Abstract

This document presents a key exchange method allowing devices managed by a controller (e.g., an SDN management station) to create private pair-wise IPsec SAs without IKEv2 or any other direct peer-to-peer session establishment messages. The method can be used when a full mesh of IKEv2 sessions between IPsec devices is not appropriate.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 2
   1.1. Requirements Language .................................. 4
2. Overview .................................................. 4
3. Generating Initial IPsec SAs ................................. 5
4. Rekey of IPsec SAs ........................................... 7
   4.1. Single IPsec Device Rekey ............................... 8
   4.2. Simultaneous Rekey of IPsec Devices .................. 10
5. IPsec Database Generation .................................... 13
   5.1. The Security Policy Database (SPD) .................... 13
   5.2. Security Association Database (SAD) ................... 13
   5.2.1. Generating Keying Material for IPsec SAs .......... 13
   5.3. Peer Authorization Database (PAD) ..................... 16
6. Policy distributed through the Controller .................... 16
   6.1. IPsec policy negotiation ............................... 17
7. Security Considerations ....................................... 18
8. IANA Considerations .......................................... 19
9. Acknowledgements ............................................. 19
10. References ................................................ 19
   10.1. Normative References ................................. 19
   10.2. Informative References ............................... 20
Appendix A. Example Controller protocols ....................... 21
   A.1. Example: I2NSF ....................................... 21
   A.2. Example: Network Controller ........................... 21
   A.3. Additional controller protocol considerations ........ 22
      A.3.1. Peer-to-peer distribution of IPsec policy ........ 22
      A.3.2. Ordering of messages distributed to a controller .. 23
Appendix B. Choosing whether to use IKEv2 or Controller IKE .... 23
Appendix C. Implementation Considerations ...................... 25
Authors’ Addresses ........................................... 25

1. Introduction

Network architectures typically have included network devices directly communicating using network control protocols such as routing and signaling protocols. Additionally, secured communications between these network devices are usually accomplished with a key agreement protocol such as IKEv2 [RFC7296], in which the network devices directly authenticate each other and agree upon security policy and keying material to protect communications between themselves. However, controller-based network architectures (sometimes called “Software-Defined Networking”) are now being defined [RFC7426] [RFC8192] and implemented. In controller-based network architectures, control protocols --including key exchange
protocols -- are not implemented directly between the network devices. Software-Defined Networks utilize the controller based network design while maximizing the scalability that it provides. The result is a significantly different trust model; rather than apply a peer-to-peer trust model, the network applies a device-to-controller trust model.

The use of IKEv2 in a device-to-controller trust model is not always optimal. Instead, a new key management method is needed for these models. Appendix B describes situations in which Controller IKE may be a better choice than IKEv2.

![Diagram of Controller-based Secure Communications]

Figure 1: Controller-based Secure Communications

Figure 1 shows an example controller based network design. Three network devices (labeled A, B, and C) setup a protected control plane connection to a Controller. The Controller distributes policy to the network devices, which enables them to securely communicate in the data plane.

When one considers adding a controller to a key exchange method, it is tempting to give it the task of generating and distributing session keys directly to network devices. However, such a design has several security considerations. Because such a controller would have all session keys it could become an active participant or a passive monitor to the secured communications. Also, for scalability reasons one might consider having a controller distribute session keys that are group keys, either a single group key or a set of group keys.
keys that devices use to protect communications between them. This document does not specify the use of group session keys.

Many key exchange methods (such as IKEv2) use a Diffie-Hellman (DH) algorithm to derive keys. When combined with an authentication method, the key exchange method allows two network devices to generate private pair-wise keys with each other. This document presents a key exchange method making use of the device-to-controller trust model, where a controller is used to distribute keying material and policy between network devices, also resulting in the devices generating private pair-wise keys with each other. DH public values are provided to controllers from IPsec devices, where the controller relays the DH public values to authorized peers of that IPsec device as defined by a centralized policy. Network devices then create and install private pair-wise IPsec session keys to be used to secure communications with their peers.

Controller-based key exchange methods can be used to create a Gateway-to-Gateway VPN [RFC7018] in either a Full-Mesh Topology or Dynamic Full-Mesh Topology.

Although IKEv2 is not used in this approach, the key management interfaces between IKEv2 and IPsec defined in RFC 7296 are maintained as much as possible.

1.1. Requirements Language

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

2. Overview

In Controller-IKE a controller acts as a trusted third party, which relays policy and keying material between IPsec devices. The controller can be a standalone device, or integrated into a management station. Communications between the controller and the IPsec devices MUST be authenticated, encrypted, and integrity-protected.

All algorithms are selected by the controller or a management station associated with the controller. The combination of a controller and a set of IPsec devices comprises a cooperating group of devices that make up a VPN, where each IPsec device is authorized to communicate with other IPsec devices in the group. Controller policy may allow
an IPsec device to communicate with all other IPsec devices in the group, or may restrict it to a subset of those devices.

DH public values are distributed to the controller from each IPsec device and redistributed from the controller to each authorized peer IPsec device. Each IPsec device creates and maintains a DH pair, which it uses to communicate with other members of the VPN. This distribution of DH public values (and other related values) is intended to be embedded into an existing network device/controller protocol. In particular, Controller-IKE provides a mechanism for secure key management and only key management. It does not provide policy information or configuration as that is assumed to be provided by the controller. One such controller protocol [I-D.ietf-i2nsf-sdn-ipsec-flow-protection] is being developed at this time in the IETF I2NSF working group. Another controller protocol [I-D.sajassi-bess-secure-evpn] is being developed by the IETF BESS working group.

3. Generating Initial IPsec SAs

When an IPsec device begins operation, it generates a DH pair, using an algorithm defined in the IKEv2 Diffie-Hellman Group Transform IDs [IKEV2-IANA]. If the device does not have any active peers it simply distributes its DH public value to the Controller, along with a nonce to be used during SA creation. Whenever a DH pair is created, a new nonce MUST also be created. Whenever DH public values are transmitted, they are transmitted with the corresponding nonce. Whenever a DH private or DH public value is used, it is used along with the corresponding nonce. However, in the diagrams and descriptions below, the nonces are often left out for the sake of clarity.

Upon receiving a peer’s DH public value and nonce, the receiver creates IPsec SAs (as described in Section 5.2). For each peer, a pair of IPsec SAs are created by combining the IPsec device’s own DH private value with the DH public number received from the Controller.
Figure 2: Generation of Initial IPsec SAs between two peers

Figure 2 shows IPsec SA generation between a pair of IPsec devices. Two IPsec devices (A and B shown in Figure 1) join the network. Each creates its own DH pair (labelled "a1" on A and "b1" on B), and distributes the DH public value (labelled a1-pub and b1-pub) to the Controller. The controller forwards the DH public value to all authorized peers, although for simplicity of exposition the figure only shows the two IPsec devices.

When each device receives the peer’s DH public value, a pair of IPsec SAs are generated: one outbound and one inbound. As shown in the figure, A generates an outbound SA labeled Tx(a1-b1), representing that it has been generated using A’s DH pair labeled a1 and B’s DH pair labeled b1. B generates the same IPsec SA as an inbound SA, which is labeled Rx(a1-b1). Similarly, A generates an inbound IPsec SA labelled Rx(b1-a1), which is the same IPsec SA on B labelled Tx(b1-a1).

This process repeats on both A and B as they discover other IPsec devices with which they are authorized to communicate.
4. Rekey of IPsec SAs

Any IPsec device may initiate a rekey at any time. Common reasons to perform a rekey include a local time or volume based policy, or may be the result of a cipher counter mode Initialization Vector (IV) counter nearing its final value. The rekey process is performed individually for each remote peer. If rekeying is performed with multiple peers simultaneously, then the decision process and rules described in this rekey are performed independently for each peer.

A decision process choosing an outbound IPsec SA is followed when certain events occur, as described in the rules below. The same decision process is followed regardless of whether the device is performing a rekey or responding to a peer’s rekey. The decision process is:

1. Determine the outbound SAs with the remote peer’s most recently distributed DH public value.

2. Determine which of those outbound SAs are "live". A "live" outbound SA is one built from a DH value from the local peer for which it has observed inbound traffic using any SA based on the same local DH pair. This proves that the remote peer is prepared to receive traffic protected by that DH pair.

3. Choose the "live" outbound SA built from the local peer’s most recent DH public value.

A rekey operation follows these four basic rules.

Rule 1 When an IPsec device needs to perform a rekey with a remote peer, it creates a new pair of IPsec SAs by combining the new DH private value with the peer’s DH public values. If the remote peer is also in the midst of a rollover and its DH public value has already been received, then this may result in creating two sets of SAs: one pair with the remote peer’s old DH public value, and one pair with the remote peer’s new DH public value.

Rule 2 When an IPsec device receives a new remote peer’s DH public value from the controller it creates and installs a new pair of IPsec SAs by combining the remote peer’s new DH public value with its own current local DH private values. If both devices are in the midst of a rollover, this may result in creating two sets of SAs with the remote peer’s new DH public value: one with the local old DH private value, and one with the local new DH private value. The outbound SA decision process is performed.
Rule 3  The first IPsec packet received by a rekeying IPsec device on an inbound SA using its new DH pair causes it to perform the outbound SA decision process. It may also shorten the lifetime of IPsec SAs using its own old DH pair that are shared with this peer, as they are no longer in use (other than the inbound SA might receive packets in transit).

Rule 4  The first IPsec packet received from a remote rekeying IPsec device using the remote peer’s new DH pair allows the IPsec device to shorten the lifetime of IPsec SAs shared with this peer using unused remote DH pairs.

Two examples follow: a single IPsec device performing a rekey with its peers, and two IPsec devices performing a simultaneous rekey. The same rekey operations described above are exhibited in both cases.

4.1. Single IPsec Device Rekey

When a single IPsec device begins a rekey, it first generates a new DH pair and generates new IPsec SA pairs for each peer with which it is communicating. It does this by combining the new DH private value with each peer’s existing DH public value. Only when the new IPsec SAs have been installed and the device is prepared to receive on those new SAs does it then distribute the new DH public value to the Controller, which forwards the new DH public value to its authorized peers. The rekeying IPsec device continues to transmit on the old SAs for each peer until it observes that peer begin to transmit on the new SAs.
In Figure 3, device A is shown as performing a rekey, and it creates a DH pair labelled "a2". The following steps are followed.

1. Rule 1 requires creating new IPsec SAs for each peer. In this example, A creates a new outbound IPsec SA to communicate with B labelled Tx(a2-b1), and a new inbound IPsec SA labelled Rx(b1-a2).
Rx(b1-a2). A continues to transmit on Tx(a1-b1) (generated as shown in Figure 2).

2. A distributes the new public value (a2-pub) to the Controller who forwards it to A’s authorized peers, which includes B. During this time, both A and B continue to use the initial IPsec SAs setup between them using a1 and b1.

3. When B receives a2 from the controller, B follows Rule 2 by creating Tx(b1-a2), Rx(a2-b1). B also follows the outbound SA decision process, which causes it to change its outbound IPsec SA to A to Tx(b1-a2).

4. When A receives a packet protected by Rx(b1-a2), it follows Rule 3 and performs the outbound SA decision process. This causes it to change its outbound IPsec SA to Use Tx(a2-b1). It also optionally shortens the lifetime of the old IPsec SAs shared with this peer.

5. When B receives a packet protected by Tx(a2-b1), it follows Rule 4, in which it may shorten the lifetime of the old IPsec SAs shared with this peer using DH pairs that are no longer in use.

At the end of the rekey, both A and B retain a single DH pair, and a single set of IPsec SAs between them.

4.2. Simultaneous Rekey of IPsec Devices

When two or more IPsec device simultaneously begin a rekey, they each follow the rekeying method described in the previous section. Every rekeying IPsec device generates a new DH pair and generates new IPsec SA pairs for each peer with which it is communicating by combining their new DH private value with each peer’s existing DH public value. When this completes on a particular IPsec device, it distributes the new DH public value to the Controller, which forwards it to its authorized peers. Each continues to transmit on the existing SAs for each peer until it observes that peer transmitting on the new SAs. During a simultaneous rekey up to four pairs of IPsec SAs may be temporarily created, but the four rules ensure that they converge on a single new set of IPsec SAs.
In Figure 4, device A and device B are both shown as performing a rekey. Their initial state corresponds to the final state shown in
Figure 2 (i.e., they are communicating using a single pair of IPsec SAs created from DH pairs "a1" and "b1".

1. A and B follow Rule 1, which includes creating new IPsec SAs for each peer. In this example, A creates a new outbound IPsec SA to communicate with B labelled Tx(a2-b1), and a new inbound IPsec SA labelled Rx(b1-a2). B creates a new outbound IPsec SA to communicate with B labelled Tx(a1-b2), and a new inbound IPsec SA labelled Rx(b2-a1). A and B continue to transmit on IPsec SAs previously created from DH pairs "a1" and "b1".

2. A distributes the new public value (a2-pub) to the Controller who forwards it to A’s authorized peers, which includes B. B also distributes the new public value (b2-pub) to the Controller who forwards it to B’s authorized peers, which includes A.

3. When A and B receive each other’s new peer DH public value from the controller they follows Rule 2. But because now there are four DH values that could be in used between A and B, they must be prepared to use IPsec SAs using each permutation of DH values: a1-b1, a1-b2, a2-b1, a2-b2. Prior to implementing Rule 2, each has already created sets of IPsec SAs matching two of the permutations, so just two more sets must be generated during Rule 2.

   * One pair is created using the IPsec device’s old DH pair with the peer’s new DH pair. This is necessary, because the peer may transmit on this pair.

   * One pair is created using the IPsec device’s new DH pair with the peer’s new DH pair. This is the set of IPsec SAs that will be used at the end of the rekey process.

Each peer begins transmitting on an IPsec SA that combines the remote peer’s new DH pair and its own old DH pair, which is the most recent "live" SA on which it can transmit. I.e., A begins transmitting on Tx(a1-b2) and B begins transmitting on Tx(b1-a2).

4. When A receives a packet protected by Rx(b1-a2), it understands that the remote peer has received its new DH public value. A also understands that because of Rule 2 that B must have created IPsec SAs using a2-b2. This allows A to follow Rule 3 and change its outbound IPsec SA to Use Tx(a2-b2). Similarly, when B receives a packet protected by Rx(a1-b2), B recognizes that it can also begin to transmit using Tx(b2-a2). Note that it also possible that A will receive a packet protected by Rx(b2-a2) or B will receive a packet protected by Rx(a2-b2), and then knows it can transmit on an IPsec SA using both of the new DH pairs.
5. Also in Rule 3, Both A and B optionally shorten the lifetime of older IPsec SAs shared with this peer derived from unused DH pairs to be cleaned up. A shortens the lifetime of SAs based on a1. B shortens the lifetime of SAs based on b1.

6. When A and B receive a packet protected by the remote peer’s latest DH pair, they shorten the lifetime of SAs based on the remote peer’s unused DH pair.

5. IPsec Database Generation

The PAD, SPD, and SAD all need to be setup as defined in the IPsec Security Architecture [RFC4301].

5.1. The Security Policy Database (SPD)

The SPD is implemented using methods outside the scope of this document. The SPD describes the type of traffic that will be protected between IPsec devices and the policy (e.g., ciphers) used to create SAs.

5.2. Security Association Database (SAD)

The SAD is constructed from IPsec policy (e.g., ciphers) obtained (depending on the controller protocol method) either from the controller or distributed by a peer (see Section 6).

Keying Material is generated following the method defined in IKEv2, and depends on SPIs, nonces, and the Diffie-Hellman shared secret.

The following sections describe how the necessary values are determined.

5.2.1. Generating Keying Material for IPsec SAs

5.2.1.1. g^ir

A DH public value is distributed from the peer.

A DH shared secret (g^ir) is computed using the peer’s public value, and the device’s private value. The DH group to be used must be known by the device. Options include distribution by an SDN controller, or distribution by the peer with the DH public value (see Section 6).
5.2.1.2. Nonces

Nonces are distributed with a DH public value, and are used only with that value. It is RECOMMENDED that nonces are generated as described in Section 2.10 of [RFC7296].

IKEv2 Key derivation specifies an initiator’s nonce (Ni) and a responder’s nonce (Nr). While neither peer is truly initiating a session, in order to fit the IKE key material models the roles must be assigned. The initiator is chosen as the peer with the larger nonce and the responder is the peer with the smaller. This does mean that the roles can change for each rekey and for each SA within a rekey.

5.2.1.3. SPIs

SPI values that are unique to each generation of keying material need to be determined. While each peer could distribute its own inbound SA value, the SPI value would be used by many peers. Although this is not a problem for an SA lookup (lookup can include the source and destination IP addresses), experience has shown that this is sub-optimal for some hardware SA lookup algorithms. Instead, this specification proposes generating values that are unpredictable and indistinguishable from randomly-generated SPI values.

SPI values are generated using the IKEv2 prf+ function, where nonces are used as the input to the prf. This produces a statistically random SPI value that should be unique. However, with a 32 bit value there is still a very small, but non-zero, chance of SPIs repeating for a given pair of peers. To prevent this and ensure uniqueness in the operational window, we also use the lower 2 bits from each peer’s rekey counter.

First the SPIs are taken from the prf+ function as 32 bit values and assigned based on which peer is taking the role of initiator and which is taking the role of responder. The p_SPI_i is taken by the device providing Ni, where p_SPI_r is taken by the other device.

\[
\{ p_{SPI_i} \mid p_{SPI_r} \} = \text{prf+(} Ni \mid Nr, "SPI generation")
\]

Next p_SPI_i and p_SPI_r are mapped from initiator and responder roles to local and remote roles based on the choice of Ni and Nr in 5.2.1.2 and are renamed to p_SPI_local and p_SPI_remote.

Then, 2 2-bit Rotation Numbers (RN) are generated from the 2 least significant bits (LSB) of the 2 rekey counter values (see Section 6). These 4 bits replace the least significant bits of p_SPI_local and p_SPI_remote with the local RN bits taking the least significant
position in p_SPI_local and the remote RN bits taking the least significant position in p_SPI_remote. This shown in the following two diagrams with RN_local shown as R_l and RN_remote shown as R_r.

```
(MSB)       (LSB)
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              p_SPI_local bits from prf+               |R_r|R_l|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
``` (MSB)       (LSB)
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              p_SPI_remote bits from prf+             |R_l|R_r|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

The reason for changing terminology from initiator/responder to local/remote is because the roles of initiator/responder can change in every rekey. The order of RN_local and RN_remote needs to remain constant. If that order was based on initiator/responder, there’s a risk that if the initiator and responder roles changed and the two peers re-keyed on different frequencies, they could end up with identical RN values.

In some circumstances additional values may also need to be added to the prf for peers to ensure that they have implemented the same policy. Appendix A.3.1 includes a discussion of when this might be needed. In these cases, only the prf+ inputs are modified and the Rotation Numbers MUST still be added as above.

Because a device is not choosing its inbound SPI, its SA lookup process needs to be aware that duplicates could occur across different peers. In that case, the inbound SA Lookup SHOULD include a source IP address in addition to the SPI value (see Section 4.1 of [RFC4301]).

5.2.1.4. IPsec key generation

As described in previous sections, a DH public value and a nonce are distributed by peers. These are used to generate IPsec keys following the method defined in the IKEv2. SKEYSEED is generated following Section 2.14 of [RFC7296]:

\[
\text{SKEYSEED} = \text{prf}(N_i \mid N_r, g^{ir})
\]
KEYMAT can be similarly derived as defined by IKEv2 (Section 2.17 of [RFC7296]), although only SK_d is required to be generated (shown in Section 2.14 of [RFC7296]).

\[ SK_d = prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr) \]

\[ KEYMAT = prf+(SK_d, Ni | Nr) \]

However, with the simplification where only SK_d is generated, it can be observed that the derivation of SK_d could be skipped entirely, and an optimized derivation of KEYMAT could be as follows:

\[ KEYMAT = prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr) \]

Note: A single specification for generating KEYMAT will be determined in a future version of this document.

5.3. Peer Authorization Database (PAD)

The PAD identifies authorized peers. PAD entries are either statically configured, or may be dynamically updated by the controller.

The PAD omits authentication data for each peer, because it has delegated authentication and authorization to the controller.

The controller protocol MUST be able to describe an identity that a receiver can match against its local PAD database, to ensure that the peer is an authorized peer.

6. Policy distributed through the Controller

An IPsec device distributes to a controller a DH public value and the associated information and policy needed to create IPsec SAs in a Device Information Message (DIM). The controller then distributes the DIM to all authorized peers of that device. The following data elements MUST be embedded in a DIM message:

- DH public number (used for key computation)
- Nonce (used for key computation and SPI generation)
- Peer identity (used to identify a peer in the PAD)
- An Indication whether this is the initial distributed policy
- A rekey counter, which increases for each unique DIM sent
In cases where a single fixed IPsec policy has been pre-distributed, it is not necessary for the peer to send or receive that policy in a DIM. However, in cases where an IPsec device needs to indicate the policy it is willing to use, the following data elements SHOULD be included in a DIM:

- An IPsec policy or policies
- A lifetime bounding the use of the DH public number. When this DH public number is used to create an IPsec SA, the shortest lifetime is used as an SA lifetime for the pair of generated IPsec SAs. When the lifetime expires, the local version of the DIM and IPsec SAs generated from it MUST be deleted.

Appendix A suggests different ways that this policy may be included in a controller protocol. This document does not define a normative protocol format, because the DIM very likely needs to be integrated into an existing controller protocol rather than be an independent key management protocol. However, the controller protocol MUST provide a strong authentication between the device and the controller, and integrity of the messages MUST be provided. Confidentiality of the messages SHOULD also be provided. It is important that the controller protocol be protected with algorithms that are at least as strong as the algorithms used to protect the IPsec packets.

### 6.1. IPsec policy negotiation

In many controller based networks, there is a single IPsec policy used by all devices and there is no need to redistribute or select policy details. However, in some circumstances, there may be a need to have multiple policy options. This could happen when a controller changes the policy and wants to smoothly migrate all devices to the new policy. Or it could happen if a network supports devices with different capabilities. In these cases, devices need to be able to choose the correct policy to use for each other device, and must do this without sending additional messages and without sending individual messages to each peer. When a device supports multiple policies, it MUST include those policies within the DIM. This is done by sending multiple distinct policies, in order of preference, where the first policy is the most preferred. The policy to use is selected by taking the receiver’s list of policies (i.e., the list advertised by the device that generates Nr), starting with the first policy, compare against the initiator’s (device that generates Ni) list, and choosing the first one found in common. The method conforms to the IKEv2 Cryptographic Algorithm Negotiation described in Section 2.7 of [RFC7296]. (However, see additional discussion when IKEv2 payloads are used in Appendix A.3.1).
If there is no match, this indicates a controller configuration error. These devices MUST NOT establish new SAs until a DIM is received that does produce a match.

When a device supports more than one DH group, then a unique DH public number MUST be specified for each in order of preference. The selection of which DH group to use follows the same logic as Policy selection, using the receiver’s list order until a match is found in the initiator’s list.

7. Security Considerations

This document proposes that a device re-use an ephemeral Diffie-Hellman exponential with multiple peers. There are some known potential vulnerabilities to this approach, which can be mitigated by the device first validating a peer’s public value to be a safe public value before combining its own private value with it. The tests which MUST be performed are described in [RFC6989]. See [REUSE] for additional security considerations when reusing ephemeral Diffie-Hellman keys.

A controller acts as a "trusted third party", which asserts that a particular Diffie-Hellman public value is associated with a particular entity. A device receiving the public key is not required to validate the assertion.

A subverted controller can act as a "man-in-the-middle" between a pair of devices. The easiest attack would be for the attacker to adjust the routing for the desired traffic through a compromised gateway and directly observe the cleartext. It is also possible that a subverted controller could provide a device with a Diffie-Hellman public value that actually belongs to a compromised gateway rather than the intended gateway, but doing so does not seem to be necessary. Nonetheless, the attack of a subverted controller can be mitigated by having a device sign its Diffie-Hellman public value (e.g, as a CMS Signed data object), where the receiver validates the digital signature on the object. However, this adds significant processing cost to a rekey and does not fit the controller-based network architecture model.

A subverted IPsec device whose DH pair has been compromised would be vulnerable to all of its IPsec traffic using that DH pair being compromised. Assuming the use of strong DH algorithms (including quantum resistant algorithms as they become available), the compromise would most likely be due to the device itself being compromised. Such a compromised device is also vulnerable to a direct plaintext compromise.
PFS is achieved between rekey periods, as DH pairs are required to be generated independently. However, because a device uses the same long-term key to generate session key with multiple peers, there is no PFS between sessions within the same rekey period. To reduce key exposure outside of a rekey period, when a connection is closed each endpoint MUST forget not only the keys used by the connection but also any information that could be used to recompute those keys. However, the DH private key value and the nonce distributed with it may be forgotten only once the last IPsec SA that uses the private key value is removed from the SAD and there is no chance that a new IPsec SA could be setup that requires the private key value.

If quantum resistance is considered to be an issue, the controller can distribute a PSK, which could be used to create the SK_d in the manner shown in [I-D.ietf-ipsecme-qr-ikev2].

8. IANA Considerations

This memo specifies no IANA actions.

9. Acknowledgements

Graham Bartlett provided many useful comments and suggestions. Rafa Marin-Lopez and Gabriel Lopez-Millan provided valuable reviews and advice regarding SDN use cases.

10. References

10.1. Normative References


10.2. Informative References


Appendix A. Example Controller protocols

This section contains suggestions of how a Controller protocol might distribute policy for the Controller-based IKE.

A.1. Example: I2NSF

IPsec devices described in this document could be implemented as an Network Security Function (NSF) in the I2NSF Framework [RFC8329]. An I2NSF Controller or NSF Manager could distribute a DIM as a new type of I2NSF Policy Rule. A YANG configuration data model would need to be defined for this. This could be a new "Case 3" defined in [I-D.ietf-i2nsf-sdn-ipsec-flow-protection].

A.2. Example: Network Controller

Site-to-site networks (e.g., an L2VPN or L3VPN) often use a network controller to share networking state between routers. When those routers use IPsec to protect the communications between themselves, this same network controller could distribute DH public number and nonces as well. For example, when a BGP Route Reflector (RR) is used in a network, a new address family (AFI) could distribute the state necessary for a controller-based IPsec key exchange. The BGP session between BGP routers and the Route Reflector (RR) would need to at least be integrity protected from a man in the middle and SHOULD be protected for confidentiality to prevent identity leakage.

The controller protocol MUST provide for adequate synchronization of the state. For example, when IPsec devices are synchronized with a key management protocol it is often necessary for the protocol to indicate when a device has rebooted and thinks that it is contacting peers for the first time. This alerts peers to destroy earlier keying state that they may still believe is current.

One possible method for distributing a DIM within a controller protocol is to use a set of IKEv2 payloads. For example, when a single set of IPsec policy has been distributed to all IPsec devices...
by a configuration server then the following minimum required data elements can be distributed using the following IKEv2 payloads.

\[ \text{ID, } \lbrack \text{N(INITIAL\_CONTACT)}, \rbrack \text{ KE, Ni} \]

When initiating a rekey, the IPsec device need only distribute its new DH public number due to Rule 1. Existing peers receiving the new DH public number need not be re-told about the previous DH public number. Any new peer that receives and acts upon a "stale" controller message (containing the old DH public number) will still be able to setup IPsec SAs using the old DH public number, and can use them until the new DH public number is received.

A.3. Additional controller protocol considerations

If the controller protocol is more complicated, there are some additional considerations.

A.3.1. Peer-to-peer distribution of IPsec policy

In some cases an IPsec device may have more than one IPsec policy under which it is willing to communicate. This would result in an IPsec device using the decision process described in Section 6.1 to determine which policy to use between itself and that peer. An IKEv2 SA payload could be used to distribute the policies, and the decision process could be used to determine which single set of policy is to be used. Note that the same decision process is followed by both peers. It is important that when an SA payload is used, that each proposal within the SA payload MUST contain at most a single transform for each Transform type (e.g., ENCR and (optionally) ESN for combined mode algorithms, ENCR, INTEG, and (optionally) ESN otherwise). This is due to the absence of a direct peer-to-peer reply message, which would alert the sender of which proposal was chosen.

1. Determine the Responder (as defined in Section 5.2.1.2).

2. Follow the negotiation rules defined in Section 2.7 of IKEv2 [RFC7296] (with the restrictions that more than one transform of each type MUST NOT be present, and no error notifications are returned to the peer). Each peer will independently compare each Proposal in the Responder’s SA payload to each Proposal in the Initiator’s SA payload and choose the first match.

3. If there is no match, then this is considered a controller error, and the IPsec devices SHOULD report the error to the controller.

Payloads distributed in the controller protocol could be as follows:
where the SA payload contains one or more Proposals, each of this includes a transform indicating the Diffie-Hellman group it is willing to use (D-H Transform), and IPsec transforms that it is willing to use (e.g., ENCR, INTEG, and ESN Transforms). The KE payload includes a DH public number matching the D-H Transform.

Because there is no direct peer-to-peer IKE messages, there is a need for peers to reliably know which Proposal in the SA payload was chosen. That is, if they do not reliably follow the same decision process they may end up installing IPsec SAs with incompatible policy. A straightforward method to verify that a peer has chosen the same policy is to include the SA Proposal number (SPN) from the SA payload in the SPI calculation.

\[
{p_{SPI_i}|p_{SPI_r}} = prf+(Ni \mid Nr, "SPI generation" \mid SPNi \mid SPNr)
\]

If a device is willing to use more than one DH group, then a single SA payload should be distributed, but the included Proposals may contain different D-H Transforms. A KE payload must be included for each D-H Group that is offered in the SA payload.

A.3.2. Ordering of messages distributed to a controller

A controller protocol may require a method of determining ordering of messages that are distributed, i.e. a Rekey Counter (RC). It is RECOMMENDED that the ordering be defined by a monotonically increasing counter value distributed with the IPsec policy. It is further RECOMMENDED that to ensure ordering after a device reboot that the counter include a "boot count", which increments after each reboot. For example, the counter could be a 64-bit counter where the high order 32 bits are a "boot count", followed by the counter that begins at 1 following the increment of the "boot count".

Appendix B. Choosing whether to use IKEv2 or Controller IKE

The following list describes the circumstances in which Controller IKE may be preferable to IKEv2. Note that Controller IKE does not replace IKEv2, but does provide an alternative for some network architectures where it is more optimal.

<table>
<thead>
<tr>
<th>Trust Model</th>
<th>Controller IKE is optimal when a device-to-controller trust model is in use. IKEv2 is a better approach when IPsec devices require a peer-to-peer trust model.</th>
</tr>
</thead>
</table>

Carrel & Weis Expires September 12, 2019 [Page 23]
Latency

Controller IKE reduces tunnel session setup latency in a device-to-controller trust model. Once controller communications have commenced, a session can be initiated with any other IPsec device managed by that controller without requiring additional key management messages. This is optimal when a group of IPsec devices are sensitive to latency.

Load Balancing

In some network architectures a full mesh of IKEv2 sessions can occur without affecting the load of IPsec and IKEv2 processing on any of the communicating IPsec devices, including having protocol state machinery to handle an IPsec peer device that is overloaded and not reliably responding. But when a set of IPsec devices is very large, this can be problematic. Also, when an IPsec device is overloaded there may be re-transmissions of IKEv2 messages, which further complicates protocol state. The simplified control plane of Controller IKE makes load balancing a purely local matter, where the installation of IPsec SAs takes into consideration only available local resources. And because there are no peer-to-peer key management messages, no re-transmissions occur.

Complexity

Full attribute negotiation is not needed in a controller environment. Controllers enforce the SA policy details, moving complexity away from end nodes. This also reduces the attack surface on the end node.

Network Shape

In some network topologies a persistent bi-directional link does not exist between all peers. Sometimes one direction has significantly reduced capabilities or is even non-existent. This can be either temporary or permanent, and sometimes is a purposefully enforced restriction. Provided that both peers can communicate with the controller, Controller IKE allows for the establishment of SAs and rekeying in these scenarios.
Appendix C. Implementation Considerations

The system architecture of many implementations includes a separation between a data plane "fastpath" and a "control plane". The data plane performs IPsec encapsulation and decapsulation in the simplest and most expedient way possible, where the control plane handles the complexity of network protocols including state machines, timers, network communication, and managing the data plane.

A typical IKEv2 implementation on Linux works this way, with IKEv2 running in user space and IPsec packet processing happening in the kernel. The kernel, or other fast path implementation, provides an interface for the control plane to manage it. This interface includes a way to create SAs, delete SAs, and observe statistics for SAs. Controller IKE is designed to be able to work with these same interfaces. For example a user space implementation of Controller IKE could work with a Linux kernel implementation of the IPsec data plane without needing kernel modifications. SA creation and deletion remains the same. SA creation occurs with Rules 1 and 2. SA deletion happens with Rules 3 and 4. Additionally Rules 3 and 4 need to observe that traffic has arrived on a particular SA. This can be done by observing packet counts on an SA and seeing when they go from zero to any positive number. Due to the asynchronous nature of Controller IKE, Rules 3 and 4 do not require immediate action when the first packets arrive, but instead they can be implemented with relaxed polling.

Authors’ Addresses

David Carrel  
Cisco Systems  
170 W. Tasman Drive  
San Jose, California 95134-1706  
USA  
Phone: +1-408-525-7852  
Email: carrel@cisco.com

Brian Weis  
Independent  
USA  
Email: bew.stds@gmail.com
BGP Usage for SDWAN Overlay Networks
draft-dunbar-bess-bgp-sdwan-usage-01

Abstract

The document describes three distinct SDWAN scenarios and discusses the applicability of BGP for each of those scenarios. The goal of the document is to make it easier for future SDWAN control plane protocols discussion.

SDWAN edge nodes are commonly interconnected by multiple underlay networks that are owned and managed by different network providers. A BGP-based control plane is chosen for handling large number of SDWAN edge nodes with little manual intervention.

Status of this Memo

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that
other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on January 8, 2009.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction...................................................3
2. Conventions used in this document..............................4
3. Use Case Scenario Description and Requirements.................5
   3.1. Requirements..............................................6
      3.1.1. Client Service Requirement...........................6
      3.1.2. SDWAN Node Provisioning..............................6
   3.2. Scenarios #1: Homogeneous WAN.............................8
   3.3. Scenario #2: SDWAN WAN ports to VPN’s PEs and to Internet.9
1. Introduction

An "SDWAN" network consists of many segments of parallel paths over different underlay networks, some of which are private networks over which traffic can traverse without encryption, others require encryption over untrusted public networks.

[Net2Cloud-Problem] describes the network related problems that enterprises face today in transitioning their IT infrastructure to support a digital economy, such as the need to connect enterprises’ branch offices to dynamic workloads in different Cloud DCs, or aggregating multiple paths provided by different service providers to achieve better performance.

Even though SDWAN has been positioned as a flexible way to reach dynamic workloads in third party data centers over multiple underlay networks, scaling becomes a major issue when there are hundreds or thousands of nodes to be interconnected by the SDWAN overlay paths.
BGP is widely used by underlay networks. This document describes using BGP to enhance the scaling properties of SDWAN overlay networks.

2. Conventions used in this document

Cloud DC: Third party data centers that host applications and workloads owned by different organizations or tenants.

Controller: Used interchangeably with SDWAN controller to manage SDWAN overlay path creation/deletion and monitor the path conditions between sites.

CPE: Customer Premise Equipment

CPE-Based VPN: Virtual Private Secure network formed among CPEs. This is to differentiate from more commonly used PE-based VPNs [RFC 4364].

Homogeneous SDWAN: A type of SDWAN network in which all traffic to/from the SDWAN edge nodes has to be encrypted regardless of underlay networks. For lack of better terminology, we call this Homogeneous SDWAN throughout this document.

ISP: Internet Service Provider

NSP: Network Service Provider. NSP usually provides more advanced network services, such as MPLS VPN, private leased lines, or managed Secure WAN connections, many times within a private trusted domain, whereas an ISP usually provides plain internet services over public untrusted domains.

PE: Provider Edge

SDWAN End-point: a port (logical or physical) of a SDWAN edge node.
SDWAN: Software Defined Wide Area Network. In this document, "SDWAN" refers to the solutions of pooling WAN bandwidth from multiple underlay networks to get better WAN bandwidth management, visibility & control. When the underlay networks are private, traffic can traverse without additional encryption; when the underlay networks are public, such as the Internet, some traffic may need to be encrypted when traversing through (depending on user provided policies).

SDWAN IPsec SA: IPsec Security Association between two SDWAN ports or nodes.

SDWAN over Hybrid Networks: SDWAN over Hybrid Networks typically have edge nodes utilizing bandwidth resources from multiple service providers. In Hybrid SDWAN network, packets over private networks can go natively without encryption and are encrypted over the untrusted network, such as the public Internet.

WAN Port: A Port or Interface facing an ISP or Network Service Provider (NSP), with address (usually public routable address) allocated by the ISP or the NSP.

C-PE: SDWAN Edge node, which can be CPE for customer managed SDWAN, or PE that is for provider managed SDWAN services.

ZTP: Zero Touch Provisioning

3. Use Case Scenario Description and Requirements

SDWAN networks can have different topologies and have different traffic patterns. To make it easier for the focused discussion in subsequent drafts on SDWAN control plane and data plane, this section describes several SDWAN scenarios that may have different need or impact to their corresponding control planes & data planes.
3.1. Requirements

3.1.1. Client Service Requirement

Client interface of SDWAN nodes can be IP or Ethernet based.

For Ethernet based client interfaces, SDWAN edge should support VLAN-based service interfaces (EVI100), VLAN bundle service interfaces (EVI200), or VLAN-Aware bundling service interfaces. EVPN service requirements are applicable to the Client traffic, as described in the Section 3.1 of RFC8388.

For IP based client interfaces, L3VPN service requirements are applicable.

3.1.2. SDWAN Node Provisioning

Unlike traditional EVPN or L3VPN where PEs are deployed for long term, SDWAN edge nodes (virtual or physical) deployment at a specific location can be ephemeral. Therefore, Zero Touch Provisioning (ZTP) is a common requirement for SDWAN. ZTP for SDWAN can include many areas, but from network connectivity perspective, ZTP should include the following:

- Upon power up, an SDWAN node can reach a central SDWAN Controller (which can be burned or preconfigured in the device) via a TLS or SSL secure channel.

- The Central SDWAN Controller can designate a Local Network Controller in the proximity of the SDWAN node; the Local Network Controller and the SDWAN nodes might be connected by third party untrusted network. The Local controller does all the following 4 tasks:

  1) ZTP
  2) Auto-discovery of Network
  3) (Auto)-Provisioning for IPsec SAs (initial provisioning part)
  4) Signaling of tenant’s routes/info

BGP is well suited for (4), using Route Reflector (RR) [RFC4456] to propagate network information among SDWAN edge nodes. The SDWAN
node can establish a secure connection (TLS, SSL, etc) to the Local Network Controller (RR).

<table>
<thead>
<tr>
<th>Peer Group 1</th>
<th>RR</th>
<th>Peer Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td></td>
<td>/</td>
</tr>
<tr>
<td>/</td>
<td></td>
<td>/</td>
</tr>
<tr>
<td>+----+</td>
<td>RR</td>
<td>+----+</td>
</tr>
<tr>
<td>C-PE 1</td>
<td></td>
<td>C-PE 2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C-PE 3</td>
<td></td>
<td>C-PE 4</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 1: Peer Groups managed by Local Controller

The SDWAN nodes (a.k.a. C-PEs throughout this document) belonging to the same Tenant can be far apart and can be connected by third party untrusted networks. Therefore, it is not appropriate for a SDWAN edge node (C-PE) to advertise its SDWAN Port properties to its neighbors. Each C-PE propagates its SDWAN Port attributes via the secure channel (TLS, SSL, etc.) established with the Local Controller.

C-PE-1 should include the following aspects in addition to managing client routes:
- Register the SDWAN node’s WAN port <-> local address mapping to its Local Controller. The Local Controller propagates the information to C-PE2 & C-PE3.
- Exchange IPsec property (capability such as the supported encryption algorithms, etc.) and ports NAT property (e.g. private addresses or dynamically assigned IP addresses) with the Local Controller.
- C-PE2 and C-PE3 can establish IPsec SA with the C-PE1 after receiving the information from the Local Controller.
- Then distribute the routes attached to the C-PE to its authorized peers.

Tenant separation is achieved by the Local Controller creating different Tenant based Peer Groups.
3.2. Scenarios #1: Homogeneous WAN

This is referring to a type of SDWAN network with edge nodes encrypting all traffic over WAN to other edge nodes, regardless of whether the underlay is private or public. For lack of better terminology, we call this Homogeneous SDWAN throughout this document.

Some typical scenarios for the use of a Homogeneous SDWAN network are as follows:

- A small branch office connecting to its HQ offices via the Internet. All sensitive traffic to/from this small branch office has to be encrypted, which is usually achieved using IPsec SAs.

- A store in a shopping mall may need to securely connect to its applications in one or more Cloud DCs via the Internet. A common way of achieving this is to establish IPsec SAs to the Cloud DC gateway to carry the sensitive data to/from the store.

As described in [SECURE-EVPN], the granularity of the IPsec SAs for Homogeneous SDWAN can be per site, per subnet, per tenant, or per address. Once the IPsec SA is established for a specific subnet/tenant/site, all traffic to/from the subnets/tenants/site are encrypted.
One of the key properties of homogeneous SDWAN is that the SDWAN Local Network Controller (RR) is connected to C-PEs via untrusted public network, therefore, requiring secure connection between RR and C-PEs (TLS, DTLS, etc.).

Homogeneous SDWAN has some similarity to commonly deployed IPsec VPN, albeit the IPsec VPN is usually point-to-point among a small number of endpoints and with heavy manual configuration for IPsec between end-points, whereas an SDWAN network can have a large number of end-points with an SDWAN controller to manage requiring zero touch provisioning upon powering up.

Existing Private VPNs (e.g. MPLS based) can use homogeneous SDWAN to extend over public network to remote sites to which the VPN operator does not own or lease infrastructural connectivity, as described in [SECURE-EVPN] and [SECURE-L3VPN]

3.3. Scenario #2: SDWAN WAN ports to VPN’s PEs and to Internet

In this scenario, SDWAN edge nodes (a.k.a. C-PEs) have some WAN ports connected to PEs of Private VPNs over which packets can be forwarded natively without encryption, and some WAN ports connected to the Internet over which sensitive traffic have to be encrypted (usually by IPsec SA).
In this scenario, the SDWAN edge nodes’ egress WAN ports are all IP/Ethernet based, either egress to PEs of the VPNs or to the Internet. Even if the VPN is a MPLS network, the VPN’s PEs have IP/Ethernet connections to the SDWAN edge (C-PEs). Throughout this document, this scenario is also called CPE based SDWAN over Hybrid Networks.

Even though IPsec SA can secure the packets traversing the Internet, it does not offer the premium SLA commonly offered by Private VPNs, especially over long distance. Clients need to have policies to specify criteria for flows only traversing private VPNs or traversing either as long as encrypted when over the Internet. For example, client can have these policies for the flows:

1. A policy or criteria for sending the flows over a private network without encryption (for better performance),
2. A policy or criteria for sending the flows over any networks as long as the packets of the flows are encrypted when traversing untrusted networks, or
3. A policy of not needing encryption at all.

If a flow traversing multiple segments, such as A<->B<->C<->D, has either Policy 2 or 3 above, the flow can traverse different underlays in different segments, such as over Private network underlay between A<->B without encryption, or over the public internet between B<->C in an IPsec SA.

As shown in the figure below, C-PE-1 has two different types of interfaces (A1 to Internet and A2 & A3 to VPN). The C-PEs’ loopback addresses and addresses attached to C-PEs may or may not be visible to the ISPs/NSPs. The addresses for the WAN ports can have addresses allocated by the service providers or dynamically assigned (e.g. by DHCP). One WAN port shown in the figure below (e.g. A1, A2, A3 etc.) is a logical representation of potential multiple physical ports on the C-PEs.
Some key characteristics of a Hybrid SDWAN overlay network are as follows:

- one C-PE may be connected to different ISPs/NSPs, with some of its WAN ports addresses being assigned by the ISPs/NSPs.

- The WAN ports connected to PEs of trusted private networks (e.g. MPLS VPN) hand off IP/Ethernet packets, just like today’s CPE that do not handle MPLS packets and do not participate in the underlay VPN networks’ control plane. Traffic can flow natively without encryption when be forwarded out through those WAN ports for better performance.

