))}
              <-------------------------
       verify HMAC_INIT                  remain stateless

               CCR: CIDs, HMAC_INIT, challenge(R), H0(R), challenge(I),
                    HMAC_CC, Ls(I), CIDT(I), H0(I)
              -------------------------->
                                         Reconstruct the hash chain.
                                         Verify HMAC_INIT and HMAC_CC.
                                         Define CIDT(R).
                CONF: CIDs, H1(R), CIDT(R)
              <--------------------------
```

verify HMAC_CC

WIMP-F can be used with routable AIDs without cryptographical properties, supporting separate ephemeral context identifier namespace. In such a case, the context establishment is based on opportunistic principle, and each host selects its context identifier during the exchange. INIT contains Initiator's CID, but Responder's CID is zero. Responder includes its CID in CC message.

The initiator triggers the exchange by sending a context initialization message, INIT, to the responder. The INIT message contains the CIDs, a challenge of the initiator and a HMAC. Optionally, it contains also an old challenge and the last revealed hash chain value of the responder (discussed later in Section 4.5.3).

The HMAC_INIT, having an anchor value as a key, is computed over the CIDs, the context identifier, the challenge and the locator set of the initiator. The context identifier and the locator set are sent later in the context check reply message (CCR). The optional values related to responder's old hash chain are not included into the HMAC_INIT computation.

Once the responder receives the INIT message, it must check whether it has already a host-pair context for the CID -pair. If a host-pair context is found, the responder should assume that the initiator has lost its state (discussed later in Section 4.5.2). If the context is not found, but the INIT message contains responder's old challenge and old hash chain value, the responder should assume it has lost its state (discussed later in Section 4.5.3). In a typical case, when the context is not found, and the INIT message does not contains any optional values, the responder must continue the normal context establishment exchange.

The responder must not establish a state after receiving the INIT, because it cannot verify the origin of the message. In order to remain stateless, the responder generates a fresh challenge and computes a temporary hash chain, as presented in Section 4.4. The context check (CC) message contains the CIDs, the received HMAC_INIT, the responder's challenge, the anchor value, and the locator set. The message is protected with the second value, H1(R), of responder's hash chain. The initiator should make reachability test for every received locator, in the Ls(R), before using it. (discussed later in Section 4.6).

The initiator replies to the CC message with a context check reply (CCR) message. The initiator proves that it was reachable at the specific location by including the HMAC_CC into the message. Using the responder's challenge and anchor value in the CCR message, the

responder can reconstruct the hash chain. The CCR message contains the required information to establish a state and verify the HMAC_INIT and HMAC_CC. The anchor value, H0(I), binds also the INIT and CCR messages together. The responder should make a reachability test for every received locator, in the Ls(I), before using it (discussed later in Section 4.6).

The responder must drop a CCR packet with an ID pair that already has a host pair-context. If the context is not found, the responder reconstructs a hash chain and verifies the HMAC_CC and HMAC_INIT (in this order). If the HMACs were valid, the responder creates a host pair context using the CIDs. It also selects a context identifier, CIDT(R), to be used in the payload packets. Finally, the responder replies with a context confirmation message (CONF) revealing the second hash value, H1(R).

The initiator verifies the HMAC_CC using the hash chain value received in the CONF message, and finalizes its state.

4.5.1 State Loss

The context establishment exchange has been designed in the way that it can be reused for secure re-synchronization. If a host receives a valid SYNC message, it should start a new handshake with the INIT message. The following scenarios describe the main use cases that are covered by the design:

- The initiator does not have a host-pair context, but the responder already has one for the CIDs. Either some host (e.g. an attacker) is already using the the initiator's identifier, ID(I), or the initiator has earlier lost its state.

- The initiator has a host-pair context, but the responder has lost its state.

4.5.2 Identity theft or the initiator has lost its state?

If an initiator reboots or times out, it has lost its host-pair context. The host does not have any prior state and sends INIT (see Figure 10). If the responder finds out an active host-pair context corresponding the CIDs, it compares the received challenge(I) with the old challenge'(I) in the existing context. If the received challenge(I) is different than the old one, it replies with the SYNC message containing the initiator's old challenge'(I) and the latest revealed hash chain value Hk'(I). If the received challenge(I) value is the same with the old value, the responder replies with CC message.

Once the initiator receives SYNC, it reconstructs the old hash chain using the challenge'(I) and verifies that the Hk'(I) is a valid value of that hash chain. If the verification fails the initiator MUST drop the packet. Either 1) the packet was sent by an attacker or 2) some other host is using the same ID(I). To protect from the former attack, the initiator SHOULD wait for a while to receive a valid CC or SYNC packet. Finally, if it does not receive a valid reply, it is possible that an attacker has established a host-pair context with the responder using the initiator's identifier (an identity theft).

The initiator has different alternatives to continue the exchange, depending on the Group-F design. If possible, the initiator SHOULD change its (ephemeral) CID and restart the exchange. If the initiator is using a cryptographical identifier (e.g. HIT), it may prove the ownership with signature included into the WIMP-F packet (TBD) or start a public key based exchange (e.g. HIP base exchange), using the same ID(I), to prove to the responder that it is the authentic owner of the identifier.

However, if the Hk'(I), received in SYNC, was a valid hash chain value, the initiator has probably lost its state. It SHOULD restart the context establishment exchange by sending a new INIT message, protected with a successor hash chain value, Hk+1'(I).

Once the responder receives the INIT message and challenge'(I) matches with the one in the existing host-pair context, it replies with CC. The message contains the old last revealed Hn(R), and the corresponding challenge(R). The HMAC_RR is protected with the successor value Hn+1(R).

The responder must drop the CCR message if the Hk+1'(I) verification fails. If Hk+1(I) is valid hash chain value the responder replies with CONF message, and reveals its successor hash chain value Hn+1(R).

