

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

D. Carrel
Cisco Systems
B. Weis
Independent
March 11, 2019

IPsec Key Exchange using a Controller
draft-carrel-ipsecme-controller-ike-01

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.

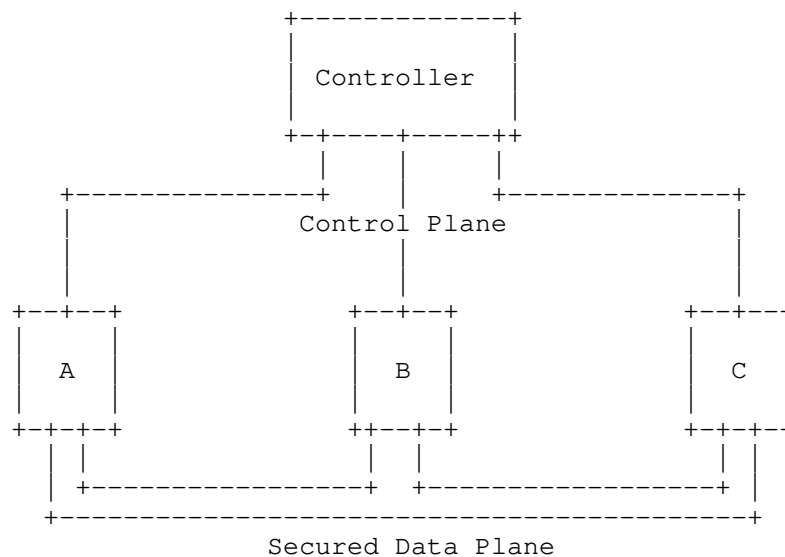


Figure 1: Controller-based Secured Communications

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 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.

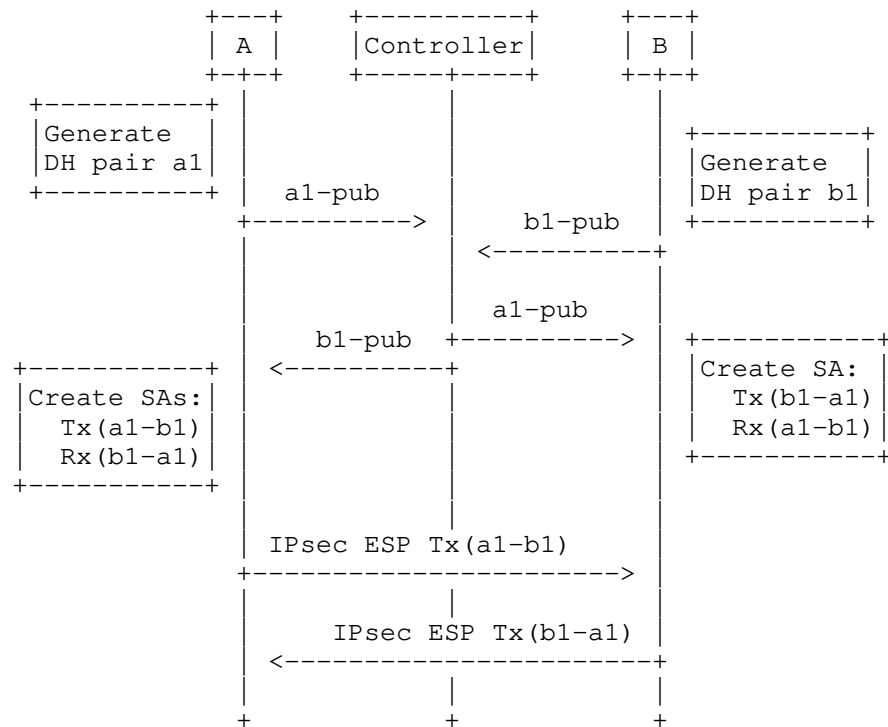


Figure 2: Generation of Initial IPsec SAs between two peers

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.