- The WAN ports connected to untrusted networks, e.g. the Internet, requires sensitive traffic to be encrypted, i.e. encrypted by IPsec SA.
- An SDWAN local Network Controller (RR) is connected to C-PEs via
  the untrusted public network, therefore, requiring secure
  connection between RR and C-PEs via TLS, DTLS, etc.

- The SDWAN nodes’ [loopback] addresses might not be routable nor
  visible in the underlay ISP/NSP networks. Routes & services
  attached to SDWAN edges at the SDWAN overlay layer are in
  different address spaces than the underlay networks.

- There could be multiple SDWAN devices sharing a common property,
  such as a geographic location. Some applications over SDWAN may
  need to traverse specific geographic locations for various
  reasons, such as to comply regulatory rules, to utilize specific
  value added services, or others.

- The underlay path selection between sites can be a local section.
  Some policies allow one service from CPE1 -> CPE2 -> CPE3 using
  one ISP/NSP underlay in the first segment (CPE1 -> CPE2), and
  using a different ISP/NSP in the second segment (CPE2-> CPE3).

- Services may not be congruent, i.e. the packets from A-> B may
  traverse one underlay network, and the packets from B -> A may
  traverse a different underlay.

- Different services, routes, or VLANs attached to SDWAN nodes can
  be aggregated over one underlay path; same service/routes/VLAN can
  spread over multiple SDWAN underlays at different times depending
  on the policies specified for the service. For example, one
  tenant’s packets to HQ need to be encrypted when sent over the
  Internet or have to be sent over private networks, while the same
  tenant’s packets to Facebook can be sent over the Internet without
  encryption.

3.4. Scenario #3: SDWAN WAN ports to MPLS VPN and the Internet

This scenario refers to existing VPN (e.g. MPLS based VPN, such as
EVPN or IPVPN) adding extra ports facing untrusted public networks
allowing PEs to offload some low priority traffic to those ports
facing public networks when the VPN MPLS paths are congested.
Throughout this document, this scenario is also called Internet
Offload for Private VPN, or PE based SDWAN.
In this scenario, it is important that the packets offloaded to untrusted public network be encrypted. In this scenario, there is a secure BGP connection between RR & PEs.

PE based SDWAN can be used by VPN service providers to temporarily increase bandwidth between sites when they are not sure if the demand will sustain for long period of time or as a temporary solution before the permanent infrastructure is built or leased.

---

<table>
<thead>
<tr>
<th>PE1</th>
<th>&lt;====&gt;</th>
<th>RR</th>
<th>&lt;====&gt;</th>
<th>PE3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>++----</td>
<td></td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>++--++</td>
<td></td>
<td>-----</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Additional Internet paths added to the VPN

Here are some key properties for PE based SDWAN:

- For MPLS based VPN, PEs continue having MPLS encapsulation handoff to existing paths.
- The BGP RR is connected to PEs in the same way as VPN, i.e. via the trusted network.
- For the added Internet ports, PEs have IP packets handoff, i.e. sending and receiving IP data frames. Internally, PEs can have the option to encapsulate the MPLS payload in IP, as specified by RFC4023.
- The ports facing public internet might get IP addresses assigned by ISPs, which may not be in the same address domain as PEs’.
- Ports facing public internet are not as secure as the ports facing private infrastructure. There could be spoofing, or DDOS attacks to the ports facing public internet. Extra consideration must be given when injecting the new routes from public network into VRFs.
- Even though packets are encrypted over public internet, the performance SLA is not guaranteed over public internet. Therefore, clients may have policies only allowing some flows to be offloaded to internet path.

4. Provisioning Model

4.1. Client Service Provisioning Model

The provisioning tasks described in Section 4 of RFC8388 are the same for the SDWAN client traffic. When client traffic are multi-homed to two (or more) C-PEs, the Non-Service-Specific parameters need to be provisioned per the Section 4.1.1 of RFC8388.

Since most SDWAN nodes are ephemeral and have small number of IP subnets or VLANs attached to the client ports of the SDWAN nodes, it is recommended to have default and simplified Service-specific parameters for each client port, remotely managed by the SDWAN Network Controller (i.e. the RR) via the secure channel (TLS/DTLS) between the controller and the C-PEs.

More details are to be added.

4.2. WAN Ports Provisioning Model

Since the deployment of PEs to MPLS VPN are for relatively long term, the common provisioning procedure for PE’s WAN ports is via CLI.

A SDWAN node deployment can be ephemeral and its location can be in remote locations, manual provisioning for its WAN ports is not acceptable. In addition, a SDWAN WAN port’s IP address can be dynamically assigned or using private addresses. Therefore, it is necessary to have a separate control protocol; something like NHRP did for ATM, for a SDWAN node to register its WAN property to its controller dynamically.
Unlike a PE to MPLS based VPN where its WAN ports are homogeneously facing MPLS private network and all traffic are egressed in MPLS data frames through its WAN ports, the WAN ports of a SDWAN node can be connected to a PE of VPN, MPLS private network directly, the public Internet, or the various combinations of all.

For Scenario #1 above, the WAN ports can face public internet or VPN.

For Scenario #2 above, WAN ports are either configured as connecting to PEs of VPN where traffic can be sent as IP/Ethernet without encryption, or configured as connecting to public Internet.

For Scenario #3 above, the WAN ports are either configured as VPN egress ports (hand off MPLS data frames), or as connecting to the public internet that requires MPLS in IP in IPsec encapsulation.

4.2.1. Why BGP as Control Plane for SDWAN WAN Ports Registration?

For a small sized SDWAN network, traditional hub & spoke model using NHRP or DSVPN/DMVPN with a hub node (or controller) managing SDWAN node WAN ports mapping (e.g. local & public addresses and tunnel identifiers mapping) can work reasonably well. However, for a large SDWAN network, say more than 100 nodes with different types of topologies, the traditional approach becomes very messy, complex and error prone.

Here are some of the compelling reasons of using BGP instead of extending NHRP/DSVPN/DMVPN. (Same as the reasons quoted by LSVR on why using BGP):

- BGP already widely deployed as sole protocol (see RFC 7938)
- Robust and simple implementation
- Wide acceptance - minimal learning
- Reliable transport
- Guaranteed in-order delivery
- Incremental updates
- Incremental updates upon session restart
- No flooding and selective filtering
- RR already has the capability to apply policies to communications among peers.

5. SDWAN Traffic Forwarding Walk Through

BGP based EVPN control plane are still applicable to routes attached to the client ports of SDWAN nodes. Section 5 of RFC8388 describes the BGP EVPN NLRI Usage for various routes of client traffic. The procedures described in the Section 6 of RFC8388 are same for the SDWAN client traffic.

The only additional consideration for SDWAN is to control how traffic egress the SDWAN edge node to various WAN ports.

5.1. SDWAN Network Startup Procedures

A SDWAN network can add or delete SDWAN edge nodes on regular basis depending on user requests.

- For Scenario #1: a SDWAN edge node in a shopping mall or Cloud DC can be added or removed on demand. The Zero Touch Provisioning described in 3.1.2 are required for the node startup.
- For Scenario #2: this can be Data Centers or enterprises upgrading their CPEs to add extra bandwidth via public internet in addition to VPN services that they already purchased. Before the node powers up or upgraded, there should be links connected to the PEs of a provider VPNs.
- For Scenario #3, the Internet facing WAN ports are added to (or removed from) existing VPN PEs.

5.2. Packet Walk-Through for Scenario #1

Upon power up, a SDWAN node can learn client routes from the Client facing ports, in the same way as EVPN described in RFC8388. Controller facilitates the IPsec SA establishment and rekey management as described in [SECURE-EVPN]. Controller manages how client's routes are associated with individual IPsec SA.

[SECURE-L3VPN] describes how to extend the RFC4364 VPN to allow some PEs being connected to other PEs via public networks. [SECURE-L3VPN] introduces the concept of Red Interface & Black Interface on those
PEs, with RED interfaces face clients’ routes within the VPN and the Black Interfaces being WAN ports over which only IPsec-protected packets to the Internet or other backbone network are sent so that eliminating the need for MPLS transport in the backbone.

[SECURE-L3VPN] assumes PEs terminate MPLS packets, and use MPLS over IPsec when sending over the Black Interfaces.

[SECURE-EVPN] describes a solution where BGP point-to-multipoint signaling is leveraged as control plane for SDWAN Scenario #1. It utilizes the BGP RR to facilitate the key and policy exchange among PE devices to create private pair-wise IPsec Security Associations without IKEv2 point-to-point signaling or any other direct peer-to-peer session establishment messages.

When C-PEs do not support MPLS, the approaches described by RFC8365 can be used, with addition of IPsec encrypting the IP packets when sending packets over the Black Interfaces.

5.3. Packet Walk-Through for Scenario #2

In this scenario, C-PEs have some WAN ports connected to the public internet and some WAN ports connected to (i.e. directly connected to) PEs of trusted VPN. The C-PEs in Scenario #2 are almost like CPEs to MPLS VPN that have the IP/Ethernet data frames egress to the PEs of the VPN, except the packets need encryption if egress to the WAN ports facing public Internet.

Users specify the policy or criteria on which flows can only egress WAN ports facing trusted VPN without encryption, which can egress the WAN ports facing the public Internet with encryption, or which can egress WAN ports facing the public Internet without encryption.

The Control Plane should not learn routes from the Public Network facing WAN ports. Should strictly follow the policies specified by the users. The internet facing WAN ports can face potential DDoS attacks, additional anti-DDoS mechanism has to be implemented on WAN ports facing those public networks.

The Scenario #2 SDWAN Control Plane has three distinct functional components:
- SDWAN node’s WAN ports property registration to the SDWAN Network Controller (BGP RR).
  - This is used to inform the SDWAN controller of all the underlay networks to which the C-PE is connected.
  - RR is responsible for propagating the C-PE WAN ports properties to authorized peers.

- Controller Facilitated IPsec SA management and NAT information distribution
  - Used by the SDWAN controller to facilitate or manage the IPsec configurations and peer authentications for all IPsec SAs terminated at the SDWAN nodes.
  - When WAN ports have private addresses, need exchange between SDWAN edges and the RR about the type of NAT, and mapping of the private addresses/ports <-> public addresses/ports.

- Attached routes distribution via BGP RR, which can be EVPN, IPVPN or others.
  - This is for the overlay layer’s route distribution, so that a C-PE can establish the overlay routing table that identifies the next hop for reaching a specific route/service attached to remote nodes. [SECURE-EVPN] describes EVPN and other options.
5.3.1. SDWAN node WAN Ports Properties Registration

In Figure 6, A1/A2/A3/B1/B2/B3 WAN ports can be from different network providers.

```
  +---+
  | 10.1.1.1 via         +-----------+  10.1.2.1 via |
  | A1/A2/A3              | RR         |  B1/B2/B3 |
  | / Untrusted          /+---++-
  | / Untrusted          +---+
  | +-----+              +-----+  packets encrypted over +-----+
  | | CN3|--| C-PE1 A2------| Untrusted +------ B1 |--| CN1|
  | +-----+  | 10.1.1.1 |  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+
  | | CN2|--| A3 PE---------+PE----B3 |--| CN3|
  | +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+
  |      |      |      |      |      |      | VPN |
  +-----+  +-----+  +-----+  +-----+  +-----+  +-----+
```

Figure 6: SDWAN Scenario #2 WAN Ports Registration

Each SDWAN edge (C-PE) needs to register its WAN ports properties along with its Loopback addresses to the SDWAN Network Controller (RR). The policies that govern the communications among peers are managed and controlled by the SDWAN Controller. Individual SDWAN edge relies on its SDWAN Controller to determine which peers can establish connections. The SDWAN controller is responsible for propagating the mapping information to the authorized peers. If C-PE-1 is not authorized to communicate with C-PE-n, C-PE-1’s WAN port<->Loopback address mapping will not be propagated to C-PE-n.

A C-PE’s Loopback addresses & attached routes may not be visible to some ISPs/NSPs to which the CPE’s WAN port is connected.

5.3.2. Controller Facilitated IPsec SA & NAT management

One IPsec SA between two end points is straightforward. However, for a network with many IPsec SAs among many end points, the configuration and IPsec Key management for the entire network can be complex.
For a 1,000-node network, each node is responsible for maintaining and managing 999 keys to all their peers, which could potentially result in 1,000,000 key exchanges to authenticate among all nodes. In addition, when an edge node has multiple tenants attached, the edge node may need to establish multiple tunnels for tenants. For example, for a network with N nodes, a node A has 5 tenants attached to it, then the node A has to maintain 5*(N-1) number of keys if each tenant needs to communicate with all other nodes.

In addition, all the IPsec keys have to be refreshed periodically, which adds more complexity. Therefore, simplification facilitated by an SDWAN controller is necessary for large-scale SDWAN deployment.

When the SDWAN IPsec SAs are fine-grained, such as per client address, per client’s VLAN, the number of IPsec SAs & Keys to be managed can go much higher, leading to more IPsec management complexity. It is better to aggregate multiple flows into one IPsec SA.

SDWAN edge nodes can rely on the SDWAN controller to facilitate the pair-wise IPsec key establishment and refreshment [RFC7296] and maintain the Security Policy Database (SPD) [RFC4301].

- In the Figure 5 SDWAN Scenario #2 above, if C-PE1 & C-PE2 each has two ports facing two different ISPs networks, and their loopback addresses are not visible to the ISPs, i.e. the C-PE1 & C-PE2 are using a provider assigned IP addresses for A1/A2/B1/B2; you are going to need minimum four IPsec SAs between C-PE1 & C-PE2.

- When C-PEs loopback addresses are visible to ISPs/NSPs, i.e. the C-PEs’ private source and destination IPs are part of a prefix exported to the ISP(s) in each site, it is possible to have one IPsec SA between C-PE1 & C-PE2.

The IP addresses of SDWAN WAN port can be dynamic (e.g. assigned by DHCP) or private IP. Some SDWAN nodes are identified by "System-ID" or Loopback addresses that are only locally significant. In some SDWAN environments, "System-ID + PortID" are used to uniquely identify a SDWAN WAN port. Sometimes, a SDWAN tunnel end-point can be associated with "private IP" + "public IP" (if NAT is used.)

When CPE WAN ports are private addresses, an additional sub-TLV has to be added to the [Tunnel-Encap] to describe the additional
information about the NAT property of SDWAN nodes’ WAN ports. A
SDWAN node can inquire STUN (Session Traversal of UDP through
Network Address Translation [RFC 3489]) Server to get the NAT
property, the public IP address and the Public Port number to pass
to the authorized peers via the SDWAN Controller.

5.3.3. BGP Based SDWAN client routes

The client routes attached to SDWAN client ports have to be
distributed to all SDWAN edge nodes, just like BGP/MPLS IP VPN
[RFC4364], so that all SDWAN edges can establish the overlay routing
table that identifies the remote SDWAN edges to reach a specific
route/service. When C-PEs do not handle MPLS, RFC8365 can be used
for packets over WAN ports, albeit applying IPsec SA encryption when
sent over the WAN ports facing the public networks.

Using the terminologies described by [SECURE-L3VPN], the RED
interface are the clients’ ports and the ports facing private
networks (e.g. connected to the PEs of MPLS VPN). Black Interfaces
are ports facing public networks. The behavior described in [SECURE-
L3VPN] applies to this scenario too, the C-PEs cannot mix the routes
learned from the Black Interfaces with the Routes from RED
Interfaces.

To minimize the burden on SDWAN edge nodes (especially low powered
virtual SDWAN edges), some SDWAN network can let SDWAN controller
take care of authenticating communications among SDWAN edge nodes
instead of pushing down policies to SDWAN edge nodes. SDWAN Edge
nodes might get clients routes from SDWAN controller instead of
learning from clients ports.

The Hybrid SDWAN control plane for distributing clients’ routes is
more similar to overlay using EVPN [RFC8365], albeit the packets
sent over the internet facing ports have to be encrypted by IPsec
SA.

[Tunnel-Encap] can be used to associate client routes with specific
tunnels:

- C-PE1 can advertise the following properties to others C-PEs
  via RR:
  - Encapsulation capability of the Ports to VPN PE
  - Encapsulation capability of the Ports to the Internet:
    GRE-IPsec, or MPLS over GRE over IPsec
- with prior established IPsec SA
- NAT information if ports are private addresses

- The Remote Endpoint sub-TLV is NOT appropriate because
  - The network to which a SDWAN port is connected might have an identifier that is more than the AS number. The SDWAN controller might use its own specific identifier for the network.
  - Suggest using an SDWAN overlay specific Transport-Network-ID to represents the connected networks.

The underlay network selections to next hop C-PE can be a local decision. Different services, routes, or VLANs can be aggregated to one underlay network between two C-PEs; the same service/routes/VLAN can spread over multiple SDWAN underlay networks at the next segment.

5.4. Packet Walk-Through for Scenario #3

The behavior described in [SECURE-L3VPN] applies to this scenario, except C-PEs not only have RED interfaces facing clients within the VPN but also have RED interface facing MPLS backbone, with additional BLACK interfaces facing the untrusted public networks. The C-PEs cannot mix the routes learned from the Black Interfaces with the Routes from RED Interfaces. The routes learned from core-facing RED interfaces are for underlay and cannot be mixed with the routes learned over access-facing RED interfaces that are for overlay. Furthermore, the routes learned over core-facing interfaces (both RED and BLACK) can be shared in the same GLOBAL route table.

There may be some added risks of the packets from the ports facing the Internet. Therefore, special consideration has to be given to the routes from WAN ports facing the Internet. RFC4364 describes using an RD to create different routes for reaching same system. A similar approach can be considered to force packets received from the Internet facing ports to go through special security functions before being sent over to the VPN backbone WAN ports.
6. Manageability Considerations

SDWAN overlay networks utilize the SDWAN controller to facilitate route distribution, central configurations, and others. To minimize the burden on SDWAN edge nodes, SDWAN Edge nodes might not need to learn the routes from clients.

7. Security Considerations

Having WAN ports facing the public Internet introduces the following security risks:

1) Potential DDoS attack to the C-PEs with ports facing internet.

2) Potential risk of provider VPN network being injected with illegal traffic coming from the public Internet WAN ports on the C-PEs.

8. IANA Considerations

None

9. References

9.1. Normative References


9.2. Informative References


[VPN-over-Internet] E. Rosen, "Provide Secure Layer L3VPNs over Public Infrastructure", draft-rosen-bess-secure-l3vpn-00, work-in-progress, July 2018


10. Acknowledgments

Acknowledgements to Jim Guichard, John Scudder, Darren Dukes, Andy Malis and Donald Eastlake for their review and contributions.

This document was prepared using 2-Word-v2.0.template.dot.
Authors’ Addresses

Linda Dunbar
Huawei
Email: ldunbar@futurewei.com

James Guichard
Huawei
Email: james.n.guichard@futurewei.com

Ali Sajassi
Cisco
Email: sajassi@cisco.com

John Drake
Juniper
Email: jdrake@juniper.net
Mathematical Mesh 3.0 Part I: Architecture Guide
draft-hallambaker-mesh-architecture-10

Abstract

The Mathematical Mesh ‘The Mesh’ is an end-to-end secure infrastructure that makes computers easier to use by making them more secure. The Mesh provides a set of protocol and cryptographic building blocks that enable encrypted data stored in the cloud to be accessed, managed and exchanged between users with the same or better ease of use than traditional approaches which leave the data vulnerable to attack.

This document provides an overview of the Mesh data structures, protocols and examples of its use.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [1] .

Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Definitions .................................................... 5
   2.1. Related Specifications .................................... 5
   2.2. Defined Terms ............................................. 6
   2.3. Requirements Language .................................. 6
   2.4. Implementation Status .................................. 6
3. Architecture .................................................. 6
   3.1. A Personal PKI ............................................ 8
       3.1.1. Device Management ................................... 9
       3.1.2. Exchange of trusted credentials ..................... 9
       3.1.3. Application configuration management .............. 10
       3.1.4. The Mesh as platform ................................ 10
   3.2. Mesh Architecture ....................................... 11
       3.2.1. Mesh Device Management ............................. 12
       3.2.2. Mesh Account ......................................... 13
       3.2.3. Mesh Service ......................................... 15
       3.2.4. Mesh Messaging ...................................... 15
   3.3. Using the Mesh with Applications ......................... 16
       3.3.1. Contact Exchange .................................... 17
       3.3.2. Confirmation Protocol ............................... 17
       3.3.3. Future Applications ................................. 18
4. Mesh Cryptography ............................................. 18
   4.1. Best Practice by Default ................................ 19
   4.2. Multi-Level Security .................................... 19
   4.3. Multi-Key Decryption ................................... 20
   4.4. Multi-Party Key Generation ............................. 20
   4.5. Data At Rest Encryption ................................ 21
       4.5.1. DARE Envelope ...................................... 21
       4.5.2. Dare Container ...................................... 21
   4.6. Uniform Data Fingerprints ................................ 22
       4.6.1. Friendly Names ...................................... 23
       4.6.2. Encrypted Authenticated Resource Locators ....... 23
       4.6.3. Secure Internet Names ............................... 24
   4.7. Personal Key Escrow ..................................... 24
5. User Experience ............................................... 26
   5.1. Creating a Mesh Profile and Administration Device .... 27
1. Introduction

The Mathematical Mesh (Mesh) is a user centered Public Key Infrastructure that uses cryptography to make computers easier to use. The Mesh provides an infrastructure that addresses the three concerns that have proved obstacles to the use of end-to-end security in computer applications:

- Device management.
- Exchange of trusted credentials.
- Application configuration management.

The infrastructure developed to address these original motivating concerns can be used to facilitate deployment and use of existing security protocols (OpenPGP, S/MIME, SSH) and as a platform for building end-to-end secure network applications. Current Mesh applications include:

- Multi-factor authentication and confirmation
- Credential management
- Bookmark/Citation management
- Task and workflow management
This document is not normative. It provides an overview of the Mesh comprising a description of the architecture, and a discussion of typical use cases and requirements. The remainder of the document series provides a summary of the principal components of the Mesh architecture and their relationship to each other.

Normative descriptions of the individual Mesh encodings, data structures and protocols are provided in separate documents addressing each component in turn.

The currently available Mesh document series comprises:

I. Architecture (This document.) Provides an overview of the Mesh as a system and the relationship between its constituent parts.

II. Uniform Data Fingerprint. Describes the UDF format used to represent cryptographic nonces, keys and content digests in the Mesh and the use of Encrypted Authenticated Resource Locators (EARLs) and Strong Internet Names (SINs) that build on the UDF platform.

III. Data at Rest Encryption. Describes the cryptographic message and append-only sequence formats used in Mesh applications and the Mesh Service protocol.

IV. Schema Reference. Describes the syntax and semantics of Mesh Profiles, Container Entries and Mesh Messages and their use in Mesh Applications.


VI. The Trust Mesh. Describes the social work factor metric used to evaluate the effectiveness of different approaches to exchange of credentials between users and organizations in various contexts and argues for a hybrid approach taking advantage of direct trust, Web of Trust and Trusted Third Party models to provide introductions.

VII. Security Considerations. Describes the security considerations for the Mesh protocol suite.

VIII Cryptographic Algorithms. Describes the recommended and required algorithm suites for Mesh applications and the implementation of the multi-party cryptography techniques used in the Mesh.

The following documents describe technologies that are used in the Mesh but do not form part of the Mesh standards suite:
JSON-BCD Encoding. Describes extensions to the JSON serialization format to allow direct encoding of binary data (JSON-B), compressed encoding (JSON-C) and extended binary data encoding (JSON-D). Each of these encodings is a superset of the previous one so that JSON-B is a superset of JSON, JSON-C is a superset of JSON-B and JSON-D is a superset of JSON-C.

DNS Web Service Discovery. Describes the means by which prefixed DNS SRV and TXT records are used to perform discovery of Web Services.

The following documents describe aspects of the Mesh Reference implementation:

Mesh Developer. Describes the reference code distribution license terms, implementation status and currently supported functions.

Mesh Platform. Describes how platform specific functionality such as secure key storage and trustworthy computing features are employed in the Mesh.

2. Definitions

This section presents the related specifications and standards on which the Mesh is built, the terms that are used as terms of art within the Mesh protocols and applications and the terms used as requirements language.

2.1. Related Specifications

Besides the documents that form the Mesh core, the Mesh makes use of many existing Internet standards, including:

Cryptographic Algorithms The RECOMMENDED and REQUIRED cryptographic algorithms for Mesh implementations are specified in [draft-hallambaker-mesh-cryptography].

In addition Mesh Devices used to administer non-Mesh applications must support the cryptographic algorithm suites specified by the application.

Transport All Mesh Services make use of multiple layers of security. Protection against traffic analysis and metadata attacks are provided by use of Transport Layer Security [RFC5246]. At present, the HTTP/1.1 [RFC7231] protocol is used to provide framing of transaction messages.
Encoding  All Mesh protocols and data structures are expressed in the JSON data model and all Mesh applications accept data in standard JSON encoding [RFC7159]. The JOSE Signature [RFC7515] and Encryption [RFC7516] standards are used as the basis for object signing and encryption.

2.2. Defined Terms

TBS

2.3. 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].

2.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

The examples in this document were created on 8/13/2019 11:36:45 AM. Out of 169 examples, 71 were not functional.

[Note: Example data is now being produced using the mesh command line tool which is currently substantially less complete than the Mesh reference code it is intended to provide an interface to. As a result, the documentation currently lags the code by more than is usual.]

3. Architecture

The Mathematical Mesh (Mesh) is a user centered Public Key Infrastructure that uses cryptography to make computers easier to use. This document describes version 3.0 of the Mesh architecture and protocols.

For several decades, it has been widely noted that most users are either unwilling or unable to make even the slightest efforts to protect their security, still less those of other parties. Yet despite this observation being widespread, the efforts of the IT security community have largely focused on changing this user behavior rather than designing applications that respect it. Real users have real work to do and have neither the time nor the inclination to use tools that will negatively impact their performance.
The Mesh is based on the principle that if the Internet is to be secure, it must become effortless to use applications securely. Rather than beginning the design process by imagining all the possible modes of attack and working out how to address these with the least possible inconvenience, we must reverse the question and ask how much security can be provided without requiring any effort on the user’s part to address it.

Today’s technology requires users to put their trust in an endless variety of devices, software and services they cannot fully understand let alone control. Even the humble television of the 20th century has been replaced by a ‘smart’ TV with 15 million lines of code. Whose undeclared capabilities may well include placing the room in which it is placed under continuous audio and video surveillance.

Every technology deployment by necessity requires some degree of trust on the owner/user’s part. But this trust should be limited and subject to accountability. If manufacturers continue to fail in this regard, they risk a backlash in which users seek to restore their rights through litigation, legislation or worst of all, simply not buying more technology that they have learned to distrust through their own experience.

The Mesh is based on the principle of radical distrust, that is, if a party is capable of defecting, we assume that they will. As the Russian proverb goes: ????????, ?? ????????: trust, but verify.

In the 1990s, the suggestion that ‘hackers’ might seek to make financial gains from their activities was denounced as ‘fear-mongering’. The suggestion that email or anonymous currencies might be abused received a similar response. Today malware, ransomware and spam have become so ubiquitous that they are no longer news unless the circumstances are particularly egregious.

We must dispense with the notion that it is improper or impolite to question the good faith of technology suppliers of any kind whether they be manufacturers, service providers, software authors or reviewers. Modern supply chains are complex, typically involving hundreds if not thousands of potential points of deliberate or accidental compromise. The technology provider who relies on the presumption of good faith on their part risks serious damage to their reputation when others assert that a capability added to their product may have malign uses.

Radical distrust means that we apply the principles of least principle and accountability at every level in the design:
o Cryptographic keys installed in a product during manufacture are only used for the limited purpose of putting that device under control of the user.

o Cryptographic keys and assertions related to management of devices are only visible to the user they belong to and are never exposed to external parties.

o Mesh Accounts belong to and are under control of the user they belong to and not the Mesh Service provider which the user can change at will with minimal inconvenience.

o Mesh Services do not have access to the plaintext of any Mesh Messages or Mesh Catalog data except for the Contacts catalog.

o All Mesh Messages are subject to access control by both the inbound and outbound Mesh Service to mitigate messaging abuse.

Security is risk management and not the elimination of all possibility of any risk. Radical distrust means that we raise the bar for attackers to the point where for most attackers the risk is greater than the reward.

In addition to distrusting technology providers the Mesh Architecture allows the user to limit the degree of trust they place in themselves. In the real world, devices are lost or stolen, passwords and activation codes are forgotten, natural or man-made catastrophes cause property and data to be lost. The Mesh permits but does not require use of escrow techniques that allow recovery from such situations.

3.1. A Personal PKI

The Mesh is a Public Key Infrastructure (PKI) that is designed to address the three major obstacles to deployment of end-to-end secure applications:

o Device management.

o Exchange of trusted credentials.

o Application configuration management.

Each Mesh user is the ultimate source of authority in their Personal Mesh which specifies the set of devices, contacts and applications that they trust and for what purposes.
The Mesh 1.0 architecture described a PKI designed to meet these limited requirements to enable use of existing end-to-end secure Internet protocols such as OpenPGP, S/MIME and SSH. Since these protocols only secure data in transit and the vast majority of data breaches involve data at rest, the Data At Rest Encryption (DARE) was added as a layered application resulting in the Mesh 2.0 architecture. This document describes the Mesh 3.0 architecture which has been entirely re-worked so that DARE provides the platform on which all other Mesh functions are built.

3.1.1. Device Management

Existing PKIs were developed in an era when the ‘personal computer’ was still coming into being. Only a small number of people owned a computer and an even smaller number owned more than one. Today, computers are ubiquitous and a typical home in the developed world contains several hundred of which a dozen or more may have some form of network access.

The modern consumer faces a problem of device management that is considerably more complex than the IT administrator of a small business might have faced in the 1990s but without any of the network management tools such an administrator would expect to have available.

One important consequence of the proliferation of devices is that end-to-end security is no longer sufficient. To be acceptable to users, a system must be ends-to-ends secure. That is, a user must be able to read their encrypted email message on their laptop, tablet, phone, or watch with exactly the same ease of use as if the mail was unencrypted.

Each personal Mesh contains a device catalog in which the cryptographic credentials and device specific application configurations for each connected device are stored.

Management of the device catalog is restricted to a subset of devices that the owner of the Mesh has specifically authorized for that purpose as administration devices. Only a device with access to a duly authorized administration key can add or remove devices from a personal Mesh.

3.1.2. Exchange of trusted credentials.

One of the most challenging, certainly the most contentious issues in PKI is the means by which cryptographic credentials are published and validated.
The Mesh does not attempt to impose criteria for accepting credentials as valid as no such set of criteria can be comprehensive. Rather the Mesh allows users to make use of the credential validation criteria that are appropriate to the purpose for which they intend to use them and Mesh Services provides protocol support for exchange of credentials between users and to synchronize credential information between all the devices belonging to a user.

In some circumstances, only a direct trust model is acceptable, in others, only a trusted third-party model is possible and in the vast majority of cases opportunistic approaches are more than sufficient. Both approaches may be reinforced by use of chained notary certificate (e.g. BlockChain) technology affords a means of establishing that a particular assertion was made before a certain date. The management of Trust in the Mesh is described in detail in [draft-hallambaker-mesh-trust].

3.1.3. Application configuration management

Configuration of cryptographic applications is typically worse than an afterthought. Configuration of one popular mail user agent to use S/MIME security requires 17 steps to be performed using four separate application programs. And since S/MIME certificates expire, the user is required to repeat these steps every few years. Contrary to the public claims made by one major software vendor it is not necessary to perform ‘usability testing’ to recognize abject stupidity.

Rather than writing down configuration steps and giving them to the user, we should turn them into code and give them to a machine. Users should never be required to do the work of the machine. Nor should any programmer be allowed to insult the user by casting their effort aside and requiring it to be re-entered.

While most computer professionals who are required to do such tasks on a regular basis will create a tool for the purpose, most users do not have that option. And of those who do write their own tools, only a few have the time and the knowledge to do the job without introducing security vulnerabilities.

3.1.4. The Mesh as platform

Meeting the core objectives of the Mesh required new naming, communication and cryptographic capabilities provided to be developed. These capabilities may in turn be used to develop new end-to-end secure applications.

For example, a DARE Catalog is a cryptographic container in which the entries represent a set of objects which may be added, updated and
deleted over time. The Mesh Service protocol allows DARE Catalogs to be synchronized between devices connected to a Mesh Account. DARE Catalogs are used as the basis for the device and contacts catalogs referred to above.

The Mesh Credentials Catalog uses the same DARE Catalog format and Mesh Service protocol to share passwords between devices with end-to-end encryption so that no password data is ever left unencrypted in the cloud.

3.2. Mesh Architecture

The Mesh has four principal components:

Mesh Device Management Each user has a personal Mesh profile that is used for management of their personal devices. A user may connect devices to or remove devices from their personal Mesh by use of a connected device that has been granted the ‘administration’ role.

Mesh Account A Mesh account is similar in concept to a traditional email or messaging account but with the important difference that it belongs to the user and not a service provider. Users may maintain multiple Mesh accounts for different purposes.

Mesh Service A Mesh Service provides a service identifier (e.g. alice@example.com) through which devices and other Mesh users may interact with a Mesh Account. It is not necessary for a Mesh Account to be connected to a Mesh Service and users may change their Mesh service provider at any time. It is even possible for a Mesh Account to be connected to multiple services at the same time but only one such account is regarded as the primary account at a given time.

Mesh Messaging Mesh Messaging allows short messages (less than 32KB) to be exchanged between Mesh devices connected to an account and between Mesh Accounts. One of the key differences between Mesh Messaging and legacy services such as SMTP is that every message received is subjected to access control.

A user’s Personal Mesh is the set of their Personal Mesh Profiles, Mesh Accounts and the Mesh Services to which they are bound.

For example, Figure X shows Alice’s personal Mesh which have separate accounts for her personal and business affairs. She has many devices, two of which are shown here. Both are linked to her personal account but only one is linked to her business account. Besides allowing Alice to separate work and pleasure, this separation
means that she does not need to worry about her business affairs being compromised if the device Alice2 is stolen.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [2].]]

Master Profile and Subordinate Devices and Accounts.

Alice’s ProfileMaster contains a Master Signature Key used to sign the profile itself and one or more Administrator Signature Keys used to sign assertions binding devices and/or assertions to her Mesh.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [3].]]

Master Profile and Associated Device and Account Connection Assertions.

If desired, Alice can escrow the master key associated with her Profile Master and delete it from her device(s), thus ensuring that compromise of the device does not result in a permanent compromise of her personal Mesh. Recovery of the Master Signature Key and the associated Master Encryption Escrow keys (not shown) allows Alice to recover her entire digital life.

To eliminate the risk of hardware failure, the escrow scheme offered by the Mesh itself uses key shares printed on paper and an encrypted escrow record stored in the cloud. Mesh users are of course free to use alternative escrow means of their choice.

3.2.1.  Mesh Device Management

Mesh devices are added to or removed from a user’s personal Mesh by adding or removing Device catalog entries from the CatalogDevice associated with the Master Profile.

Device catalog entries are created by devices that have been provisioned with an administration key specified in the corresponding ProfileMaster.

The keying material used by a device in the context of a user’s personal Mesh comes from two separate sources:
- Keying material specified in the ProfileDevice which is either generated on the device itself or installed during manufacture.

- Keying material provided by the Administration Device during the connection process.

This approach mitigates the risk of keying material used by the device being compromised during or after manufacture and the risks associated with use of weak keys. The key combination mechanism is shown in section XX below and described in detail in [draft-hallambaker-mesh-cryptography].

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [4].]]

Mapping of Device Profile and Device Private to Device Connection Keys.

In accordance with the principle of maintaining cryptographic hygiene, separate keys are generated for signature, authentication and encryption purposes.

3.2.2. Mesh Account

Mesh Accounts comprise a collection of persistent data stores associated with a particular persona associated with a personal Mesh. The connection between a Mesh Account and the personal Mesh to which it belongs may or may not be public. For example, Alice might allow her contacts to know that her business and personal accounts belong to the same personal Mesh and thus the same person but Bob might not.

Mesh Accounts afford similar functionality to the accounts provided by traditional Internet protocols and applications but with the important distinction that they belong to the user and not the service provider. A Mesh Account may be connected to one, many or no Mesh Services and the user may add or delete service providers at any time.

A Mesh Account that is not connected to a Service is called an offline account. Offline accounts cannot send or receive Mesh Messages and cannot be synchronized using the Mesh Service protocol (but may be synchronized through other means).

When a Mesh Account is connected to multiple services, only the first service is normally regarded as being primary with the others being secondary accounts for use in case of need.
Alice’s personal account is connected to two devices and two services (alice@example.com and alice@example.net).

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [5].]]

Account Profile Connected to Devices and Services.

As with the connection of the device to Alice’s personal Mesh, the connection of each device to each account requires the creation of a separate set of keying using the same key combination mechanism described above. This information is contained in the ActivationAccount record corresponding to the account in the CatalogEntryDevice.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [6].]]

Account Key Set.

Note that even though Alice’s personal account is connected to two separate Mesh Services, the same cryptographic keys are used for both. However separate keys are used for her personal and business accounts so as to prevent these accounts being linked through use of the same device keys.

3.2.2.1. Account Catalogs

Mesh Catalogs are a DARE Containers whose entries represent a set of objects with no inherent ordering. Examples of Mesh catalogs include:

Devices The devices connected to the corresponding Mesh profile.

Contacts Logical and physical contact information for people and organizations.

Application

Application

Bookmarks Web bookmarks and citations.

Credentials Username and password information for network resources.
Calendar   Appointments and tasks.

Network   Network access configuration information allowing access to WiFi networks and VPNs.

   Configuration information for applications including mail (SMTP, IMAP, OpenPGP, S/MIME, etc) and SSH.

The Devices and Contacts catalogues have special functions in the Mesh as they describe the set of devices and other users that a Mesh user interacts with.

These catalogs are also used as the basis for providing a consistent set of friendly names to the users devices and contacts that is accessible to all her devices. This (in principle) allows Alice to give a voice command to her car or her watch or her phone to call a person or open a door and expect consistent results.

3.2.3. Mesh Service

Each Mesh Service is described by a ProfileService signed by a long-lived signature key. As with the ProfileMaster, a separate set of Administrator keys is used to sign the Assertion Host objects used to credential Service Hosts.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [7].]]

Service Profile and Delegated Host Assertion.

Note that the Mesh Service Authentication mechanism only provides trust after first use. It does not provide a mechanism for secure introduction. A Mesh Service SHOULD be credentialed by means of a validation process that establishes the accountability. For example, the CA-Browser Forum Extended Validation Requirements.

3.2.4. Mesh Messaging

The Mesh Messaging layer supports the exchange of short (less than 32KB) messages. Mesh devices connected to the same Mesh profile may exchange Mesh Messages directly. Messages exchanged between Mesh Users MUST be mediated by a Mesh Service for both sending and receipt. This ‘four corner’ pattern permits ingress and egress controls to be enforced on the messages and that every message is properly recorded in the appropriate spools.
For example, To send a message to Alice, Bob posts it to one of the Mesh Services connected to the Mesh Account from which the message is to be sent. The Mesh Service checks to see that both the message and Bob’s pattern of behavior comply with their acceptable use policy and if satisfactory, forwards the message to the receiving service example.com.

Performing Access Control on Outbound Messages

The receiving service uses the recipient’s contact catalog and other information to determine if the message should be accepted. If accepted, the message is added to the recipient’s inbound message spool to be collected by her device(s) when needed.

Performing Access Control on Inbound Messages

For efficiency and to limit the scope for abuse, all inbound Mesh Messages are subject to access control and limited in size to 32KB or less. This limit has proved adequate to support transfer of complex control messages and short content messages. Transfer of data objects of arbitrary size may be achieved by sending a control message containing a URI for the main content which may then be fetched using a protocol such as HTTP.

This approach makes transfers of very large data sets (i.e. multiple Terabytes) practical as the 'push' phase of the protocol is limited to the transfer of the initial control message. The bulk transfer is implemented as a 'pull' protocol allowing support for features such as continuous integrity checking and resumption of an interrupted transfer.

3.3. Using the Mesh with Applications

The Mesh provides an infrastructure for supporting existing Internet security applications and a set security features that may be used to build new ones.
For example, Alice uses the Mesh to provision and maintain the keys she uses for OpenPGP, S/MIME, SSH and IPSEC. She also uses the credential catalog for end-to-end secure management of the usernames and passwords for her Web browsing and other purposes:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [10].]]

Each of Alice’s devices have access to the shared context of her personal account.

The Mesh design is highly modular allowing components that were originally designed to support a specific requirement within the Mesh to be applied generally.

3.3.1. Contact Exchange

One of the chief concerns in any PKI is the means by which the public keys of other users are obtained and validated. This is of particular importance in the Mesh since every Mesh Message is subject to access control and it is thus necessary for Alice to accept Bob’s credentials before Bob send most types of message to Alice.

The Mesh supports multiple mechanisms for credential exchange. If Alice and Bob meet in person and are carrying their smart phones, a secure mutual exchange of credentials can be achieved by means of a QR code mechanism. If they are at separate locations, Alice can choose between accepting Bob’s contact information with or without additional verification according to the intended use.

3.3.2. Confirmation Protocol

The basic device connection protocol requires the ability for one device to send a connection request to the Mesh service hosting the user’s profile. To accept the device connection, the user connects to the service using an administration device, reviews the pending requests and creates the necessary device connection assertion if it is accepted.

The confirmation protocol generalizes this communication pattern allowing any authorized party to post a short accept/reject question to the user who may (or may not) return a signed response. This feature can be used as improvement on traditional second factor authentication providing resistance to man-in-the-middle attacks and providing a permanent non-repudiable indication of the user’s specific intent.
3.3.3. Future Applications

Since a wide range of network applications may be reduced to
synchronization of sets and lists, the basic primitives of Catalogs
and Spools may be applied to achieve end-to-end security in an even
wider variety of applications.

For example, a Spool may be used to maintain a mailing list, track
comments on a Web forum or record annotations on a document.
Encrypting the container entries under a multi-party encryption group
allows such communications to be shared with a group of users while
maintaining full end-to-end security and without requiring every
party writing to the spool to know the public encryption key of every
recipient.

Another interesting possibility is the use of DARE Containers as a
file archive mechanism. A single signature on the final Merkle Tree
digest value would be sufficient to authenticate every file in the
archive. Updates to the archive might be performed using the same
container synchronization primitives provided by a Mesh Service.
This approach could afford a robust, secure and efficient mechanism
for software distribution and update.

4. Mesh Cryptography

All the cryptographic algorithms used in the Mesh are either industry
standards or present a work factor that is provably equivalent to an
industry standard approach.

Existing Internet security protocols are based on approaches
developed in the 1990s when performance tradeoffs were a prime
consideration in the design of cryptographic protocols. Security was
focused on the transport layer as it provided the best security
possible given the available resources.

With rare exceptions, most computing devices manufactured in the past
ten years offer either considerably more computing power than was
typical of 1990s era Internet connected machines or considerably
less. The Mesh architecture is designed to provide security
infrastructure both classes of machine but with the important
constraint that the less capable ‘constrained’ devices are considered
to be ‘network capable’ rather than ‘Internet capable’ and that the
majority of Mesh related processing will be offloaded to another
device.

For example, Alice uses her Desktop and Laptop to exchange end-to-end
secure Mesh Messages and documents but her Internet-of-Things food
blender and light bulb are limited in the range of functions they
support and the telemetry information they provide. The IoT devices connect to a Mesh Hub which acts as an always-on point of presence for the device state and allows complex cryptographic operations to be offloaded if necessary.

Constrained Devices connected through a Mesh Hub.

4.1. Best Practice by Default

Except where support for external applications demand otherwise, the Mesh requires that the following 'best practices' be followed:

Industry Standard Algorithms  All cryptographic protocols make use of the most recently adopted industry standard algorithms.

Strongest Work Factor Only the strongest modes of each cipher algorithm are used. All symmetric encryption is performed with 256-bit session keys and all digest algorithms are used in 512-bit output length mode.

Key Hygiene  Separate public key pairs are used for all cryptographic functions: Encryption, Signature and Authentication. This enables separate control regimes for the separate functions and partitioning of cryptographic functions within the application itself.

Bound Device Keys  Each device has a separate set of Encryption, Signature and Authentication key pairs. These MAY be bound to the device to which they are assigned using hardware or other techniques to prevent or discourage export.

No Optional Extras  Traditional approaches to security have treated many functions as being 'advanced' and thus suited for use by only the most sophisticated users. The Mesh rejects this approach noting that all users operate in precisely the same environment facing precisely the same threats.

4.2. Multi-Level Security

All Mesh protocol transactions are protected at the Transport, Message and Data level. This provides security in depth that cannot be achieved by applying security at the separate levels independently. Data level encryption provides end-to-end
confidentiality and non-repudiation, Message level authentication provides the basis for access control and Transport level encryption provides a degree of protection against traffic analysis.