```
    Initiator:                              Responder:

    Latest revealed value=Hk'(I)    Latest revealed value=Hn(R).
    before state loss.

    Send fresh INIT message

        INIT: CIDs, challenge(I),
          HMAC_INIT{H0(I), (CIDs || CIDT(I) || challenge(I) || Ls(I))}
           -------------------------->
                                      Host-pair context found.
                                      No match with challenge'(I).
```

```
              SYNC: CIDs, challenge'(I), Hk'(I)
              <-------------------------

       Reconstruct old hash chain
       using challenge'(I).
       Verify Hk'(I).
       Restart exchange.

           INIT: CIDs, challenge'(I),
             HMAC_INIT{Hk+1'(I), (CIDs || CIDT(I) || challenge'(I) || Ls(I))}
              -------------------------->
                                         Host-pair context found.
                                         Match with challenge'(I).

            CC: CIDs, HMAC_INIT, challenge(R), Hn(R), Ls(R),
              HMAC_CC{Hn+1(R), (CIDs || HMAC_INIT || challenge(R) || Ls(R))}
              <-------------------------


              CCR: CIDs, CIDT(I), Hk+1'(I), challenge'(I), Hn(R),
                challenge(R), HMAC_INIT, HMAC_CC, Ls(I)
              -------------------------->
                                         verify Hk+1'(I)
                                         verify HMAC_INIT and HMAC_CC
                                         update host-pair context
              CONF: CIDs, Hn+1(R), CIDT(R)
              <--------------------------

       verify HMAC_CC
       update host-pair context
```

Figure 10: Initiator has lost its state

4.5.3 Responder has lost its state

If a system receives an IP packet that does not match with any host-pair context, the host has probably lost its state. The host replies with SYNC message containing the context identifier that was received in the IP packet (see Figure 11). Every IP packet must not trigger a new SYNC message. The SYNC reply frequency must be rate limited.

Once a host receives a SYNC message containing only a context identifier, it should try to find a corresponding host-pair context. If a host-pair context is found the host should send an INIT message to verify whether the peer has lost its state. The initiator includes the last revealed hash chain value, Hn(R), and corresponding

challenge(R) of the responder to the message. The initiator should not generate new hash chain for itself, but use the successor value, Hk+1(I), of the existing hash chain to protect the INIT message.

If the responder has a host-pair context for the CIDs, the SYNC message was sent by an attacker, and the responder must drop the INIT message containing the old hash chain and challenge values. If the responder does not have a host-pair context for the CIDs, it should reconstruct the old hash chain. The responder must verify that the received hash chain value, Hn(R), is a valid member of the chain. If the verification fails the responder must drop the INIT message. If the Hn(R) is valid value, the responder includes the successor value, Hn+1(R), into CC message, to prove that it has really lost its state. The rest of the exchange is identical with the normal exchange, but the responder must reveal the successor value, Hn+2(R), in the CONF message.

```
       Initiator:                               Responder:

       Latest revealed value=Hk(I).      Latest revealed value=Hn(R)
                                         before state loss.

              IP packet including CIDT(I)
              -------------------------->
                                         No host-pair context found
              SYNC: CIDT(I)
              <-------------------------
        Find host-pair context.
        Start exchange.

              INIT: CIDs, challenge(R), Hn(R),
                HMAC_INIT{Hk+1(I), (CIDs || CIDT(I) || challenge(I) || Ls(I))}
              -------------------------->
                                         No host-pair context found.
                                         Construct old hash chain
                                         using challenge(R).
                                         Verify Hn(R).

               CC: CIDs, HMAC_INIT, challenge(R), Hn+1(R), Ls(R),
                HMAC_CC{Hn+2(R), (CIDs || HMAC_INIT || challenge(R) || Ls(R))}
              <-------------------------
         verify Hn+1(R)                  remain stateless

                CCR: CIDs, CIDT(I), Hk+1(I), challenge(I), Hn+1(R),
challenge(R),
                 HMAC_INIT, HMAC_CC, Ls(I)
              -------------------------->
                                         reconstruct the hash chain
                                         verify HMAC_INIT and HMAC_CC
                                         select CIDT(R)
                CONF: CIDs, Hn+2(R), CIDT(R)
              <--------------------------

       verify HMAC_CC
       update host-pair context
```

Figure 11: Responder has lost its state

4.6 Re-addressing exchange

Once the state has been completed, both hosts have a host-pair context. As a result, each host knows a locator set of its peer. It is good to notice that the responder may be multi-homed, even the initiator does not initially know all of the responder's locators. The hosts use re-addressing exchange to dynamically update their locator sets and to bootstrap hash chains.

The initiating party of the context establishment exchange was called the initiator. Once the host-pair contexts are established, this initial distinction is lost. Thus, the sender of the BOOTSTRAP message is called the initiator of the re-addressing exchange.

```
        Initiator                                      Responder

        construct new hash chain

        BOOTSTRAP: CIDs, Ls(I), Hn(I), H0_new(I), challenge(I),
           HMAC{Hn+1(I),(CIDs || Ls(I) || H0_new(I) || challenge(I))}
                  -------------------------->
                                     Verify H1(I).
                                     Generate a divided secret of Hk(R).
                                     Send AC per locator to be verified.