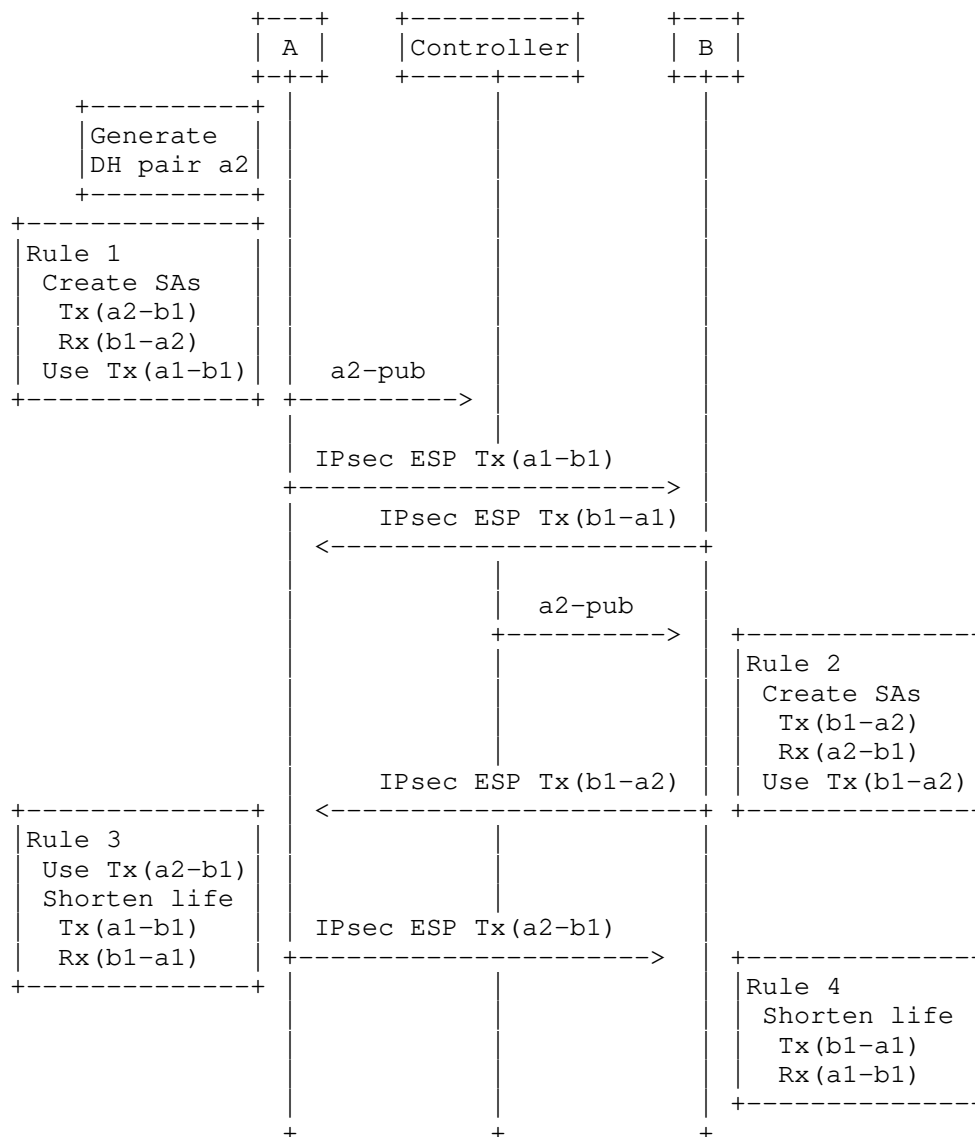


Figure 3: Single IPsec Device Rekey Protocol Flow

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). 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.