4.3. Multi-Key Decryption

Traditional public key encryption algorithms have two keys, one for encryption and another for decryption. The Mesh makes use of threshold cryptography techniques to allow the decryption key to be split into two or more parts.

For example, if we have a private key \( z \), we can use this to perform a key agreement with a public key \( S \) to obtain the key agreement value \( A \). But if \( z = (x+y) \mod g \) (where \( g \) is the order of the group), we can obtain the exact same result by applying the private keys \( x \) and \( y \) to \( S \) separately and combining the results:

\[
\text{Two key decryption.}
\]

The approach to Multi-Key Decryption used in the Mesh was originally inspired by the work of Matt Blaze et. al. on proxy re-encryption. But the approach used may also be considered a form of Torben Pedersen’s Distributed Key generation.

This technique is used in the Mesh to allow use of decryption key held by a user to be controlled by a cloud service without giving the cloud service the ability to decrypt by itself.

4.4. Multi-Party Key Generation

The mathematics that support multi-key decryption are also the basis for the multi-party key generation mechanism that is applied at multiple levels in the Mesh. The basis for the multi-party key generation used in the Mesh is that for any Diffie-Hellman type cryptographic scheme, given two keypairs \( \{ x, X \} \), \( \{ y, Y \} \), we calculate the public key corresponding to the private key \( x + y \) using just the public key values \( X \) and \( Y \).
Two party key pair generation.

Multi-party key generation ensures that keys used to bind devices to a personal Mesh or within a Mesh account are ‘safe’ if any of the contributions to the generation process are safe.

4.5. Data At Rest Encryption

The Data At Rest Encryption (DARE) format is used for all confidentiality and integrity enhancements. The DARE format is based on the JOSE Signature and Encryption formats and the use of an extended version of the JSON encoding allowing direct encoding of binary objects.

4.5.1. DARE Envelope

The DARE Envelope format offers similar capabilities to existing formats such as OpenPGP and CMS without the need for onerous encoding schemes. DARE Assertions are presented as DARE Envelopes.

A feature of the DARE Envelope format not supported in existing schemes is the ability to encrypt and authenticate sets of data attributes separately from the payload. This allows features such as the ability to encrypt a subject line or content type for a message separately from the payload.

4.5.2. Dare Container

A DARE Container is an append-only sequence of DARE Envelopes. A key feature of the DARE Container format is that entries MAY be encrypted and/or authenticated incrementally. Individual entries MAY be extracted from a DARE Container to create a stand-alone DARE Envelope.

Containers may be authenticated by means of a Merkle tree of digest values on the individual frames. This allows similar demonstrations of integrity to those afforded by Blockchain to be provided but with much greater efficiency.

Unlike traditional encryption formats which require a new public key exchange for each encrypted payload, the DARE Container format allows multiple entries to be encrypted under a single key exchange operation. This is particularly useful in applications such as
encrypting server transaction logs. The server need only perform a single key exchange operation is performed each time it starts to establish a master key. The master key is then used to create fresh symmetric keying material for each entry in the log using a unique nonce per entry.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [14].]]

DARE Container containing a transaction log.

Integrity is provided by a Merkle tree calculated over the sequence of log entries. The tree apex is signed at regular intervals to provide non-repudiation.

Three types of DARE Containers are used in the mesh

Catalogs A DARE Container whose entries track the status of a set of related objects which may be added, updated or deleted.

Spools A DARE Container whose entries track the status of a series of Mesh Messages.

Archives A DARE Container used to provide a file archive with optional confidentiality and/or integrity enhancements.

4.6. Uniform Data Fingerprints.

The Uniform Data Fingerprint (UDF) format provides a compact means of presenting cryptographic nonces, keys and digest values using Base32 encoding that resists semantic substitution attacks. UDF provides a convenient format for data entry. Since the encoding used is case-insensitive, UDFs may if necessary be read out over a voice link without excessive inconvenience.

The following are examples of UDF values:

<table>
<thead>
<tr>
<th>UDF Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBIK-OM6U-OJ4F-DOUC-AQ74-W7B6-3Z6Q</td>
</tr>
<tr>
<td>EBWM-M66M-3HA5-DPB5-DQJT-HBNE-7WUQ</td>
</tr>
<tr>
<td>SAQL-VVJQ-VJQY-MC2Y-OLIR-GWT7-AAQS-K</td>
</tr>
<tr>
<td>MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4</td>
</tr>
<tr>
<td>KCN5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN</td>
</tr>
<tr>
<td>ABUK-NC4K-SSIV-HVKGZ-A3Z7-SNPA-ZAQP</td>
</tr>
</tbody>
</table>

UDF content digests are used to support a direct trust model similar to that of OpenPGP. Every Mesh Profile is authenticated by the UDF
4.6.1. Friendly Names

Internet addressing schemes are designed to provide a globally unique (or at minimum unambiguous) name for a host, service or account. In the early days of the Internet, this resulted in addresses such as 10.2.3.4 and alice@example.com which from a usability point of view might be considered serviceable if not ideal. Today the Internet is a global infrastructure servicing billions of users and tens of billions of devices and accounts are more likely to be alice.lastname.1934@example.com than something memorable.

Friendly names provide a user or community specific means of identifying resources that may take advantage of geographic location or other cues to resolve possible ambiguity. If Alice says to her voice activated device "close the garage door" it is implicit that it is her garage door that she wishes to close. And should Alice be fortunate enough to own two houses with a garage, it is implicit that it is the garage door of the house she is presently using that she wishes to close.

The Mesh Device Catalog provides a directory mapping friendly names to devices that is available to all Alice’s connected devices so that she may give and instruction to any of her devices using the same friendly name and expect consistent results.

4.6.2. Encrypted Authenticated Resource Locators

Various schemes have been used to employ QR Codes as a means of device and/or user authentication. In many of these schemes a QR code contains a challenge nonce that is used to authenticate the connection request.

The Mesh supports a QR code connection mode employing the Encrypted Authenticated Resource Locator (EARL) format. An EARL is an identifier which allows an encrypted data object to be retrieved and decrypted. In this case, the encrypted data object contains the information needed to complete the interaction.

An EARL contains the domain name of the service providing the resolution service and an encryption master key:

```
udf://example.com/EBLD-J4FF-5G63-FS7P-TEHG-HMIW-FCMV-E7
```
The EARL may be expressed as a QR code:

QR Code representation of the EARL

An EARL is resolved by presenting the content digest fingerprint of the encryption key to a Web service hosted at the specified domain. The service returns a DARE Envelope whose payload is encrypted and authenticated under the specified master key. Since the content is stored on the service under the fingerprint of the key and not the key itself, the service cannot decrypt the plaintext. Only a party that has access to the encryption key in the QR code can decrypt the message.

4.6.3. Secure Internet Names

Secure Internet Names bind an Internet address such as a URL or an email address to a Security Policy by means of a UDF content digest of a document describing the security policy. This binding enables a SIN-aware Internet client to ensure that the security policy is applied when connecting to the address. For example, ensuring that an email sent to an address must be end-to-end encrypted under a particular public key or that access to a Web Service requires a particular set of security enhancements.

alice@example.com  Alice’s regular email address (not a SIN).

alice@mm--uuuu-uuuu-uuuu.example.com  A strong email address for Alice that can be used by a regular email client.

alice@example.com.mm--uuuu-uuuu-uuuu  A strong email address for Alice that can only be used by an email client that can process SINs.

Using an email address that has the Security Policy element as a prefix allows a DNS wildcard element to be defined that allows the address to be used with any email client. Presenting the Security Policy element as a suffix means it can only be resolved by a SIN-aware client.

4.7. Personal Key Escrow

One of the core objectives of the Mesh is to make data level encryption ubiquitous. While data level encryption provides robust
protection of data confidentiality, loss of the ability to decrypt means data loss.

For many Internet users, data availability is a considerably greater concern than confidentiality. Ten years later, there is no way to replace pictures of the children at five years old. Recognizing the need to guarantee data recovery, the Mesh provides a robust personal key escrow and recovery mechanism. Lawful access is not supported as a requirement.

Besides supporting key recovery in the case of loss, the Mesh protocols potentially support key recovery in the case of the key holder’s death. The chief difficulty faced in implementing such a scheme being developing an acceptable user interface which allows the user to specify which of their data should survive them and which should not. As the apothegm goes: Mallet wants his beneficiaries to know where he buried Aunt Agatha’s jewels but not where he buried Aunt Agatha.

The Mesh supports use of Shamir Secret Sharing to split a secret key into a set of shares, a predetermined number of which may be used to recover the original secret. For convenience secret shares are represented using UDF allowing presentation in Base32 (i.e. text format) for easy transcription or QR code presentation if preferred.

A Mesh Profile is escrowed by creating a recovery record containing the private keys corresponding to the master signature and master escrow keys associated with the profile. A master secret is then generated which is used to generate a symmetric encryption key used to encrypt the recovery record and to generate the desired number of recovery shares. For example, Alice escrows her Mesh Profile creating three recovery shares, two of which are required to recover the master secret:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [16].]]

Use of Shamir Secret Sharing to create a recovery record.

To recover the master secret, Alice presents the necessary number of key shares. These are used to recover the master secret which is used to generate the decryption key.
Use of Shamir Secret Recovery to recover a master key set.

A user may choose to store their encrypted recovery record themselves or make use of the EARL mechanism to store the information at one or more cloud services using the fingerprint of the master secret as the locator.

5. User Experience

This section describes the Mesh in use. These use cases described here are re-visited in the companion Mesh Schema Reference [draft-hallambaker-mesh-schema] and Mesh Protocol Reference [draft-hallambaker-mesh-protocol] with additional examples and details.

For clarity and for compactness, these use cases are illustrated using the command line tool meshman.

The original design brief for the Mesh was to make it easier to use the Internet securely. Over time, it was realized that users are almost never prepared to sacrifice usability or convenience for security. It is therefore insufficient to minimize the cost of security, if secure applications are to be used securely they must be at least as easy to use as those they replace. If security features are to be used, they must not require the user to make any additional effort whatsoever.

The key to meeting this constraint is that any set of instructions that can be written down to be followed by a user can be turned into code and executed by machine. Provided that the necessary authentication, integrity and confidentiality controls are provided. Thus, the Mesh is not just a cryptographic infrastructure that makes use of computer systems more secure, it is a usability infrastructure that makes computers easier to use by providing security.

The user experience is thus at the heart of the design of the Mesh and a description of the Mesh Architecture properly begins with consideration of the view of the system that matters most: that of the user.

The principle security protocols in use today were designed at a time when most Internet users made use of either a single machine or one of a number of shared machines connected to a shared file store. The
problem of transferring cryptographic keys and configuration data between machines was rarely considered and when it was considered was usually implemented badly. Today the typical user owns or makes use of multiple devices they recognize as a computer (laptop, tablet) and an even greater number of devices that they do not recognize as computers but are (almost any device with a display).

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [18].]]

Alice’s personal Mesh.

5.1. Creating a Mesh Profile and Administration Device.

The first step in using the Mesh is to create a personal profile. From the user’s point of view a profile is a collection of all the configuration data for all the Mesh enabled devices and services that they interact with.

Alice> mesh create
Device Profile UDF=MDAA-75W5-HOD4-LSSU-JNYG-WHML-34DU
Personal Profile UDF=MAOZ-3MVE-G5EN-64BI-I3RM-ODFJ-H5W4

Note that the user does not specify the cryptographic algorithms to use. Choice of cryptographic algorithm is primarily the concern of the protocol designer, not the user. The only circumstance in which users would normally be involved in algorithm selection is when there is a transition in progress from one algorithm suite to another.

5.2. Mesh Accounts

Add an account to the personal Mesh:

Alice> account create personal
Account=MAFZ-VUMP-7PU3-4TP7-7MRZ-BKBQ-VMNN

A Mesh Catalog contains a set of entries, each of which has a unique object identifier. Catalog entries may be added, updated or deleted.

By default, all catalog entries are encrypted. Applying the Default Deny principle, in normal circumstances, the Mesh Service is not capable of decrypting any catalog excepting the Contacts catalog which is used as a source of authorization data in the Access Control applied to inbound messaging requests.
For example, the entries in the credentials catalog specify username and password credentials used to access Internet services. Adding credentials to her catalog allows Alice to write scripts that access password protected resources without including the passwords in the scripts themselves:

Alice> password add ftp.example.com alice1 password
alice1@ftp.example.com = [password]
Alice> password add www.example.com alice@example.com newpassword
alice@example.com@www.example.com = [newpassword]
Alice> password list
alice1@ftp.example.com = [password]
alice@example.com@www.example.com = [newpassword]
Alice> password add ftp.example.com alice1 newpassword
alice1@ftp.example.com = [newpassword]
Alice> password get ftp.example.com
alice1@ftp.example.com = [newpassword]

5.3. Mesh Service

A Mesh Service provides an ‘always available’ point of presence that is used to exchange data between devices connected to the connected profile and send and receive Mesh Messages to and from other Mesh users.

To use a Mesh Service, a user creates a Mesh Service account. This is analogous to an SMTP email service but with the important distinction that the protocol is designed to allow users to change their Mesh Service provider at any time they choose with minimal impact.

The account is created by sending an account registration request to the chosen Mesh Service. If accepted, the Mesh Service creates a new account and creates containers to hold the associated catalogs and spools:

Alice> account register alice@example.com
ERROR - Object reference not set to an instance of an object.

As with any other Internet service provision, Mesh Service providers may impose constraints on the use of their service such as the amount of data they send, store and receive and charge a fee for their service.
5.4. Connecting and Authorizing Additional Devices

Having established a Mesh profile, a user may connect any number of devices to it. Connecting a device to a Mesh profile allows it to share data with, control and be controlled by other devices connected to the profile.

Although any type of network capable device may be connected to a Mesh profile, some devices are better suited for use with certain applications than others. Connecting an oven to a Mesh profile could allow it to be controlled through entries to the user’s recipe and calendar catalogs and alert the user when the meal is ready but attempting to use it to read emails or manage Mesh profiles.

Three connection mechanisms are currently specified, each of which provides strong mutual authentication: Direct, PIN and QR.

Since approval of a connection request requires the creation of a signed Connection Assertion, requests must be approved by a device that has access to an administration key authorized by the user’s Master Profile. Such devices are referred to as Administration devices. Administration devices must have data entry (e.g. keyboard) and output (e.g. display) affordances to support any of the currently defined connection mechanisms. The QR code connection mechanism also requires a suitable camera.

It will be noted that the process of connecting a device that contains a preconfigured set of device keys might in principle expose the user to the risk that the manufacturer has retained knowledge of these keys and that this might be used to create a ‘backdoor’.

This risk is controlled using the key co-generation technique described earlier. The original device profile is combined with a device profile provided by the user to create a composite device profile. This ensures that every device uses a unique profile even if they are initialized from a shared firmware image containing a fixed set of device key pairs.

5.4.1. Direct Connection

The direct connection mechanism requires that both the administration device and the device originating the connection request have data entry and output affordances and that it is possible for the user to compare the authentication codes presented by the two devices to check that they are identical.

The connection request is initiated on the device being connected:
Alice2> device request alice@example.com
Witness value = ZHVP-GQK3-2BUZ-NBLU-XDD4-BIJ2-PZCG
Personal Mesh = MAOZ-3MVE-G5EN-64BI-I3RM-ODFJ-H5W4

Using her administration device, Alice gets a list of pending requests. Seeing that there is a pending request matching the witness value presented by the device, Alice accepts it:

Alice> device pending
Alice> device accept NDKO-YOY7-Z6VX-A7LZ-A4SC-SI2W-ZMAX

The new device will now synchronize automatically in response to any Mesh commands. For example, listing the password catalog:

Alice2> password list
ERROR - The feature has not been implemented

5.4.2. Pin Connection

The PIN Connection mechanism is similar to the Direct connection mechanism except that the process is initiated on an administration device by requesting assignment of a new authentication PIN. The PIN is then input to the connecting device to authenticate the request.

The PIN connection mechanism begins with the issue of the PIN:

Alice> account pin
PIN=NBYN-CATU-35TJ-GE6H-CI (Expires=2019-08-14T11:36:37Z)

The PIN code is transmitted out of band to the device being connected:

Alice3> device request alice@example.com /pin=NBYN-CATU-35TJ-GE6H-CI
Witness value = Y5CA-PUCH-MIFW-H3QL-2QV7-L4W5-HLT3
Personal Mesh = MAOZ-3MVE-G5EN-64BI-I3RM-ODFJ-H5W4

Since the request was pre-authorized, it is not necessary for Alice to explicitly accept the connection request but the administration device is needed to create the connection assertion:

Alice> device pending

We can check the device connection by attempting to synchronize to the profile account:

Alice3> account sync
ERROR - The feature has not been implemented
Note that this connection mechanism could be adapted to allow a device with a camera affordance to connect by scanning a QR code on the administration device.

If the Device Profile fingerprint is known at the time the PIN is generated, this can be bound to the PIN authorization assertion to permit connection of a specific device.

5.4.3. EARL/QR Code Connection

The EARL/QR code connection mechanisms are used to connect a constrained device to a Mesh profile by means of an Encrypted Authenticated Resource Locator, typically presented as a QR code on the device itself or its packaging.

Since the meshman tool does not support QR input, it is decoded using a separate tool to recover the UDF EARL which is presented as a command line parameter:

To use the device QR code connection mechanism, we require a Web service that will host the connection document example.com and a MeshService account that the device will attempt to complete the connection by requesting synchronization devices@example.com.

To begin the process we generate a new random key and combine it with the service to create an EARL:

udf://example.com/EBLD-J4FF-5G63-FS7P-TEHG-HMIW-FCMV-E7

Next a device profile is created and preregistered on with the Mesh Service that will provide the hailing service. Since we are only preparing one device it is convenient to do this on the device itself. In a manufacturing scenario, these steps would typically be performed offline in bulk.

Alice4> device pre devices@example.com /key=udf://example.com/EBLD-J4FF-5G63-FS7P-TEHG-HMIW-FCMV-E7
ERROR - Object reference not set to an instance of an object.

Once initialized the device attempts to poll the service for a connection each time it is powered on, when a connection button affordance on the device is pressed or at other times as agreed with the Mesh Service Provider:

Alice4> account sync
ERROR - The feature has not been implemented

To connect the device to her profile, Alice scans the device with her administration device to obtain the UDF. The administration device
retrieves the connection description, decrypts it and then uses the
information in the description to create the necessary Device
Connection Assertion and connect to the device hailing Mesh Service
Account to complete the process:

Alice> device earl udf://example.com/EBLD-J4FF-5G63-FS7P-TEHG-HMIW-FCMV-E7
ERROR - Object reference not set to an instance of an object.

When the device next attempts to connect to the hailing service, it
receives the Device Connection Assertion:

Alice4> account sync
ERROR - The feature has not been implemented

5.5. Contact Requests

As previously stated, every inbound Mesh message is subject to access
control. The user’s contact catalog is used as part of the access
control authentication and authorization mechanism.

By default, the only form of inbound message that is accepted without
authorization in the contact catalog is a contact request. Though
for certain Mesh users (e.g. politicians, celebrities) even contact
requests might require some form of prior approval (e.g. endorsement
by a mutual friend).

A Mesh Contact Assertion may be limited to stating the user’s profile
fingerprint and Mesh Service Account(s). For most purposes however,
it is more convenient to present a Contact Assertion that contains at
least as much information as is typically provided on a business or
calling card:

Alice creates a contact entry for herself:

Alice> contact self email alice@example.com
{
  "Self": true,
  "Key": "NCYX-WCHS-VBNU-KDZT-NW5A-553A-U6CZ",
  "EnvelopedContact": [
    "ewogICJDb250YWN0I6IFt7CiAgI6JtYWlsdG86e2VtYWlsfSJ9XX19"
  ]
}˜˜˜˜

User’s may create multiple Contact Assertions for use in different
circumstances. A user might not want to give their home address to a
business contact or their business address to a personal friend.
5.5.1. Remote

In the most general case, the participants are remote from each other and one user must make a contact request of the other:

Bob requests Alice add him to her contacts catalog:

Bob> message contact alice@example.com
ERROR - The feature has not been implemented

When Alice next checks her messages, she sees the pending contact request from Bob and accepts it. Bob’s contact details are added to her catalog and Bob receives a response containing Alice’s credentials:

Alice> message pending
Alice> message accept tbs

5.5.2. Static QR Code

A DARE contact entry may be exchanged by means of an EARL UDF. This is typically presented by means of a QR code which may be created using the meshman tool and a QR code generator. The resulting QR code may be printed on a business card, laser engraved on a luggage tag, etc.

To accept the contact request, the recipient merely scans the code with a Mesh capable QR code reader. They are asked if they wish to accept the contact request and what privileges they wish to authorize for the new contact.

5.5.3. Dynamic QR Code

If it is possible for the device to generate a new QR code for the contact request, it is possible to support bi-directional exchange of credentials with strong mutual authentication.

For example, Alice selects the contact credential she wishes to pass to Bob on her mobile device which presents an EARL as a QR code. Bob scans the QR code with his mobile device which retrieves Alice’s credential and asks if Bob wishes to accept it and if he wishes to share his credential with Alice. If Bob agrees, his device makes a Remote Contact request authenticated under a key passed to his device with Alice’s Contact Assertion.

The Dynamic QR Code protocol may be applied to support exchange of credentials between larger groups. Enrolling the contact assertions collected in such circumstances in a notarized append only log (e.g.
a DARE Container) provides a powerful basis for building a Web of Trust that is equivalent to but considerably more convenient than participation in PGP Key Signing parties.

5.6. Sharing Confidential Data in the Cloud

As previously discussed, the Mesh makes use of multi-party encryption techniques to mitigate the risk of a device compromise leading to disclosure of confidential data. The Mesh also allows these techniques to be applied to provide Confidential Document Control. This provides data encryption capabilities that are particularly suited to ‘cloud computing’ environments.

A Mesh Encryption Group is a special type of Mesh Service Account that is controlled by one of more group administrators. The Encryption Group Key is a normal ECDH public key used in the normal manner. The decryption key is held by the group administrator. To add a user to the group, the administrator splits the group private key into two parts, a service key and a user key. These parts are encrypted under the public encryption keys of their assigned parties. The encrypted key parts form a decryption entry for the user is added to the Members Catalog of the Encryption Group at the Mesh Service.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-architecture.html [19].]]

When a user needs to decrypt a document encrypted under the group key, they make a request to the Mesh Service which checks to see that they are authorized to read that particular document, have not exceeded their decryption quota, etc. If the request is approved, the service returns the partial decryption result obtained from the service’s key part together with the encrypted user key part. To complete the decryption process, the user decrypts their key part and uses it to create a second partial decryption result which is combined with the first to obtain the key agreement value needed to complete the decryption process.

Alice creates the recryption group groupw@example.com to share confidential information with her closest friends:

Alice> group create groupw@example.com
ERROR - The feature has not been implemented

Bob encrypts a test file but he can’t decrypt it because he isn’t in the group:
Bob> dare encodeTestFile1.txt /out=TestFile1-group.dare /encrypt=groupw@example.com
ERROR - The command is not known.
Bob> dare decode TestFile1-group.dare
ERROR - The feature has not been implemented

Since she is the group administrator, Alice can decrypt the test file using the group decryption key:

Alice> dare decode TestFile1-group.dare
ERROR - The feature has not been implemented

Adding Bob to the group gives him immediate access to any file encrypted under the group key without making any change to the encrypted files:

Alice> dare decode TestFile1-group.dare
ERROR - The feature has not been implemented

Removing Bob from the group immediately withdraws his access.

Alice> group delete groupw@example.com bob@example.com
ERROR - The feature has not been implemented

Bob cannot decrypt any more files (but he may have kept copies of files he decrypted earlier).

Alice> dare decode TestFile1-group.dare
ERROR - The feature has not been implemented

Should requirements demand, the same principle may be applied to achieve separation of duties in the administration roles. Instead of provisioning the group private key to a single administrator, it may be split into two or more parts. Adding a user to the group requires each of the administrators to create a decryption entry for the user and for the service and user to apply the appropriate operations to combine the key parts available to them before use.

These techniques could even be extended to support complex authorization requirements such as the need for 2 out of 3 administrators to approve membership of the group. A set of decryption entries is complete if the sum of the key parts is equal to the private key (modulo the order of the curve).

Thus, if the set of administrators is A, B and C and the private key is k, we can ensure that it requires exactly two administrators to create a complete set of decryption entries by issuing key set \{a\} to A, the key set \{k-a, b\} to B and the key set \{k-a, k-b\} to C (where a and b are randomly generated keys).
5.7. Escrow and Recovery of Keys

One of the chief objections made against deployment of Data Level encryption is that although it provides the strongest possible protection of the confidentiality of the data, loss of the decryption keys means loss of the encrypted data. Thus, a robust and effective key escrow mechanism is essential if use of encryption is to ever become commonplace for stored data.

The use of a ‘life-long’ Mesh profiles raises a similar concern. Loss of a Master Signature Key potentially means the loss of the ability to control devices connected to the profile and the accumulated trust endorsements of other users.

Because of these requirements, Mesh users are strongly advised but not required to create a backup copy of the private keys corresponding to their Master Profile Signature and Escrow keys.

Users may use the key escrow mechanism of their choice including the escrow mechanism supported by the Mesh itself which uses Shamir Secret Sharing to escrow the encryption key for a DARE Envelope containing the private key information.

To escrow a key set, the user specifies the number of key shares to be created and the number required for recovery.

Alice> mesh escrow
ERROR - The cryptographic provider does not permit export of the private key parameters

Recovery of the key data requires the key recovery record and a quorum of the key shares:

Having recovered the Master Signature Key, the user can now create a new master profile authorizing a new administration device which can be used to authenticate access to the Mesh Service Account(s) connected to the master profile.

6. Security Considerations

The security considerations for use and implementation of Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

7. IANA Considerations

This document does not contain actions for IANA
8. Acknowledgements

Comodo Group: Egemen Tas, Melhi Abdulhayo?lu, Rob Stradling, Robin Alden.

9. References

9.1. Normative References

[draft-hallambaker-jsonbcd]

[draft-hallambaker-mesh-cryptography]

[draft-hallambaker-mesh-dare]

[draft-hallambaker-mesh-developer]

[draft-hallambaker-mesh-platform]

[draft-hallambaker-mesh-protocol]

[draft-hallambaker-mesh-schema]

[draft-hallambaker-mesh-security]
[draft-hallambaker-mesh-trust]

[draft-hallambaker-mesh-udf]

[draft-hallambaker-web-service-discovery]


9.2. URIs


Author’s Address

Phillip Hallam-Baker

Email: phill@hallambaker.com
Mathematical Mesh 3.0 Part VIII: Cryptographic Algorithms
draft-hallambaker-mesh-cryptography-02

Abstract

The Mathematical Mesh ‘The Mesh’ is an infrastructure that facilitates the exchange of configuration and credential data between multiple user devices and provides end-to-end security. This document describes the cryptographic algorithm suites used in the Mesh and the implementation of Multi-Party Encryption and Multi-Party Key Generation used in the Mesh.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [1].

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
2. Definitions ............................................. 3
   2.1. Requirements Language ............................. 3
   2.2. Defined Terms .................................... 3
   2.3. Related Specifications ......................... 4
   2.4. Implementation Status ........................... 4
3. Recommended and Required Algorithms ....................... 4
   3.1. Mesh Device ..................................... 4
   3.2. Constrained Device .............................. 5
4. Multi-Party Cryptography ................................... 6
   4.1. Application to Diffie Hellman (not normative) ... 6
   4.2. Multi-Party Key Generation ....................... 6
   4.3. Multi-Party Decryption ........................... 7
   4.4. Mutually Authenticated Key Exchange ............. 7
   4.5. Implementation .................................. 7
       4.5.1. Implementation for Ed25519 and Ed448 ....... 8
       4.5.2. Implementation for X25519 and X448 ....... 8
5. Multi-Party Key Generation ................................ 9
   5.1. Example: Provisioning the Confirmation Service .... 10
6. Multi-Party Decryption ................................... 10
   6.1. Mechanism ....................................... 12
   6.2. Implementation .................................. 13
       6.2.1. Group Creation .............................. 14
       6.2.2. Provisioning a Member ....................... 14
       6.2.3. Message Encryption and Decryption ......... 14
   6.3. Example: Messaging group .......................... 15
7. Mutually Authenticated Key Agreement ....................... 17
8. Security Considerations .................................. 18
9. IANA Considerations ..................................... 18
10. Acknowledgements ....................................... 18
11. Examples ............................................... 18
   11.1. Key Combination .................................. 18
       11.1.1. Ed25519 ................................... 18
       11.1.2. Ed448 .................................... 18
       11.1.3. X25519 ................................... 18
       11.1.4. X448 .................................... 19
   11.2. Group Encryption .................................. 19
       11.2.1. X25519 ................................... 19
       11.2.2. X448 .................................... 19
12. References ............................................. 19
1. Introduction

This document describes the cryptographic algorithm suites used in the Mesh and the implementation of Multi-Party Encryption and Multi-Party Key Generation used in the Mesh.

To allow use of Mesh capabilities on the least capable computing devices currently in use, separate schedules of recommended and required algorithms are specified for Standard Devices and Constrained Devices.

The Constrained device class may be considered to include most 8-bit CPUs equipped with sufficient memory to support the necessary operations. For example an Arduino Mega 2560 which can perform ECDH key agreement and signature operations in times ranging from 3 to 8 seconds. While such a device is clearly not suited to perform such operations routinely, a one-time connection process that takes several minutes to complete need not be of major concern.

The Standard device class may be considered to include the vast majority of general purpose and personal computing devices manufactured since 2010. Even a Raspberry Pi Zero which currently retails at $5 is capable of performing the cryptographic functions required to implement the Mesh with negligible impact on the user.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. 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].

2.2. Defined Terms

The terms of art used in this document are described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture].
2.3. Related Specifications

The architecture of the Mathematical Mesh is described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture]. The Mesh documentation set and related specifications are described in this document.

2.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

3. Recommended and Required Algorithms

To allow implementation of Mesh capabilities on the widest possible range of devices, separate algorithm requirements and recommendations are specified for four classes of device:

Administration Device A general-purpose computing device that is used for Mesh administration functions.

Mesh Device A general-purpose computing device that is not used for Mesh administration functions with sufficient memory and processing power to perform public key cryptography operations without paying particular attention to the impact on performance.

Constrained Device An embedded computing device with limited memory and computing power that offers sufficient processing capabilities to perform occasional public key operations (e.g. during device initialization) but is not suited to repeated operations.

Bridge Device A trusted device that enables Mesh Devices to interoperate with Constrained devices.

Since Administration Devices and Mesh Devices MUST support communication with Mesh Devices and Constrained devices, they MUST meet all the REQUIRED algorithms for both types of device.

3.1. Mesh Device

Support for the following algorithms is REQUIRED:

- SHA-2-512 [SHA-2]
- HMAC-SHA-2-512 [RFC2104]
- HMAC-based Extract-and-Expand Key Derivation Function [RFC5869]
Support for the following algorithms is RECOMMENDED:

- AES-CBC-256 Encryption [FIPS197]
- Advanced Encryption Standard (AES) Key Wrap Algorithm [RFC3394]
- Montgomery Curve Diffie-Hellman Key Agreement X25519 and X448 [RFC7748]
- Edwards-Curve Digital Signature Algorithm Ed25519 and Ed448 [RFC8032]

While the use of GCM is generally preferred over CBC mode in IETF security protocols, this mode is not currently supported by the reference implementation platform.

### 3.2. Constrained Device

Support for the following algorithms is REQUIRED:

- SHA-3-512 [SHA-3]
- HMAC-SHA-3-512 [SHA-3-Derived]
- Poly1305 Authenticated Encryption [RFC8439]
- ChaCha20 Encryption [RFC8439]
- Advanced Encryption Standard (AES) Key Wrap Algorithm [RFC3394]
- Edwards-Curve Digital Signature Algorithm Ed25519 [RFC8032]
- Edwards-Curve Diffie-Hellman Key Agreement Ed25519 [RFC8032]

Use of the Edwards Curves for Signature and Key Agreement allows both functions to be supported by a single library with no reduction in security.
4. Multi-Party Cryptography

The multi-party key generation and multi-party decryption mechanisms used in the Mesh protocols are made possible by the fact that Diffie Hellman key agreement and elliptic curve variants thereof support properties we call the Key Combination Law and the Result Combination Law.

Let \( \{X, x\}, \{Y, y\}, \{E, e\} \) be \( \{\text{public, private}\} \) key pairs.

The Key Combination law states that we can define an operator \( ? \) such that there is a keypair \( \{Z, z\} \) such that:

\[
Z = X ? Y \quad \text{and} \quad z = (x + y) \mod o \quad \text{(where } o \text{ is the order of the group)}
\]

The Result Combination Law states that we can define an operator \( ? \) such that:

\[
\]

4.1. Application to Diffie Hellman (not normative)

For the Diffie Hellman system in a modular field \( p \), \( o = p-1 \) and \( a ? b = a \cdot b \mod p \).

Proof:

By definition, \( X = e^x \mod p \), \( Y = e^y \mod p \), and \( Z = e^z \mod p \).

Therefore,

\[
Z = e^z \mod p = e^{x+y} \mod p = (e^{x}e^{y}) \mod p = e^{x} \mod p \cdot e^{y} \mod p = X \cdot Y
\]

A similar proof may be constructed for the operator \( ? \).

4.2. Multi-Party Key Generation

The Key Combination Law provides the basis for the Key Co-Generation technique used to ensure that the cryptographic keys used in devices connected to a Mesh profile are sufficiently random and have not been compromised by malware or other ‘backdoor’ compromise to the machine during or after manufacture.

For the Diffie Hellman system, the Key Combination law provides all the mechanism needed to implement a Key Co-Generation mechanism. If the Device key is \( \{X, x\} \), the administration device can generate a Co-Generation Key Pair \( \{Y, y\} \) and generate a Device Connection
Assertion for the final public key \( E \) calculated from knowledge of \( X \) and \( Y \) alone. Passing the value \( y \) to the device (using a secure channel) allows it to calculate the corresponding private key \( e \) required to make use of the Device Connection Assertion.

This approach ensures that a party with knowledge of either \( x \) or \( y \) but not both obtains no knowledge of \( e \).

Section REF _Ref5309729 \w \h 5 describes the implementation of these schemes in the Mesh.

### 4.3. Multi-Party Decryption

The Key Combination Law and Result Combination Law provide the basis for the Multi-Party Decryption technique used to support Mesh Encryption Groups.

Section REF _Ref5309538 \w \h 6 describes the application of this technique in the Mesh.

### 4.4. Mutually Authenticated Key Exchange.

The Result Combination Law is used to provide a Key Exchange mechanism that provides mutual authentication of the parties while preserving forward secrecy.

### 4.5. Implementation

For elliptic curve cryptosystems, the operators \( ? \) and \( ? \) are point addition.

Implementing a robust Key Co-Generation for the Elliptic Curve Cryptography schemes described in [RFC7748] and [RFC8032] requires some additional considerations to be addressed.

- The secret scalar used in the EdDSA algorithm is calculated from the private key using a digest function. It is therefore necessary to specify the Key Co-Generation mechanism by reference to operations on the secret scalar values rather than operations on the private keys.

- The Montgomery Ladder traditionally used to perform X25519 and X448 point multiplication does not require implementation of a function to add two arbitrary points. While the steps required to create such a function are fully constrained by the specification, the means of satisfying the constraints is not.
4.5.1. Implementation for Ed25519 and Ed448

The data structures used to implement co-generation of public keys are defined in the main Mesh Reference Guide. This document describes only the additional implementation details.

Note that the ‘private key’ described in [RFC8032] is in fact a seed used to generate a ‘secret scalar’ value that is the value that has the function of being the private key in the ECDH algorithm.

To provision a new public key to a device, the provisioning device:

1. Obtains the device profile of the device(s) to be provisioned to determine the type of key to perform co-generation for. Let the device {public, private} key be \( (D, d) \).
2. Generates a private key \( m \) with the specified number of bytes (32 or 57).
3. Calculates the corresponding public key \( M \).
4. Calculates the Application public key \( A = D + M \) where + is point addition.
5. Constructs the application device entry containing the private key value \( m \) and encrypts under the device encryption key \( d \).

On receipt, the device may at its option use its knowledge of the secret scalar corresponding to \( d \) and \( m \) to calculate the application secret scalar \( a \) or alternatively maintain the two secrets separately and make use of the result combination law to perform private key operations.

4.5.2. Implementation for X25519 and X448

While the point addition function can be defined for any elliptic curve system, it is not necessary to implement point addition to support ECDH key agreement.

In particular, point multiplication using the Montgomery ladder technique over Montgomery curves only operate on the x co-ordinate and only require point doubling operations.

For expediency, the current implementation of the Mesh reference code uses the Edwards curves for both signature and encryption pending announcement of platform support for both algorithms.
5. Multi-Party Key Generation

Multi-Party Key Generation is a capability that is used in the Mesh to enable provisioning of application specific private key pairs to connected devices without revealing any information concerning the application private key of the device.

For example, Alice provisions the confirmation service to her watch. The provisioning device could generate a signature key for the device and encrypt it under the encryption key of the device. But this means that we cannot attribute signatures to the watch with absolute certainty as the provisioning device has had knowledge of the watch signature key. Nor do we wish to use the device signature key for the confirmation service.

Multi-Party Key Generation allows an administration device to provision a connected device with an application specific private key that is specific to that application and no other such that the application can determine the public key of the device but has no knowledge of the private key.

Provisioning an application private key to a device requires the administration device to:

- Generate a new application public key for the device.
- Construct and publish whatever application specific credentials the device requires to use the application.
- Providing the information required to make use of the private key to the device.

Note that while the administration device needs to know the device application public key, it does not require knowledge of the device application private key.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [2].]]

Two party key pair generation.
5.1. Example: Provisioning the Confirmation Service

For example, Alice provisions the confirmation service to her watch. The device profile of the watch specifies an Ed25519 signature key. Note that for production use, Ed448 is almost certainly preferred but Ed25519 has the advantage of more compact presentation.

TBS:

The provisioning device could generate a signature key for the device and encrypt it under the encryption key of the device. But this means that we cannot attribute signatures to the watch with absolute certainty as the provisioning device has had knowledge of the watch signature key. Nor do we wish to use the device signature key for the confirmation service.

Instead, the provisioning device generates a companion keypair. A random seed is generated.

TBS:

A key derivation function (HKDF) is used to derive a 255 bit secret scalar.

TBS:

The provisioning device can calculate the public key of the composite keypair by adding the public keys of the device profile and the companion public key:

TBS:

The provisioning device encrypts the private key of the companion keypair under the encryption key of the device.

TBS:

The provisioning device calculates the private key of the composite keypair by adding the two private key values and verifies that scalar multiplication of the base point returns the composite public key value.

6. Multi-Party Decryption

A key limitation of most deployed messaging systems is that true end-to-end confidentiality is only achieved for a limited set of communication patterns. Specifically, bilateral communications (Alice sends a message to Bob) or broadcast communications to a known
set of recipients (Alice sends a message to Bob, Carol and Doug). These capabilities do not support communication patterns where the set of recipients changes over time or is confidential. Yet such requirements commonly occur in situations such as sending a message to a mailing list whose membership isn’t known to the sender, or creating a spreadsheet whose readership is to be limited to authorized members of the ‘accounting’ team.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [3].]]

Traditional End-to-End Encryption is static.

The mathematical approach that makes key co-generation possible may be applied to support a public key encryption mode in which encryption is performed as usual but decryption requires the use of multiple keys. This approach is variously described in the literature as distributed key generation and proxy re-encryption [Blaze98].

The approach specified in this document borrows aspects of both these techniques. This combined approach is called ‘recryption’. Using recryption allows a sender to send a message to a group of users whose membership is not known to the sender at the time the message is sent and can change at any time.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [4].]]

Recryption supports End-to-End Encryption in dynamic groups.

Proxy re-encryption provides a technical capability that meets the needs of such communication patterns. Conventional symmetric key cryptography uses a single key to encrypt and decrypt data. Public key cryptography uses two keys, the key used to encrypt data is separate from the key used to decrypt. Proxy re-encryption introduces a third key (the recryption key) that allows a party to permit an encrypted data packet to be decrypted using a different key without permitting the data to be decrypted.

The introduction of a recryption key permits end-to-end confidentiality to be preserved when a communication pattern requires that some part of the communication be supported by a service.
The introduction of a third type of key, the recryption key permits two new roles to be established, that of an administrator and recryption service. There are thus four parties:

- Administrator  Holder of Decryption Key, Creator of Recryption Keys
- Sender  Holder of Encryption Key
- Recryption Service  Holder of Recryption keys
- Receiver  Holder of personal decryption key

The communication between these parties is shown in Figure X below:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [5].]]

Mesh/Recrypt Parties

The information stored at the recryption service is necessary but not sufficient to decrypt the message. Thus, no disclosure of the message plaintext occurs even in the event that an attacker gains full knowledge of all the information stored by the recryption service.

6.1. Mechanism

The mechanism used to support recryption is the same as the mechanism used to support key co-generation except that this time, instead of combining two keys to create one, the private component of a decryption key (i.e. the private key) is split into two parts, a recryption key and a decryption key.

Recall that the key combination law for Diffie Hellman crypto-systems states that we can add two private keys to get a third. It follows that we can split the private key portion of a keypair \( \{G, g\} \) into two parts by choosing a random number that is less than the order of the Diffie-Hellman group to be our first key \( x \). Our second key is \( y = g - r \mod o \), where \( o \) is the order of the group.

Having generated \( x, y \), we can use these to perform private key agreement operations on a public key \( E \) and then use the result combination law to obtain the same result that we would have obtained using \( g \).
One means of applying this mechanism to recryption would be to generate a different random value x for each member of the group and store it at the recryption service and communicate the value y to the member via a secure channel. Applying this approach, we can clearly see that the recryption service gains no information about the value of the private key since the only information it holds is a random number which could have been generated without any knowledge of the group private key.

[RFC8032] requires that implementations derive the scalar secret by taking a cryptographic digest of the private key. This means that either the client or the service must use a non-compliant implementation. Given this choice, it is preferable to require that the non-standard implementation be required at the service rather than the client. This limits the scope of the non-conformant key derivation approach to the specialist recryption service and ensures that the client enforce the requirement to generate the private key component by means of a digest.

6.2. Implementation

Implementation of recryption in the Mesh has four parts:

- Creation and management of the recryption group.
- Provisioning of members to a recryption group.
- Message encryption.
- Message decryption.

These operations are all performed using the same catalog and messaging infrastructure provided by the Mesh for other purposes.

Each recryption group has its own independent Mesh account. This has many advantages:

- Administration of the recryption group may be spread across multiple Mesh users or transferred from one user to another without requiring specification of a separate management protocol to support these operations.
- The recryption account address can be used by Mesh applications such as group messaging, conferencing, etc. as a contact address.
- The contact request service can be used to notify members that they have been granted membership in the group.
6.2.1. Group Creation

Creation of a Recryption group requires the steps of:

- Generating the recryption group key pair
- Creating the recryption group account
- Generating administrator record for each administrator.
- Publishing the administrator records to the recryption catalog.

Note that in principle, we could make use of the key combination law to enable separation of duties controls on administrators so that provisioning of members required multiple administrators to participate in the process. This is left to future versions.

6.2.2. Provisioning a Member

To provision a user as a member of the recryption group, the administrator requires their current recryption profile. The administrator MAY obtain this by means of a contact service request. As with any contact service request, this request is subject to access control and MAY require authorization by the intended user before the provisioning can proceed.

Having obtained the user’s recryption profile, the administration tool generates a decryption private key for the user and encrypts it under the member’s key to create the encrypted decryption key entry.

The administration tool then computes the secret scalar from the private key and uses this together with the secret scalar of the recryption group to compute the recryption key for the member. This value and the encrypted decryption key entry are combined to form the recryption group membership record which is published to the catalog.

6.2.3. Message Encryption and Decryption

Encryption of a messages makes use of DARE Message in exactly the same manner as any other encryption. The sole difference being that the recipient entry for the recryption operation MUST specify the recryption group address an not just the key fingerprint. This allows the recipient to determine which recryption service to contact to perform the recryption operation.

To decrypt a message, the recipient makes an authenticated recryption request to the specified recryption service specifying:
o The recipient entry to be used for decryption
o The fingerprint of the decryption key(s) the device would like to make use of.
o Whether or not the encrypted decryption key entry should be returned.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-cryptography.html [6].]]

Two key decryption.