            AC1: CIDs, Key_count, Key_mask, key_piece, challenge(I)
            ...   <-------------------------
            ACn   <-------------------------

        combine the key pieces
        verify the combined key

            ACR: CIDs, Key_combined, Key_mask, Hn+1(I)
                  -------------------------->
                                                verify the combined key Hk(R)
                                                verify HMAC
```

The re-addressing exchange is a three-way handshake. The BOOTSTRAP

message has two purposes. First, it informs the responder about the currently active locator set. Second, it bootstraps a new hash chain.

Once the responder receives a BOOTSTRAP message, it verifies that the hash chain value, Hn(I), belongs to the initiator. In addition, the responder stores the initiator's new anchor value, H0_new(I), into the host-pair context. (Note: a host may verify locators after the context establishment exchange by directly sending AC messages to the peer).

The responder divides its next unused hash chain value, Hk(R), into pieces using the secret splitting technique (See Section 3.4). The responder MAY verify the locators in the locator set on demand basis or all at once. In the latter case, the hash chain value is divided into as many parts as the were locators in the received locator set. The responder defines a key mask for each of the key pieces (see Section 6.2.9). The key mask will later allow the responder to check if all pieces reached the initiator. It is possible that the initiator is not reachable via all of the locators in the locator set.

The responder sends one address check message (AC) per locator that it wants to verify. Each AC message contains one piece of the divided Hk(R). The initiator should wait a while (TBD) for the upcoming AC messages. The key count in every AC packet defines the total amount of pieces, sent by the responder.

If the initiator receives all the pieces of the hash chain value, it should verify that the combined pieces form a successor value of the responder's hash chain. Otherwise, the initiator MAY stop the re-addressing exchange. The initiator makes an OR-operation with all of the received key masks and includes the result of the operation with the combined key to the address check reply message (ACR). The ACR message includes also hash chain value, Hn+1(I) that was used to protect the HMAC.

The responder identifies the locators which were reachable, using the combined key and the key mask. The responder authenticates also the BOOTSTRAP message with hash chain value, Hn+1. Finally, the responder marks the verified locators valid in the host-pair context.

5. Packets

There are eight basic WIMP-F packets. Four are for the context establishment exchange, and three are for re-addressing, and one is for re-synchronization.

Packets consist of the fixed header as described in Section 6.1, followed by parameters. The parameter part, in turn, consists of zero or more TLV coded parameters.

Packet representation uses the following operations:

()	parameter
[]	optional parameter

An upper layer payload MAY follow the WIMP-F header. The payload proto field in the header indicates if there is additional data following the WIMP-F header.

5.1 INIT - the context initiator packet

The WIMP-F header values for the INIT packet:

```
   Header:
     Packet Type = 1
     SRC CID = Initiator's CID
     DST CID = Responder's CID

   IP( WIMP-F (CHALLENGE(I), HMAC-INIT, [CHALLENGE(R), HASVAL(R)]))
```

Valid control bits: P

5.2 CC - the context check packet

The WIMP-F header values for the CC packet:

```
   Header:
     Packet Type = 2
     SRC CID = Responder's CID
     DST CID = Initiator's CID

   IP ( WIMP-F (CHALLENGE(R), HASVAL(R), LSET(R), HMAC-INIT, HMAC-CC ))
```

Valid control bits: P

The CIDs MUST be the same as received in INIT.

The responder copies the HMAC-INIT TLV from the INIT message to the CC message.

5.3 CCR - the context check reply packet

The WIMP-F header values for the CCR packet:

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

      IP ( WIMP-F ( HASHVAL(I), HASHVAL(R), CIDT(I), CHALLENGE(I),
CHALLENGE(R),
                  LSET(I), HMAC-INIT, HMAC-CC))
```

Valid control bits: P

The CIDs used MUST match the ones used in the INIT and CC messages.

The Initiator copies the HMAC-INIT, and HMAC-CC TLVs from the CC message to the CCR message.

5.4 CONF - the context confirm packet

The WIMP-F header values for the CONF packet:

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

      IP ( WIMP-F ( HASHVAL(R), CIDT(R) ))
```

Valid control bits: P

The CIDs used MUST match the ones used in the CCR message.

5.5 BOOTSTRAP - The bootstrapping packet

The WIMP-F header values for the BOOTSTRAP packet:

```
      Header:
        Packet Type = 10
        SRC CID = Initiator's CID
        DST CID = Responder's CID

      IP ( WIMP-F ( LSET(I), HASHVAL(I), ANCHOR(I), CHALLENGE(I), HMAC-
BOOTSTRAP ))
```

Valid control bits: P

5.6 AC - The address check packet

The WIMP-F header values for the AC packet:

```
Header:
  Packet Type = 11
  SRC CID = Responder's CID
  DST CID = Initiator's CID
```

IP (WIMP-F (KEY(R), CHALLENGE(I)))

Valid control bits: P

The responder copies the CHALLENGE(I) from the BOOTSTRAP message to the AC message(s)

5.7 ACR - The address check reply packet

The WIMP-F header values for the AC packet:

```
Header:
  Packet Type = 20
  SRC CID = Initiator's CID
  DST CID = Responder's CID
```

IP (WIMP-F (KEY(I), HASHVAL(I)))

Valid control bits: P

The KEY TLV is constructed of the received key pieces and their key masks.

5.8 SYNC - The re-synchronization packet

The WIMP-F header values for the SYNC packet:

```
Header:
  Packet Type = 12
  SRC CID = Initiator's CID
  DST CID = Responder's CID
```

IP (WIMP-F (CIDT(I), [CHALLENGE(I), HASHVAL(I)]))

Valid control bits: -

6. Message formats

6.1 Header format

All WIMP-F packets start with a fixed header.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | Next Header   |  Payload Len  |     Type      |  VER. |  RES. |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Controls             |           Checksum            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Sender's Identifier (CID)                |
   |                                                               |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Receiver's Identifier (CID)              |
   |                                                               |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   /                        WIMP-F Parameters                      /
   /                                                               /
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The WIMP-F header is carried in UDP payload followed by a next header that is defined in Next Header field. If no next headers follow, the decimal 59, IPPROTO_NONE, the IPV6 no next header value, is used.

The Header Length field contains the length of the WIMP-F header and the length of parameters in 8 bytes units, excluding the first 8 bytes. Since all WIMP-F headers MUST contain the sender's and receiver's CID fields, the minimum value for this field is 4, and conversely, the maximum length of the WIMP-F Parameters field is (255*8)-32 = 2008 bytes.

The Packet Type indicates the WIMP-F packet type. The individual packet types are defined in the relevant sections. If a WIMP-F host receives a packet that contains an unknown packet type, it MUST drop the packet.

The Version is four bits. The current version is 1. The version number is expected to be incremented only if there are incompatible changes to the protocol. Most extensions can be handled by defining new packet types, new parameter types, or new controls.

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

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

6.1.1 WIMP-F Controls

The WIMP-F control section transfers information about the structure of the packet and capabilities of the host.

The following fields have been defined:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | | | | | | | | | | | |P| | | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

P - Piggy backing The sending host is capable of sending and
   receiving additional data in WIMP-F packets.

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

6.1.2 Checksum

If WIMP-F messages are sent in UDP datagrams the checksum field in the WIMP-F header is set to zero.

If WIMP-F messages are implemented as IP extension headers, the checksum is calculated in the following way. The pseudo-header [1] contains the source and destination IPv6 addresses, WIMP-F packet length in the pseudo-header length field, a zero field, and the WIMP-F protocol number (TBD) in the Next Header field. The length field is in bytes and can be calculated from the WIMP-F header length field: (WIMP-F Header Length + 1) * 8.

6.2 TLV format

The TLV encoded parameters are described in the following subsections. The type-field value also describes the order of these fields in the packet. The parameters MUST be included into the packet so that the types form an increasing order. If the order does not follow this rule, the packet is considered to be malformed and it MUST be discarded.

All the TLV parameters have a length which is a multiple of 8 bytes. When needed, padding MUST be added to the end of the parameter so that the total length becomes a multiple of 8 bytes. This rule ensures proper alignment of data. If padding is added, the Length field MUST NOT include the padding. Any added padding bytes MUST be set zero by the sender, but their content SHOULD NOT be checked on the receiving end.

Consequently, the Length field indicates the length of the Contents field (in bytes). The total length of the TLV parameter (including Type, Length, Contents, and Padding) is related to the Length field according to the following formula:

Total Length = 11 + Length - (Length + 3) % 8;

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

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

Critical parameters MUST be recognized by the recipient. If a recipient encounters a critical parameter that it does not recognize, it MUST NOT process the packet any further.

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

6.2.1 HMAC-INIT

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                             HMAC                              |
   |                                                               |
   |                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type 65500

Length 20

HMAC

HMAC-SHA1 is computed over:

- WIMP-F common header,
- CIDT TLV, of the initiator,
- LSET TLV, of the initiator
- CHALLENGE TLV, of the initiator,

excluding the HMAC-INIT TLV. The checksum field MUST be set to zero and the WIMP-F header length in the WIMP-F common header MUST be calculated to cover all the included parameters when the HMAC-SHA1 is calculated. The key is the first unrevealed hash value of the Initiator.

HMAC-INIT calculation and verification process:

INIT message sender:

1. Create the INIT message, containing CIDT, LSET and CHALLENGE TLVs, without the HMAC-INIT TLV.

2. Calculate the length field in the WIMP-F header.

3. Compute the HMAC-SHA1.

4. remove the CIDT and LSET TLVs.

5. Add the HMAC-INIT TLV and optional TLVs to the packet.

6. Recalculate the length field in the WIMP-F header.

CCR message receiver:

1. Create the INIT message, containing CIDT, LSET and CHALLENGE TLVs, without the HMAC-INIT TLV.

2. Calculate the length in the WIMP-F header and clear the checksum field (set it to all zeros).

3. Compute the HMAC-SHA1 and verify it against the received HMAC-INIT TLV.

6.2.2 HMAC-CC

The TLV structure is the same as in Section 6.2.1. The fields are:

Type	65502
Length	20
HMAC	HMAC-SHA1 is computed over: - WIMP-F common header. - HMAC-INIT TLV, received in the INIT message, - CHALLENGE TLV, of the responder, - LSET TLV, of the responder, excluding the HMAC-CC parameter. The checksum field MUST be set to zero and the WIMP-F header length in the WIMP-F common header MUST be calculated to cover all the included parameters when the SHA1 is calculated. The key is the the first unrevealed hash value of the responder.

HMAC-CC calculation and verification process:

CC message sender:

1. Create the CC message, containing HMAC-INIT, CHALLENGE and LSET TLVs, without HMAC-CC TLV.

2. Calculate the length field in the WIMP-F header.

3. Compute the HMAC-SHA1.

4. Add the HMAC-CC TLV and HASVAL TLV to the packet.

5. Recalculate the length field in the WIMP-F header.

CCR and CONF message receiver:

1. Create the CC message, containing HMAC-INIT, CHALLENGE and LSET TLVs, without HMAC-CC and HASHVAL TLVs.

2. Calculate the length field in the WIMP-F header and clear the checksum field (set it to all zeros).

3. Compute the HMAC-SHA1 and verify it against the received HMAC-CC.

6.2.3 HMAC-BOOTSTRAP

The TLV structure is the same as in Section 6.2.1. The fields are:

```
   Type           65504
   Length         20
   HMAC
                 HMAC-SHA1 is computed over:
                    - WIMP-F common header,
                    - LSET TLV, of the initiator,
                    - ANCHOR TLV, of the initiator,
                    - CHALLENGE TLV, of the initiator,

                  excluding the HMAC-BOOTSTRAP parameter. The checksum
                  field MUST be set to zero and the WIMP-F header length
                  in the WIMP-F common header MUST be calculated to cover
                  all the included parameters when the SHA1 is calculated.
                  The key is the first unrevealed hash value of the
initiator's
                  hash chain.
```

HMAC-BOOTSTRAP calculation and verification process.

BOOTSTRAP message sender:

1. Create the BOOTSTRAP message, containing LSET, ANCHOR, and CHALLENGE TLVs, without HMAC-BOOTSTRAP and HASHVAL TLVs.

2. Calculate the length field in the WIMP-F header.

3. Compute the HMAC-SHA1.

4. Add the HMAC-CC and HASHVAL TLVs to the packet.

5. Recalculate the length field in the WIMP-F header.

ACR message receiver:

1. Create the BOOTSTRAP message, containing LSET, ANCHOR, and CHALLENGE TLVs, without HMAC-BOOTSTRAP and HASHVAL TLVs.

2. Calculate the length field in the WIMP-F header and clear the checksum field (set it to all zeros).

3. Compute the HMAC-SHA1 and verify it against the received HMAC-BOOTSTRAP.

6.2.4 HASHVAL

```
   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            |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |             Chain ID          |           Count               |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                            reserved                           |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                                                               |
  |                                                               |
  |                              Hash                             |
  |                                                               |
  |                                                               |
  |                                                               |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	12
Length	28
Chain ID	Identifier of the Hash Chain.
Count	The number of a hash chain value in the hash chain. zero means an anchor value.
Reserved	Zero when sent, ignored when received
Hash	160 bit SHA1 hash value.

6.2.5 ANCHOR

The TLV structure is the same as in Section 6.2.4. The fields are:

Type	10
Length	28
Count	0 (an anchor value).
Reserved	Zero when sent, ignored when received
Hash	160 bit SHA1 hash value.

6.2.6   CIDT

```
       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             |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             Context ID                |       padding         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	16
Length	(variable)
Context ID	context identifier (e.g. flowid or SPI)

6.2.7   CHALLENGE

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             Type              |            Length             |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                            Reserved                           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                            Challenge                          |
      |                                                               |
      |                                                               |
      |                                                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	20
Length	20
Reserved	Zero when sent, ignored when received
Challenge	128 bit (16 bytes) random number

6.2.8 LSET

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Reserved                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Locator #1                          |
   /                                                               /
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                              ...                              /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Locator #n                          |
   /                                                               /
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type	22
Length	variable
Reserved	Zero when sent, ignored when received
Locator	IPv6 address (or IPv4 address in IPv6 format)

6.2.9 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            |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |           Chain ID            |      Hash count               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |           AC ID               |      Key Count                |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                           Reserved                            |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                           Key mask                            |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   Key piece 160 bit (AC message) /            |
       /                   Combined key 160 bit (ACR message)          /
       |                                                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

       Type          26
       Length        40
       Reserved      Zero when sent, ignored when received
       Chain ID      Identifier of the Hash Chain.
       Hash Count    The number of a hash chain value in the responder's
                     hash chain. The hash chain value that is divided into key
                     pieces.
       AC ID         An increasing counter that identifies the exchange with
CIDs.
       Key count     The total number of key pieces sent/received
       Key Mask
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                     The responder defines a bit in the mask for a key piece.

       Key piece     A hash chain value is divided into (160 bit) key pieces
                     by the responder.
       Combined key  The initiator constructs a combined key using the key
                     pieces.
```

The AC ID is related to a divided secret, i.e., a hash chain value. Every key piece related to that value must have the same AC ID.

Responder sending AC message:

The key count field value in the AC packet is the total number of key pieces sent by the responder, i.e. the total number of AC packets. Each KEY-MASK TLV in one AC packet contains a key piece having a corresponding bit on in the key mask field

(Note: The responder is able to make a reachability test maximum for 32 locators per divided key. However, it typically verifies the locators on demand basis.)

Initiator sending ACR message:

The initiator makes an OR -operation for all received key masks in the AC packets. The result of the operation is included in the KEY-MASK TLV in the ACR message.The key count field in KEY-MASK TLV in the ACR message indicates the number of key pieces the initiator received from the responder. The combined key is included into the KEY-MASK TLV.

7. Packet processing

7.1 Processing outgoing application data

In an WIMP-F host, an application can send application level data using AIDs as source and destination identifiers. However, whenever there is such outgoing data, the stack has to send the resulting datagram using appropriate source and destination IP addresses.

The following steps define the conceptual processing rules for outgoing datagrams destinated to a AID(R).

1. If the datagram has a specified source AID(I), it MUST be one of the locators.

2. If the datagram has an unspecified AID(I), the implementation MUST choose a suitable source AID(I) for the datagram. In selecting a proper AID(I), the implementation MUST consult the table of currently active WIMP-F sessions, and preferably select an AID(I) that already has an active session with the target AID(R).