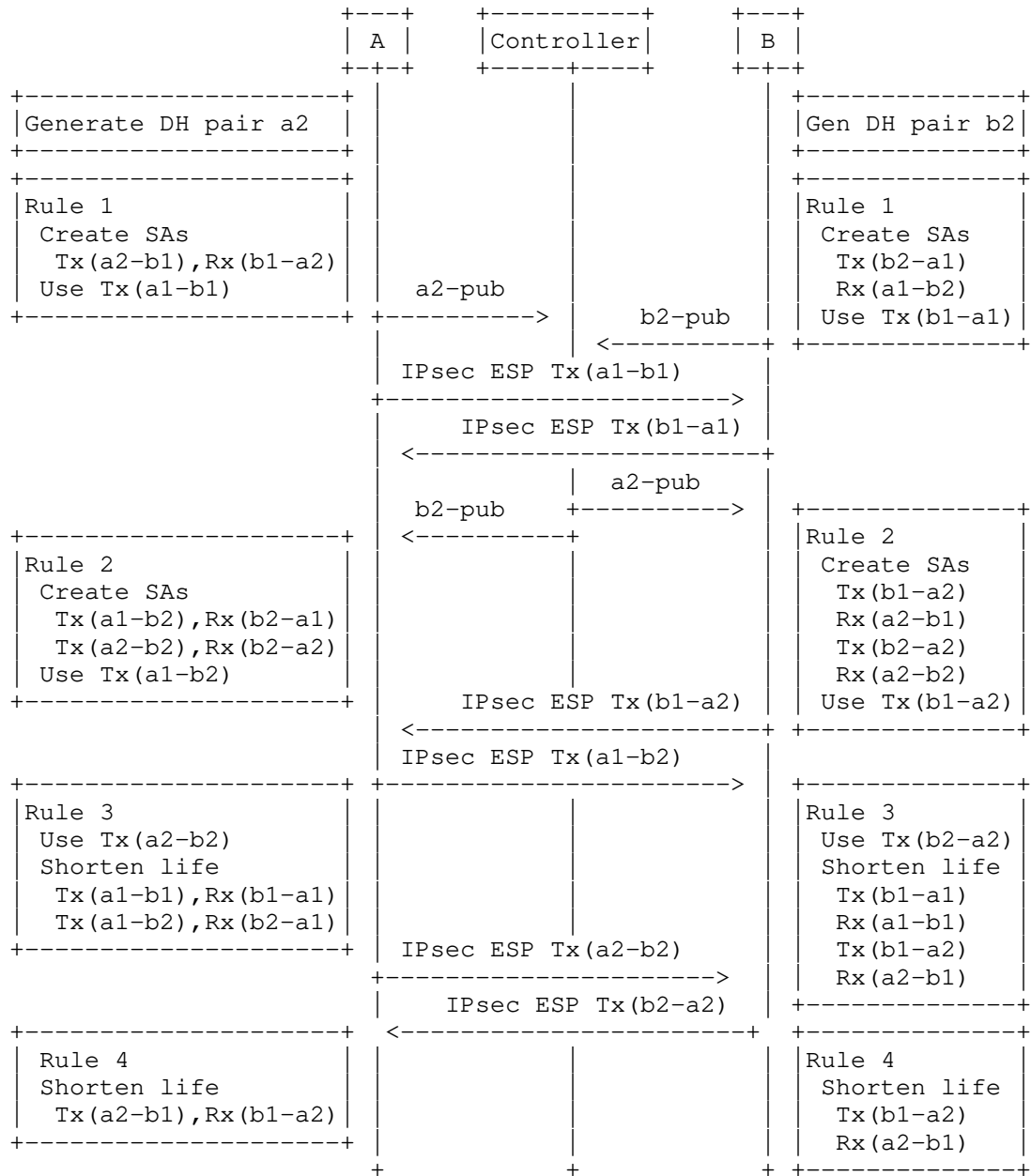


Figure 4: Simultaneous IPsec Device Rekey Protocol Flow

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 A 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 follow 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 is 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 shortens 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.