The recryption service searches the catalog for the corresponding recryption group to find a matching entry. If found and if the recipient and proposed decryption key are dully authorized for the purpose, the service performs the key agreement operation using the recryption key specified in the entry and returns the result to the recipient.

The recipient then decrypts the recryption data entry using its device decryption key and uses the group decryption key to calculate the other half of the result. The two halves of the result are then added to obtain the key agreement value that is then used to decrypt the message.

6.3. Example: Messaging group

Alice creates a recryption group. The group encryption and signature key parameters are:

TBS:

To verify the proper function of the group, Alice creates a test message and encrypts it under the group key.

TBS:
TBS:

Alice decides to add Bob to the group. Bob’s recryption profile is:

TBS:

The decryption key is specified in the same way as any other Ed25519 private key using the hash of a private key seed value:
TBS:

The recryption key is the group secret scalar minus (mod p) the secret scalar of Bob’s private key:

TBS:

The Recryption entry consists of Bob’s address, the recryption key and the decryption key encrypted under Bob’s encryption key:

TBS:

The group administration tool creates a notification request to tell Bob that he has been made a member of the new group and signs it using the group signature key. The recryption entry and the notification are then sent to the recryption service:

TBS:

The notification message contains a link to the test message. When he accepts membership of the group, Bob clicks on the link to test that his membership has been fully provisioned. Providing an explicit test mechanism avoids the problem that might otherwise occur in which the message spool fills up with test messages being posted.

Bob’s Web browser requests the recryption data for the test message. The request is authenticated and encrypted under Bob’s device keys. The plaintext of the message is:

TBS:

The plaintext of the response contains the additional information Bob’s Web browser needs to complete the decryption process:

TBS:

The Web browser decrypts the private key and uses it to calculate the decryption value:

TBS:

The key agreement value is obtained by point addition of the recryption and decryption values:

TBS:

This value allows the test message to be decrypted.
7. Mutually Authenticated Key Agreement

Diffie Hellman key agreement using the authenticated public keys of the principals provides mutual authentication of those principals.

For example, if Alice’s key pair is \( \{a, A\} \) and Bob’s key pair is \( \{b, B\} \), the Diffie Hellman key agreement value \( \text{DH} (a, B) = \text{DH} (b, A) \) can only be generated from the public information if \( a \) or \( b \) is known.

The chief disadvantage of this approach is that it only allows Alice and Bob to establish a single shared secret that will never vary and does not provide forward secrecy. To avoid this, cryptographic protocols usually perform the key agreement against an ephemeral key and either accept that the client key is not authenticated or perform multiple key agreements and combine the results.

Using the Result Combination Law allows a key agreement mechanism to combine the benefits of mutual authentication with the use of ephemeral keys without the need for multiple private key operations or additional round trips.

In its simplest form, the key exchange has two parties which we refer to as the client and the server. The client being the party that initiates the protocol exchange and the server being the party that responds. Let the public key pair of the client be \( \{a, A\} \) and that of the server \( \{b, B\} \).

Two versions of the key agreement mechanism are specified:

Client ephemeral The client contributes an ephemeral key pair \( \{n_A, N_A\} \). The effective public key of the client is \( A \otimes N_A \).

The server uses its public key \( B \).

The key agreement value is \( \text{DH} (a + n_A, B) = \text{DH} (b, A \otimes N_A) \)

Dual ephemeral The client contributes an ephemeral key pair \( \{n_A, N_A\} \). The effective public key of the client is \( A \otimes N_A \).

The server contributes an ephemeral key pair \( \{n_B, N_B\} \). The effective public key of the client is \( B \otimes N_B \).

The key agreement value is \( \text{DH} (a + n_A, B \otimes N_B) = \text{DH} (b + n_B, A \otimes N_A) \)

The function of the ephemeral key is effectively that of a nonce but it is shared with the counter-party as a public key value.
The dual ephemeral approach has the advantage that it limits the scope for side channel attacks as both sides have contributed unknown information to the key agreement value. The disadvantage of this approach is that the key agreement value can only be calculated after the server has provided its ephemeral.

Implementations MAY take advantage of the result combination law to enable private key operations involving the authenticated key (or a contribution to it) to be performed in trustworthy hardware.

An advantage of this key exchange mechanism over the traditional TLS key exchange approach is that no signature operation is involved, thus ensuring that either party can repudiate the exchange and thus the claim that they were in communication.

The master secret is calculated from the key agreement value in the usual fashion. For ECDH algorithms, this comprises the steps of converting the key agreement value to an octet string which forms the input to a Key Derivation Function.

8. Security Considerations

The security considerations for use and implementation of Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

9. IANA Considerations

This document requires no IANA actions.

10. Acknowledgements

A list of people who have contributed to the design of the Mesh is presented in [draft-hallambaker-mesh-architecture].

11. Examples

11.1. Key Combination

11.1.1. Ed25519

11.1.2. Ed448

11.1.3. X25519
11.1.4. X448

11.2. Group Encryption

11.2.1. X25519

11.2.2. X448

12. References

12.1. Normative References


draft-hallambaker-mesh-architecture]

draft-hallambaker-mesh-security]


12.2. Informative References

[Blaze98] "[Reference Not Found!]".


12.3. URIs


Author’s Address

Phillip Hallam-Baker

Email: phill@hallambaker.com
Abstract

This document describes the Data At Rest Encryption (DARE) Envelope and Container syntax.

The DARE Envelope syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary content data.

The DARE Container syntax describes an append-only sequence of entries, each containing a DARE Envelope. DARE Containers may support cryptographic integrity verification of the entire data container content by means of a Merkle tree.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [1].

Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................ 4
   1.1. Encryption and Integrity .......................... 5
       1.1.1. Key Exchange .................................. 5
       1.1.2. Data Erasure .................................. 6
   1.2. Signature ......................................... 7
       1.2.1. Signing Individual Plaintext Envelopes ........ 7
       1.2.2. Signing Individual Encrypted Envelopes ........ 7
       1.2.3. Signing sequences of envelopes ................ 8
   1.3. Container ......................................... 8
       1.3.1. Container Format .............................. 8
       1.3.2. Write ......................................... 9
       1.3.3. Encryption and Authentication ................. 10
       1.3.4. Integrity and Signature ..................... 10
       1.3.5. Redaction ................................... 11
       1.3.6. Alternative approaches ....................... 11
       1.3.7. Efficiency ................................... 12
2. Definitions .......................................... 12
   2.1. Related Specifications ........................... 12
   2.2. Requirements Language ............................ 13
   2.3. Defined terms .................................... 13
3. DARE Envelope Architecture ............................ 14
   3.1. Processing Considerations ........................ 15
   3.2. Content Metadata and Annotations ................. 15
   3.3. Encoded Data Sequence ............................ 16
   3.4. Encryption and Integrity ........................ 17
       3.4.1. Key Exchange .................................. 18
       3.4.2. Key Identifiers ............................... 18
       3.4.3. Salt Derivation ................................ 19
       3.4.4. Key Derivation ................................ 19
   3.5. Signature ......................................... 20
   3.6. Algorithms ....................................... 20
       3.6.1. Field: kwd ................................... 20
4. DARE Container Architecture .......................... 21
   4.1. Container Navigation .............................. 21
       4.1.1. Tree .......................................... 22
       4.1.2. Position Index ................................. 22
4.1.3. Metadata Index .................................. 22
4.2. Integrity Mechanisms .................................. 23
4.2.1. Digest Chain calculation ............................ 23
4.2.2. Binary Merkle tree calculation ....................... 23
4.2.3. Signature ........................................ 23
5. DARE Schema ........................................... 24
5.1. Message Classes ...................................... 24
5.1.1. Structure: DareEnvelopeSequence .................... 24
5.2. Header and Trailer Classes ............................ 25
5.2.1. Structure: DareTrailer ............................. 25
5.2.2. Structure: DareHeader .............................. 25
5.3. Cryptographic Data .................................... 27
5.3.1. Structure: DareSigner .............................. 27
5.3.2. Structure: X509Certificate ......................... 27
5.3.3. Structure: DareSignature ........................... 27
5.3.4. Structure: DareRecipient ........................... 28
6. DARE Container Schema ................................ 28
6.1. Container Headers ................................... 28
6.1.1. Structure: ContainerEntry ........................ 28
6.1.2. Structure: ContainerHeaderFirst ................... 28
6.1.3. Structure: ContainerHeader ......................... 29
6.2. Content Metadata Structure .......................... 30
6.2.1. Structure: ContentMeta ............................ 30
6.3. Index Structures .................................... 30
6.3.1. Structure: ContainerIndex ......................... 30
6.3.2. Structure: IndexPosition .......................... 30
6.3.3. Structure: KeyValue ............................... 31
6.3.4. Structure: IndexMeta .............................. 31
7. Dare Container Applications ............................. 31
7.1. Catalog ............................................. 31
7.2. Spool .............................................. 32
7.3. Archive ............................................ 33
8. Future Work ........................................... 33
8.1. Terminal integrity check .............................. 33
8.2. Terminal index record ............................... 33
8.3. Deferred indexing .................................. 33
9. Security Considerations ................................. 34
9.1. Encryption/Signature nesting .......................... 34
9.2. Side channel ....................................... 34
9.3. Salt reuse ......................................... 34
10. IANA Considerations .................................. 34
11. Acknowledgements ..................................... 34
12. Appendix A: DARE Envelope Examples and Test Vectors .... 34
13. Test Examples ....................................... 34
13.1. Plaintext Message .................................. 35
13.2. Plaintext Message with EDS ......................... 35
13.3. Encrypted Message ................................. 35
13.4. Signed Message .................................... 37
1. Introduction

This document describes the Data At Rest Encryption (DARE) Envelope and Container Syntax. The DARE Envelope syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content. The DARE Container syntax describes an append-only sequence of data frames, each containing a DARE Envelope that supports efficient incremental signature and encryption.

The DARE Envelope Syntax is based on a subset of the JSON Web Signature [RFC7515] and JSON Web Encryption [RFC7516] standards and shares many fields and semantics. The processing model and data structures have been streamlined to remove alternative means of specifying the same content and to enable multiple data sequences to be signed and encrypted under a single master encryption key without compromise to security.

A DARE Envelope consists of a Header, Payload and an optional Trailer. To enable single pass encoding and decoding, the Header contains all the information required to perform cryptographic processing of the Payload and authentication data (digest, MAC, signature values) MAY be deferred to the Trailer section.

A DARE Container is an append-only log format consisting of a sequence of frames. Cryptographic enhancements (signature, encryption) may be applied to individual frames or to sets of frames. Thus, a single key exchange may be used to provide a master key to encrypt multiple frames and a single signature may be used to authenticate all the frames in the container up to and including the frame in which the signature is presented.

The DARE Envelope syntax may be used either as a standalone cryptographic message syntax or as a means of presenting a single
DARE Container frame together with the complete cryptographic context required to verify the contents and decrypt them.

1.1. Encryption and Integrity

A key innovation in the DARE Envelope Syntax is the separation of key exchange and data encryption operations so that a Master Key (MK) established in a single exchange to be applied to multiple data sequences. This means that a single public key operation MAY be used to encrypt and/or authenticate multiple parts of the same DARE Envelope or multiple frames in a DARE Container.

To avoid reuse of the key and to avoid the need to communicate separate IVs, each octet sequence is encrypted under a different encryption key (and IV if required) derived from the Master Key by means of a salt that is unique for each octet sequence that is encrypted. The same approach is used to generate keys for calculating a MAC over the octet sequence if required. This approach allows encryption and integrity protections to be applied to the envelope payload, to header or trailer fields or to application defined Enhanced Data Sequences in the header or trailer.

1.1.1. Key Exchange

Traditional cryptographic containers describe the application of a single key exchange to encryption of a single octet sequence. Examples include PCKS#7/CMS [RFC2315], OpenPGP [RFC4880] and JSON Web Encryption [RFC7516].

To encrypt data using RSA, the encoder first generates a random encryption key and initialization vector (IV). The encryption key is encrypted under the public key of each recipient to create a per-recipient decryption entry. The encryption key, plaintext and IV are used to generate the ciphertext (figure 1).

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [2].]]

Monolithic Key Exchange and Encrypt

This approach is adequate for the task of encrypting a single octet stream. It is less than satisfactory when encrypting multiple octet streams or very long streams for which a rekeying operation is desirable.
In the DARE approach, key exchange and key derivation are separate operations and keys MAY be derived for encryption or integrity purposes or both. A single key exchange MAY be used to derive keys to apply encryption and integrity enhancements to multiple data sequences.

The DARE key exchange begins with the same key exchange used to produce the CEK in JWE but instead of using the CEK to encipher data directly, it is used as one of the inputs to a Key Derivation Function (KDF) that is used to derive parameters for each block of data to be encrypted. To avoid the need to introduce additional terminology, the term ‘CEK’ is still used to describe the output of the key agreement algorithm (including key unwrapping if required) but it is more appropriately described as a Master Key (figure 2).

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [3].]]

Exchange of Master Key

A Master Key may be used to encrypt any number of data items. Each data item is encrypted under a different encryption key and IV (if required). This data is derived from the Master Key using the HKDF function [RFC5869] using a different salt for each data item and separate info tags for each cryptographic function (figure 3).

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [4].]]

Data item encryption under Master Key and per-item salt.

This approach to encryption offers considerably greater flexibility allowing the same format for data item encryption to be applied at the transport, message or field level.

1.1.2. Data Erasure

Each encrypted DARE Envelope specifies a unique Master Salt value of at least 128 bits which is used to derive the salt values used to derive cryptographic keys for the envelope payload and annotations.

Erasure of the Master Salt value MAY be used to effectively render the envelope payload and annotations undecipherable without altering
the envelope payload data. The work factor for decryption will be \( O(2^{128}) \) even if the decryption key is compromised.

1.2. Signature

As with encryption, DARE Envelope signatures MAY be applied to an individual envelope or a sequence of envelope.

1.2.1. Signing Individual Plaintext Envelopes

When an individual plaintext envelope is signed, the digest value used to create the signature is calculated over the binary value of the payload data. That is, the value of the payload before the encoding (Base-64, JSON-B) is applied.

1.2.2. Signing Individual Encrypted Envelopes

When an individual plaintext envelope is signed, the digest value used to create the signature is calculated over the binary value of the payload data. That is, the value of the payload after encryption but before the encoding (Base-64, JSON-B) is applied.

Use of signing and encryption in combination presents the risk of subtle attacks depending on the order in which signing and encryption take place [Davis2001].

Naïve approaches in which an envelope is encrypted and then signed present the possibility of a surreptitious forwarding attack. For example, Alice signs an envelope and sends it to Mallet who then strips off Alice’s signature and sends the envelope to Bob.

Naïve approaches in which an envelope is signed and then encrypted present the possibility of an attacker claiming authorship of a ciphertext. For example, Alice encrypts a ciphertext for Bob and then signs it. Mallet then intercepts the envelope and sends it to Bob.

While neither attack is a concern in all applications, both attacks pose potential hazards for the unwary and require close inspection of application protocol design to avoid exploitation.

To prevent these attacks, each signature on an envelope that is signed and encrypted MUST include a witness value that is calculated by applying a MAC function to the signature value as described in section XXX.
1.2.3. Signing sequences of envelopes

To sign multiple envelopes with a single signature, we first construct a Merkle tree of the envelope payload digest values and then sign the root of the Merkle tree.

[This is not yet implemented but will be soon.]

1.3. Container

DARE Container is a message and file syntax that allows a sequence of data frames to be represented with cryptographic integrity, signature, and encryption enhancements to be constructed in an append only format.

The format is designed to meet the requirements of a wide range of use cases including:

- Recording transactions in persistent storage.
- Synchronizing transaction logs between hosts.
- File archive.
- Message spool.
- Signing and encrypting single data items.
- Incremental encryption and authentication of server logs.

1.3.1. Container Format

A Container consists of a sequence of variable length Frames. Each frame consists of a forward length indicator, the framed data and a reverse length indicator. The reverse length indicator is written out backwards allowing the length and thus the frame to be read in the reverse direction:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [5].]]

JBCD Bidirectional Frame

Each frame contains a single DARE Envelope consisting of a Header, Payload and Trailer (if required). The first frame in a container describes the container format options and defaults. These include
the range of encoding options for frame metadata supported and the container profiles to which the container conforms.

All internal data formats support use of pointers of up to 64 bits allowing containers of up to 18 exabytes to be written.

Five container types are currently specified:

Simple  The container does not provide any index or content integrity checks.

Tree  Frame headers contain entries that specify the start position of previous frames at the apex of the immediately enclosing binary tree. This enables efficient random access to any frame in the file.

Digest  Each frame trailer contains a PayloadDigest field. Modification of the payload will cause verification of the PayloadDigest value to fail on that frame.

Chain  Each frame trailer contains PayloadDigest and ChainDigest fields allowing modifications to the payload data to be detected. Modification of the payload will cause verification of the PayloadDigest value to fail on that frame and verification of the ChainDigest value to fail on all subsequent frames.

Merkle Tree  Frame headers contain entries that specify the start position of previous frames at the apex of the immediately enclosing binary tree. Frame Trailers contain TreeDigestPartial and TreeDigestFinal entries forming a Merkle digest tree.

1.3.2.  Write

In normal circumstances, Containers are written as an append only log. As with Envelopes, integrity information (payload digest, signatures) is written to the entry trailer. Thus, large payloads may be written without the need to buffer the payload data provided that the content length is known in advance.

Should exceptional circumstances require, Container entries MAY be erased by overwriting the Payload and/or parts of the Header content without compromising the ability to verify other entries in the container. If the entry Payload is encrypted, it is sufficient to erase the container salt value to render the container entry effectively inaccessible (though recovery might still be possible if the original salt value can be recovered from the storage media.)
1.3.3. Encryption and Authentication

Frame payloads and associated attributes MAY be encrypted and/or authenticated in the same manner as Envelopes.

Incremental encryption is supported allowing encryption parameters from a single public key exchange operation to be applied to encrypt multiple frames. The public key exchange information is specified in the first encrypted frame and subsequent frames encrypted under those parameters specify the location at which the key exchange information is to be found by means of the ExchangePosition field which MUST specify a location that is earlier in the file.

To avoid cryptographic vulnerabilities resulting from key re-use, the DARE key exchange requires that each encrypted sequence use an encryption key and initialization vector derived from the master key established in the public key exchange by means of a unique salt.

Each Envelope and by extension, each Container frame MUST specify a unique salt value of at least 128 bits. Since the encryption key is derived from the salt value by means of a Key Derivation Function, erasure of the salt MAY be used as a means of rendering the payload plaintext value inaccessible without changing the payload value.

1.3.4. Integrity and Signature

Signatures MAY be applied to a payload digest, the final digest in a chain or tree. The chain and tree digest modes allow a single signature to be used to authenticate all frame payloads in a container.

The tree signature mode is particularly suited to applications such as file archives as it allows files to be verified individually without requiring the signer to sign each individually. Furthermore, in applications such as code signing, it allows a single signature to be used to verify both the integrity of the code and its membership of the distribution.

As with DARE Envelope, the signature mechanism does not specify the interpretation of the signature semantics. The presence of a signature demonstrates that the holder of the private key applied it to the specified digest value but not their motive for doing so. Describing such semantics is beyond the scope of this document and is deferred to future work.
1.3.5. Redaction

The chief disadvantage of using an append-only format is that containers only increase in size. In many applications, much of the data in the container becomes redundant or obsolete and a process analogous to garbage collection is required. This process is called redaction.

The simplest method of redaction is to create a new container and sequentially copy each entry from the old container to the new, discarding redundant frames and obsolete header information.

For example, partial index records may be consolidated into a single index record placed in the last frame of the container. Unnecessary signature and integrity data may be discarded and so on.

While redaction could in principle be effected by moving data in-place in the existing container, supporting this approach in a robust fashion is considerably more complex as it requires backward references in subsequent frames to be overridden as each frame is moved.

1.3.6. Alternative approaches

Many file proprietary formats are in use that support some or all of these capabilities but only a handful have public, let alone open, standards. DARE Container is designed to provide a superset of the capabilities of existing message and file syntaxes, including:

- Cryptographic Message Syntax [RFC5652] defines a syntax used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

- The.ZIP File Format specification [ZIPFILE] developed by Phil Katz.

- The BitCoin Block chain [BLOCKCHAIN].

- JSON Web Encryption and JSON Web Signature

Attempting to make use of these specifications in a layered fashion would require at least three separate encoders and introduce unnecessary complexity. Furthermore, there is considerable overlap between the specifications providing multiple means of achieving the same ends, all of which must be supported if decoders are to work reliably.
1.3.7. Efficiency

Every data format represents a compromise between different concerns, in particular:

- **Compactness** The space required to record data in the encoding.
- **Memory Overhead** The additional volatile storage (RAM) required to maintain indexes etc. to support efficient retrieval operations.
- **Number of Operations** The number of operations required to retrieve data from or append data to an existing encoded sequence.
- **Number of Disk Seek Operations** Optimizing the response time of magnetic storage media to random access read requests has traditionally been one of the central concerns of database design. The DARE Container format is designed to the assumption that this will cease to be a concern as solid state media replaces magnetic.

While the cost of storage of all types has declined rapidly over the past decades, so has the amount of data to be stored. DARE Container represents a pragmatic balance of these considerations for current technology. In particular, since payload volumes are likely to be very large, memory and operational efficiency are considered higher priorities than compactness.

2. Definitions

2.1. Related Specifications

The DARE Envelope and Container formats are based on the following existing standards and specifications.

- **Object serialization** The JSON-B [draft-hallambaker-jsonbcd] encoding is used for object serialization. This encoding is an extension of the JavaScript Object Notation (JSON) [RFC7159].

- **Message syntax** The cryptographic processing model is based on JSON Web Signature (JWS) [RFC7515], JSON Web Encryption (JWE) [RFC7516] and JSON Web Key (JWK) [RFC7517].

- **Cryptographic primitives**. The HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] and Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] are used.

The Uniform Data Fingerprint method of presenting data digests is used for key identifiers and other purposes [draft-hallambaker-mesh-udf].
Cryptographic algorithms The cryptographic algorithms and identifiers described in JSON Web Algorithms (JWA) [RFC7518] are used together with additional algorithms as defined in the JSON Object Signing and Encryption IANA registry [IANAJOSE].

2.2. 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].

2.3. Defined terms

The terms "Authentication Tag", "Content Encryption Key", "Key Management Mode", "Key Encryption", "Direct Key Agreement", "Key Agreement with Key Wrapping" and "Direct Encryption" are defined in the JWE specification [RFC7516].


Annotated Envelope A DARE Envelope that contains an Annotations field with at least one entry.

Authentication Data A Message Authentication Code or authentication tag.

Complete Envelope A DARE envelope that contains the key exchange information necessary for the intended recipient(s) to decrypt it.

Detached Envelope A DARE envelope that does not contain the key exchange information necessary for the intended recipient(s) to decrypt it.

Encryption Context The master key, encryption algorithms and associated parameters used to generate a set of one or more enhanced data sequences.

Encoded data sequence (EDS) A sequence consisting of a salt, content data and authentication data (if required by the encryption context).

Enhancement Applying a cryptographic operation to a data sequence. This includes encryption, authentication and both at the same time.
Generator  The party that generates a DARE envelope.

Group Encryption Key  A key used to encrypt data to be read by a group of users. This is typically achieved by means of some form of proxy re-encryption or distributed key generation.

Group Encryption Key Identifier  A key identifier for a group encryption key.

Master Key (MK)  The master secret from which keys are derived for authenticating enhanced data sequences.

Recipient  Any party that receives and processes at least some part of a DARE envelope.

Related Envelope  A set of DARE envelopes that share the same key exchange information and hence the same Master Key.

Uniform Data Fingerprint (UDF)  The means of presenting the result of a cryptographic digest function over a data sequence and content type identifier specified in the Uniform Data Fingerprint specification [draft-hallambaker-mesh-udf]

3. DARE Envelope Architecture

A DARE Envelope is a sequence of three parts:

Header  A JSON object containing information a reader requires to begin processing the envelope.

Payload  An array of octets.

Trailer  A JSON object containing information calculated from the envelope payload.

For example, the following sequence is a JSON encoded Envelope with an empty header, a payload of zero length and an empty trailer:

[ {}, "", {} ]

DARE Envelopes MAY be encoded using JSON serialization or a binary serialization for greater efficiency.

JSON  Offers compatibility with applications and libraries that support JSON. Payload data is encoded using Base64 incurring a 33% overhead.
JSON-B  A superset of JSON encoding that permits binary data to be encoded as a sequence of length-data segments. This avoids the Base64 overhead incurred by JSON encoding. Since JSON-B is a superset of JSON encoding, an application can use a single decoder for either format.

JSON-C  A superset of JSON-C which provides additional efficiency by allowing field tags and other repeated string data to be encoded by reference to a dictionary. Since JSON-C is a superset of JSON and JSON-B encodings, an application can use a single decoder for all three formats.

DARE Envelope processors MUST support JSON serialization and SHOULD support JSON-B serialization.

3.1. Processing Considerations

The DARE Envelope Syntax supports single pass encoding and decoding without buffering of data. All the information required to begin processing a DARE envelope (key agreement information, digest algorithms), is provided in the envelope header. All the information that is derived from envelope processing (authentication codes, digest values, signatures) is presented in the envelope trailer.

The choice of envelope encoding does not affect the semantics of envelope processing. A DARE Envelope MAY be reserialized under the same serialization or converted from any of the specified serialization to any other serialization without changing the semantics or integrity properties of the envelope.

3.2. Content Metadata and Annotations

A header MAY contain header fields describing the payload content. These include:

ContentType  Specifies the IANA Media Type [RFC6838] .

Annotations  A list of Encoded Data Sequences that provide application specific annotations to the envelope.

For example, consider the following mail message:

From: Alice@example.com
To: bob@example.com
Subject: TOP-SECRET Product Launch Today!

The CEO told me the product launch is today. Tell no-one!
Existing encryption approaches require that header fields such as the subject line be encrypted with the body of the message or not encrypted at all. Neither approach is satisfactory. In this example, the subject line gives away important information that the sender probably assumed would be encrypted. But if the subject line is encrypted together with the message body, a mail client must retrieve at least part of the message body to provide a ‘folder’ view.

The plaintext form of the equivalent DARE Message encoding is:

```
[{
    "cty":"application/example-mail",
    "Annotations":["iAEBiBgdGcm9tOiBBbGlj2UB1eGFtcGxlLmNvbYgA",
                  "iAECiBNUbzogYm9iQGV4YW1wbGUuY29tIAA",
                  "iAEDiClTdWJqZWN0OiBUT1AtU0VDUkVUIFByb2R1Y3QgTGF1bmNoIFRvZGF5
                  iYgA"
                ],
    "VGhlIENFTyB0b2xkIG1lIHRoZSBwcm9kdWN0IGxhdW5jaCBpcyB0b2RheS4gVG
                VsBCBubylvbmUh"
}]
```

This contains the same information as before but the mail message headers are now presented as a list of Encoded Data Sequences.

3.3. Encoded Data Sequence

An encoded data sequence (EDS) is a sequence of octets that encodes a data sequence according to cryptographic enhancements specified in the context in which it is presented. An EDS MAY be encrypted and MAY be authenticated by means of a MAC. The keys and other cryptographic parameters used to apply these enhancements are derived from the cryptographic context and a Salt prefix specified in the EDS itself.

An EDS sequence contains exactly three binary fields encoded in JSON-B serialization as follows:

- **Salt Prefix** A sequence of octets used to derive the encryption key, Initialization Vector and MAC key as required.
- **Body** The plaintext or encrypted content.
- **Authentication Tag** The authentication code value in the case that the cryptographic context specifies use of authenticated encryption or a MAC, otherwise is a zero-length field.
Requiring all three fields to be present, even in cases where they are unnecessary simplifies processing at the cost of up to six additional data bytes.

The encoding of the ‘From’ header of the previous example as a plaintext EDS is as follows:

```
88 01
01
88 17
46 72 6f 6d 3a 20 41 6c 69 63 65 40 65 78 61 6d 70 6c 65 2e 63 6f 6d
88 00
```

3.4. Encryption and Integrity

Encryption and integrity protections MAY be applied to any DARE Envelope Payload and Annotations.

The following is an encrypted version of the message shown earlier. The payload and annotations have both increased in size as a result of the block cipher padding. The header now includes Recipients and Salt fields to enable the content to be decoded.

```
[{
  "enc":"A256CBC",
  "Salt":"2TJKJ5AVAag9RVleG4vplQ",
  "cty":"application/example-mail",
  "Annotations":"
  "iAEBiCBs9DEywpxAYyiC6UM2drKV6PRFmdsrdd6wBdkvictcIAaQ",
  "iAEcICahBbNB2baok3ra4TwfrFxuSjXFBCK_Wwke0Z2K-160Hg",
  "iAEd1DDsJYBwpIP7YN3y10fDnnNHdZqTwWoLZrveQ-f9iByJv2uT8UI95GG7wEp9WJOs5s
  "},
  "recipients":{
    "kid":"MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
    "epk":{
      "PublicKeyECDH":{
        "crv":"Ed25519",
        "Public":"WiIfCOrs2DEiD7wTIZVdqDA0e6hOl_4fjrHPvz2sUXg"},
        "wmk":"oAnxLpiXmAvqXcDgrEwCwKtjJRG1i_5_s8qQ5AxXiIv258jll9E0
      yQ"
    },
    "ywvEL8p7OMpYzd-sv0g3zE8SmBmvi4coxEjMy151g9phuIhmsujRUJP8igoH060W
L_myk3oXR1oxrMRVogAf8w"
  }
}]
```
3.4.1. Key Exchange

The DARE key exchange is based on the JWE key exchange except that encryption modes are intentionally limited and the output of the key exchange is the DARE Master Key rather than the Content Encryption Key.

A DARE Key Exchange MAY contain any number of Recipient entries, each providing a means of decrypting the Master Key using a different private key.

If the Key Exchange mechanism supports message recovery, Direct Key Agreement is used, in all other cases, Key Wrapping is used.

This approach allows envelopes with one intended recipient to be handled in the exact same fashion as envelopes with multiple recipients. While this does require an additional key wrapping operation, that could be avoided if an envelope has exactly one intended recipient, this is offset by the reduction in code complexity.

If the key exchange algorithm does not support message recovery (e.g. Diffie Hellman and Elliptic Curve Diffie-Hellman), the HKDF Extract-and-Expand Key Derivation Function is used to derive a master key using the following info tag:

"dare-master" [64 61 72 65 2d 6d 61 73 74 65 72]  Key derivation info field used when deriving a master key from the output of a key exchange.

The master key length is the maximum of the key size of the encryption algorithm specified by the key exchange header, the key size of the MAC algorithm specified by the key exchange header (if used) and 256.

3.4.2. Key Identifiers

The JWE/JWS specifications define a kid field for use as a key identifier but not how the identifier itself is constructed. All DARE key identifiers are either UDF key fingerprints [draft-hallambaker-mesh-udf] or Mesh/Recrypt Group Key Identifiers.

A UDF fingerprint is formed as the digest of an IANA content type and the digested data. A UDF key fingerprint is formed with the content type application/pkix-keyinfo and the digested data is the ASN.1 DER encoded PKIX certificate keyInfo sequence for the corresponding public key.
A Group Key Identifier has the form `<fingerprint>@<domain>`. Where `<fingerprint>` is a UDF key fingerprint and `<domain>` is the DNS address of a service that provides the encryption service to support decryption by group members.

3.4.3. Salt Derivation

A Master Salt is a sequence of 16 or more octets that is specified in the Salt field of the header.

The Master Salt is used to derive salt values for the envelope payload and associated encoded data sequences as follows.

- **Payload Salt** = Master Salt
- **EDS Salt** = Concatenate (Payload Salt Prefix, Master Salt)

Encoders SHOULD NOT generate salt values that exceed 1024 octets.

The salt value is opaque to the DARE encoding but MAY be used to encode application specific semantics including:

- Frame number to allow reassembly of a data sequence split over a sequence of envelopes which may be delivered out of order.
- Transmit the Master Key in the manner of a Kerberos ticket.
- Identify the Master Key under which the Enhanced Data Sequence was generated.
- Enable access to the plaintext to be eliminated by erasure of the encryption key.

For data erasure to be effective, the salt MUST be constructed so that the difficulty of recovering the key is sufficiently high that it is infeasible. For most purposes, a salt with 128 bits of appropriately random data is sufficient.

3.4.4. Key Derivation

Encryption and/or authentication keys are derived from the Master Key using a Extract-and-Expand Key Derivation Function as follows:

1. The Master Key and salt value are used to extract the PRK (pseudorandom key)
2. The PRK is used to derive the algorithm keys using the application specific information input for that key type.
The application specific information inputs are:

"dare-encrypt" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an encryption or encryption with authentication key.

"dare-iv" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an initialization vector.

"dare-mac" [dare-mac] To generate a Message Authentication Code key.

3.5. Signature

While encryption and integrity enhancements can be applied to any part of a DARE Envelope, signatures are only applied to payload digest values calculated over one or more envelope payloads.

The payload digest value for an envelope is calculated over the binary payload data. That is, after any encryption enhancement has been applied but before the envelope encoding is applied. This allows envelopes to be converted from one encoding to another without affecting signature verification.

Single Payload The signed value is the payload digest of the envelope payload.

Multiple Payload. The signed value is the root of a Merkle Tree in which the payload digest of the envelope is one of the leaves.

Verification of a multiple payload signature naturally requires the additional digest values required to construct the Merkle Tree. These are provided in the Trailer in a format that permits multiple signers to reference the same tree data.

3.6. Algorithms

3.6.1. Field: kwd

The key wrapping and derivation algorithms.

Since the means of public key exchange is determined by the key identifier of the recipient key, it is only necessary to specify the algorithms used for key wrapping and derivation.

The default (and so far only) algorithm is kwd-aes-sha2-256-256.

Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] is used to wrap the Master Exchange Key. AES 256 is used.
HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] is used for key derivation. SHA-2-256 is used for the hash function.

4. DARE Container Architecture

4.1. Container Navigation

Three means of locating frames in a container are supported:

Sequential  Access frames sequentially starting from the start or the end of the container.

Binary search  Access any container frame by frame number in \( O(\log_2(n)) \) time by means of a binary tree constructed while the container is written.

Index  Access and container frame by frame number or by key by means of an index record.

All DARE Containers support sequential access. Only tree and Merkle tree containers support binary search access. An index frame MAY be written appended to any container and provides \( O(1) \) access to any frame listed in the index.

Two modes of compilation are considered:

Monolithic  Frames are added to the container in a single operation, e.g. file archives,

Incremental  Additional frames are written to the container at various intervals after it was originally created, e.g. server logs, message spools.

In the monolithic mode, navigation requirements are best met by writing an index frame to the end of the container when it is complete. It is not necessary to construct a binary search tree unless a Merkle tree integrity check is required.

In the incremental mode, Binary search provides an efficient means of locating frames by frame number but not by key. Writing a complete index to the container every \( m \) write operations provides \( O(m) \) search access but requires \( O(n^2) \) storage.

Use of partial indexes provides a better compromise between speed and efficiency. A partial index is written out every \( m \) frames where \( m \) is a power of two. A complete index is written every time a binary tree apex record is written. This approach provides for \( O(\log_2(n)) \)
search with incremental compilation with approximately double the overhead of the monolithic case.

4.1.1. Tree

Binary search is supported by means of the TreePosition parameter specified in the FrameHeader. This parameter specifies the value of the immediately preceding apex.

Calculation of the immediately preceding apex is most easily described by representing the array index in binary with base of 1 (rather than 0). An array index that is a power of 2 (2, 4, 8, 16, etc.) will be the apex of a complete tree. Every other array index has the value of the sum of a set of powers of 2 and the immediately preceding apex will be the value of the next smallest power of 2 in the sum.

For example, to find the immediately preceding apex for frame 5, we add 1 to get 6. 6 = 4 + 2, so we ignore the 2 and the preceding frame is 4.

The values of Tree Position are shown for the first 8 frames in figure xx below:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-dare.html [6].]]

Merkle Tree Integrity check

An algorithm for efficiently calculating the immediately preceding apex is provided in Appendix C.

4.1.2. Position Index

Contains a table of frame number, position pairs pointing to prior locations in the file.

4.1.3. Metadata Index

Contains a list of IndexMeta entries. Each entry contains a metadata description and a list of frame indexes (not positions) of frames that match the description.
4.2. Integrity Mechanisms

Frame sequences in a DARE container MAY be protected against a frame insertion attack by means of a digest chain, a binary Merkle tree or both.

4.2.1. Digest Chain calculation

A digest chain is simple to implement but can only be verified if the full chain of values is known. Appending a frame to the chain has O(1) complexity but verification has O(n) complexity:

\[ \text{Hash chain integrity check} \]

The value of the chain digest for the first frame (frame 0) is \( H(H(\text{null})+H(\text{Payload}_0)) \), where null is a zero length octet sequence and payloadn is the sequence of payload data bytes for frame n.

The value of the chain digest for frame n is \( H(H(\text{Payload}_{n-1} + H(\text{Payload}_n)) \), where A+B stands for concatenation of the byte sequences A and B.

4.2.2. Binary Merkle tree calculation

The tree index mechanism describe earlier may be used to implement a binary Merkle tree. The value TreeDigest specifies the apex value of the tree for that node.

Appending a frame to the chain has O(log₂ (n)) complexity provided that the container format supports at least the binary tree index. Verifying a chain has O(log₂ (n)) complexity, provided that the set of necessary digest inputs is known.

To calculate the value of the tree digest for a node, we first calculate the values of all the sub trees that have their apex at that node and then calculate the digest of that value and the immediately preceding local apex.

4.2.3. Signature

Payload data MAY be signed using a JWS [RFC7515] as applied in the Envelope.
Signatures are specified by the Signatures parameter in the content header. The data that the signature is calculated over is defined by the typ parameter of the Signature as follows.

Payload  The value of the PayloadDigest parameter
Chain    The value of the ChainDigest parameter
Tree     The value of the TreeDigestFinal parameter

If the typ parameter is absent, the value Payload is implied.

A frame MAY contain multiple signatures created with the same signing key and different typ values.

The use of signatures over chain and tree digest values permit multiple frames to be validated using a single signature verification operation.

5. DARE Schema

A DARE Envelope consists of a Header, an Enhanced Data Sequence (EDS) and an optional trailer. This section describes the JSON data fields used to construct headers, trailers and complete messages.

Wherever possible, fields from JWE, JWS and JWK have been used. In these cases, the fields have the exact same semantics. Note however that the classes in which these fields are presented have different structure and nesting.

5.1. Message Classes

A DARE Message contains a single DAREMessageSequence in either the JSON or Compact serialization as directed by the protocol in which it is applied.

5.1.1. Structure: DareEnvelopeSequence

A DARE Message containing Header, EDS and Trailer in JSON object encoding. Since a DAREMessage is almost invariably presented in JSON sequence or compact encoding, use of the DAREMessage subclass is preferred.

Although a DARE Message is functionally an object, it is serialized as an ordered sequence. This ensures that the message header field will always precede the body in a serialization, this allowing processing of the header information to be performed before the entire body has been received.
Header: DareHeader (Optional)  The message header. May specify the key exchange data, pre-signature or signature data, cloaked headers and/or encrypted data sequences.

Body: Binary (Optional)  The message body

Trailer: DareTrailer (Optional)  The message trailer. If present, this contains the signature.

5.2.  Header and Trailer Classes

A DARE Message sequence MUST contain a (possibly empty) DAREHeader and MAY contain a DareTrailer.

5.2.1.  Structure: DareTrailer

A DARE Message Trailer

Signatures: DareSignature [0..Many]  A list of signatures. A message trailer MUST NOT contain a signatures field if the header contains a signatures field.

SignedData: Binary (Optional)  Contains a DareHeader object

PayloadDigest: Binary (Optional)  If present, contains the digest of the Payload.

ChainDigest: Binary (Optional)  If present, contains the digest of the PayloadDigest values of this frame and the frame immediately preceding.

TreeDigest: Binary (Optional)  If present, contains the Binary Merkle Tree digest value.

5.2.2.  Structure: DareHeader

Inherits: DareTrailer

A DARE Message Header. Since any field that is present in a trailer MAY be placed in a header instead, the message header inherits from the trailer.

EncryptionAlgorithm: String (Optional)  The encryption algorithm as specified in JWE

DigestAlgorithm: String (Optional)  Digest Algorithm. If specified, tells decoder that the digest algorithm is used to construct a signature over the message payload.
Salt: Binary (Optional)  Salt value used to derive cryptographic parameters for the content data.

Malt: Binary (Optional)  Hash of the Salt value used to derive cryptographic parameters for the content data. This field SHOULD NOT be present if the Salt field is present. It is used to allow the salt value to be erased (thus rendering the payload content irrecoverable) without affecting the ability to calculate the payload digest value.

Signed: Binary (Optional)  Contains signed headers.

Cloaked: Binary (Optional)  If present in a header or trailer, specifies an encrypted data block containing additional header fields whose values override those specified in the message and context headers.

When specified in a header, a cloaked field MAY be used to conceal metadata (content type, compression) and/or to specify an additional layer of key exchange. That applies to both the Message body and to headers specified within the cloaked header.

Processing of cloaked data is described in...

ContentType: String (Optional)  The content type field as specified in JWE

EDSS: Binary [0..Many]  If present, the Annotations field contains a sequence of Encrypted Data Segments encrypted under the message Master Key. The interpretation of these fields is application specific.

Signers: DareSigner [0..Many]  A list of ‘presignature’

Recipients: DareRecipient [0..Many]  A list of recipient key exchange information blocks.

UniqueID: String (Optional)  Unique object identifier

Filename: String (Optional)  The original filename under which the data was stored.

Event: String (Optional)  Operation on the header

Labels: String [0..Many]  List of labels that are applied to the payload of the frame.
KeyValues: KeyValue [0..Many]  List of key/value pairs describing the payload of the frame.

5.3.  Cryptographic Data

DARE Message uses the same fields as JWE and JWS but with different structure. In particular, DARE messages MAY have multiple recipients and multiple signers.

5.3.1.  Structure: DareSigner

The signature value

Dig: String (Optional)  Digest algorithm hint. Specifying the digest algorithm to be applied to the message body allows the body to be processed in streaming mode.

Alg: String (Optional)  Key exchange algorithm

KeyIdentifier: String (Optional)  Key identifier of the signature key.

Certificate: X509Certificate (Optional)  PKIX certificate of signer.

Path: X509Certificate (Optional)  PKIX certificates that establish a trust path for the signer.

5.3.2.  Structure: X509Certificate

X5u: String (Optional)  URL identifying an X.509 public key certificate

X5: Binary (Optional)  An X.509 public key certificate

5.3.3.  Structure: DareSignature

Inherits: DareSigner

The signature value

Manifest: Binary (Optional)  The data description that was signed.

SignatureValue: Binary (Optional)  The signature value as an Enhanced Data Sequence under the message Master Key.

WitnessValue: Binary (Optional)  The signature witness value used on an encrypted message to demonstrate that the signature was
authorized by a party with actual knowledge of the encryption key used to encrypt the message.

5.3.4. Structure: DareRecipient

Recipient information

KeyIdentifier: String (Optional)  Key identifier for the encryption key.

The Key identifier MUST be either a UDF fingerprint of a key or a Group Key Identifier

KeyWrapDerivation: String (Optional)  The key wrapping and derivation algorithms.

WrappedMasterKey: Binary (Optional)  The wrapped master key. The master key is encrypted under the result of the key exchange.

RecipientKeyData: String (Optional)  The per-recipient key exchange data.

6. DARE Container Schema

TBS stuff

6.1. Container Headers

TBS stuff

6.1.1. Structure: ContainerEntry

Inherits: ContainerHeader

Inherits: ContainerHeader

Body: Binary (Optional)  The container data.

6.1.2. Structure: ContainerHeaderFirst

Inherits: ContainerHeader

Inherits: ContainerHeader

DataEncoding: String (Optional)  Specifies the data encoding for the header section of for the following frames. This value is ONLY valid in Frame 0 which MUST have a header encoded in JSON.
6.1.3. Structure: ContainerHeader

Inherits: DareHeader

Describes a container header. A container header MAY contain any DARE Message header.

Index: Integer (Optional) The record index within the file. This MUST be unique and satisfy any additional requirements determined by the ContainerType.

ContainerType: String (Optional) Specifies the container type for the following records.

IsMeta: Boolean (Optional) If true, the current frame is a meta frame and does not contain a payload.

Note: Meta frames MAY be present in any container. Applications MUST accept containers that contain meta frames at any position in the file. Applications MUST NOT interpret a meta frame as a data frame with an empty payload.

Default: Boolean (Optional) If set true in a persistent container, specifies that this record contains the default object for the container.