3. If there is no active WIMP-F session with the AID(R), one may be created by running the context establishment exchange. The INIT packet is sent to the AID(R). The host selects a CID(I) and optionally CID(R) for the context establishment exchange. At latest, the context must be established when the AID(I) differs from the selected source locator.

4. Once there is an active WIMP session for the given AID(R), the AID(R) must match one of the locators in the context. The CIDs are represented by CIDTs in the IP payload packets.

5. The AIDs in the datagram are replaced with suitable locators.

7.2 Processing incoming application data

Incoming WIMP-F datagrams arrive as normal IP packets containing CIDT. In the usual case the receiving host has a corresponding host-pair context, identified by the CIDT and a locator-pair in the packet. However, if the host has crashed or otherwise lost its state, it may not have such a host-pair context.

The following steps define the conceptual processing rules for incoming datagrams targeted to a WIMP-F host-pair context.

1. If the packet does not contains CIDT, it is sent to the upper layer without processing.

2. If a proper host-pair context is detected, using the CIDT and locator-pair, the packet is processed by the wedge layer. If there are no host-pair context identified with the specific CIDT, the host MAY reply with a SYNC packet.

3. If a proper host-pair context is found, the packet is processed by WIMP-F. The locators in the datagram are replaced with the AIDs associated with the host-pair context.

8. State Machine

Each host is assumed to have a separate WIMP-F implementation that manages the host-pair contexts and handles requests for new ones. Each host-pair context is governed by a state machine. WIMP-F implementation can simultaneously maintain host-pair contexts with more than one host. Furthermore, WIMP-F implementation may have more than one active host-pair context with another host; in this case, host-pair contexts are distinguished by their respective CIDs. It is not possible to have more than one host-pair contexts between any given pair of CIDs. Consequently, the only way for two hosts to have more than one parallel associations is to use different CIDs, at least in one end.

The processing of packets depends on the state of the host-pair context(s) with respect to the originator of the packet. A WIMP-F implementation determines whether it has an active association with the originator of the packet based on the CIDs or the context identifier, CIDT, in the IP packet.

The state machine is presented in a single system view, representing either an Initiator or a Responder. There is not a complete overlap of processing logic here and in the packet definitions. Both are needed to completely implement WIMP-F.

Implementors must understand that the state machine, as described here, is informational. Specific implementations are free to implement the actual functions differently.

States:

START , state machine start

INIT-sent , INIT sent by initiator

CCR-sent , CCR sent by initiator

ESTABLISHED , host-pair context established

ESTABLISHED-BOOTSTRAP-sent , BOOTSTRAP sent

ESTABLISHED-AC-sent , AC sent without receiving BOOTSTRAP message.