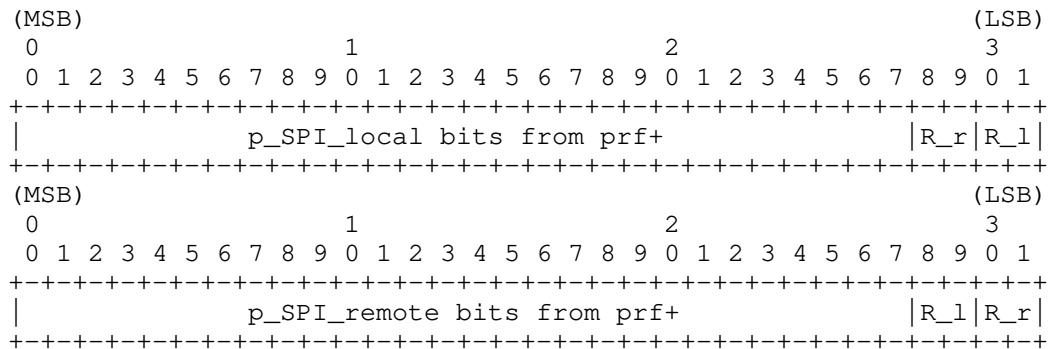
First the SPIs are taken from the prf+ function as 32 bit values and assigned based on which peer is taking the role of initiator and which is taking the role of responder. The p_SPI_i is taken by the device providing Ni, where p_SPI_r is taken by the other device.

$$\{p_SPI_i \mid p_SPI_r\} = \text{prf+}(Ni \mid Nr, \text{"SPI generation"})$$

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`.



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]:

$$\text{SKEYSEED} = \text{prf}(\text{Ni} \parallel \text{Nr}, g^{\text{ir}})$$

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]).

$$\text{SK_d} = \text{prf+} (\text{SKEYSEED}, \text{Ni} \mid \text{Nr} \mid \text{SPIi} \mid \text{SPIr})$$
$$\text{KEYMAT} = \text{prf+}(\text{SK_d}, \text{Ni} \mid \text{Nr})$$

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:

$$\text{KEYMAT} = \text{prf+} (\text{SKEYSEED}, \text{Ni} \mid \text{Nr} \mid \text{SPIi} \mid \text{SPIr})$$

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:

- o DH public number (used for key computation)
- o Nonce (used for key computation and SPI generation)
- o Peer identity (used to identify a peer in the PAD)
- o An Indication whether this is the initial distributed policy
- o 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:

- o An IPsec policy or policies
- o 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

[IKEV2-IANA]

IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters", February 2016, <<http://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-8>>.

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

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.

- [RFC6989] Sheffer, Y. and S. Fluhrer, "Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 6989, DOI 10.17487/RFC6989, July 2013, <<https://www.rfc-editor.org/info/rfc6989>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.ietf-i2nsf-sdn-ipsec-flow-protection]
Lopez, R. and G. Lopez-Millan, "Software-Defined Networking (SDN)-based IPsec Flow Protection", draft-ietf-i2nsf-sdn-ipsec-flow-protection-03 (work in progress), October 2018.
- [I-D.ietf-ipsecme-qr-ikev2]
Fluhrer, S., McGrew, D., Kampanakis, P., and V. Smyslov, "Postquantum Preshared Keys for IKEv2", draft-ietf-ipsecme-qr-ikev2-07 (work in progress), January 2019.
- [I-D.sajassi-bess-secure-evpn]
Sajassi, A., Banerjee, A., Thoria, S., Carrel, D., and B. Weis, "Secure EVPN", draft-sajassi-bess-secure-evpn-00 (work in progress), October 2018.
- [REUSE] Menezes, A. and B. Ustaoglu, "On Reusing Ephemeral Keys In Diffie-Hellman Key Agreement Protocols", December 2008, <<http://www.cacr.math.uwaterloo.ca/techreports/2008/cacr2008-24.pdf>>.
- [RFC7018] Manral, V. and S. Hanna, "Auto-Discovery VPN Problem Statement and Requirements", RFC 7018, DOI 10.17487/RFC7018, September 2013, <<https://www.rfc-editor.org/info/rfc7018>>.
- [RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <<https://www.rfc-editor.org/info/rfc7426>>.

- [RFC8192] Hares, S., Lopez, D., Zarny, M., Jacquenet, C., Kumar, R., and J. Jeong, "Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases", RFC 8192, DOI 10.17487/RFC8192, July 2017, <<https://www.rfc-editor.org/info/rfc8192>>.
- [RFC8329] Lopez, D., Lopez, E., Dunbar, L., Strassner, J., and R. Kumar, "Framework for Interface to Network Security Functions", RFC 8329, DOI 10.17487/RFC8329, February 2018, <<https://www.rfc-editor.org/info/rfc8329>>.