ContentMeta: ContentMeta (Optional) Content meta data.

TreePosition: Integer (Optional) Position of the frame containing the apex of the preceding sub-tree.

IndexPosition: Integer (Optional) Specifies the position in the file at which the last index entry is to be found

ExchangePosition: Integer (Optional) Specifies the position in the file at which the key exchange data is to be found

ContainerIndex: ContainerIndex (Optional) An index of records in the current container up to but not including this one.

First: Integer (Optional) Frame number of the first object instance value.

Previous: Integer (Optional) Frame number of the immediately prior object instance value
6.2. Content Metadata Structure

TBS stuff

6.2.1. Structure: ContentMeta

Information describing the object instance

ContentType: String (Optional)  The content type field as specified in JWE

Paths: String [0..Many]  List of filename paths for the payload of the frame.

UniqueID: String (Optional)  Unique object identifier

Created: DateTime (Optional)  Initial creation date.

Modified: DateTime (Optional)  Date of last modification.

6.3. Index Structures

TBS stuff

6.3.1. Structure: ContainerIndex

A container index

Full: Boolean (Optional)  If true, the index is complete and contains position entries for all the frames in the file. If absent or false, the index is incremental and only contains position entries for records added since the last frame containing a ContainerIndex.

Positions: IndexPosition [0..Many]  List of container position entries

Metas: IndexMeta [0..Many]  List of container position entries

6.3.2. Structure: IndexPosition

Specifies the position in a file at which a specified record index is found

Index: Integer (Optional)  The record index within the file.

Position: Integer (Optional)  The record position within the file relative to the index base.
6.3.3. Structure: KeyValue

Specifies a key/value entry

Key: String (Optional)  The key

Value: String (Optional)  The value corresponding to the key

6.3.4. Structure: IndexMeta

Specifies the list of index entries at which a record with the specified metadata occurs.

Index: Integer [0..Many]  List of record indices within the file where frames matching the specified criteria are found.

ContentType: String (Optional)  Content type parameter

Paths: String [0..Many]  List of filename paths for the current frame.

Labels: String [0..Many]  List of labels that are applied to the current frame.

7. Dare Container Applications

DARE Containers are used to implement two forms of persistence store to support Mesh operations:

Catalogs  A set of related items which MAY be added, modified or deleted at any time.

Spools  A list of related items whose status MAY be changed at any time but which are immutable once added.

Since DARE Containers are an append only log format, entries can only be modified or deleted by adding items to the log to change the status of previous entries. It is always possible to undo any operation on a catalog or spool unless the underlying container is purged or the individual entries modified.

7.1. Catalog

Catalogs contain a set of entries, each of which is distinguished by a unique identifier.

Three operations are supported:
Add  Addition of the entry to the catalog
Update  Modification of the data associated with the entry excluding the identifier
Delete  Removal of the entry from the catalog

The set of valid state transitions is defined by the Finite State machine:

(Add-Update*-Delete)*

Catalogs are used to represent sets of persistent objects associated with a Mesh Service Account. The user’s set of contacts for example. Each contact entry may be modified many times over time but refers to the same subject for its entire lifetime.

SchemaCatalog

7.2.  Spool

Spools contain lists of entries, each of which is distinguished by a unique identifier.

Four operations are supported:
Post  Addition of the entry to the spool
Processed  Marks the entry as having been processed.
Unprocessed  Returns the entry to the unread state.
Delete  Mark the entry as deleted allowing recovery of associated storage in a subsequent purge operation.

The set of valid state transitions is defined by the Finite State machine:

Post-(Processed| Unprocessed| Delete *)

Spools are used to represent time sequence ordered entries such as lists of messages being sent or received, task queues and transaction logs.

SchemaCatalog
7.3. Archive

A DARE Archive is a DARE Container whose entries contain files. This affords the same functionality as a traditional ZIP or tar archive but with the added cryptographic capabilities provided by the DARE format.

8. Future Work

The current specification describes an approach in which containers are written according to a strict append-only policy. Greater flexibility may be achieved by loosening this requirement allowing record(s) at the end of the container to be overwritten.

8.1. Terminal integrity check

A major concern when operating a critical service is the possibility of a hardware or power failure occurring during a write operation causing the file update to be incomplete. While most modern operating systems have effective mechanisms in place to prevent corruption of the file system itself in such circumstances, this does not provide sufficient protection at the application level.

Appending a null record containing a container-specific magic number provides an effective means of detecting this circumstance that can be quickly verified.

If a container specifies a terminal integrity check value in the header of frame zero, the container is considered to be in an incomplete write state if the final frame is not a null record specifying the magic number.

When appending new records to such containers, the old terminal integrity check record is overwritten by the data being added and a new integrity check record appended to the end.

8.2. Terminal index record

A writer can maintain a complete (or partial) index of the container in its final record without additional space overhead by overwriting the prior index on each update.

8.3. Deferred indexing

The task of updating terminal indexes may be deferred to a time when the machine is not busy. This improves responsiveness and may avoid the need to re-index containers receiving a sequence of updates.
This approach may be supported by appending new entries to the end of the container in the usual fashion and maintaining a record of containers to be updated as a separate task.

When updating the index on a container that has been updated in this fashion, the writer must ensure that no data is lost even if the process is interrupted. The use of guard records and other precautions against loss of state is advised.

9. Security Considerations

This section describes security considerations arising from the use of DARE in general applications.

Additional security considerations for use of DARE in Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

9.1. Encryption/Signature nesting

9.2. Side channel

9.3. Salt reuse

10. IANA Considerations

11. Acknowledgements

A list of people who have contributed to the design of the Mesh is presented in [draft-hallambaker-mesh-architecture].

The name Data At Rest Encryption was proposed by Melhi Abdulhayo?lu.

12. Appendix A: DARE Envelope Examples and Test Vectors

13. Test Examples

In the following examples, Alice’s encryption private key parameters are:

```
{  "PrivateKeyECDH":{
      "crv":"Ed25519",
      "Private":"i4OxYpYWjPL1j6GVk0TNwn9F9MKix2e9NJCIRm2Qrgw"
    }}
```

Alice’s signature private key parameters are:
The body of the test message is the UTF8 representation of the following string:

"This is a test long enough to require multiple blocks"

The EDS sequences, are the UTF8 representation of the following strings:

"Subject: Message metadata should be encrypted"
"2018-02-01"

13.1. Plaintext Message

A plaintext message without associated EDS sequences is an empty header followed by the message body:

```
{  
  "DareEnvelope":[]
}
```

13.2. Plaintext Message with EDS

If a plaintext message contains EDS sequences, these are also in plaintext:

```
{  
  "DareEnvelope":{
    "Annotations":["iAEBiC1TdWJq2WN0OiBNZXNzYWdlIG1ldGVsZCBiZSBlbmNyeXB0ZSBBibG9ja3M",
                  "iAECiAoyMDE4LTAyLTAxiAA"],
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdWx0aXBsZSBBibG9ja3M"
  }
}
```

13.3. Encrypted Message

The creator generates a master session key:
For each recipient of the message:

The creator generates an ephemeral key:

```json
{  
  "PrivateKeyECDH":{  
    "crv":"Ed25519",  
    "Private":"WtOjaU3fKpP7BcIchAlaETHChkbXMPrbip003z90ITvQ"}
}
```

The key agreement value is calculated:

```
14 68 2C CE 27 FD 4F A0 8D F4 42 4B 46 FD C8 39
27 F7 E1 AD D0 89 C3 F7 97 DC 4B 84 94 44 3D 3E
```

The key agreement value is used as the input to a HKDF key derivation function with the info parameter master to create the key used to wrap the master key:

```
20 EF BA 11 EA CD 32 34 47 56 84 B2 D4 8D 2A 61
E3 21 3E FD AB 8B 77 1F B1 86 BF 25 C9 0E 22 DA
```

The wrapped master key is:

```
12 A1 79 F3 69 80 86 89 D9 1E 4A 17 49 44 12 4D
6D B4 DF DA 97 6B F3 5B C1 B1 53 C8 54 6B 2A 59
A7 38 1E D7 98 6A 16 6E
```

This information is used to calculate the Recipient information shown in the example below.

To encrypt a message, we first generate a unique salt value:

```
8B 96 1D 69 4C EB 6F B7 B1 30 1B 47 F7 18 C9 39
```

The salt value and master key are used to generate the payload encryption key:

```
F3 F7 BF 79 0D DF 4D FE C5 7A 8E 8E 2A 0F 11 C9
E6 43 F2 B9 09 A8 92 D6 D2 EA 7B 32 99 D8 01 C9
```
Since AES is a block cipher, we also require an initialization vector:

4C C6 DC AB EF E6 DB 24 A8 71 A4 ED C3 75 72 4B

The output sequence is the encrypted bytes:

3B 01 AC 77 C7 23 C2 44 AD 46 45 3C 43 45 DA 83
97 AC FB 14 77 9C 3C 2E 2C 8D 34 AB B3 6A F4 FD
9C A0 46 E0 45 A5 19 E6 2A 43 EC E6 EC 55 BD B8
DC F3 D8 84 81 76 AE D5 3C D1 F2 EA 8A 80 A3 6C

Since the message is not signed, there is no need for a trailer. The completed message is:

```
{"DareEnvelope":{
  "enc":"A256CBC",
  "Salt":"i5YdaUzrb7exMBtH9xjJOQ",
  "recipients":{
    "kid":"MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
    "epk":{
      "PublicKeyECDH":{
        "crv":"Ed25519",
        "Public":"EOMMvoCXaHrUzXqtfqzdQD3TpYvUImG2sAOcsod2o8A"},
      "wmk":"EqF582mAhonZHkoXSUQSTW2039qXa_NbwFTyFrKlmmOB7XmG
oWbg"}
    },
  "OwGsd8cjwkStRkU8Q0Xag5es-xR3nDwuLi00q7Nq9F2coEbgRaUZ5ipD7obsVb
243PPYhIF2rtU80fLqicCjbA"
}}
```

13.4. Signed Message

Signed messages specify the digest algorithm to be used in the header and the signature value in the trailer. Note that the digest algorithm is not optional since it serves as notice that a decoder should digest the payload value to enable signature verification.
13.5. Signed and Encrypted Message

A signed and encrypted message is encrypted and then signed. The signer proves knowledge of the payload plaintext by providing the plaintext witness value.

```json
{ "DareEnvelope":{
    "enc":"A256CBC",
    "dig":"S512",
    "Salt":"r2ZmivI52we7wUT4RFCUKA",
    "recipients":{
        "kid":"MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
        "epk":{
            "PublicKeyECDH":{
                "crv":"Ed25519",
                "Public":"akbyCWQnALpo3Nxc-UctbvNQk3ziDhvCP1-Gj8Nua7Q"},
            "wmk":"RTrtjl_85Dntow0BYuTIkK2yKM-o5wniU0aIB9fP7R7rI31X6wVPkg"
        },
        "hkWnMoUgIjGc8qLAR4n-NwCFfQCYEhCDiRtdYjgE-ua86JYVRbC2IkwCwdx4dxC_A07LA2C3AoWZQgAeRwUVg",
        "signatures":{
            "signature":"XWTe9zxAMNVPWDhSCeFkyWBnqYuw0Pxx4y9vdfGiT3pzz2cCBp1WcTiY14cJKA7z-rvfcFHy3Q8fIK35BDw",
            "witness":"_j_Cptkznea925FHgc41_LX3mgo7zaBCBaJM0-FMOYB4"
        },
        "PayloadDigest":"8HaAO13paSETh5MLLgFx1WLtPT2bLGTWClVwtyq5qdo0J07YjiqkNKZAin70r2Ug-WhZVtXK331NmpATjyLL-9Q"
    }
}}
```
14. Appendix B: DARE Container Examples and Test Vectors

The data payloads in all the following examples are identical, only the authentication and/or encryption is different.

- Frame 1..n consists of 300 bytes being the byte sequence 00, 01, 02, etc. repeating after 256 bytes.

For conciseness, the raw data format is omitted for examples after the first, except where the data payload has been transformed, (i.e. encrypted).

14.1. Simple container

the following example shows a simple container with first frame and a single data frame:

```
f4 5d
f0 59
f0 00
5d f4
f5 01 40
f0 0f
f1 01 2c
40 01 f5
```

Since there is no integrity check, there is no need for trailer entries. The header values are:

Frame 0

```
{  
  "Index": 0,  
  "ContainerType": "List",  
  "ContentMeta": {},  
  "DataEncoding": "JSON"}
```

[Empty trailer]

Frame 1

```
{  
  "Index": 1}
```

[Empty trailer]
14.2. Payload and chain digests

The following example shows a chain container with a first frame and three data frames. The headers of these frames is the same as before but the frames now have trailers specifying the PayloadDigest and ChainDigest values:

Frame 0

```
{
  "Index": 0,
  "ContainerType": "Chain",
  "ContentMeta": {},
  "DataEncoding": "JSON"
}
```
[Empty trailer]

Frame 1

```
{
  "Index": 1
}
```
```
"PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjZD1ZeawQ1BszYho88-ekpNXpZ2iY96zTRI229zaJ5sw",
"ChainDigest": "T7S1FcrgY3AaWd4L-t5W1K-3XYkPTcOdGEGygj1TD6yMYVRVz9tn_KQc6GdA-P4VSRigBygV650Ed2Vv3Ydhw"
```

Frame 2

```
{
  "Index": 2
}
```
```
"PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjZD1ZeawQ1BszYho88-ekpNXpZ2iY96zTRI229zaJ5sw",
"ChainDigest": "T7S1FcrgY3AaWd4L-t5W1K-3XYkPTcOdGEGygj1TD6yMYVRVz9tn_KQc6GdA-P4VSRigBygV650Ed2Vv3Ydhw"
```

Frame 3
14.3.  Merkle Tree

The following example shows a chain container with a first frame and six data frames. The trailers now contain the TreePosition and TreeDigest values:

Frame 0

{  
  "Index": 0,  
  "ContainerType": "Merkle",  
  "ContentMeta": {},  
  "DataEncoding": "JSON"
}

[Empty trailer]

Frame 1

{  
  "Index": 1,  
  "TreePosition": 0
}

{  
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEGKBjzXBRwppu6qbtmA16UjZDlZeawQ1BzYohpU88-ekpNxp22iy96zTRI229zaJ5sw",  
  "TreeDigest": "T7S1Fcrgy3aWD4L-t5W1K-3XYkPToDGEgygj1TD6yMYVRVz9tn_KQc6GdA-P4VSRigBygV65OEd2Vv3Ydhw"
}

Frame 2

{  
  "Index": 2,  
  "TreePosition": 325
}

{  
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEGKBjzXBRwppu6qbtmA16UjZDlZeawQ1BzYohpU88-ekpNxp22iy96zTRI229zaJ5sw",  
  "TreeDigest": "7fHmkE1sPkn6sDyAOLvpIjN5Dg3F3xDAaq-112kh8722kakkFn2QcYcjuYrC71aHXRi18q-1PnfRkmwryG-bhqQ"
}
Frame 3

{
    "Index": 3,
    "TreePosition": 325
}

{
    "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjzD1ZeawQ1BszYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest": "T7S1FcrY3AaWD4L-t5W1K-3XYkPTcOdGEGyjg1TD6yMYVRVz9tn_KQc6Gda-P4VSRigBygV650Ed2Vv3YDhw"}

Frame 4

{
    "Index": 4,
    "TreePosition": 1469
}

{
    "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjzD1ZeawQ1BszYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest": "vJ6ngNATvZcXSMALi5Uqz11GBxBnTNVcC87VL_BhMRChAvKsj8gs0VFgxxLk2myrtaDIwhHoswiTiBMNLWug"}

Frame 5

{
    "Index": 5,
    "TreePosition": 1469
}

{
    "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjzD1ZeawQ1BszYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest": "vJ6ngNATvZcXSMALi5Uqz11GBxBnTNVcC87VL_BhMRChAvKsj8gs0VFgxxLk2myrtaDIwhHoswiTiBMNLWug"}

Frame 6

{
    "Index": 6,
    "TreePosition": 2616
}

{
    "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA16UjzD1ZeawQ1BszYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest": "WgHlz3EHczVPqgtp39Arv7CF1sCbFVsk8wg0j2qL1Efur9S0mdr6SkA-HF0Qx8gg_DAoi3wUrwADDXyyVJCg"}
14.4. Signed container

The following example shows a tree container with a signature in the final record. The signing key parameters are:

```json
{
  "PrivateKeyECDH": {
    "crv": "Ed25519",
    "Private": "CpM8Q0Tddslp9vUpB91RUjQ8tIex7FYku7AqFBX8o"
  }
}
```

The container headers and trailers are:

Frame 0

```json
{
  "Index": 0,
  "ContainerType": "Merkle",
  "ContentMeta": {},
  "DataEncoding": "JSON"
}
[Empty trailer]
```

Frame 1

```json
{
  "Index": 1,
  "TreePosition": 0
}
```

```json
{
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA1UujZD1ZeawQ18sYHou88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "T7S1FcrqY3AaWD4L-t5W1K-3XYkPTcOdGEGyjgI1TD6yMYVRRnK_Qc6GdA-P4VSRigBygV650Ed2Vv3YDhww"
}
```

Frame 2

```json
{
  "Index": 2,
  "TreePosition": 325
}
```

```json
{
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmA1UujZD1ZeawQ18sYHou88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "7fHmkEIsPkn6sDYA0LoVpIjn5Dg3PxDAAaq-112kh8v722kokkFv3QyJuVC71aHNXI18q-lPnfRkmwryGbhpQ"
}
```
14.5. Encrypted container

The following example shows a container in which all the frame payloads are encrypted under the same master secret established in a key agreement specified in the first frame.

Frame 0

{
    "enc": "A256CBC",
    "Salt": "KrAJLsza8qtDOBlbNKmDcg",
    "recipients": [{
        "kid": "MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
        "epk": {
            "PublicKeyECDH": {
                "crv": "Ed25519",
                "Public": "6unfe0fePLCG1psc6nf_tgck8QyoDWxdSWxoc45K5U4"},
            "wmk": "-0bqijEjs0XTjF6W6Fl1p1PQNKb-t8WeiFmD_BTM_HP02iFVmsMg6JpQ"},
        "Index": 0,
        "ContainerType": "List",
        "ContentMeta": {},
        "DataEncoding": "JSON"
    }
}

[Empty trailer]

Frame 1

{
    "enc": "A256CBC",
    "Salt": "vw7kqW-snWPSKG7mSynUBw",
    "Index": 1
}

[Empty trailer]

Frame 2

{
    "enc": "A256CBC",
    "Salt": "AlyQ7XoO8LAv5Sf1ZfA",
    "Index": 2
}

[Empty trailer]

Here are the container bytes. Note that the content is now encrypted and has expanded by 25 bytes. These are the salt (16 bytes), the AES padding (4 bytes) and the JSON-B framing (5 bytes).
The following example shows a container in which all the frame payloads are encrypted under separate key agreements specified in the payload frames.

Frame 0

```
{
    "Index": 0,
    "ContainerType": "List",
    "ContentMeta": {},
    "DataEncoding": "JSON"
}
```

[Empty trailer]

Frame 1

```
{
    "enc": "A256CBC",
    "Salt": "FxVy8pBxwnYttJ07FFvMyA",
    "recipients": [{
        "kid": "MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
        "epk": {
            "PublicKeyECDH": {
                "crv": "Ed25519",
                "Public": "yenFnuLKZ71I1G9BV1ZsS6ViyEJmnFD7Rim1xO7U4eo"},
            "wmk": "nZz0-quCr17o0fTDJEgxnLGusr-d0234U9KQtz4bzwjw__jwKQA"},
        "Index": 1}
```

[Empty trailer]

Frame 2
{  
    "enc": "A256CBC",
    "Salt": "vnbl-fL_CW1v-MWjQ8hNFQ",
    "recipients": [{
        "kid": "MDE7-76UB-4XXJ-YQRI-TIMB-QW5I-2LZG",
        "epk": {
            "PublicKeyECDH": {
                "crv": "Ed25519",
                "Public": "HcCByC4bjfFFDmtrWb5ntRpgijp1CIBT8qLX282Uanw"},
            "wmk": "vVrjPhAHDeku7F6t-QOFKgJaQd0T-QII-aCkWdIPQbslyQPAm6nqg"},
        "Index": 2}
    ]
}

15. Appendix C: Previous Frame Function

    public long PreviousFrame (long Frame) {
        long x2 = Frame + 1;
        long d = 1;

        while (x2 > 0) {
            if ((x2 & 1) == 1) {
                return x2 == 1 ? (d / 2) - 1 : Frame - d;
            }
            d = d * 2;
            x2 = x2 / 2;
        }
        return 0;
    }

16. Appendix D: Outstanding Issues

    The following issues need to be addressed.
<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X25519</td>
<td>The examples currently use Edwards Curve25519 for encryption. This should be Curve X25519</td>
</tr>
<tr>
<td>Indexing</td>
<td>No examples are given of indexing a container</td>
</tr>
<tr>
<td>Archive</td>
<td>Should include a file archive example</td>
</tr>
<tr>
<td>File Path</td>
<td>Mention the file path security issue in the security considerations</td>
</tr>
<tr>
<td>Security</td>
<td>Write Security considerations</td>
</tr>
<tr>
<td>Considerations</td>
<td>AES-GCM Switch to using AES GCM in the examples</td>
</tr>
<tr>
<td>Witness</td>
<td>Complete handling of witness values.</td>
</tr>
<tr>
<td>Schema</td>
<td>Complete the schema documentation</td>
</tr>
<tr>
<td>Container Redo</td>
<td>Rework the container/header objects so that these are separate classes and Header is an entry in the Container header.</td>
</tr>
</tbody>
</table>

Table 1

17. References

17.1. Normative References

[draft-hallambaker-jsonbcd]

[draft-hallambaker-mesh-architecture]

[draft-hallambaker-mesh-security]

[draft-hallambaker-mesh-udf]

[IANAJOSE]
"[Reference Not Found!]".


17.2. Informative References

[BLOCKCHAIN] Chain.com, "Blockchain Specification".
17.3. URIs


Author’s Address

Phillip Hallam-Baker

Email: phill@hallambaker.com
Abstract

The Mathematical Mesh ‘The Mesh’ is an end-to-end secure infrastructure that facilitates the exchange of configuration and credential data between multiple user devices. The core protocols of the Mesh are described with examples of common use cases and reference data.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [1].

Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
Table of Contents

1. Introduction ............................................. 4
2. Definitions ............................................. 5
   2.1. Requirements Language ............................... 5
   2.2. Defined Terms ....................................... 5
   2.3. Related Specifications .............................. 5
   2.4. Implementation Status .............................. 5
3. Mesh Assertions ............................................ 5
   3.1. Encoding ............................................ 6
   3.2. Mesh Profiles ....................................... 6
   3.3. Mesh Connections .................................... 7
   3.4. Mesh Private Declarations .......................... 7
4. Architecture .............................................. 7
   4.1. Device Management .................................. 8
      4.1.1. Master Profile ................................. 9
      4.1.2. Mesh Devices .................................. 11
   4.2. Mesh Accounts ....................................... 17
      4.2.1. Creating a ProfileAccount ...................... 19
      4.2.2. Connecting a Device to an Account ............. 19
      4.2.3. Binding and Account to a Service .............. 20
   4.3. Mesh Services ....................................... 20
      4.3.1. Creating a ProfileService ..................... 21
      4.3.2. Creating a ProfileHost ......................... 22
      4.3.3. Creating a ConnectionHost ..................... 22
   4.4. Mesh Messaging ...................................... 22
      4.4.1. Traffic Analysis ............................... 23
5. Mesh Catalogs ............................................. 24
   5.1. Application .......................................... 25
      5.1.1. Mesh Account .................................... 25
      5.1.2. SSH ............................................. 25
      5.1.3. Mail ........................................... 26
   5.2. Device ............................................... 26
   5.3. Contact .............................................. 26
   5.4. Credential ........................................... 27
   5.5. Bookmark ............................................ 27
   5.6. Task ................................................ 27
   5.7. Network .............................................. 28
6. Mesh Messages ............................................. 28
   6.1. Completion .......................................... 28
   6.2. Connection .......................................... 28
   6.3. Contact .............................................. 29
   6.4. Confirmation ........................................ 30
7. Schema .................................................. 31
    7.1. Shared Classes ........................................ 31
        7.1.1. Classes describing keys ......................... 31
        7.1.2. Structure: PublicKey .......................... 31
        7.1.3. Structure: KeyComposite ......................... 31
        7.1.4. Structure: KeyOverlay .......................... 31
        7.1.5. Structure: EscrowedKeySet ....................... 31
        7.1.6. Structure: DeviceRecryptionKey ................. 32
    7.2. Assertion classes ..................................... 32
        7.2.1. Structure: Assertion ............................ 32
        7.2.2. Structure: Condition ............................ 32
        7.2.3. Profile Classes .................................. 32
        7.2.4. Structure: Profile ............................... 32
        7.2.5. Structure: ProfilePersonal ....................... 33
        7.2.6. Structure: ProfileDevice ......................... 33
        7.2.7. Structure: ProfileService ....................... 33
        7.2.8. Structure: ProfileAccount ....................... 34
        7.2.9. Structure: ProfileGroup ........................ 34
        7.2.10. Structure: ProfileHost ......................... 34
        7.2.11. Connection Classes ............................. 34
        7.2.12. Structure: Connection ......................... 34
        7.2.13. Structure: Permission ......................... 35
        7.2.15. Structure: ConnectionAccount ................... 35
        7.2.16. Structure: ConnectionService ................... 36
        7.2.17. Structure: ConnectionHost ...................... 36
        7.2.18. Structure: ConnectionApplication ............... 36
        7.2.19. Activation Classes ............................ 36
        7.2.20. Structure: Activation ......................... 36
        7.2.21. Structure: ActivationDevice .................... 36
        7.2.22. Structure: ActivationAccount .................. 37
    7.3. Cataloged items ...................................... 37
        7.3.1. Data Structures ................................. 37
        7.3.2. Structure: Contact .............................. 37
        7.3.3. Structure: Role .................................. 38
        7.3.4. Structure: Address .............................. 39
        7.3.5. Structure: Location ............................. 39
        7.3.6. Structure: Reference ............................ 39
        7.3.7. Structure: Task .................................. 40
    7.4. Catalog Entries ........................................ 41
        7.4.1. Structure: CatalogedEntry ....................... 41
        7.4.2. Structure: CatalogedDevice ...................... 41
        7.4.3. Structure: CatalogedCredential .................. 41
        7.4.4. Structure: CatalogedNetwork ..................... 42
        7.4.5. Structure: CatalogedContact .................... 42
        7.4.6. Structure: CatalogedContactRecryption .......... 42
        7.4.7. Structure: CatalogedBookmark ................... 43
        7.4.8. Structure: CatalogedTask ....................... 43
1. Introduction

This document describes the data structures of the Mathematical Mesh with illustrative examples. For an overview of the Mesh objectives and architecture, consult the accompanying Architecture Guide [draft-hallambaker-mesh-architecture]. For information on the implementation of the Mesh Service protocol, consult the accompanying Protocol Reference [draft-hallambaker-mesh-protocol].

This document has two main sections. The first section presents examples of the Mesh assertions, catalog entry and messages in use. The second section contains the schema reference. All the material in both sections is generated from the Mesh reference implementation [draft-hallambaker-mesh-developer].

Although some of the services described in this document could be used to replace existing Internet protocols including FTP and SMTP, the principal value of any communication protocol lies in the size of the audience it allows them to communicate with. Thus, while the Mesh Messaging service is designed to support efficient and reliable transfer of messages ranging in size from a few bytes to multiple terabytes, the near-term applications of these services will be to...
applications that are not adequately supported by existing protocols if at all.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. 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].

2.2. Defined Terms

The terms of art used in this document are described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture].

2.3. Related Specifications

The architecture of the Mathematical Mesh is described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture]. The Mesh documentation set and related specifications are described in this document.

2.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

3. Mesh Assertions

Mesh Assertions are signed DARE Envelopes that contain one or more claims. Mesh Assertions provide the basis for trust in the Mathematical Mesh.

Mesh Assertions are divided into two classes. Mesh Profiles are self-signed assertions. Assertions that are not self-signed are called declarations. The only type of declaration currently defined is a Connection Declaration describing the connection of one profile to another. Currently, five profile and four connection types are defined:
Profiles And Connections

3.1. Encoding

The payload of a Mesh Assertion is a JSON encoded object that is a subclass of the Assertion class which defines the following fields:

Identifier  An identifier for the assertion.

Updated  The date and time at which the assertion was issued or last updated.

NotaryToken  An assertion may optionally contain one or more notary tokens issued by a Mesh Notary service. These establish a proof that the assertion was signed after the date the notary token was created.

Conditions  A list of conditions that MAY be used to verify the status of the assertion if the relying party requires.

The implementation of the NotaryToken and Conditions mechanisms is to be specified in [draft-hallambaker-mesh-notary] at a future date.

Note that the implementation of Conditions differs significantly from that of SAML. Relying parties are required to process condition clauses in a SAML assertion to determine validity. Mesh Relying parties MAY verify the conditions clauses or rely on the trustworthiness of the provider.

The reason for weakening the processing of conditions clauses in the Mesh is that it is only ever possible to validate a conditions clause of any type relative to a ground truth. In SAML applications, the relying party almost invariably has access to an independent source of ground truth. A Mesh device connected to a Mesh Service does not. Thus the types of verification that can be achieved in practice are limited to verifying the consistency of current and previous statements from the Mesh Service.

3.2. Mesh Profiles

Mesh Profiles perform a similar role to X.509v3 certificates but with important differences:
Profiles describe credentials, they do not make identity statements.

Profiles do not expire, there is therefore no need to support renewal processing.

Profiles may be modified over time, the current and past status of a profile being recorded in an append only log.

Profiles provide the axioms of trust for the Mesh PKI. Unlike in the PKIX model in which all trust flows from axioms of trust held by a small number of Certificate Authorities, every part in the Mesh contributes their own axiom of trust.

It should be noted however that the role of Certificate Authorities is redefined rather than eliminated. Rather than making assertions whose subject is represented by identities which are inherently mutable and subjective, Certificate Authorities can now make assertions about immutable cryptographic keys.

Every Profile MUST contain a SignatureKey field and MUST be signed by the key specified in that field.

A Profile is valid if and only if:

- There is a SignatureKey field.
- The profile is signed under the key specified in the SignatureKey field.

A profile has the status current if and only if:

- The Profile is valid
- Every Conditions clause in the profile is understood by the relying party and evaluates to true.

3.3. Mesh Connections

3.4. Mesh Private Declarations

4. Architecture

The Mesh architecture has four principal components:

Mesh Device Management Binds a collection of devices that the owner of the Mesh uses together to function as a single personal Mesh.
Mesh Account  Contains all the information (contacts, calendar entries, inbound and outbound messages, etc.) related to a particular persona used by the owner.

Mesh Service  Provides a service identifier (e.g. alice@example.com) through which devices and other Mesh users may interact with a Mesh Account.

Mesh Messaging

Allows short messages (less than 32KB) to be exchanged between Mesh devices connected to an account and between Mesh Accounts.

Device management and Accounts components are defined by a data schema alone. The Service and Messaging components are defined by a data schema and a service protocol.

The separation of accounts and services as separate components is a key distinction between the Mesh and earlier Internet applications. A Mesh account belongs to the owner of the Mesh and not the account service provider which the user may change at any time of their choosing. A Mesh account may be connected to multiple service providers to provide backup capability or to none.

For example, Alice’s personal Mesh has one Master Profile, multiple device profiles (two of which are shown here) and two Account Profiles. Her Personal account is connected to two Mesh services while her Business account is not currently connected to any service:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [3].]]

Alice’s Personal Mesh

In normal circumstances, a user will create a personal Mesh and add their first device, account and service at once. These are shown here as separate operations to illustrate the separation of the Mesh components.

4.1. Device Management

Device Management provides the foundation for all Mesh functions allowing a collection of devices belonging to a user to function as a single personal Mesh.
The device management layer of a personal Mesh consists of exactly one Master Profile and a catalog containing the entries describing the connected devices.

4.1.1. Master Profile

A Mesh master profile provides the axiom of trust for a mesh user. It contains a Master Signature Key and one or more Administration Signature Keys. The unique identifier of the master profile is the UDF of the Master Signature Key.

A Master Profile MAY contain one or more MasterEscrowKeys. These MAY be used to escrow private keys used for encryption. They SHOULD NOT be used to escrow authentication keys and MUST NOT be used to escrow signature keys.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [4].]]

Master Profile and Associated Device and Account Connection Assertions.

A user should not need to replace their master profile unless they intend to establish a separate identity. To minimize the risk of disclosure, the Master Signature Key is only ever used to sign updates to the master profile itself. This allows the user to secure their Master Signature Key by either keeping it on hardware token or device dedicated to that purpose or by using the escrow mechanism and paper recovery keys as described in this document.

Alice creates a ProfileMaster with one administration key and one master escrow key:
{
  "ProfilePersonal": {
    "KeyOfflineSignature": {
      "UDF": "MAOZ-3MVE-G5EN-64BI-I3RM-ODFJ-H5W4",
      "PublicParameters": {
        "PublicKeyECDH": {
          "crv": "Ed448",
          "Public": "0Vxq105QJ4y0IJu_pE2iBm6y7NTruQQkZhUIJGYu-6mBhNJ
CTKjnSikk6SK6aLR1zpWv-UWCzMA")
      }
    }
  }
}

4.1.1.1. Creating a ProfileMaster

Creating a ProfileMaster comprises the steps of:

1. Creating a Master Signature key.
2. Creating an Online Signing Key
3. Signing the ProfileMaster using the Master Signature Key
4. Persisting the ProfileMaster on the administration device to the CatalogHost.
5. (Optional) Connecting at least one Administration Device and granting it the ActivationAdministration activation.

4.1.1.2. Updating a ProfileMaster

Updating a ProfileMaster comprises the steps of:

1. Making the necessary changes.
2. Signing the ProfileMaster using the Master Signature Key
3. Persisting the ProfileMaster on the administration device to the CatalogHost.

4.1.1.3. The Device Catalog

Each personal Mesh has a Device Catalog CatalogDevice associated with it. The Device Catalog is used to manage the connection of devices to the Personal Mesh and has a CatalogEntryDevice for each device currently connected to the catalog.

Each Administration Device MUST have access to an up to date copy of the Device Catalog in order to manage the devices connected to the Mesh. The Mesh Service protocol MAY be used to synchronize the Device Catalog between administration devices in the case that there is more than one administration device.

The CatalogEntryDevice contains fields for the device profile, device private and device connection.

4.1.2. Mesh Devices

The principle of radical distrust requires us to consider the possibility that a device might be compromised during manufacture. Once consequence of this possibility is that when an administration device connects a new device to a user’s personal Mesh, we cannot put our full trust in either the device being connected or the administration device connecting it.

This concern is resolved by (at minimum) combining keying material generated from both sources to create the keys to be used in the context of the user’s personal Mesh with the process being fully verified by both parties.

Additional keying material sources could be added if protection against the possibility of compromise at both devices was required but this is not supported by the current specifications.

A device profile provides the axiom of trust and the key contributions of the device:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [5].]]

Mapping of Device Profile and Device Private to Device Connection Keys.
Unless exceptional circumstances require, a device should not require more than one Device profile even if the device supports use by multiple users under different accounts. But a device MAY have multiple profiles if this approach is more convenient for implementation.

Alice’s Device Profile specifies keys for encryption, signature and exchange:

```
{
  "ProfileDevice":{
    "KeyOfflineSignature":{
      "UDF":"MABG-6ZLL-I2FM-KUKX-TM7E-OVAJ-M5QP",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"9Fot0c6ztURHkCY0pLeW5-9cjGv72EztCSx0xwh0IstC-I
          kb7fdzwv_vcEA6f6eyJF4HD-mGSeA"},
        "KeyEncryption":{
          "UDF":"MB3L-TNWX-HNZI-FFHN-IJRZ-HSOF-NKZL",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"Ed448",
              "Public":"CUG3q5c8uFglVWyGeMnxybLprQAKHqvq-V9UbRQ5OS3LB9J
j1H7k8i9hQpiyhNvCZ5mcb5Rti_6A"},
            "KeyAuthentication":{
              "UDF":"MD6N-VQ4H-6EQQ-DT3S-IUMN-C5OJ-CDXR",
              "PublicParameters":{
                "PublicKeyECDH":{
                  "crv":"Ed448",
                  "Public":"IodXkMxsyg6kUsKFSlS8mDvxe5KSuSU0r-1XoY2TvPOCM
F34NA1OViRPaJPhVFJLdgaLspYA"}}}}
    }},
  Since the Device Profile keys are ultimately under the control of the device and/or software provider, these are considered insufficiently trustworthy and the administration device creates key contributions to be added to the device keys to establish the key set to be used in the context of the user’s personal Mesh:
{
   "ActivationDevice":{
      "KeysSignature":{
         "UDF":"MDAA-75W5-HOD4-LSSU-JNYG-WHML-34DU",
         "BaseUDF":"MABG-62LL-I2FM-KUKX-TM7E-OVAJ-M5QP",
         "Overlay":{
            "PrivateKeyECDH":{
               "crv":"Ed448",
               "Private":"hchEs5Is89D_B89_04Y24dau12P7hVvG1IY5JJVRevREWGLZsuRaDnsJTDaW7yE6kW0QM5hky5k"},
            "KeyEncryption":{
               "UDF":"MDBS-VUEM-T4LN-V6SG-OLAZ-DBXD-BEKX",
               "BaseUDF":"MB3L-TNWX-HNZI-FFHN-IJRZ-HSOF-NKZL",
               "Overlay":{
                  "PrivateKeyECDH":{
                     "crv":"Ed448",
                     "Private":"idvk3TtbV_tE1P0jEFyWMAnrwDQdow6TnvusuNpGUIrKteQaRHL3tLa53Nca7Fyl2_8NpA85Ps"},
                  "KeyAuthentication":{
                     "UDF":"MC20-YPA4-NZXY-4XYX-LV7S-TV.CV-ODF",
                     "BaseUDF":"MD6N-VQ4H-6E5O-DT3S-IUMN-C50J-CDXR",
                     "Overlay":{
                        "PrivateKeyECDH":{
                           "crv":"Ed448",
                           "Private":"Sq5L14DMmsC0WC4gBzC3RGm4Q50r11DBautaEDSNQFLQP RFD4CR2h1Iq057HKuNk5IDViidysA"}}}}
         }}
   }
}

The resulting key set is specified in the device connection:
Internet-Draft

Mesh Schema Reference

August 2019

{
"ConnectionDevice":{
"KeySignature":{
"UDF":"MDAA-75W5-HOD4-LSSU-JNYG-WHML-34DU",
"PublicParameters":{
"PublicKeyECDH":{
"crv":"Ed448",
"Public":"CHU3vyEJRMDuHnzZev5GqO-0iK-lOBxowBEUsOUaWN-KUe0
RZHa5d-a3xBEQtG-0o0rq5LwKvJwA"}}},
"KeyEncryption":{
"UDF":"MDBS-VUEM-T4LN-V6SG-OLAZ-DBXD-BEKR",
"PublicParameters":{
"PublicKeyECDH":{
"crv":"Ed448",
"Public":"VW2pEXUUgHkvV7e4DhsX8qjLr4Z5-qGkyBsfCRUdmtyGqdq
RGYXA8qeGvP4GcrMlhlmtJ_tEQBcA"}}},
"KeyAuthentication":{
"UDF":"MC2O-YPA4-NZYJ-4XYX-LV7S-TVCV-ODIH",
"PublicParameters":{
"PublicKeyECDH":{
"crv":"Ed448",
"Public":"Yz1ljn0ykFMNussK778qoua202kBugeeCRj35OL7VqBxYOw
8nG4BcmHTH3cIKdJohQB4H3T1ikOA"}}}}}
All the above are combined to form the CatalogedDevice entry:
{
"CatalogedDevice":{
"UDF":"MDAA-75W5-HOD4-LSSU-JNYG-WHML-34DU",
"DeviceUDF":"MABG-6ZLL-I2FM-KUKX-TM7E-OVAJ-M5QP",
"EnvelopedProfileDevice":[{
"dig":"S512",
"cty":"application/mmm"},
"ewogICJQcm9maWxlRGV2aWNlIjogewogICAgIktleU9mZmxpbmVTaWduYXR1
cmUiOiB7CiAgICAgICJVREYiOiAiTUFCRy02WkxMLUkyRk0tS1VLWC1UTTdFLU9WQ
UotTTVRUCIsCiAgICAgICJQdWJsaWNQYXJhbWV0ZXJzIjogewogICAgICAgICJQdW
JsaWNLZXlFQ0RIIjogewogICAgICAgICAgImNydiI6ICJFZDQ0OCIsCiAgICAgICA
gICAiUHVibGljIjogIjlGb3QwYzZ6dFVSSGtDWTBwTGV3NS05Y2pHdlA3MkV6dENT
eDB4d2gwSXN0Qy1Ja2I3ZmQKICB6d3ZfdmNFQTZmNmV5SkY0SEQtbUdTZUEifX19L
AogICAgIktleUVuY3J5cHRpb24iOiB7CiAgICAgICJVREYiOiAiTUIzTC1UTldYLU
hOWkktRkZITi1JSlJaLUhTT0YtTktaTCIsCiAgICAgICJQdWJsaWNQYXJhbWV0ZXJ
zIjogewogICAgICAgICJQdWJsaWNLZXlFQ0RIIjogewogICAgICAgICAgImNydiI6
ICJFZDQ0OCIsCiAgICAgICAgICAiUHVibGljIjogIkNVRzNxNWM4dUZnbFZXeUdlT
W54eWJMcHJRQUtIdnFiLVY5VWJSUTVPUzNMQjlKajFIN2sKICA4aTlIUXBpeWhOdk
NaNW1jYjVSdGlfNkEifX19LAogICAgIktleUF1dGhlbnRpY2F0aW9uIjogewogICA
gICAiVURGIjogIk1ENk4tVlE0SC02RVFPLURUM1MtSVVNTi1DNU9KLUNEWFIiLAog
ICAgICAiUHVibGljUGFyYW1ldGVycyI6IHsKICAgICAgICAiUHVibGljS2V5RUNES
CI6IHsKICAgICAgICAgICJjcnYiOiAiRWQ0NDgiLAogICAgICAgICAgIlB1YmxpYy

Hallam-Baker

Expires February 14, 2020

[Page 14]


"signatures": [{
  "signature": "zTXCLitDmFOuBcza1RqH_DgOHpy7PwpVx3ndtT7OA
N1xf7Rqc7nflcmfKLqeyu1NT04YFpW9C4ac5Fv0M1_jzOPdBcicpuAu+-nH6ArIcD
uxiat7kbfSjm4vdMjV_gzIdny6fYx6WH6SA7VvSpAxA"
},

"PayloadDigest": "qFInJ114BiGkbKq-TvoQ3b0ztUyVu_lqm1wUed3nj
mYeguyZov8tI9NdvuFmGfyKxOW2n-USz-4PHvHAyDg"
},

"EnvelopedConnectionDevice": [{
  "dig": "S512",
  "ewogICJDb25uZWN0aW9uW9uRGV2aWNIjogewogICAgIktleVNpZ25hdHVyZSI6
iHsKICAgIcAgI1VERiI61IHYEFBLtCvZUtSSE9ENC1MU13VLjUopWO0CtV00NTC0zn
ERlVikHICAgIcAgIb1xmpyX1BhcmFtZXRlcnMiOlBI7C1AgIcAgIcAgIcAgIb1xmpyX0
UeUVDREgiO1BiC1AgIcAgIcAiY3J2IjogIkVlKNDQ4iwKICAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIC
wQajWjaMiOlAiQ00VHm5Gx526Vw2NUdXyT0waUslbE9CeG93QkVc099
YVsDolUXVZTSWkhkNhQ0IgQTYN40KvrdCetMG8zcE1Hd3lpQ3JS9jfx0ScAgI
CAI2555X5Wcn1wDglvbi16HsKICAgIcAgI1VERiI6I1HYEFBLtCvZUtSSE9ENC1Mj
lWzQWVRU0tVDRMT1w1W1NH1LQ9MQVotREYR1CRU7SiwiKICAgIcAgIb1xmpyX1BhcmFtZXRlcnMiOlIB
7C1AgIcAgIcAgIb1xmpyX0UeUVDREgiO1BiC1AgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIC
nDJw4aMiOlAiQ00VHm5Gx526Vw2NUdXyT0waUslbE9CeG93QkVc099
YVsDolUXVZTSWkhkNhQ0IgQTYN40KvrdCetMG8zcE1Hd3lpQ3JS9jfx0ScAgI
CAI2555X5Wcn1wDglvbi16HsKICAgIcAgI1VERiI6I1HYEFBLtCvZUtSSE9ENC1Mj
lWzQWVRU0tVDRMT1w1W1NH1LQ9MQVotREYR1CRU7SiwiKICAgIcAgIb1xmpyX1BhcmFtZXRlcnMiOlIB
7C1AgIcAgIcAgIb1xmpyX0UeUVDREgiO1BiC1AgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIcAgIC
]

"signatures": [{
  "signature": "LakNIBxHRHIWfUdZuKKt_UzGpwJPxLXSDHsS43pzFO
G8Q4FJzaza0lMs3pIw56Mjwcq8kvgZz7QarwsewDVCXg5sF-IO43z1k8iEeJBA8csWz
ika412bjiVST--nOjKYtaeRLBycm656e7I0djwt_sR9A"
},

"PayloadDigest": "sUSwjnOWhRs5n7CUZCr72SHFGJYyNgxtg8fICUeEq
u1Mhjw3z11-UAj1lyRQQnmsZ_YEF_eBzSwRK077LwXzw"
},

"EnvelopedActivationDevice": {
  "enc": "none",
  "Salt": "cJKbnuGuELszHVogMXTj3Q",
  "cty": "application/mmm",
  "recipients": [{
    "kid": "MB3L-TNWX-HNZI-FFH-IRJZ-HSOF-NKZL",
    "epk": {
      "PublicKeyECDH": {
        "crv": "Ed448",
        "pub": "04Ed448PublicKey"
      }
    }
  }]
}
The derivation of the Connection encryption and signature keys from the Profile and Private contributions in this example is shown in [draft-hallambaker-mesh-cryptography].

4.1.2.1. Creating a ProfileDevice

Creating a ProfileDevice comprises the steps of:

1. Creating the necessary key

2. Signing the ProfileDevice using the Master Signature Key
3. Once created, a ProfileDevice is never changed. In the unlikely event that any modification is required, a completely new ProfileDevice MUST be created.

4.1.2.2. Connection to a Personal Mesh

Devices are only connected to a personal Mesh by administration device. This comprises the steps of:

1. Generating the PrivateDevice keys.

2. Creating the ConnectionDevice data from the public components of the ProfileDevice and PrivateDevice keys and signing it using the administration key.

3. Creating the Activations for the device and signing them using the administration key.

4. Creating the CatalogEntryDevice for the device and adding it to the CatalogDevice of the Personal Mesh.

5. If the Personal Mesh has accounts that are connected to a Mesh Service, synchronizing the CatalogEntryDevice to those services.

4.2. Mesh Accounts

Mesh Accounts contains all the stateful information (contacts, calendar entries, inbound and outbound messages, etc.) related to a particular persona used by the owner.

A Mesh Profile MAY be connected to multiple accounts at the same time allowing the user to maintain separate personas for separate purposes.

Unlike traditional Internet application accounts, Mesh accounts are created by and belong to the user, not the Mesh Service provider. A user MAY change their Mesh Service provider at any time and disconnect the profile from all Mesh Services (e.g. to archive the account).

Alice’s personal account is connected to two Mesh services:

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [6].]]

Account Profile Connected to Devices and Services.
The account profile specifies the online and offline signature keys used to maintain the profile and the encryption key to be used by the account.