FAILED , host-pair context establishment failed

State Processes:

```
+---------+
|  START  |
+---------+

Context establishment request, send INIT and go to INIT-send
Receive INIT, send CC and stay at START
Receive CCR, process
     if successfull, send CONF and go to ESTABLISHED
     if fail, stay at START
Receive IP packet without context, send SYNC and stay at START
Receive ANYOTHER, drop and stay at START

+-----------+
| INIT-sent |
+-----------+

Receive INIT, send CC and stay at INIT-sent
Receive CCR, process
     if successful, send CONF and go to ESTABLISHED
     if fail, stay at INIT-sent
Receive CC, process
     if successful, send CCR and go to CCR-sent
     if fail, stay at INIT-sent
Receive SYNC, process
     if successful, send INIT, and stay at INIT-sent
     if fail, change CID(I), send INIT, and stay at INIT-sent, or
```

```
        if fail, start another exchange supporting strong authentication,
           and stay at INIT-sent, or
        if fail, go to FAILED
   Receive ANYOTHER, drop and stay at INIT-sent
   Timeout, increment timeout counter
        If counter is less than INIT_RETRIES_MAX, send INIT,
           and stay at INIT-sent
        If counter is greater than INIT_RETRIES_MAX, go to FAILED

   +----------+
   | CCR-sent |
   +----------+

   Receive INIT, send CC and stay at CCR-sent
   Receive CCR, process
        if successful, send CONF and go to ESTABLISHED
        if fail, stay at CCR-SENT
   Receive CONF, process
        if successful, go to ESTABLISHED
        if fail, stay at CCR-SENT
   Receive ANYOTHER, drop and stay at CCR-sent
   Timeout, increment timeout counter
        If counter is less than CCR_RETRIES_MAX, send CCR and stay at CCR-sent
        If counter is greater than CCR_RETRIES_MAX, go to FAILED

   +-------------+
   | ESTABLISHED |
   +-------------+

   BOOTSTRAP to send, go to ESTABLISHED-BOOTSTRAP-sent
   AC to send, go to ESTABLISHED-AC-sent
   Receive BOOTSTRAP, process
        if successful, send AC(s), and stay at ESTABLISHED
        if fail, drop, and stay at ESTABLISHED
   Receive AC, process
        if successful, send ACR, and stay at ESTABLISHED
        if fail, drop and stay at ESTABLISHED
   Receive ACR, process
        if successful, update context, and stay at ESTABLISHED
        if fail, drop, and stay at ESTABLISHED
   Receive IP packet for context, process and stay at ESTABLISHED
   Receive SYNC, process
        if successful, send INIT, and stay at ESTABLISHED
        if fail, drop, and stay at ESTABLISHED
   Receive CC, process
        if successful, update context, send CCR, and stay at ESTABLISHED
        if fail, drop, and stay at ESTABLISHED
   Receive CONF, process
```

```
        if successful, update context, and stay at ESTABLISHED
        if fail, drop, and stay at ESTABLISHED
   Receive ANYOTHER, drop and stay at ESTABLISHED

   +------------------------------+
   | ESTABLISHED-BOOTSTRAP-sent  | (for full re-addresing exchange)
   +------------------------------+
   Receive AC, process
        if successful, send ACR, and go to ESTABLISHED
        if fail, drop and cycle at ESTABLISHED-BOOTSTRAP-sent
   Receive BOOTSTRAP, process
        if successful, send AC(s), and stay at ESTABLISHED-BOOTSTRAP-sent
        if fail, drop, and stay at ESTABLISHED-BOOTSTRAP-sent
   Receive ACR, process
        if successful, update context, and stay at
           ESTABLISHED-BOOTSTRAP-sent
        if fail, drop, and stay at ESTABLISHED-BOOTSTRAP-sent
   Receive SYNC, process
        if successful, send INIT, and stay at ESTABLISHED-BOOTSTRAP-sent
        if fail, drop, and stay at ESTABLISHED-BOOTSTRAP-sent
   Receive CC, process
        if successful, update context, send CCR, and go to ESTABLISHED
        if fail, drop, and stay at ESTABLISHED-BOOTSTRAP-sent
   Receive IP packet for context, process and stay at ESTABLISHED-BOOTSTRAP-
sent
   Receive ANYOTHER, drop and stay at ESTABLISHED-BOOTSTRAP-sent
   Timeout, increment timeout counter
        If counter is less than BOOTSTRAP_RETRIES_MAX,
            send BOOTSTRAP to another locator,
            and stay at ESTABLISHED-BOOTSTRAP-sent
        If counter is greater than BOOTSTRAP_RETRIES_MAX, go to FAILED

   +-----------------------+
   | ESTABLISHED-AC-sent  | (for dynamic AC-ACR reachability test)
   +-----------------------+
   Receive AC, process
        if successful, send ACR, and stay at ESTABLISHED-AC-sent
        if fail, drop and cycle at ESTABLISHED-AC-sent
   Receive BOOTSTRAP, process
        if successful, send AC, and stay at ESTABLISHED-AC-sent
        if fail, drop, and stay at ESTABLISHED-AC-sent
   Receive ACR, process
        if successful, update context, and go to ESTABLISHED
        if fail, drop, and stay at ESTABLISHED-AC-sent
   Receive SYNC, process
        if successful, send INIT, and stay at ESTABLISHED-AC-sent
        if fail, drop, and stay at ESTABLISHED-AC-sent
   Receive CC, process
```

```
          if successful, update context, send CCR, and go to ESTABLISHED
```

```
     if fail, drop, and stay at ESTABLISHED-AC-sent
Receive IP packet for context, process and stay at ESTABLISHED-AC-sent
Receive ANYOTHER, drop and stay at ESTABLISHED-AC-sent
Timeout, increment timeout counter
     If counter is less than AC_RETRIES_MAX, send AC to another locator,
         and stay at ESTABLISHED-AC-sent
     If counter is greater than AC_RETRIES_MAX, go to FAILED
```

9. Security Considerations

The main objective in WIMP-F is to use one-way hash chains instead of public key cryptography. The reason for this is that there exists already a group of authenticated Diffie-Hellman key exchange protocols. Protocols using public key cryptography are often vulnerable for different kind of CPU related denial-of-service attacks. The trade-off between hash chain based message authentication and signatures is that the former is vulnerable for a class of man-in-the-middle attacks. However, if we accept that the first round-trip of the context establishment exchange is open for such attacks we obtain several advantages. The hash chain and HMAC computation are very lightweight operations compared to public key signing and signature verification.

9.1 Context establishment exchange

9.1.1 Man-in-the-Middle attacks

The context establishment exchange is based on opportunistic authentication. In other words, both hosts, the initiator and the responder, blindly trust to each other during the first round-trip. The responder trusts that the INIT message comes from an authentic initiator. In a similar way, the initiator trusts that the CC message comes from an authentic responder. Thus, the first round-trip is vulnerable for the MitM attacks.

Basically, the first round-trip of the exchange also bootstraps the hash chains. The initiator's anchor is sent in the third message, but the HMAC binds the first and the third messages together. A MitM attacker cannot later change any values in the CCR message, because the parameters are authenticated with the initiator's HMAC-INIT. Further, the HMAC-INIT is protected with the responder's HMAC-CC. In this way, the responder does not need to establish a state once it receives INIT message.

An eavesdropper, listening INIT messages, cannot reserve a host-pair context by sending CCR message before the initiator, because the HMACs bind the messages together. Basically, the hash chains have

two main purposes. They bind messages together, and they are used for authentication. The hash chain properties were discussed in more detail in Section 3.

9.1.2 Denial-of-Service attacks

The initiator may use any kind of identifier, CID(I), it wants with the responder, an ephemeral or a fixed identifier. The ephemeral identifier protects the initiator from a specific form of DoS attack. That is, an attacker cannot guess the initiator's identifier and reserve a state at the responder. If the initiator decides to use fixed identifiers it MAY need prove to the responder, using public key cryptography, that it owns the identifier.

The beginning of the WIMP-F context establishment exchange is stateless for the responder, in order to avoid attacks where the attacker is trying to create inconsistent states at the responder. The responder makes a kind of reachability test before establishing a state with the initiator. The initiator has to prove that it is reachable at the location where it sent the initial packet. The responder MAY optionally restrict establishing a host-pair context with CID(I) if the responder already has several host-pair contexts related to the corresponding locator. This restrictions reduces the need of using any challenge puzzle (See HIP) in the CC message.