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.

ID, [N(INITIAL_CONTACT),] 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:

ID, [N(INITIAL_CONTACT),] SA, KE, Ni

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.

$$\{p_SPI_i | p_SPI_r\} = \text{prf}+(Ni \mid Nr, \text{"SPI generation"} \mid SPNi \mid SPNr)$$

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.

ID, [N(INITIAL_CONTACT),] [SA,] KE, [KE,] Ni

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.

Trust Model	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.
-------------	--

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 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

Network Working Group
Internet Draft
Intended status: Informational
Expires: Dec 2019

L. Dunbar
J. Guichard
Huawei
Ali Sajassi
Cisco
J. Drake
Juniper
Ayan Barnerjee
D. Carrel
Cisco

July 8, 2019

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/lid-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

3.4. Scenario #3: SDWAN WAN ports to MPLS VPN and the Internet	12
4. Provisioning Model.....	14
4.1. Client Service Provisioning Model.....	14
4.2. WAN Ports Provisioning Model.....	14
4.2.1. Why BGP as Control Plane for SDWAN WAN Ports Registration?.....	15
5. SDWAN Traffic Forwarding Walk Through.....	16
5.1. SDWAN Network Startup Procedures.....	16
5.2. Packet Walk-Through for Scenario #1.....	16
5.3. Packet Walk-Through for Scenario #2.....	17
5.3.1. SDWAN node WAN Ports Properties Registration.....	19
5.3.2. Controller Facilitated IPsec SA & NAT management....	19
5.3.3. BGP Based SDWAN client routes.....	21
5.4. Packet Walk-Through for Scenario #3.....	22
6. Manageability Considerations.....	23
7. Security Considerations.....	23
8. IANA Considerations.....	23
9. References.....	23
9.1. Normative References.....	23
9.2. Informative References.....	24
10. Acknowledgments.....	25

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).

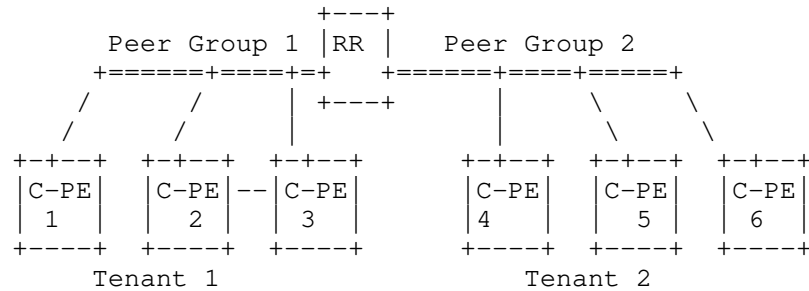


Figure 1: Peer Groups managed by Local Controller

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.