```
{  
  "ProfileAccount":{  
    "MeshProfileUDF":"MAOZ-3MVE-G5EN-64BI-I3RM-ODFJ-H5W4",  
    "KeyEncryption":{  
      "UDF":"MAFZ-VUMP-7PU3-4TP7-7MRZ-BKBQ-VMNN",  
      "PublicParameters":{  
        "PublicKeyECDH":{  
          "crv":"Ed448",
          "Public":"l1cZUNjHGJWkPwMS9MH6GYNfDSZG041Gk5Xp7xgH--SGLQE4E_s-0b0PI8m0fjiz1MUEYD_I4")}
      
      "ProfileAccount":{  
        "AccountUDF":"MAFZ-VUMP-7PU3-4TP7-7MRZ-BKBQ-VMNN",
        "EnvelopedAssertionAccountConnection":{  
          "PayloadDigest":"oKJFUfV-FrLaHCbGvk_GTyUVxRPGsGWVNdpF1cH2PtIN0AFk2d2QMFZ1N2o4ckwcwZnMudLWUp1CIaqj1JEUTROMETUVFkUUVMGmjDaHxxWFNBI19fx19",  
          "signatures":{  
            "signature":"jm8gei5o49kwavBEBFGB9E8PxdmIpo7Z7F0Xq7l2jGF8edCScFbDQQk0Okx7v95Aw?Ssz2z7p_KLysAajjg51y8W02jwhQhLa4C2fsthW8A2yvyHeNMFysD47aZj2BEXpS8e_OKKcy-Y5syj2q_gAA"
          }
    }
  }
```

Each device using the account requires an activation record:

```
{  
  "ActivationAccount":{  
    "AccountUDF":"MAFZ-VUMP-7PU3-4TP7-7MRZ-BKBQ-VMNN",
    "EnvelopedAssertionAccountConnection":{  
      "PayloadDigest":"oKJFUfV-FrLaHCbGvk_GTyUVxRPGsGWVNdpF1cH2PtIN0AFk2d2QMFZ1N2o4ckwcwZnMudLWUp1CIaqj1JEUTROMETUVFkUUVMGmjDaHxxWFNBI19fx19",  
      "signatures":{  
        "signature":"jm8gei5o49kwavBEBFGB9E8PxdmIpo7Z7F0Xq7l2jGF8edCScFbDQQk0Okx7v95Aw?Ssz2z7p_KLysAajjg51y8W02jwhQhLa4C2fsthW8A2yvyHeNMFysD47aZj2BEXpS8e_OKKcy-Y5syj2q_gAA"
      }
    }  
  }  
}
"KeyEncryption":{
  "Public":{
    "PublicKeyECDH":{
      "crv":"Ed448",
      "Public":"1lcZUNjHGJWkPWmS9MHeGYNfDSZGUO41Gk5Xpq7XgH--SGLQE4E_s-0b0PI8m0fjtz1MUEYD1_IA"},
    "Part":{
      "PrivateKeyECDH":{
        "crv":"Ed448",
        "Private":"0wzAsNH-Ft8i3GcK585BxuECOxwQCS1V_bPJxhJHURA7weSPHflh-ThorKAF6Em0jtST4mG0FBi"},
      "KeyAuthentication":{
        "UDF":"MCNW-QURN-W75E-THE6-P2RA-Q7PN-IMTY",
        "BaseUDF":"MD6N-VQ4H-6EQO-DT3S-IUMN-C5OJ-CDXR",
        "Overlay":{
          "PrivateKeyECDH":{
            "crv":"Ed448",
            "Private":"8E50eTfRcL9ghQUyOnSvI3wagGsYGQaKXj09KxNRmzq5TDaKD-E-HroSc8wJapds-bHDSocShgM"},
          "KeySignature":{
            "UDF":"MDYL-6G2A-Y3EG-U3FG-HAY3-SNLN-NYP2",
            "BaseUDF":"MABG-6ZLL-I2FM-KUKX-TM7E-OVAJ-M5QP",
            "Overlay":{
              "PrivateKeyECDH":{
                "crv":"Ed448",
                "Private":"Ye28IFBvRxvKKc5r5c-cAPYjGhLFw_dCt5ScuFzAx7bAltWviYpsk_bd5ndt3cKReFUT30EpFOg"}}}}}}},
  "KeySignature":{
    "UDF":"MDYL-6G2A-Y3EG-U3FG-HAY3-SNLN-NYP2",
    "BaseUDF":"MABG-6ZLL-I2FM-KUKX-TM7E-OVAJ-M5QP",
    "Overlay":{
      "PrivateKeyECDH":{
        "crv":"Ed448",
        "Private":"Ye28IFBvRxvKKc5r5c-cAPYjGhLFw_dCt5ScuFzAx7bAltWviYpsk_bd5ndt3cKReFUT30EpFOg"}}}}}}

4.2.1. Creating a ProfileAccount

Creating a ProfileAccount comprises the steps of:

1. [TBS]
2. .
3. Signing the ProfileMaster using the Master Signature Key

4.2.2. Connecting a Device to an Account

Adding a device to an account comprises the steps of:

1. Creating a PrivateAccount instance for the device.
2. Creating a ConnectionAccountDevice for the device using the public keys from the PrivateAccount instance and the ProfileDevice.
3. Creating an ActivationAccount for the device containing the PrivateAccount and ConnectionAccountDevice instances.

4. Adding the ActivationAccount to the CatalogEntryDevice of the device.

5. If the Personal Mesh has accounts that are connected to a Mesh Service, synchronizing the CatalogEntryDevice to those services.

4.2.3. Binding and Account to a Service

Binding a ProfileAccount to a Mesh Service the steps of:

1. [TBS]

2. [TBS]

3. Signing the ProfileMaster using the Master Signature Key

4.3. Mesh Services

A service profile provides the axiom of trust and cryptographic keys for a Mesh Service. A Mesh Service Host SHOULD return a copy of its ProfileHost and the parent ProfileService in response to a Hello transaction request.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html [7].]]

Service Profile and Delegated Host Assertion.

The credentials provided by the ProfileService and ProfileHost are distinct from those provided by the WebPKI that typically services TLS requests. WebPKI credentials provide service introduction and authentication while a Mesh ProfileHost only provides authentication.

Unless exceptional circumstances require, a service should not need to revise its Service Profile unless it is intended to change its identity. Service Profiles MAY be countersigned by Trusted Third Parties to establish accountability.

The service profile
The host also has a profile

```json
{
  "ProfileHost":{
    "KeyOfflineSignature":{
      "UDF":"MDKQ-VTTD-HQKB-YH6O-H57C-4KDH-6KWI",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"ouDlKyIUksS_CWUhgoWaPPF93Kaa9Zg53dQCC44gauG0BB9S12eHlz2KoX9-PctQH31567TcmMDCA"},
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"4BfeW0WQB503EOrjW4LvAdkLz71BtJs1s3wzw0acuH5EdupWWT8vE3MObkfmd_4tP27B6VanteA"}},
    "KeyAuthentication":{
      "UDF":"MC3D-ID62-L5VZ-INLC-WRLX-7FTT-F7ST",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"4BfeW0WQB503EOrjW4LvAdkLz71BtJs1s3wzw0acuH5EdupWWT8vE3MObkfmd_4tP27B6VanteA"}}
  }
}
```

And there should be a connection of the host to the service but this isn’t implemented yet:

```
$$$$ Empty $$$$
```

### 4.3.1. Creating a ProfileService

[TBS]

Creating a ProfileService comprises the steps of:

1. [TBS]
2. .
3. [TBS]
4.
5. Signing the ProfileMaster using the Master Signature Key

4.3.2. Creating a ProfileHost

Creating a ProfileHost comprises the steps of:

1. [TBS]
2. .
3. [TBS]
4. 
5. Signing the ConnectionHost using the Master Signature Key of the ProfileService.

4.3.3. Creating a ConnectionHost

Creating a ConnectionHost comprises the steps of:

1. [TBS]
2. .
3. Signing the ConnectionHost using the Master Signature Key of the ProfileService.

4.4. Mesh Messaging

Mesh Messaging is an end-to-end secure messaging system used to exchange short (32KB) messages between Mesh devices and services. In cases where exchange of longer messages is required, Mesh Messaging MAY be used to provide a control plane to advise the intended message recipient(s) of the type of data being offered and the means of retrieval (e.g an EARL).

A four-corner messaging model is enforced. Mesh Services only accept outbound messages from devices connected to accounts that it services. Inbound messages are only accepted from other Mesh Services. This model enables access control at both the outbound and inbound services.
Performing Access Control on Outbound Messages

The outbound Mesh Service checks to see that the message request does not violate its acceptable use policy. Accounts that make a large number of message requests that result in complaints SHOULD be subject to consequences ranging from restriction of the number and type of messages sent to suspending or terminating messaging privileges.

Performing Access Control on Inbound Messages

4.4.1. Traffic Analysis

The Mesh Messaging protocol as currently specified provides only limited protection against traffic analysis attacks. The use of TLS to encrypt communication between Mesh Services limits the effectiveness of naïve traffic analysis mechanisms but does not prevent timing attacks unless dummy traffic is introduced to obfuscate traffic flows.

The limitation of the message size is in part intended to facilitate use of mechanisms capable of providing high levels of traffic analysis such as mixmaster and onion routing but the current Mesh
Service Protocol does not provide support for such approaches and there are no immediate plans to do so.

5. Mesh Catalogs

Catalogs track sets of persistent objects associated with a Mesh Service Account. The Mesh Service has no access to the entries in any Mesh catalog except for the Device and Contacts catalog which are used in device authentication and authorization of inbound messages.

Each Mesh Catalog managed by a Mesh Account has a name of the form:

<<prefix>_<name>

Where <<prefix> is the IANA assigned service name. The assigned service name for the Mathematical Mesh is mmm. Thus, all catalogs specified by the Mesh schema have names prefixed with the sequence mmm_.

The following catalogs are currently specified within the Mathematical Mesh.

Application: mmm_CatalogApplication  Contains configuration information for applications including mail (SMTP, IMAP, OpenPGP, S/MIME, etc) and SSH and for the MeshAccount application itself.

Device: mmm_CatalogDevice  Contains descriptions of devices connected to the account and the permissions assigned to them

Contact: mmm_CatalogContact  Contains logical and physical contact information for people and organizations.

Credential: mmm_CatalogCredential  Contains credentials used to access network resources.

Bookmark: mmm_CatalogBookmark  Contains Web bookmarks and other citations allowing them to be shared between devices connected to the profile.

Task: mmm_CatalogTask  Contains tasks assigned to the user including calendar entries and to do lists.

Network: mmm_CatalogNetwork  Contains network settings such as WiFi access points, IPSEC and TLS VPN configurations, etc.

In many cases, the Mesh Catalog offers capabilities that represent a superset of the capabilities of an existing application. For example, the task catalog supports the appointment tracking functions
of a traditional calendar application and the task tracking function of the traditional ‘to do list’ application. Combining these functions allows tasks to be triggered by other events other than the passage of time such as completion of other tasks, geographical presence, etc.

In such cases, the Mesh Catalog entries are designed to provide a superset of the data representation capabilities of the legacy formats and (where available) recent extensions. Where a catalog entry is derived from input presented in a legacy format, the original data representation MAY be attached verbatim to facilitate interoperability.

5.1. Application

The application catalog mmm_CatalogApplication contains CatalogEntryApplication entries which describe the use of specific applications under the Mesh Service Account. Multiple application accounts for a single application MAY be connected to a single Mesh Service Account. Each account being specified in a separate entry.

The CatalogEntryApplication entries only contain configuration information for the application as it applies to the account as a whole. If the application requires separate configuration for individual devices, this is specified in separate activation records specified in the corresponding ConnectionDevice.

5.1.1. Mesh Account

Mesh Accounts are described by CatalogEntryAccount entries. The corresponding activation records for the connected devices contain the contributions used to derive the private keys for use of the account.

The CatalogEntryAccount entry is described in the section describing Mesh accounts above.

5.1.2. SSH

SSH configuration profiles are described by CatalogEntryApplicationSSH entries. The corresponding activation records for the connected devices contain the contributions used to derive the private keys.

A user may have separate SSH configurations for separate purposes within a single Mesh Account. This allows a system administrator servicing multiple clients to maintain separate SSH profiles for each
of her customers allowing credentials to be easily (and verifiably) revoked at contract termination.

The SSH profile contains the information that is stored in the known_hosts and authorized_keys files of SSH clients and servers.

5.1.3. Mail

Mail configuration profiles are described by one or more CatalogEntryApplicationMail entries, one for each email account connected to the Mesh profile. The corresponding activation records for the connected devices contain information used to provide the device with the necessary decryption information.

Entries specify the email account address(es), the inbound and outbound server configuration and the cryptographic keys to be used for S/MIME and OpenPGP encryption.

5.2. Device

The device catalog mmm_CatalogDevice contains CatalogEntryDevice entries which describe the devices connected to the account and the permissions assigned to them.

The management of the device catalog is described in the section describing Mesh Device Management.

5.3. Contact

The contacts catalog contains CatalogEntryContact entries which describe

The fields of the contact catalog provide a superset of the capabilities of vCard [RFC2426].

The Contact catalog is typically used by the MeshService as a source of authorization information to perform access control on inbound and outbound message requests. For this reason, Mesh Service SHOULD be granted read access to the contacts catalog by providing a decryption entry for the service.
5.4. Credential

The credential catalog contains CatalogEntryCredential entries which describe credentials used to access network resources.

Only username/password credentials are stored in the credential catalog. If public key credentials are to be used, these SHOULD be managed as an application profile allowing separate credentials to be created for each device.

```
{ 
  "CatalogedCredential":{
    "Service":"ftp.example.com",
    "Username":"alice1",
    "Password":"newpassword"}
}
```

5.5. Bookmark

The bookmark catalog contains CatalogEntryBookmark entries which describe Web bookmarks and other citations allowing them to be shared between devices connected to the profile.

The fields currently supported by the Bookmarks catalog are currently limited to the fields required for tracking Web bookmarks. Specification of additional fields to track full academic citations is a work in progress.

```
{ 
  "CatalogedBookmark":{
    "Uri":"http://example.net/Bananas",
    "Title":"\"Banana",
    "Path":"Folder1/2"}}
```

5.6. Task

The Task catalog contains CatalogEntryTask entries which describe tasks assigned to the user including calendar entries and to do lists.

The fields of the task catalog currently reflect those offered by the iCalendar specification [RFC5545]. Specification of additional fields to allow task triggering on geographic location and/or completion of other tasks is a work in progress.

```
{ 
  "CatalogedTask":{
    "Key":"CalID1"}}
```
5.7. Network

The network catalog contains CatalogEntryNetwork entries which describe network settings, IPSEC and TLS VPN configurations, etc.

$$$$ Empty $$ $$

6. Mesh Messages

All communications between Mesh accounts take the form of a Mesh Message carried in a Dare Envelope. Mesh Messages are stored in two spools associated with the account, the SpoolOutbound and the SpoolInbound containing the messages sent and received respectively.

This document only describes the representation of the messages within the message spool. The Mesh Service protocol by which the messages are exchanged between devices and services and between services is described in [draft-hallambaker-mesh-protocol].

6.1. Completion

Completion messages are dummy messages that are added to a Mesh Spool to change the status of messages previously posted. Any message that is in the inbound spool and has not been erased or redacted MAY be marked as read, unread or deleted. Any message in the outbound spool MAY be marked as sent, received or deleted.

Services MAY erase or redact messages in accordance with local site policy. Since messages are not removed from the spool on being marked deleted, they may be undeleted by marking them as read or unread. Marking a message deleted MAY make it more likely that the Service will purge the message however.

Having processed a message, a completion message is added to the spool so that other devices can see that it has been read:

[NYI]

6.2. Connection

Connection requests are sent by a device requesting connection to a Mesh Service Account.

The MessageConnectionRequest is originally sent by the device requesting connection to the Mesh Service associated with the account.
If the connection request is accepted by the Mesh Service, it creates a MessageConnectionResponse containing the ServerNonce and Witness values used in the authentication of the response together with a verbatim copy of the original request. The MessageConnectionResponse is then returned to the device that made the original request and placed on the SpoolInbound of the account to which the request was directed.

Further details of this mechanism are described in [draft-hallambaker-mesh-protocol].

The connection process begins with the assignment of a time-limited PIN value. This is described in a Message sent by the administration device to allow other admin devices to accept the request made.

[NYI]

The initial request is sent to the service

[NYI]

The service returns an acknowledgement giving the Witness value. Note that this is not a ‘reply’ since it comes from the service, not the user.

[NYI]

[Note, this mechanism should be revised to ensure that there is perfect forward secrecy. The device should provide a nonce key as a mixin]

6.3. Contact

A contact request presents a proposed contact entry and requests that it be added to the Contacts catalog of the specified Mesh Service Account. A contact request is usually sent by the party requesting that their contact be added but this is not necessarily the case.

The MessageContact contains a DARE Envelope containing the Contact information of the requester. If the request is accepted, this information will be added to the contact catalog of the relevant account. If the Reply field has the value ‘true’, this indicates that the sender is asking for the recipient to return their own credentials in reply.

Since the sender requires the user’s contact information before the request can be made, the MessageContact message MAY be encrypted under either the user’s account encryption key (if known) or theMesh
Service encryption key (which may be obtained from the service on request.

Bob asks Alice to send her contact details and sends his.

[NYI]

Alice responds with her details:

[NYI]

[Note that this exchange could be performed automatically on Alice’s behalf by the service if she delegates this action to it.]

The current protocol assumes that all contact management will be performed end-to-end through the Mesh Services themselves. If the number of Mesh users were to become very large, additional infrastructure to facilitate contact management will be required. These topics are discussed at a high level in [draft-hallambaker-mesh-trust].

In situations where a user is well known and has a very large number of contacts, they are likely to make use of a tiered approach to contact management in which they keep separate accounts for their ‘public’ and ‘restricted’ personas and delegate management of their public account to a subordinate or to their Mesh Service provider.

6.4. Confirmation

Confirmation messages are used to provide an improved form of second factor authentication capability.

Two confirmation messages are specified, a request and response.

A confirmation request is initiated by sending a MessageConfirmationRequest to the Mesh Service hosting the recipient Mesh Service Account. The request specifies the question that is to be put to the user.

To respond to a confirmation request, a user generates a MessageConfirmationResponse. This MUST be signed by a device authorized to respond to confirmation requests by a Device Connection Assertion with the Confirmation privilege.

The confirmation request

[NYI]
The confirmation response

[NYI]

7. Schema

7.1. Shared Classes

The following classes are used as common elements in Mesh profile specifications.

7.1.1. Classes describing keys

7.1.2. Structure: PublicKey

The PublicKey class is used to describe public key pairs and trust assertions associated with a public key.

UDF: String (Optional)  UDF fingerprint of the public key parameters/
X509Certificate: Binary (Optional)  List of X.509 Certificates
X509CSR: Binary (Optional)  X.509 Certificate Signing Request.

7.1.3. Structure: KeyComposite

Service: String (Optional)  Service holding the additional contribution

7.1.4. Structure: KeyOverlay

UDF: String (Optional)  Fingerprint of the resulting composite key
(to allow verification)

BaseUDF: String (Optional)  Fingerprint specifying the base key

7.1.5. Structure: EscrowedKeySet

A set of escrowed keys.

[No fields]
7.1.6. Structure: DeviceRecryptionKey

UDF: String (Optional)  The fingerprint of the encryption key
RecryptionKey: PublicKey (Optional)  The recryption key
EnvelopedRecryptionKeyDevice: DareEnvelope (Optional)  The decryption key encrypted under the user’s device key.

7.2. Assertion classes

Classes that are derived from an assertion.

7.2.1. Structure: Assertion

Parent class from which all assertion classes are derived

Names: String [0..Many]  Fingerprints of index terms for profile retrieval. The use of the fingerprint of the name rather than the name itself is a precaution against enumeration attacks and other forms of abuse.

Updated: DateTime (Optional)  The time instant the profile was last modified.

NotaryToken: String (Optional)  A Uniform Notary Token providing evidence that a signature was performed after the notary token was created.

7.2.2. Structure: Condition

Parent class from which all condition classes are derived.

[No fields]

7.2.3. Profile Classes

Profiles are self signed assertions.

7.2.4. Structure: Profile

Inherits: Assertion

Parent class from which all profile classes are derived

KeyOfflineSignature: PublicKey (Optional)  The permanent signature key used to sign the profile itself. The UDF of the key is used as the permanent object identifier of the profile. Thus, by
definition, the KeySignature value of a Profile does not change under any circumstance. The only case in which a
KeysOnlineSignature: PublicKey [0..Many] A Personal profile contains at least one OSK which is used to sign device administration application profiles.

7.2.5. Structure: ProfilePersonal

Inherits: Profile

Describes the long term parameters associated with a personal profile.

KeysMasterEscrow: PublicKey [0..Many] A Personal Profile MAY contain one or more PMEK keys to enable escrow of private keys used for stored data.

KeyEncryption: PublicKey (Optional) Key used to pass encrypted data to the device such as a DeviceUseEntry

7.2.6. Structure: ProfileDevice

Inherits: Profile

Describes a mesh device.

Description: String (Optional) Description of the device

KeyEncryption: PublicKey (Optional) Key used to pass encrypted data to the device such as a DeviceUseEntry

KeyAuthentication: PublicKey (Optional) Key used to authenticate requests made by the device.

7.2.7. Structure: ProfileService

Inherits: Profile

Profile of a Mesh Service

KeyAuthentication: PublicKey (Optional) Key used to authenticate service connections.
7.2.8. Structure: ProfileAccount

Inherits: Profile

Account assertion. This is signed by the service hosting the account.

ServiceIDs: String [0..Many] Service address(es).

MeshProfileUDF: String (Optional) Master profile of the account being registered.

KeyEncryption: PublicKey (Optional) Key used to encrypt data under this profile

7.2.9. Structure: ProfileGroup

Inherits: Profile

Describes a group. Note that while a group is created by one person who becomes its first administrator, control of the group may pass to other administrators over time.

[No fields]

7.2.10. Structure: ProfileHost

Inherits: Profile

Inherits: Profile

KeyAuthentication: PublicKey (Optional) Key used to authenticate service connections.

7.2.11. Connection Classes

7.2.12. Structure: Connection

Inherits: Assertion

Inherits: Assertion

SubjectUDF: String (Optional) UDF of the connection target.

AuthorityUDF: String (Optional) UDF of the connection source.
7.2.13. Structure: Permission

    Name: String (Optional)
    Name: String (Optional)
    Role: String (Optional)
    Role: String (Optional)

    Capabilities: DareEnvelope (Optional)  Keys or key contributions enabling the operation to be performed

7.2.14. Structure: ConnectionDevice

    Inherits: Connection
    Inherits: Connection

    Permissions: Permission [0..Many]  List of the permissions that the device has been granted.

    KeySignature: PublicKey (Optional)  The signature key for use of the device under the profile

    KeyEncryption: PublicKey (Optional)  The encryption key for use of the device under the profile

    KeyAuthentication: PublicKey (Optional)  The authentication key for use of the device under the profile

7.2.15. Structure: ConnectionAccount

    Inherits: Connection
    Inherits: Connection

    Permissions: Permission [0..Many]  List of the permissions that the device has been granted.

    KeySignature: PublicKey (Optional)  The signature key for use of the device under the profile

    KeyEncryption: PublicKey (Optional)  The encryption key for use of the device under the profile

    KeyAuthentication: PublicKey (Optional)  The authentication key for use of the device under the profile
7.2.16. Structure: ConnectionService

Inherits: Connection
[No fields]

7.2.17. Structure: ConnectionHost

Inherits: Connection
[No fields]

7.2.18. Structure: ConnectionApplication

Inherits: Connection
[No fields]

7.2.19. Activation Classes

7.2.20. Structure: Activation

Inherits: Assertion
Contains the private activation information for a Mesh application running on a specific device
[No fields]

7.2.21. Structure: ActivationDevice

Inherits: Assertion
Inherits: Assertion

EnvelopedAssertionDeviceConnection: DareEnvelope (Optional) The signed AssertionDeviceConnection.

KeySignature: KeyOverlay (Optional) The key overlay used to generate the account signature key from the device signature key

KeyEncryption: KeyOverlay (Optional) The key overlay used to generate the account encryption key from the device encryption key

KeyAuthentication: KeyOverlay (Optional) The key overlay used to generate the account authentication key from the device authentication key
7.2.22. Structure: ActivationAccount

Inherits: Activation
Inherits: Activation

AccountUDF: String (Optional)  The UDF of the account

EnvelopedAssertionAccountConnection: DareEnvelope (Optional)  The account connection assertion

KeyEncryption: KeyComposite (Optional)  The key contribution for the decryption key for the device. NB this is NOT an overlay on the device signature key, it is an overlay on the corresponding recryption key.

KeyAuthentication: KeyOverlay (Optional)  The key overlay used to generate the account authentication key from the device authentication key

KeySignature: KeyOverlay (Optional)  The key overlay used to generate the account signature key from the device signature key

7.3. Cataloged items

7.3.1. Data Structures

Classes describing data used in cataloged data.

7.3.2. Structure: Contact

Inherits: Assertion
Inherits: Assertion

Identifier: String (Optional)
Identifier: String (Optional)

FullName: String (Optional)
FullName: String (Optional)

Title: String (Optional)
Title: String (Optional)

First: String (Optional)
First: String (Optional)
Middle: String (Optional)
Middle: String (Optional)
Last: String (Optional)
Last: String (Optional)
Suffix: String (Optional)
Suffix: String (Optional)
Labels: String [0..Many]
Labels: String [0..Many]
AssertionAccounts: ProfileAccount [0..Many]
AssertionAccounts: ProfileAccount [0..Many]
Addresses: Address [0..Many]
Addresses: Address [0..Many]
Locations: Location [0..Many]
Locations: Location [0..Many]
Roles: Role [0..Many]

7.3.3. Structure: Role

CompanyName: String (Optional)
CompanyName: String (Optional)
Addresses: Address [0..Many]
Addresses: Address [0..Many]
Locations: Location [0..Many]
Locations: Location [0..Many]
7.3.4. Structure: Address

URI: String (Optional)

URI: String (Optional)

Labels: String [0..Many]

7.3.5. Structure: Location

Appartment: String (Optional)

Appartment: String (Optional)

Street: String (Optional)

Street: String (Optional)

District: String (Optional)

District: String (Optional)

Locality: String (Optional)

Locality: String (Optional)

County: String (Optional)

County: String (Optional)

Postcode: String (Optional)

Postcode: String (Optional)

Country: String (Optional)

7.3.6. Structure: Reference

MessageID: String (Optional) The received message to which this is a response

ResponseID: String (Optional) Message that was generated in response to the original (optional).

Relationship: String (Optional) The relationship type. This can be Read, Unread, Accept, Reject.
7.3.7. Structure: Task

Key: String (Optional)  Unique key.
Start: DateTime (Optional)
Start: DateTime (Optional)
Finish: DateTime (Optional)
Finish: DateTime (Optional)
StartTravel: String (Optional)
StartTravel: String (Optional)
FinishTravel: String (Optional)
FinishTravel: String (Optional)
TimeZone: String (Optional)
TimeZone: String (Optional)
Title: String (Optional)
Title: String (Optional)
Description: String (Optional)
Description: String (Optional)
Location: String (Optional)
Location: String (Optional)
Trigger: String [0..Many]
Trigger: String [0..Many]
Conference: String [0..Many]
Conference: String [0..Many]
Repeat: String (Optional)
Repeat: String (Optional)
Busy: Boolean (Optional)

7.4. Catalog Entries

7.4.1. Structure: CatalogedEntry

Base class for cataloged Mesh data.

[No fields]

7.4.2. Structure: CatalogedDevice

Inherits: CatalogedEntry

Public device entry, indexed under the device ID

AccountUDFs: String [0..Many] The accounts to which this device is bound.

UDF: String (Optional) UDF of the signature key of the device in the Mesh

DeviceUDF: String (Optional) UDF of the signature key of the device

EnvelopedProfileDevice: DareEnvelope (Optional) The device profile

EnvelopedConnectionDevice: DareEnvelope (Optional) The public assertion demonstrating connection of the Device to the Mesh

EnvelopedActivationDevice: DareEnvelope (Optional) The device profile

7.4.3. Structure: CatalogedCredential

Inherits: CatalogedEntry

Inherits: CatalogedEntry

Protocol: String (Optional)

Protocol: String (Optional)

Service: String (Optional)

Service: String (Optional)

Username: String (Optional)
Username: String (Optional)
Password: String (Optional)

7.4.4. Structure: CatalogedNetwork

Inherits: CatalogedEntry
Inherits: CatalogedEntry
Protocol: String (Optional)
Protocol: String (Optional)
Service: String (Optional)
Service: String (Optional)
Username: String (Optional)
Username: String (Optional)
Password: String (Optional)

7.4.5. Structure: CatalogedContact

Inherits: CatalogedEntry
Inherits: CatalogedEntry
Self: Boolean (Optional)  If true, this catalog entry is for the user who created the catalog. To be valid, such an entry MUST be signed by an administration key for the Mesh profile containing the account to which the catalog belongs.
Key: String (Optional)  Unique key.
Permissions: Permission [0..Many]  List of the permissions that the contact has been granted.
EnvelopedContact: DareEnvelope (Optional)  The (signed) contact data.

7.4.6. Structure: CatalogedContactRecryption

Inherits: CatalogedContact
[No fields]
7.4.7. Structure: CatalogedBookmark

- Inherits: CatalogedEntry
- Uri: String (Optional)
- Title: String (Optional)
- Path: String (Optional)

7.4.8. Structure: CatalogedTask

- Inherits: CatalogedEntry
- EnvelopedTask: DareEnvelope (Optional)
- Key: String (Optional) Unique key.

7.4.9. Structure: CatalogedApplication

- Inherits: CatalogedEntry
- Key: String (Optional)

7.4.10. Structure: CatalogedApplicationAccount

- Wrapper for a signed AccountAssertion
- EnvelopedProfileAccount: DareEnvelope (Optional) The account assertion
7.4.11. Structure: CatalogedMember
    UDF: String (Optional)
    UDF: String (Optional)
    Inherits: CatalogedEntry

7.4.12. Structure: CatalogedGroup
    Inherits: CatalogedApplication
    [No fields]

7.4.13. Structure: CatalogedApplicationSSH
    Inherits: CatalogedApplication
    [No fields]

7.4.14. Structure: CatalogedApplicationMail
    Inherits: CatalogedApplication
    [No fields]

7.4.15. Structure: CatalogedApplicationNetwork
    Inherits: CatalogedApplication
    [No fields]

7.5. Messages

7.5.1. Structure: Message
    MessageID: String (Optional)
    MessageID: String (Optional)
    Sender: String (Optional)
    Sender: String (Optional)
    Recipient: String (Optional)
    Recipient: String (Optional)
7.5.2. Structure: MessageComplete

Inherits: Message

[No fields]

7.5.3. Structure: MessagePIN

Account: String (Optional)
Account: String (Optional)
Inherits: Message
Inherits: Message
Expires: DateTime (Optional)
Expires: DateTime (Optional)
PIN: String (Optional)

7.5.4. Structure: RequestConnection

Connection request message. This message contains the information

Inherits: Message
Inherits: Message
ServiceID: String (Optional)
ServiceID: String (Optional)
EnvelopedProfileDevice: DareEnvelope (Optional) Device profile of
the device making the request.
ClientNonce: Binary (Optional)
ClientNonce: Binary (Optional)
PinUDF: String (Optional) Fingerprint of the PIN value used to
authenticate the request.
7.5.5. Structure: AcknowledgeConnection

Connection request message generated by a service on receipt of a valid MessageConnectionRequestClient

Inherits: Message
Inherits: Message

EnvelopedRequestConnection: DareEnvelope (Optional)  The client connection request.
ServerNonce: Binary (Optional)
ServerNonce: Binary (Optional)
Witness: String (Optional)

7.5.6. Structure: RequestContact

Inherits: Message
Inherits: Message

Reply: Boolean (Optional)
Reply: Boolean (Optional)
Self: DareEnvelope (Optional)  The contact data.

7.5.7. Structure: RequestConfirmation

Inherits: Message
Inherits: Message

Text: String (Optional)

7.5.8. Structure: ResponseConfirmation

Inherits: Message
Inherits: Message

ResponseID: String (Optional)
ResponseID: String (Optional)
Accept: Boolean (Optional)

7.5.9. Structure: RequestTask

Inherits: Message

[No fields]

8. Security Considerations

The security considerations for use and implementation of Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

9. IANA Considerations

All the IANA considerations for the Mesh documents are specified in this document.

10. Acknowledgements

A list of people who have contributed to the design of the Mesh is presented in [draft-hallambaker-mesh-architecture].

11. References

11.1. Normative References

[draft-hallambaker-mesh-architecture]

[draft-hallambaker-mesh-cryptography]

[draft-hallambaker-mesh-notary]
"[Reference Not Found!]".

[draft-hallambaker-mesh-protocol]
11.2. Informative References

[draft-hallambaker-mesh-developer]

[draft-hallambaker-mesh-trust]


11.3. URIs


Author’s Address

Phillip Hallam-Baker

Email: phill@hallambaker.com
Abstract

This document describes the naming and addressing schemes used in the Mathematical Mesh. The means of generating Uniform Data Fingerprint (UDF) values and their presentation as text sequences and as URIs are described.

A UDF consists of a binary sequence, the initial eight bits of which specify a type identifier code. Type identifier codes have been selected so as to provide a useful mnemonic indicating their purpose when presented in Base32 encoding.

Two categories of UDF are described. Data UDFs provide a compact presentation of a fixed length binary data value in a format that is convenient for data entry. A Data UDF may represent a cryptographic key, a nonce value or a share of a secret. Fingerprint UDFs provide a compact presentation of a Message Digest or Message Authentication Code value.

A Strong Internet Name (SIN) consists of a DNS name which contains at least one label that is a UDF fingerprint of a policy document controlling interpretation of the name. SINs allow a direct trust model to be applied to achieve end-to-end security in existing Internet applications without the need for trusted third parties.

UDFs may be presented as URIs to form either names or locators for use with the UDF location service. An Encrypted Authenticated Resource Locator (EARL) is a UDF locator URI presenting a service from which an encrypted resource may be obtained and a symmetric key that may be used to decrypt the content. EARLs may be presented on paper correspondence as a QR code to securely provide a machine-readable version of the same content. This may be applied to automate processes such as invoicing or to provide accessibility services for the partially sighted.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-udf.html [1].
Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 4
   1.1. UDF Types ................................................. 4
       1.1.1. Cryptographic Keys and Nonces ..................... 5
       1.1.2. Fingerprint type UDFS .............................. 6
   1.2. UDF URIs ................................................ 6
       1.2.1. Name Form .................................... 7
       1.2.2. Locator Form ................................ 7
   1.3. Secure Internet Names .................................. 9
2. Definitions .................................................... 9
   2.1. Requirements Language .................................... 10
   2.2. Defined Terms .......................................... 10
   2.3. Related Specifications ................................. 11
   2.4. Implementation Status .................................. 11
3. Architecture .................................................. 11
3.1. Base32 Presentation ........................................... 11
  3.1.1. Precision Improvement ................................. 12
3.2. Type Identifier ............................................ 12
3.3. Content Type Identifier ................................. 13
3.4. Truncation ............................................... 14
  3.4.1. Compression ................................. 14
3.5. Presentation ........................................... 15
3.6. Alternative Presentations ................................. 15
  3.6.1. Word Lists ........................................... 15
  3.6.2. Image List ........................................... 16
4. Fixed Length UDFs ............................................. 16
  4.1. Nonce Type ........................................... 16
  4.2. Encryption/Authentication Type .......................... 16
  4.3. Shamir Shared Secret .................................... 17
    4.3.1. Secret Generation .................................. 17
7. Strong Internet Names ........................................ 32
9. Security Considerations ..................................... 33
  8.1. Confidentiality ........................................... 33
  8.2. Availability ........................................... 33
  8.3. Integrity ............................................. 33
1. Introduction

A Uniform Data Fingerprint (UDF) is a generalized format for presenting and interpreting short binary sequences representing cryptographic keys or fingerprints of data of any specified type. The UDF format provides a superset of the OpenPGP [RFC4880] fingerprint encoding capability with greater encoding density and readability.

This document describes the syntax and encoding of UDFs, the means of constructing and comparing them and their use in other Internet addressing schemes.

1.1. UDF Types

Two categories of UDF are described. Data UDFs provide a compact presentation of a fixed length binary data value in a format that is convenient for data entry. A Data UDF may represent a cryptographic key or nonce value or a part share of a key generated using a secret sharing mechanism. Fingerprint UDFs provide a compact presentation of a Message Digest or Message Authentication Code value.
Both categories of UDF are encoded as a UDF binary sequence, the first octet of which is a Type Identifier and the remaining octets specify the binary value according to the type identifier and data referenced.

UDFs are typically presented to the user as a Base32 encoded sequence in groups of five characters separated by dashes. This format provides a useful balance between compactness and readability. The type identifier codes have been selected so as to provide a useful mnemonic when presented in Base32 encoding.

The following are examples of UDF values:

- NB1K-0M6U-OJ4F-DOUC-AQ74-W7B6-3Z6Q
- EBWM-M66M-3HA5-DPB5-DQJT-HBNE-7WUQ
- SAQL-VVJO-VJQY-MC2Y-OLIR-GWT7-AAQS-K
- MB5S-R4AJ-3FBT-7NH0-TZ62-2E6Y-WFH4
- KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN
- ABUK-NC4K-SSIV-HVKZ-A3Z7-SNPA-ZAQP

Like email addresses, UDFs are not a Uniform Resource Identifier (URI) but may be expressed in URI form by adding the scheme identifier (UDF) for use in contexts where an identifier in URI syntax is required. A UDF URI MAY contain a domain name component allowing it to be used as a locator.

1.1.1. Cryptographic Keys and Nonces

A Nonce (N) UDF represents a short, fixed length randomly chosen binary value.

Nonce UDFs are used within many Mesh protocols and data formats where it is necessary to represent a nonce value in text form.

Nonce UDF:
- NB1K-0M6U-OJ4F-DOUC-AQ74-W7B6-3Z6Q

An Encryption/Authentication (E) UDF has the same format as a Random UDF but is identified as being intended to be used as a symmetric key for encryption and/or authentication.

Key Value:
- 6C C6 7B CC D9 C1 D1 BC 3D 1C 13 33 85 A4 FD A9

Encryption/Authenticator UDF:
- EBWM-M66M-3HA5-DPB5-DQJT-HBNE-7WUQ
A Share (S) UDF also represents a short, fixed length binary value but only provides one share in secret sharing scheme. Recovery of the binary value requires a sufficient number of shares.

Share UDFs are used in the Mesh to support key and data escrow operations without the need to rely on trusted hardware. A share UDF can be copied by hand or printed in human or machine-readable form (e.g. QR code).

Key: EBWM-M66M-3HA5-DPB5-DQJT-HBNE-7WUQ
Share 0: SAQL-VVJQ-VJQY-MC2Y-OLIR-GWT7-AAQS-K
Share 1: SAQQ-RY7F-Q7UU-URHU-VCDB-HALY-LNCG-4
Share 2: SARF-N4U2-MVYQ-47UQ-3Y5R-HKDR-WZT6-U

1.1.2. Fingerprint type UDFs

Fingerprint type UDFs contains a fingerprint value calculated over a content data item and an IANA media type.

A Content Digest type UDF is a fingerprint type UDF in which the fingerprint is formed using a cryptographic algorithm. Two digest algorithms are currently supported, SHA-2-512 (M, for Merkle Damgard) and SHA-3-512 (K, for Keccak).

The inclusion of the media type in the calculation of the UDF value provides protection against semantic substitution attacks in which content that has been found to be trustworthy when interpreted as one content type is presented in a context in which it is interpreted as a different content type in which it is unsafe.

SHA-2-512: MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
SHA-3-512: KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN

An Authentication UDF (A) is formed in the same manner as a fingerprint but using a Message Authentication Code algorithm and a symmetric key.

Authentication UDFs are used to express commitments and to provide a means of blinding fingerprint values within a protocol by means of a nonce.

SHA-2-512: ABUK-NC4K-SSIV-HVKZ-A3Z7-SNPA-ZAQp

1.2. UDF URIs

The UDF URI scheme allows use of a UDF in contexts where a URF is expected. The UDF URI scheme has two forms, name and locator.
1.2.1. Name Form

Name form UDF URIs identify a data resource but do not provide a means of discovery. The URI is simply the scheme (udf) followed by the UDF value:

```
udf:MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
```

1.2.2. Locator Form

Locator form UDF URIs identify a data resource and provide a hint that MAY provide a means of discovery. If the content is not available from the location indicated, content obtained from a different source that matches the fingerprint MAY be used instead.

```
udf://MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
```

UDF locator form URIs presenting a fingerprint type UDF provide a tight binding of the content to the locator. This allows the resolved content to be verified and rejected if it has been modified.

UDF locator form URIs presenting an Encryptor/Authenticator type UDF provide a mechanism for identification, discovery and decryption of encrypted content. UDF locators of this type are known as Encrypted/Authenticated Resource Locators (EARLs).

Regardless of the type of the embedded UDF, UDF locator form URIs are resolved by first performing DNS Web Service Discovery to identify the Web Service Endpoint for the mmm-udf service at the specified domain.