The context establishment protocol is vulnerable for a context identifier, CIDT(R), related attack. The only parameter that is not protected with HMAC is the responder's CIDT in the last message. The reason for this is that the responder cannot reserve any values before it creates a state. The state is created after receiving the CCR message. As a result, the responder cannot include the CIDT into its HMAC in the CC message. a MitM attacker may change the responder's CIDT on the fly in the last message and course a denial-of-service situation. In other words, the responder will send IP packets with unrecognized context identifier to the initiator. However, the main objective of the protocol is to protect the hosts from the re-direction attacks and the presented attack does not open new vulnerabilities for that part of the protocol.

9.1.3 Cryptanalysis based on the state loss procedure

An attacker may apply the re-synchronization procedure to make cryptanalysis. The attacker may try to pump up a full hash chain of the responder. The attacker sends a storm of INIT packets, each of them containing different random challenge, CHALLENGE(R), and hash chain, HASHVAL(R), values. The destination identifier must match with the responder's identifier, but the initiator may freely select its own identifier. The responder drops the INIT packet, if the

received hash value is not a member of the reconstructed hash chain. However, if the attacker manages to make a correct guess, the responder responds with CC packet containing the successor value of its hash chain. The attacker can send a new INIT message containing the received successor value and the challenge that was used to generate that hash chain. Finally, the attacker finds out the whole hash chain.

The attacker must make the attack beforehand, because the responder does not reply with CC message, if it has already a context for the CIDs. In addition, the responder generates a fresh challenge for every new hash chain, and thus it is statistically hard to guess the correct combination that matches with the CIDs and the challenge. An ephemeral identifier of the initiator makes the attack more difficult.

It may be possible that an attacker is able to make cryptanalysis about the responder's secret applying the previous attack. (The probability and difficultness of such an attack is TBD. Thus, the secret should be changed every TBD. If the attacker is able to find out the responder's secret, it may impersonate itself to a responder, and launch an attack by sending SYNC message to the initiator. The initiator sends the INIT packet to one of the responder's locators and verifies if the responder has rebooted. This attack requires that the attacker is able to listen the traffic between the inititor and the responder. Otherwise, the attacker cannot reply with the correct CC message, including the initiator's HMAC_INIT. A successful attack results in that the initiator updates its context with the attacker. (Note: It is still good to notice that WIMP-F is a semi-strong authentication protocol that does not require much computation power. If strong authentication is required some public key based protocol, should be used.)

9.2 Re-addressing exchange

A host must inform its peers about the new locator(s) after site renumbering. The sender of the new locator set is called the initiator. Once the responder receives BOOTSTRAP message it cannot trust the initiator is reachable at the new location(s). To avoid different kind of DoS and re-direction attacks the responder must verify that the initiator is indeed reachable at the claimed location(s).

WIMP-F takes advantage of parallel paths between the hosts. The responder splits its hash chain value into pieces and includes one piece to each challenge (AC) message. The challenge messages are sent to different locators. As a result, an attacker has to locate at different topological locations, at the same time, to be able to

answer to the challenge. The initiator, in turn, is able to verify that all of the pieces came from the authentic responder. The secret splitting works only if there are more than one destination locators and the messages are routed via different paths to the initiator.

10. IANA Considerations

TBD.

11. Acknowledgments

The initial version of this draft was written after a discussion between the authors, Pekka Nikander and Jari Arkko at the 58th IETF. The focus in this draft has been on authenticating the hosts to their peers with one-way hash chains. Instead of inventing the wheel once again, we took several ideas from the existing multi-homing protocol proposals. Thus, there are certain similarities, e.g., with NOID and HIP design factors. We would like to thank all contributers in those drafts who have affected the work.

The authors would especially like to thank the following people who have given valuable feedback about WIMP (the names are in alphabetical order): Jari Arkko, Marcelo Bagnulo, Iljitsch van Beijnum, Gonzalo Camarillo, Brian E. Carpenter, Dave Crocker, Joel M. Halpern, Pekka Nikander, Ayyasamy Senthilkumar, and Margaret Wasserman.

12. References

12.1 Normative references

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

12.2 Informative references

[2] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[3] Nordmark, E. and T. Li, "Threats relating to IPv6 multihoming solutions", draft-nordmark-multi6-threats-00.txt (work in progress), October 2003.

[4] Shamir, A., "How to Share a Secret", Comm. of the ACM 22(11):612- 613, November 1979.

[5] Blakely, G., "Safeguarding Cryptographic Keys", In Proc. AFIPS National Computer Conference pp. 313-317, 1979.

[6] Canetti, R., Song, D. and D. Tygar, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels", , May 2000.

[7] Lamport, L., "Password authentication with insecure communication", Commun. Mag. of ACM 24 (11), pp. 770-772, 1981.

Authors' Addresses

Jukka Ylitalo
Ericsson Research Nomadiclab

Jorvas FIN-02420
FINLAND

Phone: +358 9 299 1
EMail: jukka.ylitalo@ericsson.com

Vesa Torvinen
Ericsson Research Nomadiclab

Turku FIN-20520
FINLAND

Phone: +358 9 299 1
EMail: vesa.torvinen@ericsson.com

Erik Nordmark
Sun Microsystems, Inc.
17 Network Circle
Mountain View, CA
USA

Phone: +1 650 786 2921
EMail: erik.nordmark@sun.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.