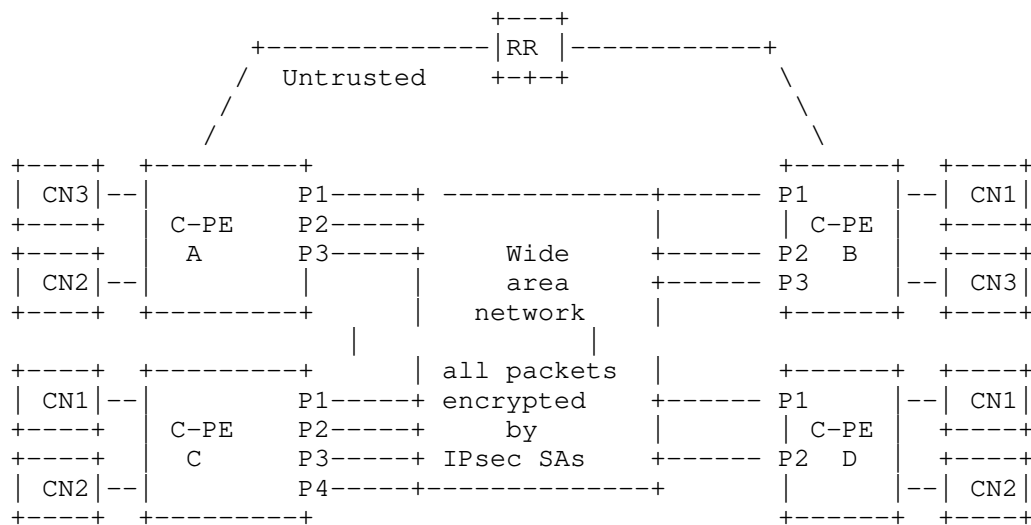
3.2. Scenarios #1: Homogeneous WAN

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.



CN: Client Networks, which is same as Tenant Networks used by NVo3

Figure 1: Homogeneous SDWAN

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).

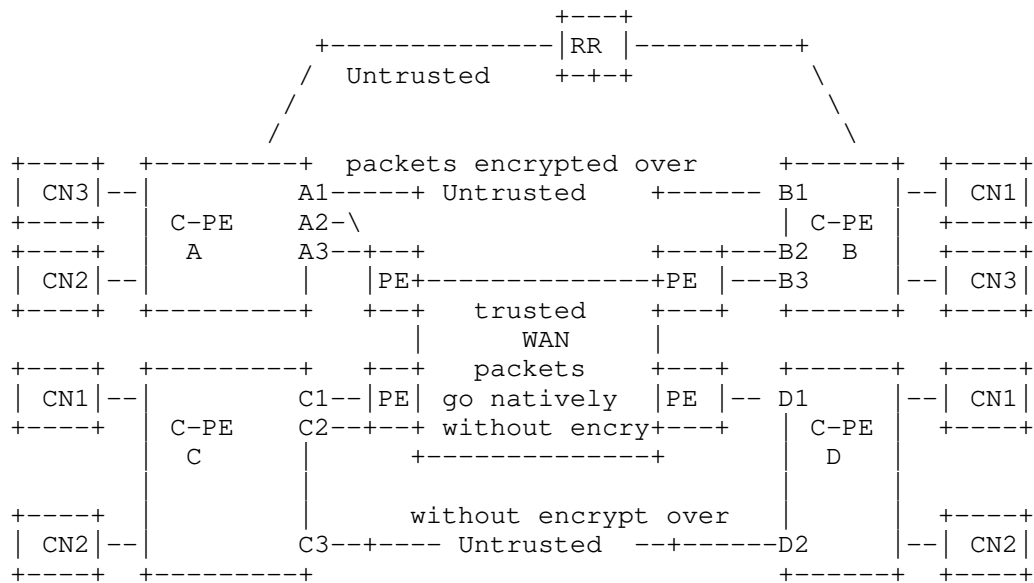
In this scenario, the SDWAN edge nodes' egress WAN ports are all IP/Ethernet based, either egress to PEs of the VPNs or to the Internet. Even if the VPN is a MPLS network, the VPN's PEs have IP/Ethernet connections to the SDWAN edge (C-PEs). Throughout this document, this scenario is also called CPE 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 those 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.



CN: Client Network

Figure 2: Hybrid SDWAN

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.

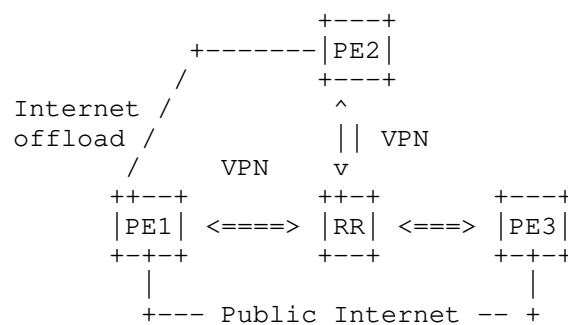


Figure 3: Additional Internet paths added to the VPN

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:

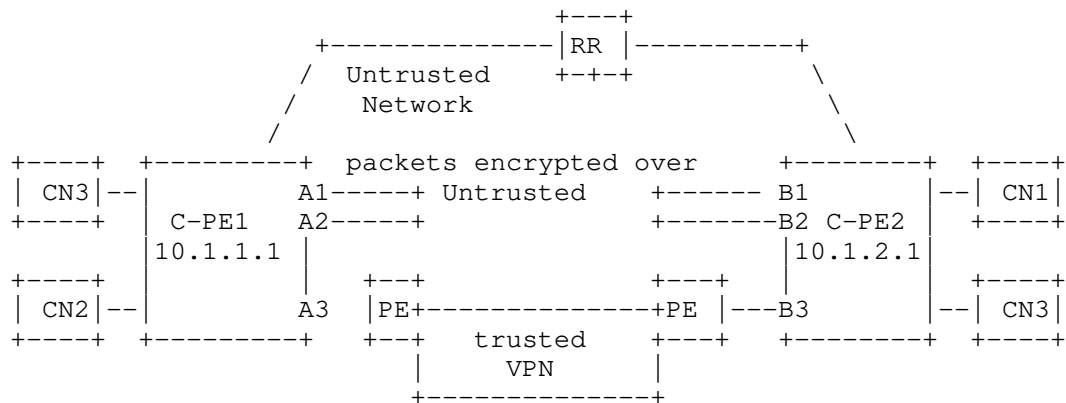


Figure 5: SDWAN Scenario #2

- SDWAN node's WAN ports property registration to the SDWAN Network Controller (BGP RR).
 - o This is used to inform the SDWAN controller of all the underlay networks to which the C-PE is connected.
 - o RR is responsible for propagating the C-PE WAN ports properties to authorized peers.
- Controller Facilitated IPsec SA management and NAT information distribution
 - o Used by the SDWAN controller to facilitate or manage the IPsec configurations and peer authentications for all IPsec SAs terminated at the SDWAN nodes.
 - o 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.
 - o 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.

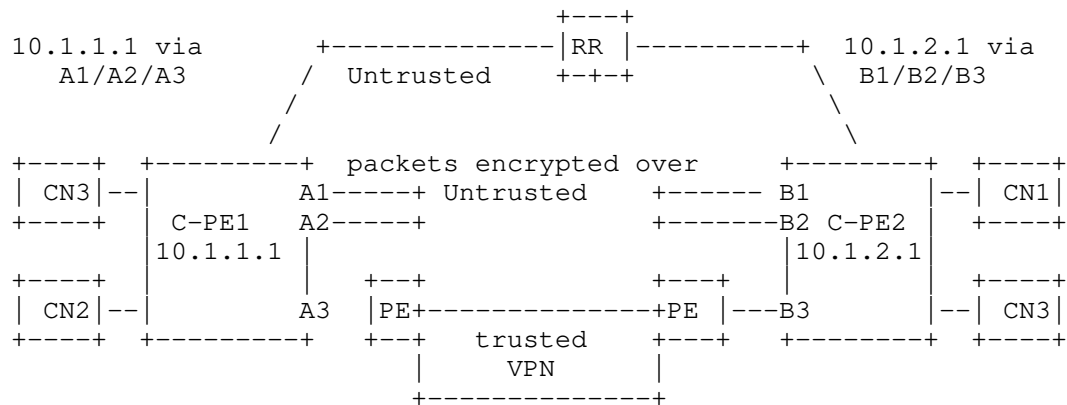


Figure 6: SDWAN Scenario #2 WAN Ports Registration

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 app 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 with 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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4364] E. rosen, Y. Rekhter, "BGP/MPLS IP Virtual Private networks (VPNs)", Feb 2006.
- [RFC7296] C. Kaufman, et al, "Internet Key Exchange Protocol Version 2 (IKEv2)", Oct 2014.
- [RFC7432] A. Sajassi, et al, "BGP MPLS-Based Ethernet VPN", Feb 2015.
- [RFC8365] A. Sajassi, et al, "A network Virtualization Overlay Solution Using Ethernet VPN (EVPN)", March 2018.