Resolution is completed by presenting the Content Digest Fingerprint of the UDF value specified in the URI to the specified Web Service Endpoint and performing a GET method request on the result.

For example, Alice subscribes to Example.com, a purveyor of cat and kitten images. The company generates paper and electronic invoices on a monthly basis.

To generate the paper invoice, Example.com first creates a new encryption key:

```
EBIR-U3OZ-3FMP-Y5XV-G26I-5W32-ADKJ-BL
```

One or more electronic forms of the invoice are encrypted under the key EBIR-U3OZ-3FMP-Y5XV-G26I-5W32-ADKJ-BL and placed on the Example.com Web site so that the appropriate version is returned if Alice scans the QR code.
The key is then converted to form an EARL for the example.com UDF resolution service:

```
udf://example.com/EBIR-U3OZ-3FMP-Y5XV-G26I-5W32-ADKJ-BL
```

The EARL is then rendered as a QR code:

```
QR Code with embedded decryption and location key
```

A printable invoice containing the QR code is now generated and sent to Alice.

When Alice receives the invoice, she can pay it by simply scanning the invoice with a device that recognizes at least one of the invoice formats supported by Example.com.

The UDF EARL locator shown above is resolved by first determining the Web Service Endpoint for the mmm-udf service for the domain example.com.

```
```

Next the fingerprint of the source UDF is obtained.

```
UDF (EBIR-U3OZ-3FMP-Y5XV-G26I-5W32-ADKJ-BL) = MBPX-ZK4S-DKGJ-HOZT-7IYK-TJ7D-QXNF-LLC6-MAXI-NSKS-D2FC-W6OS-CKGP-45KO
```

Combining the Web Service Endpoint and the fingerprint of the source UDF provides the URI from which the content is obtained using the normal HTTP GET method:

```
```

Having established that Alice can read postal mail sent to a physical address and having delivered a secret to that address, this process might be extended to provide a means of automating the process of enrolment in electronic delivery of future invoices.
1.3. Secure Internet Names

A SIN is an Internet Identifier that contains a UDF fingerprint of a security policy document that may be used to verify the interpretation of the identifier. This permits traditional forms of Internet address such as URIs and RFC822 email addresses to be used to express a trusted address that is independent of any trusted third party.

This document only describes the syntax and interpretation of the identifiers themselves. The means by which the security policy documents bound to an address govern interpretation of the name is discussed separately in [draft-hallambaker-mesh-trust].

For example, Example Inc holds the domain name example.com and has deployed a private CA whose root of trust is a PKIX certificate with the UDF fingerprint MB2GK-6DUF5-YGYYL-JNY5E-RWSHZ.

Alice is an employee of Example Inc., she uses three email addresses:

alice@example.com  A regular email address (not a SIN).

alice@mm--mb2gk-6duf5-ygyyl-jny5e-rwshz.example.com  A strong email address that is backwards compatible.

alice@example.com.mm--mb2gk-6duf5-ygyyl-jny5e-rwshz  A strong email address that is backwards incompatible.

All three forms of the address are valid RFC822 addresses and may be used in a legacy email client, stored in an address book application, etc. But the ability of a legacy client to make use of the address differs. Addresses of the first type may always be used. Addresses of the second type may only be used if an appropriate MX record is provisioned. Addresses of the third type will always fail unless the resolver understands that it is a SIN requiring special processing.

These rules allow Bob to send email to Alice with either 'best effort' security or mandatory security as the circumstances demand.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.
2.1. 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].

2.2. Defined Terms

Cryptographic Digest Function  A hash function that has the properties required for use as a cryptographic hash function. These include collision resistance, first pre-image resistance and second pre-image resistance.

Content Type  An identifier indicating how a Data Value is to be interpreted as specified in the IANA registry Media Types.

Commitment  A cryptographic primitive that allows one to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later.

Data Value  The binary octet stream that is the input to the digest function used to calculate a digest value.

Data Object  A Data Value and its associated Content Type

Digest Algorithm  A synonym for Cryptographic Digest Function

Digest Value  The output of a Cryptographic Digest Function

Data Digest Value  The output of a Cryptographic Digest Function for a given Data Value input.

Fingerprint  A presentation of the digest value of a data value or data object.

Fingerprint Presentation  The representation of at least some part of a fingerprint value in human or machine-readable form.

Fingerprint Improvement  The practice of recording a higher precision presentation of a fingerprint on successful validation.

Fingerprint Work Hardening  The practice of generating a sequence of fingerprints until one is found that matches criteria that permit a compressed presentation form to be used. The compressed fingerprint thus being shorter than but presenting the same work factor as an uncompressed one.
Hash  A function which takes an input and returns a fixed-size output. Ideally, the output of a hash function is unbiased and not correlated to the outputs returned to similar inputs in any predictable fashion.

Precision  The number of significant bits provided by a Fingerprint Presentation.

Work Factor  A measure of the computational effort required to perform an attack against some security property.

2.3.  Related Specifications

This specification makes use of Base32 [RFC4648] encoding, SHA-2 [SHA-2] and SHA-3 [SHA-3] digest functions in the derivation of basic fingerprints. The derivation of keyed fingerprints additionally requires the use of the HMAC [RFC2014] and HKDF [RFC5869] functions.

Resolution of UDF URI Locators makes use of DNS Web Service Discovery [draft-hallambaker-web-service-discovery].

2.4.  Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

3.  Architecture

A Uniform Data Fingerprint (UDF) is a presentation of a UDF Binary Data Sequence.

This document specifies seven UDF Binary Data Sequence types and one presentation.

The first octet of a UDF Binary Data Sequence identifies the UDF type and is referred to as the Type identifier.

UDF Binary Data Sequence types are either fixed length or variable length. A variable length Binary Data Sequence MUST be truncated for presentation. Fixed length Binary Data Sequences MUST not be truncated.

3.1.  Base32 Presentation

The default UDF presentation is Base32 Presentation.
Variable length Binary Data Sequences are truncated to an integer multiple of 20 bits that provides the desired precision before conversion to Base32 form.

Fixed length Binary Data Sequences are converted to Base32 form without truncation.

After conversion to Base32 form, dash ‘-’ characters are inserted between groups of 4 characters to aid reading. This representation improves the accuracy of both data entry and verification.

3.1.1. Precision Improvement

Precision improvement is the practice of using a high precision UDF (e.g. 260 bits) calculated from content data that has been validated according to a lower precision UDF (e.g. 120 bits).

This allows a lower precision UDF to be used in a medium such as a business card where space is constrained without compromising subsequent uses.

Applications SHOULD make use of precision improvement wherever possible.

3.2. Type Identifier

A Version Identifier consists of a single byte.

The byte codes have been chosen so that the first character of the Base32 presentation of the UDF provides a mnemonic for its type. A SHA-2 fingerprint UDF will always have M (for Merkle Damgard) as the initial letter, a SHA-3 fingerprint UDF will always have K (for Keccak) as the initial letter, and so on.

The following version identifiers are specified in this document:
3.3. Content Type Identifier

A secure cryptographic digest algorithm provides a unique digest value that is probabilistically unique for a particular byte sequence but does not fix the context in which a byte sequence is interpreted. While such ambiguity may be tolerated in a fingerprint format designed for a single specific field of use, it is not acceptable in a general-purpose format.

For example, the SSH and OpenPGP applications both make use of fingerprints as identifiers for the public keys used but using different digest algorithms and data formats for representing the public key data. While no such vulnerability has been demonstrated to date, it is certainly conceivable that a crafty attacker might construct an SSH key in such a fashion that OpenPGP interprets the data in an insecure fashion. If the number of applications making use of fingerprint format that permits such substitutions is sufficiently large, the probability of a semantic substitution vulnerability being possible becomes unacceptably large.

A simple control that defeats such attacks is to incorporate a content type identifier within the scope of the data input to the hash function.

Table 1

<table>
<thead>
<tr>
<th>Type ID</th>
<th>Initial</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>HMAC-SHA-2-512</td>
</tr>
<tr>
<td>32</td>
<td>E</td>
<td>HKDF-AES-512</td>
</tr>
<tr>
<td>80</td>
<td>K</td>
<td>SHA-3-512</td>
</tr>
<tr>
<td>81</td>
<td>K</td>
<td>SHA-3-512 (20 bits compressed)</td>
</tr>
<tr>
<td>82</td>
<td>K</td>
<td>SHA-3-512 (30 bits compressed)</td>
</tr>
<tr>
<td>83</td>
<td>K</td>
<td>SHA-3-512 (40 bits compressed)</td>
</tr>
<tr>
<td>84</td>
<td>K</td>
<td>SHA-3-512 (50 bits compressed)</td>
</tr>
<tr>
<td>96</td>
<td>M</td>
<td>SHA-2-512</td>
</tr>
<tr>
<td>97</td>
<td>M</td>
<td>SHA-2-512 (20 bits compressed)</td>
</tr>
<tr>
<td>98</td>
<td>M</td>
<td>SHA-2-512 (30 bits compressed)</td>
</tr>
<tr>
<td>99</td>
<td>M</td>
<td>SHA-2-512 (40 bits compressed)</td>
</tr>
<tr>
<td>100</td>
<td>M</td>
<td>SHA-2-512 (50 bits compressed)</td>
</tr>
<tr>
<td>104</td>
<td>N</td>
<td>Nonce data</td>
</tr>
<tr>
<td>144</td>
<td>S</td>
<td>Shamir Secret Sharing</td>
</tr>
</tbody>
</table>
3.4. Truncation

Different applications of fingerprints demand different tradeoffs between compactness of the representation and the number of significant bits. A larger the number of significant bits reduces the risk of collision but at a cost to convenience.

Modern cryptographic digest functions such as SHA-2 produce output values of at least 256 bits in length. This is considerably larger than most uses of fingerprints require and certainly greater than can be represented in human readable form on a business card.

Since a strong cryptographic digest function produces an output value in which every bit in the input value affects every bit in the output value with equal probability, it follows that truncating the digest value to produce a fingerprint is at least as strong as any other mechanism if digest algorithm used is strong.

Using truncation to reduce the precision of the digest function has the advantage that a lower precision fingerprint of some data content is always a prefix of a higher prefix of the same content. This allows higher precision fingerprints to be converted to a lower precision without the need for special tools.

3.4.1. Compression

The Content Digest UDF types make use of work factor compression. Additional type identifiers are used to indicate digest values with 20, 30, 40 or 50 trailing zero bits allowing a UDF fingerprint offering the equivalent of up to 150 bits of precision to be expressed in 20 characters instead of 30.

To use compressed UDF identifiers, it is necessary to search for content that can be compressed. If the digest algorithm used is secure, this means that by definition, the fastest means of search is brute force. Thus, the reduction in fingerprint size is achieved by transferring the work factor from the attacker to the defender. To maintain a work factor of $2^{120}$ with a $2^{80}$ bits, it is necessary for the content generator to perform a brute force search at a cost of the order of $2^{40}$ operations.

For example, the smallest allowable work factor for a UDF presentation of a public key fingerprint is 92 bits. This would normally require a presentation with 20 significant characters. Reducing this to 16 characters requires a brute force search of approximately $10^6$ attempts. Reducing this to 12 characters would require $10^{12}$ attempts and to 10 characters, $10^{15}$ attempts.
Omission of support for higher levels of compression than $2^{50}$ is intentional.

In addition to allowing use of shorter presentations, work factor compression MAY be used as evidence of proof of work.

3.5. Presentation

The presentation of a fingerprint is the format in which it is presented to either an application or the user.

Base32 encoding is used to produce the preferred text representation of a UDF fingerprint. This encoding uses only the letters of the Latin alphabet with numbers chosen to minimize the risk of ambiguity between numbers and letters (2, 3, 4, 5, 6 and 7).

To enhance readability and improve data entry, characters are grouped into groups of four. This means that each block of four characters represents an increase in work factor of approximately one million times.

3.6. Alternative Presentations

Applications that support UDF MUST support use of the Base32 presentation. Applications MAY support alternative presentations.

3.6.1. Word Lists

The use of a Word List to encode fingerprint values was introduced by Patrick Juola and Philip Zimmerman for the PGPfone application. The PGP Word List is designed to facilitate exchange and verification of fingerprint values in a voice application. To minimize the risk of misinterpretation, two-word lists of 256 values each are used to encode alternative fingerprint bytes. The compact size of the lists used allowed the compilers to curate them so as to maximize the phonetic distance of the words selected.

The PGP Word List is designed to achieve a balance between ease of entry and verification. Applications where only verification is required may be better served by a much larger word list, permitting shorter fingerprint encodings.

For example, a word list with 16384 entries permits 14 bits of the fingerprint to be encoded at once, 65536 entries permits encoding of 16 bits. These encodings allow a 120 bit fingerprint to be encoded in 9 and 8 words respectively.
3.6.2. Image List

An image list is used in the same manner as a word list affording rapid visual verification of a fingerprint value. For obvious reasons, this approach is not suited to data entry but is preferable for comparison purposes. An image list of 1,048,576 images would provide a 20 bit encoding allowing 120 bit precision fingerprints to be displayed in six images.

4. Fixed Length UDFs

Fixed length UDFs are used to represent cryptographic keys, nonces and secret shares and have a fixed length determined by their function that cannot be truncated without loss of information.

All fixed length Binary Data Sequence values are an integer multiple of eight bits.

4.1. Nonce Type

A Nonce Type UDF consists of the type identifier octet 136 followed by the Binary Data Sequence value.

The Binary Data Sequence value is an integer number of octets that SHOULD have been generated in accordance with processes and procedures that ensure that it is sufficiently unpredictable for the purposes of the protocol in which the value is to be used. Requirements for such processes and procedures are described in [RFC4086].

Nonce Type UDFs are intended for use in contexts where it is necessary for a randomly chosen value to be unpredictable but not secret. For example, the challenge in a challenge/response mechanism.

4.2. Encryption/Authentication Type

An Encryption/Authentication Type UDF consists of the type identifier octet 104 followed by the Binary Data Sequence value.

The Binary Data Sequence value is an integer number of octets that SHOULD have been generated in accordance with processes and procedures that ensure that it is sufficiently unpredictable and unguessable for the purposes of the protocol in which the value is to be used. Requirements for such processes and procedures are described in [RFC4086].
Encryption/Authentication Type UDFs are intended to be used as a means of specifying secret cryptographic keying material. For example, the input to a Key Derivation Function used to encrypt a document. Accordingly, the identifier UDF corresponding to an Encryption/Authentication type UDF is a UDF fingerprint of the Encryption/Authentication Type UDF in Base32 presentation with content type 'application/udf-encryption'.

4.3. Shamir Shared Secret

The UDF format MAY be used to encode shares generated by a secret sharing mechanism. The only secret sharing mechanism currently supported is the Shamir Secret Sharing mechanism [Shamir79]. Each secret share represents a point represents a point on \((x, f(x))\), a polynomial in a modular field \(p\). The secret being shared is an integer multiple of 32 bits represented by the polynomial value \(f(0)\).

A Shamir Shared Secret Type UDF consists of the type identifier octet 144 followed by the Binary Data Sequence value describing the share value.

The first octet of the Binary Data Sequence value specifies the threshold value and the \(x\) value of the particular share:

- Bits 4-7 of the first byte specify the threshold value.
- Bits 0-3 of the first byte specify the \(x\) value minus 1.

The remaining octets specify the value \(f(x)\) in network byte (big-endian) order with leading padding if necessary so that the share has the same number of bytes as the secret.

The algorithm requires that the value \(p\) be a prime larger than the integer representing the largest secret being shared. For compactness of representation we chose \(p\) to be the smallest prime that is greater than \(2^n\) where \(n\) is an integer multiple of 32. This approach leaves a small probability that a set of chosen polynomial parameters cause one or more share values be larger than \(2^n\). Since it is the value of the secret rather than the polynomial parameters that is of important, such parameters MUST NOT be used.

4.3.1. Secret Generation

To share a secret of \(L\) bits with a threshold of \(n\) we use a \(f(x)\) a polynomial of degree \(n\) in the modular field \(p\):

\[
f(x) = a_0 + a_1.x + a_2.x^2 + \ldots + a_n.x^n
\]
where:

L  Is the length of the secret in bits, an integer multiple of 32.

n  Is the threshold, the number of shares required to reconstitute
    the secret.

a0 Is the integer representation of the secret to be shared.

a1 ... an  Are randomly chosen integers less than p

p  Is the smallest prime that is greater than 2^L.

For L=128, p = 2^128+51.

The values of the key shares are the values f(1), f(2),... f(n).

The most straightforward approach to generation of Shamir secrets is
    to generate the set of polynomial coefficients, a_0, a_1, ... a_n and
    use these to generate the share values f(1), f(2),... f(n).

Note that if this approach is adopted, there is a small probability
    that one or more of the values f(1), f(2),... f(n) exceeds the range
    of values supported by the encoding. Should this occur, at least one
    of the polynomial coefficients MUST be replaced.

An alternative means of generating the set of secrets is to select up
    to n-1 secret share values and use secret recovery to determine at
    least one additional share. If n shares are selected, the shared
    secret becomes an output of rather than an input to the process.

4.3.2. Recovery

To recover the value of the shared secret, it is necessary to obtain
    sufficient shares to meet the threshold and recover the value f(0) =
    a_0.

Applications MAY employ any approach that returns the correct result.
    The use of Lagrange basis polynomials is described in Appendix C.

Alice decides to encrypt an important document and split the
    encryption key so that there are five key shares, three of which will
    be required to recover the key.

Alice’s master secret is
    0C CE BA 49  7C 13 3F 5D  E1 96 61 B3  CA 8F 22 85

This has the UDF representation:
The master secret is converted to an integer applying network byte order conventions. Since the master secret is 128 bits, it is guaranteed to be smaller than the modulus. The resulting value becomes the polynomial value a0.

Since a threshold of three shares is required, we will need a second order polynomial. The co-efficients of the polynomial a1, a2 are random numbers smaller than the modulus:

\[
\begin{align*}
  a0 &= 17024127452518717767628838933325881989 \\
  a1 &= 136365534840891325614195791189102741129 \\
  a2 &= 4976424005849144061671329931296451151
\end{align*}
\]

The master secret is the value \( f(0) = a0 \). The key shares are the values \( f(1), f(2), \ldots, f(5) \):

\[
\begin{align*}
  f(1) &= 20309608629925918744349596053725074269 \\
  f(2) &= 148298526236759481779331133604948957344 \\
  f(3) &= 192913814185958064238508967018765742721 \\
  f(4) &= 336941950146854934821029460295175430400 \\
  f(5) &= 2401005671985116306351800602409808874
\end{align*}
\]

The first byte of each share specifies the recovery information (quorum, x value), the remaining bytes specify the share value in network byte order:

\[
\begin{align*}
  f(1) &= \text{30 98 CA E2 27 C2 0F C9 44 40 90 67 25 2C FD 13 5D} \\
  f(2) &= \text{31 6F 91 41 DA 66 B1 F4 45 92 B8 74 99 BF 19 58 A0} \\
  f(3) &= \text{32 91 21 D9 61 69 F9 C0 61 D8 0E 8A 11 80 E3 F2 81} \\
  f(4) &= \text{33 FD 7C A8 BC CB E7 2D 99 10 92 A7 8C 72 5C E1 00} \\
  f(5) &= \text{34 B4 A1 AF EC 8C 7A 3B EB 3C 44 CD 0A 93 84 23 EA}
\end{align*}
\]

The UDF presentation of the key shares is thus:
To recover the value $f(0)$ from any three shares, we need to fit a polynomial curve to the three points and use it to calculate the value at $x=0$ using the Lagrange polynomial basis.

5. Variable Length UDFs

Variable length UDFs are used to represent fingerprint values calculated over a content type identifier and the cryptographic digest of a content data item. The fingerprint value MAY be specified at any integer multiple of 20 bits that provides a work factor sufficient for the intended purpose.

Two types of fingerprint are specified:

- **Digest fingerprints**  Are computed with the same cryptographic digest algorithm used to calculate the digest of the content data.
- **Message Authentication Code fingerprints**  Are computed using a Message Authentication Code.

For a given algorithm (and key, if requires), if two UDF fingerprints are of the same content data and content type, either the fingerprint values will be the same or the initial characters of one will be exactly equal to the other.

5.1. Content Digest

A Content Digest Type UDF consists of the type identifier octet followed by the Binary Data Sequence value.

The type identifier specifies the digest algorithm used and the compression level. Two digest algorithms are currently specified with four compression levels for each making a total of eight possible type identifiers.

The Content Digest UDF for given content data is generated by the steps of:

1. Applying the digest algorithm to determine the Content Digest Value
2. Applying the digest algorithm to determine the Typed Content Digest Value

3. Determining the compression level from bytes 0-3 of the Typed Content Digest Value.

4. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.

5. Truncating bytes 4-63 of the Typed Content Digest Value to determine the Binary Data Sequence value.

5.1.1. Content Digest Value

The Content Digest Value (CDV) is determined by applying the digest algorithm to the content data:

\[ \text{CDV} = H(<\text{Data}>) \]

Where

- \( H(x) \) is the cryptographic digest function
- \(<\text{Data}>\) is the binary data.

5.1.2. Typed Content Digest Value

The Typed Content Digest Value (TCDV) is determined by applying the digest algorithm to the content type identifier and the CDV:

\[ \text{TCDV} = H(<\text{Content-ID}> + ?:+ + \text{CDV}) \]

Where

- \( A + B \) represents concatenation of the binary sequences \( A \) and \( B \).
- \(<\text{Content-ID}>\) is the IANA Content Type of the data in UTF8 encoding

The two-step approach to calculating the Type Content Digest Value allows an application to attempt to match a set of content data against multiple types without the need to recalculate the value of the content data digest.

5.1.3. Compression

The compression factor is determined according to the number of trailing zero bits in the first 8 bytes of the Typed Content Digest Value as follows:
19 or fewer leading zero bits  Compression factor = 0
29 or fewer leading zero bits  Compression factor = 20
39 or fewer leading zero bits  Compression factor = 30
49 or fewer leading zero bits  Compression factor = 40
50 or more leading zero bits  Compression factor = 50

The least significant bits of each octet are regarded to be 'trailing'.

Applications MUST use compression when creating and comparing UDFs. Applications MAY support content generation techniques that search for UDF values that use a compressed representation. Presentation of a content digest value indicating use of compression MAY be used as an indicator of 'proof of work'.

5.1.4. Presentation

The type identifier is determined by the algorithm and compression factor as follows:

<table>
<thead>
<tr>
<th>Type ID</th>
<th>Initial</th>
<th>Algorithm</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>K</td>
<td>SHA-3-512</td>
<td>0</td>
</tr>
<tr>
<td>81</td>
<td>K</td>
<td>SHA-3-512</td>
<td>20</td>
</tr>
<tr>
<td>82</td>
<td>K</td>
<td>SHA-3-512</td>
<td>30</td>
</tr>
<tr>
<td>83</td>
<td>K</td>
<td>SHA-3-512</td>
<td>40</td>
</tr>
<tr>
<td>84</td>
<td>K</td>
<td>SHA-3-512</td>
<td>50</td>
</tr>
<tr>
<td>96</td>
<td>M</td>
<td>SHA-2-512</td>
<td>0</td>
</tr>
<tr>
<td>97</td>
<td>M</td>
<td>SHA-2-512</td>
<td>20</td>
</tr>
<tr>
<td>98</td>
<td>M</td>
<td>SHA-2-512</td>
<td>30</td>
</tr>
<tr>
<td>99</td>
<td>M</td>
<td>SHA-2-512</td>
<td>40</td>
</tr>
<tr>
<td>100</td>
<td>M</td>
<td>SHA-2-512</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2

The Binary Data Sequence value is taken from the Typed Content Digest Value starting at the 9th octet and as many additional bytes as are required to meet the presentation precision.
5.1.5. Example Encoding

In the following examples, <Content-ID> is the UTF8 encoding of the string "text/plain" and <Data> is the UTF8 encoding of the string "UDF Data Value"

Data = 55 44 46 20 44 61 74 61 20 56 61 6C 75 65

ContentType = 74 65 78 74 2F 70 6C 61 69 6E

5.1.6. Using SHA-2-512 Digest

H(<Data>) = 48 DA 47 CC AB FE A4 5C 76 61 D3 21 BA 34 3E 58
10 87 2A 03 B4 02 9D AB 84 7C CE D2 22 B6 9C AB
02 38 D4 E9 1E 2F 6B 36 A0 9E ED 11 09 8A EA AC
99 D9 E0 BD EA 47 93 15 BD 7A E9 E1 2E AD C4 15

<Content-ID> + ':' + H(<Data>) = 74 65 78 74 2F 70 6C 61 69 6E 3A 48 DA 47 CC AB
FE A4 5C 76 61 D3 21 BA 34 3E 58 10 87 2A 03 B4
02 9D AB 84 7C CE D2 22 B6 9C AB 02 38 D4 E9 1E
2F 6B 36 A0 9E ED 11 09 8A EA AC 99 D9 E0 BD EA
47 93 15 BD 7A E9 E1 2E AD C4 15

H(<Content-ID> + ':' + H(<Data>)) = C6 AF B7 C0 FE BE 04 E5 AE 94 E3 7B AA 5F 1A 40
5B A3 CE CC 97 4D 55 C0 9E 61 E4 B0 EF 9C AE F9
EB 83 BB 9D 5F 0F 39 F6 5F AA 0E DC 67 2A 67 71
4F FF 8F 83 C4 55 38 36 38 AE 42 7A 82 9C 85 BB

The prefixed Binary Data Sequence is thus
60 C6 AF B7 C0 FE BE 04 E5 AE 94 E3 7B AA 5F 1A 40 5B

The 125 bit fingerprint value is MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF

This fingerprint MAY be specified with higher or lower precision as appropriate.

100 bit precision MDDK-7N6A-727A-JZNO-STRX

120 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7

200 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF-XI60-ZSLU-2VOA
5.1.7. Using SHA-3-512 Digest

\[ H(<Data>) = \]
\[
6D 2E CF E6 93 5A 0C FC F2 A9 1A 49 E0 0C D8 07 \\
A1 4E 70 AB 72 94 6E CC BB 47 48 F1 8E 41 49 95 \\
07 1D F3 6E 0D 0C 8B 60 39 C1 8E B4 0F 6E C8 08 \\
65 B4 C4 45 9B A2 7E 97 74 7B BE 68 BC A8 C2 17
\]

\[ <\text{Content-ID}> + ':' + H(<Data>) = \]
\[
74 65 78 74 2F 70 6C 61 69 6E 3A 6D 2E CF E6 93 \\
5A 0C FC F2 A9 1A 49 E0 0C D8 07 A1 4E 70 AB 72 \\
94 6E CC BB 47 48 F1 8E 41 49 95 07 1D F3 6E 0D \\
0C 8B 60 39 C1 8E B4 0F 6E C8 08 65 B4 C4 45 9B \\
A2 7E 97 74 7B BE 68 BC A8 C2 17
\]

\[ H(<\text{Content-ID}> + ':' + H(<Data>)) = \]
\[
8A 86 8A 06 1C 54 6E 7E 3F 75 5F 3F 88 F9 FD 2F \\
8E C8 45 93 1B 80 A8 2F 29 16 7B A3 BE 21 1F 8A \\
75 61 88 A1 D5 7F 07 D5 9D 68 A4 2D 17 F4 4D 23 \\
F9 E4 OB B2 1A 8D B9 F5 8D FC EC BD 01 F4 37 7C
\]

The prefixed Binary Data Sequence is thus
\[
50 8A 86 8A 06 1C 54 6E 7E 3F 75 5F 39 88 F9 FD 2F \\
8E
\]

The 125 bit fingerprint value is KCFI-NCQG-DRKG-47R7-OVPT-TCH2-7UXY

5.1.8. Using SHA-2-512 Digest with Compression

The content data "UDF Compressed Document 4187123" produces a UDF Content Digest SHA-2-512 binary value with 20 trailing zeros and is therefore presented using compressed presentation:

\[ \text{Data} = " \]
\[
55 44 46 20 43 6F 6D 70 72 65 73 73 65 64 20 44 \\
6F 63 75 6D 65 6E 74 20 34 31 38 37 31 32 33"
\]

The UTF8 Content Digest is given as:
H(<Data>) =
36 21 FA 2A C5 D8 62 5C 2D 0B 45 FB 65 93 FC 69
C1 ED F7 00 AE 6F E3 3D 38 13 FE AB 76 AA 74 13
6D 5A 2B 20 DE D6 A5 CF 6C D4 E6 56 3F F3 C0 C7
C4 1D 3F 43 DD DC F1 A5 67 A7 E0 67 9A B0 C6 B7

H(<Content-ID> + ':' + H(<Data>)) =
74 65 78 74 2F 70 6C 61 69 6E 3A 36 21 FA 2A C5
D8 6B 5C 2D 0B 45 FB 65 93 FC 69 C1 ED F7 00 AE
6F E3 3D 38 13 FE AB 76 AA 74 13 6D 5A 2B 20 DE
D6 A5 CF 6C 04 E6 56 3F F3 C0 C7 C4 1D 3F 43 DD
DC F1 A5 67 A7 E0 67 9A B0 C6 B7

The prefixed Binary Data Sequence is thus
61 8E 14 D9 19 19 4E D6 02 12 C3 30 A7 BB 5F C7 17 6D AE

The 125 bit fingerprint value is MGHB-JWIZ-J3LA-EEWD-GCT3-WX6H-C5W2

5.1.9. Using SHA-3-512 Digest with Compression

The content data "UDF Compressed Document 774665" produces a UDF Content Digest SHA-3-512 binary value with 20 trailing zeros and is therefore presented using compressed presentation:

Data =
55 44 46 20 43 6F 6D 70 72 65 73 73 65 64 20 44
6F 63 75 6D 65 6E 74 20 37 37 34 36 36 35

The UTF8 SHA-3-512 Content Digest is KEJI-Y225-BDUG-XX22-MXKE-5ITF-YVYM

5.2. Authenticator UDF

An authenticator Type UDF consists of the type identifier octet followed by the Binary Data Sequence value.

The type identifier specifies the digest and Message Authentication Code algorithm. Two algorithm suites are currently specified. Use of compression is not supported.
The Authenticator UDF for given content data and key is generated by the steps of:

1. Applying the digest algorithm to determine the Content Digest Value

2. Applying the MAC algorithm to determine the Authentication value

3. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.

4. Truncating the Authentication value to determine the Binary Data Sequence value.

The key used to calculate and Authenticator type UDF is always a UNICODE string. If use of a binary value as a key is required, the value MUST be converted to a string format first. For example, by conversion to an Encryption/Authentication type UDF.

5.2.1. Content Digest Value

The Content Digest Value (CDV) is determined in the exact same fashion as for a Content Digest UDF by applying the digest algorithm to the content data:

\[
\text{CDV} = H(<\text{Data}>)
\]

Where

\(H(x)\) is the cryptographic digest function

<Data> is the binary data.

5.2.2. Authentication Value

The Authentication Value (AV) is determined by applying the digest algorithm to the content type identifier and the CDV:

\[
\text{AV} = \text{MAC} (<\text{OKM}>, (<\text{Content-ID}> + ?:? + \text{CDV}>)
\]

Where

<OKM> is the authentication key as specified below

\(\text{MAC( <Key>, <data>)}\) is the result of applying the Message Authentication Code algorithm to with Key <Key> and data <data>

The value is calculated as follows:
IKM = UTF8 (Key)
PRK = MAC (UTF8 ("KeyedUDFMaster"), IKM)
OKM = HKDF-Expand (PRK, UTF8 ("KeyedUDFExpand"), HashLen)

Where the function UTF8(string) converts a string to the binary UTF8 representation, HKDF-Expand is as defined in [RFC5869] and the function MAC(k,m) is the HMAC function formed from the specified hash H(m) as specified in [RFC2014].

Keyed UDFs are typically used in circumstances where user interaction requires a cryptographic commitment type functionality.

In the following example, <Content-ID> is the UTF8 encoding of the string "text/plain" and <Data> is the UTF8 encoding of the string "Konrad is the traitor". The randomly chosen key is NDD7-6CMX-H2FW-ISAL-K4VB-DQ3E-PEDM.

Data =
4B 6F 6E 72 61 64 20 69 73 20 74 68 65 20 74 72 61 69 74 6F 72
61 69 74 6F 72

ContentType =
74 65 78 74 74 2F 70 6C 61 69 6E

Key =
4E 44 44 37 2D 36 43 4D 58 2D 48 32 46 57 2D 49 53 41 4C 2D 4B 34 56 42 2D 44 51 33 45 2D 50 45
44 4D

Processing is performed in the same manner as an unkeyed fingerprint except that compression is never used:
H(<Data>) =
93 FC DA F9 FA FD 1E 26 50 26 C3 C1 28 43 40 73
D8 BC 3D 62 87 73 2B 73 B8 EC 93 B6 DE 80 FF DA
70 0A D1 CE E8 F4 36 68 EF 4E 71 63 41 53 91 5C
CE 8C 5C CE C7 9A 46 94 6A 35 79 F9 33 70 85 01

<Content-ID> + ':' + H(<Data>) =
74 65 78 74 2F 70 6C 61 6E 3A 93 FC DA F9 FA
FD 1E 26 50 26 C3 C1 28 43 40 73 D8 BC 3D 62 87
73 2B 73 B8 EC 93 B6 DE 80 FF DA 70 0A D1 CE E8
F4 36 68 EF 4E 71 63 41 53 91 5C CE 8C 5C CE C7
9A 46 94 6A 35 79 F9 33 70 85 01

PRK(Key) =
77 D3 0A 08 39 BD 9D C0 97 44 DA 33 15 0A 42 5E
CD 17 80 03 B3 CF CC 89 7A C7 84 12 B4 51 5B 25
DC 26 F5 E1 1B 20 F3 89 2E 9A 1A 7B 0E 73 23 39
0E C3 4C EF 2D 40 DA 05 B4 70 C6 1C 82 C1 49 33

HKDF(Key) =
BF A9 B4 58 9C 1D 68 D7 9A B7 11 F6 C8 98 59 14
20 D7 82 67 C5 84 22 E5 A0 F9 93 52 B1 C3 87 EB
05 06 CB C4 E4 D6 E6 EE 1F F0 D4 7A 97 68 5E CE
28 1C CA AF D8 B5 D1 24 4A 71 EC E3 AC B5 D2 04

MAC(<key>, <Content-ID> + ':' + H(<Data>)) =
4C C3 7F D3 F9 9E 52 CF 07 90 74 53 84 65 95 BC
1A 2B A5 D1 68 9D 05 6D 06 C5 CA BF 17 CB E0 49
95 39 57 08 79 C4 E5 49 D3 3A 59 A3 32 05 45 A6
30 26 25 AE 8A F4 47 C6 1F B5 33 7F AD 69 A6 30

The prefixed Binary Data Sequence is thus
00 4C C3 7F D3 F9 9E 52 CF 07 90 74 53 84 65 95 BC 1A

The 125 bit fingerprint value is ABGM-G76T-7GPF-FTYH-SB2F-HBDF-SW6B

5.3. Content Type Values

While a UDF fingerprint MAY be used to identify any form of static data, the use of a UDF fingerprint to identify a public key signature key provides a level of indirection and thus the ability to identify dynamic data. The content types used to identify public keys are thus of particular interest.

As described in the security considerations section, the use of fingerprints to identify a bare public key and the use of
fingerprints to identify a public key and associated security policy information are very different.

5.3.1. PKIX Certificates and Keys

UDF fingerprints MAY be used to identify PKIX certificates, CRLs and public keys in the ASN.1 encoding used in PKIX certificates.

Since PKIX certificates and CRLs contain security policy information, UDF fingerprints used to identify certificates or CRLs SHOULD be presented with a minimum of 200 bits of precision. PKIX applications MUST not accept UDF fingerprints specified with less than 200 bits of precision for purposes of identifying trust anchors.

PKIX certificates, keys and related content data are identified by the following content types:

- application/pkix-cert  A PKIX Certificate
- application/pkix-crl  A PKIX CRL
- application/pkix-keyinfo  The KeyInfo structure defined in the PKIX certificate specification.

5.3.2. OpenPGP Key

OpenPGPv5 keys and key set content data are identified by the following content type:

- application/pgp-keys  An OpenPGP key set.

5.3.3. DNSSEC

DNSSEC record data consists of DNS records which are identified by the following content type:

- application/dns  A DNS resource record in binary format

6. UDF URIs

The UDF URI scheme describes a means of constructing URIs from a UDF value.

Two forms or UDF URI are specified, Name and Locator. In both cases the URI MUST specify the scheme type "UDF", and a UDF fingerprint and MAY specify a query identifier and/or a fragment identifier.
By definition a Locator form URI contains an authority field which
MUST be a DNS domain name. The use of IP address forms for this
purpose is not permitted.

Name Form URIs allow static content data to be identified without
specifying the means by which the content data may be retrieved.
Locator form URIs allow static content data or dynamic network
resources to be identified and the means of retrieval.

The syntax of a UDF URI is a subset of the generic URI syntax
specified in [RFC3986]. The use of userinfo and port numbers is not
supported and the path part of the uri is a UDF in base32
presentation.

```
URI           = "UDF:" udf [ "?" query ] [ "#" fragment ]
udf           = name-form / locator-form
name-form     = udf-value
locator-form  = "//" authority "/" udf-value
authority     = host
host          = reg-name
```

6.1. Name form

Name form UDF URIs provide a means of presenting a UDF value in a
context in which a URI form of a name is required without providing a
means of resolution.

Adding the UDF scheme prefix to a UDF fingerprint does not change the
semantics of the fingerprint itself. The semantics of the name
result from the context in which it is used.

For example, a UDF value of any type MAY be used to give a unique
targetNamespace value in an XML Schema [XMLSchema]

6.2. Locator form

The locator form of an unkeyed UDF URI is resolved by the following
steps:

- Use DNS Web service discovery to determine the Web Service
  Endpoint.
- Determine the content identifier from the source URI.
o Append the content identifier to the Web Service Endpoint as a suffix to form the target URI.

o Retrieve content from the Web Service Endpoint by means of a GET method.

o Perform post processing as specified by the UDF type.

6.2.1. DNS Web service discovery

DNS Web Discovery is performed as specified in [draft-hallambaker-web-service-discovery] for the service mmm-udf and domain name specified in the URI. For a full description of the discovery mechanism, consult the referenced specification.

The use of DNS Web Discovery permits service providers to make full use of the load balancing and service description capabilities afforded by use of DNS SRV and TXT records in accordance with the approach described in [RFC6763].

If no SRV or TXT records are specified, DNS Web Discovery specifies that the Web Service Endpoint be the Well Known Service [RFC5785] with the prefix /.well-known/srv/mmm-udf.

6.2.2. Content Identifier

For all UDF types other than Secret Share, the Content Identifier value is the UDF SHA-2-512 Content Digest of the canonical form of the UDF specified in the source URI presented at twice the precision to a maximum of 440 bits.

If the UDF is of type Secret Share, the shared secret MUST be recovered before the content identifier can be resolved. The shared secret is then expressed as a UDF of type Encryption/Authentication and the Content Identifier determined as for an Encryption/Authentication type UDF.

6.2.3. Target URI

The target URI is formed by appending a slash separator ‘/’ and the Content Identifier value to the Web Service Endpoint.

Since the path portion of a URI is case sensitive, the UDF value MUST be specified in upper case and MUST include separator marks.
6.2.4. Postprocessing

After retrieving the content data, the resolver MUST perform post processing as indicated by the content type:

Nonce  No additional post processing is required.

Content Digest  The resolver MUST verify that the content returned matches the UDF fingerprint value.

Authenticator  The resolver MUST verify that the content returned matches the UDF fingerprint value.

Encryption/Authentication  The content data returned is decrypted and authenticated using the key specified in the UDF value as the initial keying material (see below).

Secret Share (set)  The content data returned is decrypted and authenticated using the shared secret as the initial keying material (see below).

6.2.5. Decryption and Authentication

The steps performed to decode cryptographically enhanced content data depends on the content type specified in the returned content. Two formats are currently supported:

- DARE Envelope format as specified in [draft-hallambaker-mesh-dare]
- Cryptographic Message Syntax (CMS) Symmetric Key Package as specified in [RFC6031]

6.2.6. QR Presentation

Encoding of a UDF URI as a QR code requires only the characters in alphanumeric encoding, thus achieving compactness with minimal overhead.

7. Strong Internet Names

A Strong Internet Name is an Internet address that is bound to a policy governing interpretation of that address by means of a Content Digest type UDF of the policy expressed as a UDF prefixed DNS label within the address itself.

The Reserved LDH labels as defined in [RFC5890] that begin with the prefix mm-- are reserved for use as Strong Internet Names. The
characters following the prefix are a Content Digest type UDF in Base32 presentation.

Since DNS labels are limited to 63 characters, the presentation of the SIN itself is limited to 59 characters and thus 240 bits of precision.

8. Security Considerations

This section describes security considerations arising from the use of UDF in general applications.

Additional security considerations for use of UDFs in Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

8.1. Confidentiality

Encrypted locator is a bearer token

8.2. Availability

Corruption of a part of a shared secret may prevent recovery

8.3. Integrity

Shared secret parts do not contain context information to specify which secret they relate to.

8.4. Work Factor and Precision

A given UDF data object has a single fingerprint value that may be presented at different precisions. The shortest legitimate precision with which a UDF fingerprint may be presented has 96 significant bits

A UDF fingerprint presents the same work factor as any other cryptographic digest function. The difficulty of finding a second data item that matches a given fingerprint is $2^n$ and the difficulty or finding two data items that have the same fingerprint is $2^{n/2}$. Where $n$ is the precision of the fingerprint.

For the algorithms specified in this document, $n = 512$ and thus the work factor for finding collisions is $2^{256}$, a value that is generally considered to be computationally infeasible.

Since the use of 512 bit fingerprints is impractical in the type of applications where fingerprints are generally used, truncation is a practical necessity. The longer a fingerprint is, the less likely it
is that a user will check every character. It is therefore important to consider carefully whether the security of an application depends on second pre-image resistance or collision resistance.

In most fingerprint applications, such as the use of fingerprints to identify public keys, the fact that a malicious party might generate two keys that have the same fingerprint value is a minor concern. Combined with a flawed protocol architecture, such a vulnerability may permit an attacker to construct a document such that the signature will be accepted as valid by some parties but not by others.

For example, Alice generates keypairs until two are generated that have the same 100 bit UDF presentation (typically $2^{48}$ attempts). She registers one keypair with a merchant and the other with her bank. This allows Alice to create a payment instrument that will be accepted as valid by one and rejected by the other.

The ability to generate of two PKIX certificates with the same fingerprint and different certificate attributes raises very different and more serious security concerns. For example, an attacker might generate two certificates with the same key and different use constraints. This might allow an attacker to present a highly constrained certificate that does not present a security risk to an application for purposes of gaining approval and an unconstrained certificate to request a malicious action.

In general, any use of fingerprints to identify data that has security policy semantics requires the risk of collision attacks to be considered. For this reason, the use of short, ‘user friendly’ fingerprint presentations (Less than 200 bits) SHOULD only be used for public key values.

8.5. Semantic Substitution

Many applications record the fact that a data item is trusted, rather fewer record the circumstances in which the data item is trusted. This results in a semantic substitution vulnerability which an attacker may exploit by presenting the trusted data item in the wrong context.

The UDF format provides protection against high level semantic substitution attacks by incorporating the content type into the input to the outermost fingerprint digest function. The work factor for generating a UDF fingerprint that is valid in both contexts is thus the same as the work factor for finding a second preimage in the digest function ($2^{512}$ for the specified digest algorithms).
It is thus infeasible to generate a data item such that some applications will interpret it as a PKIX key and others will accept as an OpenPGP key. While attempting to parse a PKIX key as an OpenPGP key is virtually certain to fail to return the correct key parameters it cannot be assumed that the attempt is guaranteed to fail with an error message.

The UDF format does not provide protection against semantic substitution attacks that do not affect the content type.

8.6. QR Code Scanning

The act of scanning a QR code SHOULD be considered equivalent to clicking on an unlabeled hypertext link. Since QR codes are scanned in many different contexts, the mere act of scanning a QR code MUST NOT be interpreted as constituting an affirmative acceptance of terms or conditions or as creating an electronic signature.

If such semantics are required in the context of an application, these MUST be established by secondary user actions made subsequent to the scanning of the QR code.

There is a risk that use of QR codes to automate processes such as payment will lead to abusive practices such as presentation of fraudulent invoices for goods not ordered or delivered. It is therefore important to ensure that such requests are subject to adequate accountability controls.

9. IANA Considerations

Registrations are requested in the following registries:

- Service Name and Transport Protocol Port Number
- well-known URI registry
- Uniform Resource Identifier (URI) Schemes
- Media Types

In addition, the creation of the following registry is requested: Uniform Data Fingerprint Type Identifier Registry.

9.1. Protocol Service Name

The following registration is requested in the Service Name and Transport Protocol Port Number Registry in accordance with [RFC6355]
Service Name (REQUIRED)  mmm-udf

Transport Protocol(s) (REQUIRED)  TCP

Assignee (REQUIRED)  Phillip Hallam-Baker, phill@hallambaker.com

Contact (REQUIRED)  Phillip Hallam-Baker, phill@hallambaker.com

Description (REQUIRED)  mmm-udf is a Web Service protocol that resolves Mathematical Mesh Uniform Data Fingerprints (UDF) to resources. The mmm-udf service name is used in service discovery to identify a Web Service endpoint to perform resolution of a UDF presented in URI locator form.

Reference (REQUIRED)  [This document]

Port Number (OPTIONAL)  None

Service Code (REQUIRED for DCCP only)  None

Known Unauthorized Uses (OPTIONAL)  None

Assignment Notes (OPTIONAL)  None

9.2. Well Known

The following registration is requested in the well-known URI registry in accordance with [RFC5785]

URI suffix  srv/mmm-udf

Change controller  Phillip Hallam-Baker, phill@hallambaker.com

Specification document(s):  [This document]

Related information

[draft-hallambaker-web-service-discovery]

9.3. URI Registration

The following registration is requested in the Uniform Resource Identifier (URI) Schemes registry in accordance with [RFC7595]

Scheme name:  UDF

Status:  Provisional
Applications/protocols that use this scheme name: Mathematical Mesh Service protocols (mmm)

Contact: Phillip Hallam-Baker mailto:phill@hallambaker.com

Change controller: Phillip Hallam-Baker

References: [This document]

9.4. Media Types Registrations

9.4.1. Media Type: application/pkix-keyinfo

Type name: application

Subtype name: pkix-keyinfo

Required parameters: None

Optional parameters: None

Encoding considerations: None

Security considerations: Described in [This]

Interoperability considerations: None

Published specification: [This]

Applications that use this media type: Uniform Data Fingerprint

Fragment identifier considerations: None

Additional information: Deprecated alias names for this type: None

Magic number(s): None

File extension(s): None

Macintosh file type code(s): None

Person & email address to contact for further information:
Phillip Hallam-Baker @hallambaker.com>

Intended usage: Content type identifier to be used in constructing UDF Content Digests and Authenticators and related cryptographic purposes.
Restrictions on usage: None
Author: Phillip Hallam-Baker
Change controller: Phillip Hallam-Baker
Provisional registration? (standards tree only): Yes

9.4.2. Media Type: application/udf-encryption

Type name: application
Subtype name: udf-encryption
Required parameters: None
Optional parameters: None
Encoding considerations: None
Security considerations: Described in [This]
Interoperability considerations: None
Published specification: [This]
Applications that use this media type: Uniform Data Fingerprint
Fragment identifier considerations: None
Additional information: Deprecated alias names for this type: None
Magic number(s): None
File extension(s): None
Macintosh file type code(s): None

Person & email address to contact for further information:
Phillip Hallam-Baker @hallambaker.com>

Intended usage: Content type identifier to be used in constructing
UDF Content Digests and Authenticators and related cryptographic
purposes.

Restrictions on usage: None
Author: Phillip Hallam-Baker
9.4.3. Media Type: application/udf-secret

Type name: application
Subtype name: udf-secret
Required parameters: None
Optional parameters: None
Encoding considerations: None
Security considerations: Described in [This]
Interoperability considerations: None
Published specification: [This]
Applications that use this media type: Uniform Data Fingerprint
Fragment identifier considerations: None
Additional information: Deprecated alias names for this type: None
   Magic number(s): None
   File extension(s): None
   Macintosh file type code(s): None

Person & email address to contact for further information:
   Phillip Hallam-Baker @hallambaker.com

Intended usage: Content type identifier to be used in constructing
   UDF Content Digests and Authenticators and related cryptographic
   purposes.
Restrictions on usage: None
Author: Phillip Hallam-Baker
Change controller: Phillip Hallam-Baker
Provisional registration? (standards tree only): Yes
9.5. Uniform Data Fingerprint Type Identifier Registry

This document describes a new extensible data format employing fixed length version identifiers for UDF types.

9.5.1. The name of the registry

Uniform Data Fingerprint Type Identifier Registry

9.5.2. Required information for registrations

Registrants must specify the Type identifier code(s) requested, description and RFC number for the corresponding standards action document.

The standards document must specify the means of generating and interpreting the UDF Data Sequence Value and the purpose(s) for which it is proposed.

Since the initial letter of the Base32 presentation provides a mnemonic function in UDFs, the standards document must explain why the proposed Type Identifier and associated initial letter are appropriate. In cases where a new initial letter is to be created, there must be an explanation of why this is appropriate. If an existing initial letter is to be created, there must be an explanation of why this is appropriate and/or acceptable.

9.5.3. Applicable registration policy

Due to the intended field of use (human data entry), the code space is severely constrained. Accordingly, it is intended that code point registrations be as infrequent as possible.

Registration of new digest algorithms is strongly discouraged and should not occur unless, (1) there is a known security vulnerability in one of the two schemes specified in the original assignment and (2) the proposed algorithm has been subjected to rigorous peer review, preferably in the form of an open, international competition and (3) the proposed algorithm has been adopted as a preferred algorithm for use in IETF protocols.

Accordingly, the applicable registration policy is Standards Action.

9.5.4. Size, format, and syntax of registry entries

Each registry entry consists of a single byte code,
9.5.5. Initial assignments and reservations

The following entries should be added to the registry as initial assignments:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>HMAC and SHA-2-512</td>
<td>[This document]</td>
</tr>
<tr>
<td>32</td>
<td>HKDF-AES-512</td>
<td>[This document]</td>
</tr>
<tr>
<td>80</td>
<td>SHA-3-512</td>
<td>[This document]</td>
</tr>
<tr>
<td>81</td>
<td>SHA-3-512 with 20 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>82</td>
<td>SHA-3-512 with 30 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>82</td>
<td>SHA-3-512 with 40 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>83</td>
<td>SHA-3-512 with 50 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>96</td>
<td>SHA-2-512</td>
<td>[This document]</td>
</tr>
<tr>
<td>97</td>
<td>SHA-2-512 with 20 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>98</td>
<td>SHA-2-512 with 30 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>99</td>
<td>SHA-2-512 with 40 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>100</td>
<td>SHA-2-512 with 50 trailing zeros</td>
<td>[This document]</td>
</tr>
<tr>
<td>104</td>
<td>Random nonce</td>
<td>[This document]</td>
</tr>
<tr>
<td>144</td>
<td>Shamir Secret Share</td>
<td>[This document]</td>
</tr>
</tbody>
</table>

10. Acknowledgements

A list of people who have contributed to the design of the Mesh is presented in [draft-hallambaker-mesh-architecture].

Thanks are due to Viktor Dukhovni, Damian Weber and an anonymous member of the cryptography@metzdowd.com list for assisting in the compilation of the table of prime values.

11. Appendix A: Prime Values for Secret Sharing

The following are the prime values to be used for sharing secrets of up to 512 bits.

If it is necessary to share larger secrets, the corresponding prime may be found by choosing a value $2^{32}n$ that is larger than the secret to be encoded and determining the next largest number that is prime.
For example, the prime to be used to share a 128 bit value is $2^{128} + 51$.

12. Recovering Shamir Shared Secret

The value of a Shamir Shared secret may be recovered using Lagrange basis polynomials.

To share a secret with a threshold of $n$ shares and $L$ bits we constructed $f(x)$ a polynomial of degree $n$ in the modular field $p$ where $p$ is the smallest prime greater than $2^L$:

$$f(x) = a_0 + a_1.x + a_2.x^2 + ... + a_n.x^n$$

The shared secret is the binary representation of the value $a_0$

Given $n$ shares $(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})$, The corresponding the Lagrange basis polynomials $l_0, l_1, \ldots l_{n-1}$ are given by:

$$l_m = \left(\frac{(x - x(m_0)) \cdot (x - x(m_1)) \cdot \ldots \cdot (x - x(m_{n-2}))}{(x(m) - x(m_0)) \cdot (x(m) - x(m_1)) \cdot \ldots \cdot (x(m) - x(m_{n-2}))}\right)$$

Where the values $m_0, m_1, \ldots m_{n-2}$, are the integers $0, 1, \ldots n-1$, excluding the value $m$. 

Table 3

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Offset = Primen - 2n</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td>64</td>
<td>13</td>
</tr>
<tr>
<td>96</td>
<td>61</td>
</tr>
<tr>
<td>128</td>
<td>51</td>
</tr>
<tr>
<td>160</td>
<td>7</td>
</tr>
<tr>
<td>192</td>
<td>133</td>
</tr>
<tr>
<td>224</td>
<td>735</td>
</tr>
<tr>
<td>256</td>
<td>297</td>
</tr>
<tr>
<td>288</td>
<td>127</td>
</tr>
<tr>
<td>320</td>
<td>27</td>
</tr>
<tr>
<td>352</td>
<td>55</td>
</tr>
<tr>
<td>384</td>
<td>231</td>
</tr>
<tr>
<td>416</td>
<td>235</td>
</tr>
<tr>
<td>448</td>
<td>211</td>
</tr>
<tr>
<td>480</td>
<td>165</td>
</tr>
<tr>
<td>512</td>
<td>75</td>
</tr>
</tbody>
</table>
These can be used to compute $f(x)$ as follows:

$$f(x) = y_0l_0 + y_1l_1 + \ldots y_n-1l_{n-1}$$

Since it is only the value of $f(0)$ that we are interested in, we compute the Lagrange basis for the value $x = 0$:

$$l_z_m = \left(\frac{(x(m_1))}{(x(m) - x(m_1))}\right) \cdot \left(\frac{(x(m_2))}{(x(m) - x(m_2))}\right)$$

Hence,

$$a_0 = f(0) = y_0lz_0 + y_1lz_1 + \ldots y_n-1l_{n-1}$$

The following C# code recovers the values.

```csharp
using System;
using System.Collections.Generic;
using System.Numerics;

namespace Examples {
    class Examples {
        /// Combine a set of points $(x, f(x))$ on a polynomial of degree $n$ in a
discrete field modulo prime $p$ to recover the value $f(0)$ using Lagrange basis polynomials.

        /// The values $f(x)$.
        /// The values for $x$.
        /// The modulus.
        /// The polynomial degree.
        /// The value $f(0)$.
        static BigInteger CombineNK(BigInteger[] fx, int[] x, BigInteger p, int n) {
            if (fx.Length < n) {
                throw new Exception("Insufficient shares");
            }

            BigInteger accumulator = 0;
            for (var formula = 0; formula < n; formula++) {
                var value = fx[formula];
                BigInteger numerator = 1, denominator = 1;

                \text{...}
            }
        }
    }
}
```
for (var count = 0; count < n; count++) {
    if (formula == count) {
        continue; // If not the same value
    }

    var start = x[formula];
    var next = x[count];

    numerator = (numerator * -next) % p;
    denominator = (denominator * (start - next)) % p;
}

var InvDenominator = ModInverse(denominator, p);

accumulator = Modulus((accumulator +
    (fx[formula] * numerator * InvDenominator)), p);
}

return accumulator;
}

///
/// Compute the modular multiplicative inverse of the value
/// modulo
///
/// The value to find the inverse of
/// The modulus.
///
/// static BigInteger ModInverse(
///     BigInteger k,
///     BigInteger p) {
///     var m2 = p - 2;
///     if (k < 0) {
///         k = k + p;
///     }
///
///     return BigInteger.ModPow(k, m2, p);
/// }

///
/// Calculate the modulus of a number with correct handling
/// for negative numbers.
///
/// Value
/// The modulus.
/// x mod p
public static BigInteger Modulus(
    BigInteger x,
BigInteger p) {
    var Result = x % p;
    return Result.Sign >= 0 ? Result : Result + p;
}
}

13. References

13.1. Normative References

[draft-hallambaker-mesh-architecture]

[draft-hallambaker-mesh-dare]

[draft-hallambaker-mesh-security]

[draft-hallambaker-web-service-discovery]


13.2. Informative References

[draft-hallambaker-mesh-developer]

[draft-hallambaker-mesh-trust]


[Shamir79] "[Reference Not Found!]".


13.3. URIs


Author’s Address

Phillip Hallam-Baker

Email: phill@hallambaker.com
Clarification of Enrollment over Secure Transport (EST): transfer encodings and ASN.1
draft-richardson-lamps-rfc7030est-clarify-02

Abstract

This document updates RFC7030: Enrollment over Secure Transport (EST) to resolve some errata that was reported, and which has proven to have interoperability when RFC7030 has been extended.

This document deprecates the specification of "Content-Transfer-Encoding" headers for EST endpoints, providing a way to do this in an upward compatible way. This document additional defines a GRASP discovery mechanism for EST endpoints, and specifies requirements for them.

Finally, this document fixes some syntactical errors in ASN.1 that was presented.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 20, 2019.
1. Introduction

[RFC7030] defines the Enrollment over Secure Transport, or EST protocol.

This specification defines a number of HTTP endpoints for certificate enrollment and management. The details of the transaction were defined in terms of MIME headers as defined in [RFC2045], rather than in terms of the HTTP protocol as defined in [RFC2616] and [RFC7230].

[RFC2616] and later [RFC7231] Appendix A.5 has text specifically deprecating Content-Transfer-Encoding.

[RFC7030] calls it out this header incorrectly.
[I-D.ietf-anima-bootstrapping-keyinfra] extends [RFC7030], adding new functionality, and interop testing of the protocol has revealed that unusual processing called out in [RFC7030] causes confusion.

EST is currently specified as part of IEC 62351, and is widely used in Government, Utilities and Financial markets today.

Changes to [RFC7030] to bring it inline with typical HTTP processing would change the on-wire protocol in a way that is not backwards compatible. Reports from the field suggest that many implementations do not send the Content-Transfer-Encoding, and many of them ignore it.

This document therefore revises [RFC7030] to reflect the field reality, deprecating the extraneous field.

This document deals with errata numbers [errata4384], [errata5107], and [errata5108].

2. Terminology

The abbreviation "CTE" is used to denote the Content-Transfer-Encoding header, and the abbreviation "CTE-base64" is used to denote a request or response whose Content-Transfer-Encoding header contains the value "base64".

3. Requirements Language

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant STuPiD implementations.

4. Changes to EST endpoint processing

The [RFC7030] sections 4.1.3 (CA Certificates Response, /cacerts), 4.3.1/4.3.2 (Full CMC, /fullcmc), 4.4.2 (Server-Side Key Generation, /serverkeygen), and 4.5.2 (CSR Attributes, /csrsattrs) specify the use of base64 encoding with a Content-Transfer-Encoding for requests and response.

This document updates [RFC7030] to require the POST request and payload response of all endpoints in to be [RFC4648] section 4 Base64 encoded DER. This format is to be used regardless of whether there is any Content-Transfer-Encoding header, and any value in that header is to be ignored.
5. Clarification of ASN.1 for Certificate Attribute set.
   errata 4384.

6. Clarification of error messages for certificate enrollment operations
   errata 5108.

7. Privacy Considerations
   This document does not disclose any additional identifies to either active or passive observer would see with [RFC7030].

8. Security Considerations
   This document clarifies an existing security mechanism. An option is introduced to the security mechanism using an implicit negotiation.

9. IANA Considerations
   This document does not require any registrations.

10. Acknowledgements
   This work was supported by the Huawei Technologies.

11. References

11.1. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra]


11.2. Informative References


Authors’ Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Abstract

The applications of EVPN-based solutions ([RFC7432] and [RFC8365]) have become pervasive in Data Center, Service Provider, and Enterprise segments. It is being used for fabric overlays and inter-site connectivity in the Data Center market segment, for Layer-2, Layer-3, and IRB VPN services in the Service Provider market segment, and for fabric overlay and WAN connectivity in Enterprise networks. For Data Center and Enterprise applications, there is a need to provide inter-site and WAN connectivity over public Internet in a secured manner with same level of privacy, integrity, and authentication for tenant’s traffic as IPsec tunneling using IKEv2. This document presents a solution where BGP point-to-multipoint signaling is leveraged for key and policy exchange among PE devices to create private pair-wise IPsec Security Associations without IKEv2 point-to-point signaling or any other direct peer-to-peer session establishment messages.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction .................................................. 6
2 Requirements .................................................. 7
  2.1 Tenant’s Layer-2 and Layer-3 data & control traffic .......... 7
  2.2 Tenant’s Unicast & Multicast Data Protection ............... 7
  2.3 P2MP Signaling for SA setup and Maintenance ............... 7
  2.4 Granularity of Security Association Tunnels ............... 7
  2.5 Support for Policy and DH-Group List ..................... 8
3 BGP Component .................................................. 8
  3.1 Zero Touch Bring-up (ZTB) .................................. 8
  3.2 Configuration Management .................................. 8
  3.3 Orchestration ............................................... 9
  3.4 Signaling .................................................. 9
4 Solution Description ........................................... 9
  4.1 Inheritance of Security Policies ........................... 10
  4.2 Distribution of Public Keys and Policies .................. 11
    4.2.1 Minimal DIM ........................................... 11
  4.2.2 Multiple Policies ....................................... 12
    4.2.2.1 Multiple DH-groups ................................. 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2.2</td>
<td>Multiple or Single ESP SA policies</td>
<td>12</td>
</tr>
<tr>
<td>4.3</td>
<td>Initial IPsec SAs Generation</td>
<td>13</td>
</tr>
<tr>
<td>4.4</td>
<td>Re-Keying</td>
<td>13</td>
</tr>
<tr>
<td>4.5</td>
<td>IPsec Databases</td>
<td>13</td>
</tr>
<tr>
<td>5.1</td>
<td>Standard ESP Encapsulation</td>
<td>14</td>
</tr>
<tr>
<td>5.2</td>
<td>ESP Encapsulation within UDP packet</td>
<td>15</td>
</tr>
<tr>
<td>6.1</td>
<td>The Base (Minimal Set) DIM Sub-TLV</td>
<td>16</td>
</tr>
<tr>
<td>6.2</td>
<td>Key Exchange Sub-TLV</td>
<td>17</td>
</tr>
<tr>
<td>6.3</td>
<td>ESP SA Proposals Sub-TLV</td>
<td>18</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Transform Substructure</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Applicability to other VPN types</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Acknowledgements</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Security Considerations</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>IANA Considerations</td>
<td>20</td>
</tr>
<tr>
<td>11.1</td>
<td>Normative References</td>
<td>20</td>
</tr>
<tr>
<td>11.2</td>
<td>Informative References</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Authors’ Addresses</td>
<td>22</td>
</tr>
</tbody>
</table>

Terminology

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

AC: Attachment Circuit.

ARP: Address Resolution Protocol.

BD: Broadcast Domain. As per [RFC7432], an EVI consists of a single or multiple BDs. In case of VLAN-bundle and VLAN-based service models (see [RFC7432]), a BD is equivalent to an EVI. In case of VLAN-aware bundle service model, an EVI contains multiple BDs. Also, in this document, BD and subnet are equivalent terms.

BD Route Target: refers to the Broadcast Domain assigned Route Target [RFC4364]. In case of VLAN-aware bundle service model, all the BD instances in the MAC-VRF share the same Route Target.

BT: Bridge Table. The instantiation of a BD in a MAC-VRF, as per [RFC7432].

DGW: Data Center Gateway.
Ethernet A-D route: Ethernet Auto-Discovery (A-D) route, as per [RFC7432].

Ethernet NVO tunnel: refers to Network Virtualization Overlay tunnels with Ethernet payload. Examples of this type of tunnels are VXLAN or GENEVE.

EVI: EVPN Instance spanning the NVE/PE devices that are participating on that EVPN, as per [RFC7432].

EVPN: Ethernet Virtual Private Networks, as per [RFC7432].

GRE: Generic Routing Encapsulation.

GW IP: Gateway IP Address.

IPL: IP Prefix Length.

IP NVO tunnel: it refers to Network Virtualization Overlay tunnels with IP payload (no MAC header in the payload).

IP-VRF: A VPN Routing and Forwarding table for IP routes on an NVE/PE. The IP routes could be populated by EVPN and IP-VPN address families. An IP-VRF is also an instantiation of a layer 3 VPN in an NVE/PE.

IRB: Integrated Routing and Bridging interface. It connects an IP-VRF to a BD (or subnet).

MAC-VRF: A Virtual Routing and Forwarding table for Media Access Control (MAC) addresses on an NVE/PE, as per [RFC7432]. A MAC-VRF is also an instantiation of an EVI in an NVE/PE.

ML: MAC address length.

ND: Neighbor Discovery Protocol.

NVE: Network Virtualization Edge.

GENEVE: Generic Network Virtualization Encapsulation, [GENEVE].

NVO: Network Virtualization Overlays.

RT-2: EVPN route type 2, i.e., MAC/IP advertisement route, as defined in [RFC7432].

RT-5: EVPN route type 5, i.e., IP Prefix route. As defined in Section 3 of [EVPN-PREFIX].
SBD: Supplementary Broadcast Domain. A BD that does not have any ACs, only IRB interfaces, and it is used to provide connectivity among all the IP-VRFs of the tenant. The SBD is only required in IP-VRF-to-IP-VRF use-cases (see Section 4.4.).

SN: Subnet.

TS: Tenant System.

VA: Virtual Appliance.

VNI: Virtual Network Identifier. As in [RFC8365], the term is used as a representation of a 24-bit NVO instance identifier, with the understanding that VNI will refer to a VXLAN Network Identifier in VXLAN, or Virtual Network Identifier in GENEVE, etc. unless it is stated otherwise.

VTEP: VXLAN Termination End Point, as in [RFC7348].

VXLAN: Virtual Extensible LAN, as in [RFC7348].

This document also assumes familiarity with the terminology of [RFC7432], [RFC8365] and [RFC7365].
1 Introduction

The applications of EVPN-based solutions have become pervasive in Data Center, Service Provider, and Enterprise segments. It is being used for fabric overlays and inter-site connectivity in the Data Center market segment, for Layer-2, Layer-3, and IRB VPN services in the Service Provider market segment, and for fabric overlay and WAN connectivity in the Enterprise networks. For Data Center and Enterprise applications, there is a need to provide inter-site and WAN connectivity over public Internet in a secured manner with the same level of privacy, integrity, and authentication for tenant’s traffic as used in IPsec tunneling using IKEv2. This document presents a solution where BGP point-to-multipoint signaling is leveraged for key and policy exchange among PE devices to create private pair-wise IPsec Security Associations without IKEv2 point-to-point signaling or any other direct peer-to-peer session establishment messages.

EVPN uses BGP as control-plane protocol for distribution of information needed for discovery of PEs participating in a VPN, discovery of PEs participating in a redundancy group, customer MAC addresses and IP prefixes/addresses, aliasing information, tunnel encapsulation types, multicast tunnel types, multicast group memberships, and other info. The advantages of using BGP control plane in EVPN are well understood including the following:

1) A full mesh of BGP sessions among PE devices can be avoided by using Route Reflector (RR) where a PE only needs to setup a single BGP session between itself and the RR as opposed to setting up N BGP sessions to N other remote PEs; therefore, reducing number of BGP sessions from O(N^2) to O(N) in the network. Furthermore, RR hierarchy can be leveraged to scale the number of BGP routes on the RR.

2) MP-BGP route filtering and constrained route distribution can be leveraged to ensure that the control-plane traffic for a given VPN is only distributed to the PEs participating in that VPN.

For setting up point-to-point security association (i.e., IPsec tunnel) between a pair of EVPN PEs, it is important to leverage BGP point-to-multipoint singling architecture using the RR along with its route filtering and constrain mechanisms to achieve the performance and the scale needed for large number of security associations (IPsec tunnels) along with their frequent re-keying requirements. Using BGP signaling along with the RR (instead of peer-to-peer protocol such as IKEv2) reduces number of message exchanges needed for SAs establishment and maintenance from O(N^2) to O(N) in the network.
2 Requirements

The requirements for secured EVPN are captured in the following subsections.

2.1 Tenant’s Layer-2 and Layer-3 data & control traffic

Tenant’s layer-2 and layer-3 data and control traffic must be protected by IPsec cryptographic methods. This implies not only tenant’s data traffic must be protected by IPsec but also tenant’s control and routing information that are advertised in BGP must also be protected by IPsec. This in turn implies that BGP session must be protected by IPsec.

2.2 Tenant’s Unicast & Multicast Data Protection

Tenant’s layer-2 and layer-3 unicast traffic must be protected by IPsec. In addition to that, tenant’s layer-2 broadcast, unknown unicast, and multicast traffic as well as tenant’s layer-3 multicast traffic must be protected by IPsec when ingress replication or assisted replication are used. The use of BGP P2MP signaling for setting up P2MP SAs in P2MP multicast tunnels is for future study.

2.3 P2MP Signaling for SA setup and Maintenance

BGP P2MP signaling must be used for IPsec SAs setup and maintenance. The BGP signaling must follow P2MP signaling framework per [CONTROLLER-IKE] for IPsec SAs setup and maintenance in order to reduce the number of message exchanges from $O(N^2)$ to $O(N)$ among the participant PE devices.

2.4 Granularity of Security Association Tunnels

The solution must support the setup and maintenance of IPsec SAs at the following level of granularities:

1) Per PE: A single IPsec tunnel between a pair of PEs to be used for all tenants’ traffic supported by the pair of PEs.

2) Per tenant: A single IPsec tunnel per tenant per pair of PEs. For example, if there are 1000 tenants supported on a pair of PEs, then 1000 IPsec tunnels are required between that pair of PEs.

3) Per subnet: A single IPsec tunnel per subnet (e.g., per VLAN/EVI) of a tenant on a pair of PEs.

4) Per IP address: A single IPsec tunnel per pair of IP addresses of a tenant on a pair of PEs.
5) Per MAC address: A single IPsec tunnel per pair of MAC addresses of a tenant on a pair of PEs.

6) Per Attachment Circuit: A single IPsec tunnel per pair of Attachment Circuits between a pair of PEs.

2.5 Support for Policy and DH-Group List

The solution must support a single policy and DH group for all SAs as well as supporting multiple policies and DH groups among the SAs.

3 BGP Component

The architecture that encompasses device-to-controller trust model, has several components among which is the signaling component. Secure EVPN Signaling, as defined in this document, is the BGP signaling component of the overall Architecture. We will briefly describe this Architecture here to further facilitate understanding how Secure EVPN fits into the overall architecture. The Architecture describes the components needed to create BGP based SD-WANs and how these components work together. Our intention is to list these components here along with their brief description and to describe this Architecture in details in a separate document where to specify the details for other parts of this architecture besides the BGP signaling component which is described in this document.

The Architecture consists of four components. These components are Zero Touch Bring-up, Configuration Management, Orchestration, and Signaling. In addition to these components, secure communications must be provided between the edge nodes and all servers/devices providing the architecture components.

3.1 Zero Touch Bring-up (ZTB)

The first component is a zero touch capability that allows an edge device to find and join its SD-WAN with little to no assistance other than power and network connectivity. The goal is to use existing work in this area. The requirements are that an edge device can locate its ZTB server/component of its SD-WAN controller in a secure manner and to proceed to receive its configuration.

3.2 Configuration Management

After an edge device joins its SD-WAN, it needs to be configured.
Configuration covers all device configuration, not just the configuration related to Secure EVPN. The previous Zero Touch Bring-up component will have directed the edge device, either directly or indirectly, to its configuration server/component. One example of a configuration server is the I2NSF Controller. After a device has been configured, it can engage in the next two components. Configuration may include updates over time and is not a one-time-only component.

3.3 Orchestration

This component is optional. It allows for more dynamic updates of configuration and statistics information. Orchestration can be more dynamic than configuration.

3.4 Signaling

Signaling is the component described in this document. The functionality of a Route Reflector is well understood. Here we describe the signaling component of BGP SD-WAN Architecture and the BGP extension/signaling for IPsec key management and policy.

4 Solution Description

This solution uses BGP P2MP signaling where an originating PE only send a message to the Route Reflector (RR) and then the RR reflects that message to the interested recipient PEs. The framework for such signaling is described in [CONTROLLER-IKE] and it is referred to as device-to-controller trust model. This trust model is significantly different than the traditional peer-to-peer trust model where a P2P signaling protocol such as IKEv2 [RFC7296] is used in which the PE devices directly authenticate each other and agree upon security policy and keying material to protect communications between themselves. The device-to-controller trust model leverages P2MP signaling via the controller (e.g., the RR) to achieve much better scale and performance for establishment and maintenance of large number of pair-wise Security Associations (SAs) among the PEs.

This device-to-controller trust model first secures the control channel between each device and the controller using peer-to-peer protocol such as IKEv2 [RFC7296] to establish P2P SAs between each PE and the RR. It then uses this secured control channel for P2MP signaling in establishment of P2P SAs between each pair of PE devices.
Each PE advertises to other PEs via the RR the information needed in establishment of pair-wise SAs between itself and every other remote PEs. These pieces of information are sent as Sub-TLVs of IPSec tunnel type in BGP Tunnel Encapsulation attribute. These Sub-TLVs are detailed in section 5 and are based on the DIM message components from [CONTROLLER-IKE] and the IKEv2 specification [RFC7296]. The IPSec tunnel TLVs along with its Sub-TLVs are sent along with the BGP route (NLRI) for a given level of granularity.

If only a single SA is required per pair of PE devices to multiplex user traffic for all tenants, then IPSec tunnel TLV is advertised along with IPv4 or IPv6 NLRI representing loopback address of the originating PE. It should be noted that this is not a VPN route but rather an IPv4 or IPv6 route.

If a SA is required per tenant between a pair of PE devices, then IPSec tunnel TLV can be advertised along with EVPN IMET route representing the tenant or can be advertised along with a new EVPN route representing the tenant.

If a SA is required per tenant’s subnet (e.g., per VLAN) between a pair of PE devices, then IPSec tunnel TLV is advertised along with EVPN IMET route.

If a SA is required between a pair of tenant’s devices represented by a pair of IP addresses, then IPSec tunnel TLV is advertised along with EVPN IP Prefix Advertisement Route or EVPN MAC/IP Advertisement route.

If a SA is required between a pair of tenant’s devices represented by a pair of MAC addresses, then IPSec tunnel TLV is advertised along with EVPN MAC/IP Advertisement route.

If a SA is required between a pair of Attachment Circuits (ACs) on two PE devices (where an AC can be represented by <VLAN, port>), then IPSec tunnel TLV is advertised along with EVPN Ethernet AD route.

4.1 Inheritance of Security Policies

Operationally, it is easy to configure a security association between a pair of PEs using BGP signaling. This is the default security association that is used for traffic that flows between peers. However, in the event more finer granularity of security association is desired on the traffic flows, it is possible to set up SAs between a pair of tenants, a pair of subnets within a tenant, a pair of IPs between a subnet, and a pair of MACs between a subnet using the appropriate EVPN routes as described above. In the event, there are no security TLVs associated with an EVPN route, there is a strict
order in the manner security associations are inherited for such a route. This results in an EVPN route inheriting the security associations of the parent in a hierarchical fashion. For example, traffic between an IP pair is protected using security TLVs announced along with the EVPN IP Prefix Advertisement Route or EVPN MAC/IP Advertisement route as a first choice. If such TLVs are missing with the associated route, then one checks to see if the subnets the IPs are associated with has security TLVs with the EVPN IMET route. If they are present, those associations are used in securing the traffic. In the absence of them, the peer security associations are used. The order in which security associations are inherited are from the granular to the coarser, namely, IP/MAC associated TLVs with the EVPN route being the first preference, and the subnet, the tenant, and the peer associations preferred in that fashion.

It should be noted that when a security association is made it is possible for it to be re-used by a large number of traffic flows. For example, a tenant security association may be associated with a number of child subnet routes. Clearly it is mandatory to keep a tenant security association alive, if there are one or more subnet routes that want to use that association. Logically, the security associations between a pair of entities creates a single secure tunnel. It is thus possible to classify the incoming traffic in the most granular sense (IP/MAC, subnet, tenant, peer) to a particular secure tunnel that falls within its route hierarchy. The policy that is applied to such traffic is independent from its use of an existing or a new secure tunnel. It is clear that since any number of classified traffic flows can use a security association, such a security association will not be torn down, if at least there is one policy using such a secure tunnel.

4.2 Distribution of Public Keys and Policies

One of the requirements for this solution is to support a single DH group and a single policy for all SAs as well as to support multiple DH groups and policies among the SAs. The following subsections describe what pieces of information (what Sub-TLVs) are needed to be exchanged to support a single DH group and a single policy versus multiple DH groups and multiple policies.

4.2.1 Minimal DIM

For SA establishment, at the minimum, a PE needs to advertise to other PEs, its DIM values as specified in [CONTROLLER-IKE]. These include:

<table>
<thead>
<tr>
<th>ID</th>
<th>Tunnel ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Nonce</td>
</tr>
</tbody>
</table>
RC  Rekey Counter
I  Indication of initial policy distribution
KE  DH public value.

When this minimal set of DIM values is sent, then it is assumed that all peer PEs share the same policy for which DH group to use, as well as which IPSec SA policy to employ. Section 5.1 defines the Minimal DIM sub-TLV as part of IPsec tunnel TLV in BGP Tunnel Encapsulation Attribute.

4.2.2 Multiple Policies

There can be scenarios for which there is a need to have multiple policy options. This can happen when there is a need for policy change and smooth migration among all PE devices to the new policy is required. It can also happen if different PE devices have different capabilities within the network. In these scenarios, PE devices need to be able to choose the correct policy to use for each other. This multi-policy scheme is described in section 6 of [CONTROLLER-IKE]. In order to support this multi-policy feature, a PE device MUST distribute a policy list. This list consists of multiple distinct policies in order of preference, where the first policy is the most preferred one. The receiving PE selects the policy by taking the received list (starting with the first policy) and comparing that against its own list and choosing the first one found in common. If there is no match, this indicates a configuration error and the PEs MUST NOT establish new SAs until a message is received that does produce a match.

4.2.2.1 Multiple DH-groups

It can be the case that not all peers use the same DH group. When multiple DH groups are supported, the peer may include multiple KE Sub-TLVs. The order of the KE Sub-TLVs determines the preference. The preference and selection methods are specified in Section 6 of [CONTROLLER-IKE].

4.2.2.2 Multiple or Single ESP SA policies

In order to specify an ESP SA Policy, a DIM may include one or more SA Sub-TLVs. When all peers are configured by a controller with the same ESP SA policy, they MAY leave the SA out of the DIM. This minimizes messaging when group configuration is static and known. However, it may also be desirable to include the SA. If a single SA is included, the peer is indicating what ESP SA policy it uses, but is not willing to negotiate. If multiple SA Sub-TLVs are included, the peer is indicating that it is willing to negotiate. The order of
the SA Sub-TLVs determines the preference. The preference and selection methods are specified in Section 6 of [CONTROLLER-IKE].

4.3 Initial IPsec SAs Generation

The procedure for generation of initial IPsec SAs is described in section 3 of [CONTROLLER-IKE]. This section gives a summary of it in context of BGP signaling. When a PE device first comes up and wants to setup an IPsec SA between itself and each of the interested remote PEs, it generates a DH pair along for each [what word here? "tennant"?] using an algorithm defined in the IKEv2 Diffie-Hellman Group Transform IDs [IKEv2-IANA]. The originating PE distributes the DH public value along with the other values in the DIM (using IPsec Tunnel TLV in Tunnel Encapsulation Attribute) to other remote PEs via the RR. Each receiving PE uses this DH public number and the corresponding nonce in creation of IPsec SA pair to the originating PE — i.e., an outbound SA and an inbound SA. The detail procedures are described in section 5.2 of [CONTROLLER-IKE].

4.4 Re-Keying

A PE can initiate re-keying at any time due to local time or volume based policy or due to the result of cipher counter nearing its final value. The rekey process is performed individually for each remote PE. If rekeying is performed with multiple PEs simultaneously, then the decision process and rules described in this rekey are performed independently for each PE. Section 4 of [CONTROLLER-IKE] describes this rekeying process in details and gives examples for a single IPsec device (e.g., a single PE) rekey versus multiple PE devices rekey simultaneously.

4.5 IPsec Databases

The Peer Authorization Database (PAD), the Security Policy Database (SPD), and the Security Association Database (SAD) all need to be setup as defined in the IPsec Security Architecture [RFC4301]. Section 5 of [CONTROLLER-IKE] gives a summary description of how these databases are setup for the controller-based model where key is exchanged via P2MP signaling via the controller (i.e., the RR) and the policy can be either signaled via the RR (in case of multiple policies) or configured by the management station (in case of single policy).

5 Encapsulation
Vast majority of Encapsulation for Network Virtualization Overlay (NVO) networks in deployment are based on UDP/IP with UDP destination port ID indicating the type of NVO encapsulation (e.g., VxLAN, GPE, GENEVE, GUE) and UDP source port ID representing flow entropy for load-balancing of the traffic within the fabric based on n-tuple that includes UDP header. When encrypting NVO encapsulated packets using IP Encapsulating Security Payload (ESP), the following two options can be used: a) adding a UDP header before ESP header (e.g., UDP header in clear) and b) no UDP header before ESP header (e.g., standard ESP encapsulation). The following subsection describe these encapsulation in further details.

5.1 Standard ESP Encapsulation

When standard IP Encapsulating Security Payload (ESP) is used (without outer UDP header) for encryption of NVO packets, it is used in transport mode as depicted below. When such encapsulation is used, for BGP signaling, the Tunnel Type of Tunnel Encapsulation TLV is set to ESP-Transport and the Tunnel Type of Encapsulation Extended Community is set to NVO encapsulation type (e.g., VxLAN, GENEVE, GPE, etc.). This implies that the customer packets are first encapsulated using NVO encapsulation type and then it is further encapsulated & encrypted using ESP-Transport mode.
5.2 ESP Encapsulation within UDP packet

In scenarios where NAT traversal is required ([RFC3948]) or where load balancing using UDP header is required, then ESP encapsulation within UDP packet as depicted in the following figure is used. The ESP for NVO applications is in transport mode. The outer UDP header (before the ESP header) has its source port set to flow entropy and its destination port set to 4500 (indicating ESP header follows). A non-zero SPI value in ESP header implies that this is a data packet (i.e., it is not an IKE packet). The Next Protocol field in the ESP trailer indicates what follows the ESP header, is a UDP header. This inner UDP header has a destination port ID that identifies NVO encapsulation type (e.g., VxLAN). Optimization of this packet format where only a single UDP header is used (only the outer UDP header) is for future study.

When such encapsulation is used, for BGP signaling, the Tunnel Type of Tunnel Encapsulation TLV is set to ESP-in-UDP-Transport and the Tunnel Type of Encapsulation Extended Community is set to NVO.
encapsulation type (e.g., VxLAN, GENEVE, GPE, etc.). This implies that the customer packets are first encapsulated using NVO encapsulation type and then it is further encapsulated & encrypted using ESP-in-UDP with Transport mode.

[RFC3948]

---

**Figure 4: VxLAN Encapsulation within ESP Within UDP**

6 BGP Encoding

This document defines two new Tunnel Types along with its associated sub-TLVs for The Tunnel Encapsulation Attribute [TUNNEL-ENCAP]. These tunnel types correspond to ESP-Transport and ESP-in-UDP-Transport as described in section 4. The following sub-TLVs apply to both tunnel types unless stated otherwise.

6.1 The Base (Minimal Set) DIM Sub-TLV
The Base DIM is described in 3.2.1. One and only one Base DIM may be sent in the IPSec Tunnel TLV.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   ID Length   |       Nonce Length            |I|   Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Rekey                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Counter                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Originator ID + (Tenant ID) + (Subnet ID) + (Tenant Address) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Nonce Data                                                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: The Base DIM Sub-TLV

ID Length (16 bits) is the length of the Originator ID + (Tenant ID) + (Subnet ID) + (Tenant Address) in bytes.

Nonce Length (8 bits) is the length of the Nonce Data in bytes.

I (1 bit) is the initial contact flag from [CONTROLLER-IKE]

Flags (7 bits) are reserved and MUST be set to zero on transmit and ignored on receipt.

The Rekey Counter is a 64 bit rekey counter as specified in [CONTROLLER-IKE]

The Originator ID + (Tenant ID) + (Subnet ID) + (Tenant Address) is the tunnel identifier and uniquely identifies the tunnel. Depending on the granularity of the tunnel, the fields in () may not be used - i.e., for a tunnel at the PE level of granularity, only Originator ID is required.

The Nonce Data is the nonce described in [CONTROLLER-IKE]. Its length is a multiple of 32 bits. Nonce lengths should be chosen to meet minimum requirements described in IKEv2 [RFC7296].

6.2 Key Exchange Sub-TLV
The KE Sub-TLV is described in 3.2.1 and 3.2.2.1. A KE is always required. One or more KE Sub-TLVs may be included in the IPSec Tunnel TLV.

Diffie-Hellman Group Num 916 bits) identifies the Diffie-Hellman group in the Key Exchange Data was computed. Diffie-Hellman group numbers are discussed in IKEv2 [RFC7296] Appendix B and [RFC5114].

The Key Exchange payload is constructed by copying one’s Diffie-Hellman public value into the "Key Exchange Data" portion of the payload. The length of the Diffie-Hellman public value is described for MOPD groups in [RFC7296] and for ECP groups in [RFC4753].

6.3 ESP SA Proposals Sub-TLV

The SA Sub-TLV is described in 3.2.2.2. Zero or more SA Sub-TLVs may be included in the IPSec Tunnel TLV.

Num Transforms is the number of transforms included.

Reserved is not used and MUST be set to zero on transmit and MUST be ignored on receipt.
6.3.1 Transform Substructure

The Transform Attr Length is the length of the Transform Attributes field.

The Transform Type is from Section 3.3.2 of [RFC7296] and [IKEV2IANA]. Only the values ENCR, INTEG, and ESN are allowed.

The Transform ID specifies the transform identification value from [IKEV2IANA].

Reserved is unused and MUST be zero on transmit and MUST be ignored on receipt.

The Transform Attributes are taken directly from 3.3.5 of [RFC7296].

7 Applicability to other VPN types

Although P2MP BGP signaling for establishment and maintenance of SAs among PE devices is described in this document in context of EVPN, there is no reason why it cannot be extended to other VPN technologies such as IP-VPN [RFC4364], VPLS [RFC4761] & [RFC4762], and MVPN [RFC6513] & [RFC6514] with ingress replication. The reason EVPN has been chosen is because of its pervasiveness in DC, SP, and Enterprise applications and because of its ability to support SA establishment at different granularity levels such as: per PE, Per tenant, per subnet, per Ethernet Segment, per IP address, and per MAC. For other VPN technology types, a much smaller granularity levels can be supported. For example for VPLS, only the granularity of per PE and per subnet can be supported. For per-PE granularity level, the mechanism is the same among all the VPN technologies as IPsec tunnel type (and its associated TLV and sub-TLVs) are sent along with the PE’s loopback IPv4 (or IPv6) address. For VPLS, if
per-subnet (per bridge domain) granularity level needs to be supported, then the IPsec tunnel type and TLV are sent along with VPLS AD route.

The following table lists what level of granularity can be supported by a given VPN technology and with what BGP route.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>EVPN</th>
<th>IP-VPN</th>
<th>MVPN</th>
<th>VPLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>per tenant</td>
<td>IMET (or new)</td>
<td>lpbk (or new)</td>
<td>I-PMSI</td>
<td>N/A</td>
</tr>
<tr>
<td>per subnet</td>
<td>IMET</td>
<td>N/A</td>
<td>N/A</td>
<td>VPLS AD</td>
</tr>
<tr>
<td>per IP</td>
<td>EVPN RT2/RT5</td>
<td>VPN IP rt</td>
<td>*,G or S,G</td>
<td>N/A</td>
</tr>
<tr>
<td>per MAC</td>
<td>EVPN RT2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

8 Acknowledgements

9 Security Considerations

10 IANA Considerations

A new transitive extended community Type of 0x06 and Sub-Type of TBD for EVPN Attachment Circuit Extended Community needs to be allocated by IANA.

10 References

11.1 Normative References


[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC2119
11.2 Informative References


6513, February 2012.


Authors' Addresses

Ali Sajassi
Cisco
Email: sajassi@cisco.com

Ayan Banerjee
Cisco
Email: ayabaner@cisco.com

Samir Thoria
Cisco
Email: sthoria@cisco.com

David Carrel
Cisco
Email: carrel@cisco.com

Brian Weis
Individual

Sajassi et al. Expires January 8, 2020
Email: bew.stds@gmail.com

John Drake
Juniper
Email: jdrake@juniper.net