9.2. Informative References

- [RFC8192] S. Hares, et al, "Interface to Network Security Functions (I2NSF) Problem Statement and Use Cases", July 2017
- [RFC5521] P. Mohapatra, E. Rosen, "The BGP Encapsulation Subsequent Address Family Identifier (SAFI) and the BGP Tunnel Encapsulation Attribute", April 2009.
- [BGP-SDWAN-Port] L. Dunbar, H. Wang, W. Hao, "BGP Extension for SDWAN Overlay Networks", draft-dunbar-idr-bgp-sdwan-overlay-ext-03, work-in-progress, Nov 2018.
- [Net2Cloud-Gap] L. Dunbar, A. Malis, C. Jacquenet, "Gap Analysis of Interconnecting Underlay with Cloud Overlay", draft-dm-net2cloud-gap-analysis-02, work in progress, Oct. 2018.
- [VPN-over-Internet] E. Rosen, "Provide Secure Layer L3VPNs over Public Infrastructure", draft-rosen-bess-secure-l3vpn-00, work-in-progress, July 2018
- [DMVPN] Dynamic Multi-point VPN:
<https://www.cisco.com/c/en/us/products/security/dynamic-multipoint-vpn-dmvpn/index.html>
- [DSVPN] Dynamic Smart VPN:
<http://forum.huawei.com/enterprise/en/thread-390771-1-1.html>
- [SECURE-EVPN] A. Sajassi, et al, "Secure EVPN", draft-sajassi-bess-secure-evpn-01, Work-in-progress, March 2019.
- [SECURE-L3VPN] E. Rosen, R. Bonica, "Secure Layer L3VPN over Public Infrastructure", draft-rosen-bess-secure-l3vpn-00, Work-in-progress, June 2018.
- [ITU-T-X1036] ITU-T Recommendation X.1036, "Framework for creation, storage, distribution and enforcement of policies for network security", Nov 2007.

[Net2Cloud-Problem] L. Dunbar and A. Malis, "Seamless Interconnect Underlay to Cloud Overlay Problem Statement", draft-dm-net2cloud-problem-statement-02, June 2018

[Net2Cloud-gap] L. Dunbar, A. Malis, and C. Jacquenet, "Gap Analysis of Interconnecting Underlay with Cloud Overlay", draft-dm-net2cloud-gap-analysis-02, work-in-progress, Aug 2018.

[Tunnel-Encap] E. Rosen, et al "The BGP Tunnel Encapsulation Attribute", draft-ietf-idr-tunnel-encaps-10, Aug 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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
ThresholdSecrets.com
20 April 2022

Mathematical Mesh 3.0 Part I: Architecture Guide
draft-hallambaker-mesh-architecture-20

Abstract

The Mathematical Mesh is a Threshold Key Infrastructure that makes computers easier to use by making them more secure. Application of threshold cryptography to key generation and use enables users to make use of public key cryptography across multiple devices with minimal impact on the user experience.

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

[Note to Readers] Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

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

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 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.

Table of Contents

1. Introduction	3
2. Definitions	7
2.1. Related Specifications	7
2.2. Defined Terms	7
2.3. Requirements Language	7
2.4. Implementation Status	7
3. Requirements	8
3.1. The Device Management Challenge	10
3.2. Exchange of trusted credentials.	11
3.3. Application configuration management	11
3.4. Mesh Service Provider	12
3.5. The Mesh as platform	12
3.6. Security	12
3.7. Enterprise Deployment	13
4. User Experience	13
4.1. Creating a Mesh Account	13
4.1.1. Encrypting and Decrypting files.	15
4.1.2. Catalogs	15
4.2. Adding devices	16
4.2.1. Direct Connection	17
4.2.2. PIN Connection	18
4.2.3. Making use of the new device	19
4.2.4. Applications	20
4.2.5. Threshold Key Devices	21
4.3. Mesh Messaging	22
4.3.1. Contact exchange	23
4.3.2. Confirmation service	25
4.4. Encryption Groups	27
4.5. Deleting Devices	29
4.6. Escrow and Recovery	30
4.7. Future Directions	30
4.7.1. Device Disconnection	30
4.7.2. Service Transition	31
4.7.3. Threshold User Account	31
4.7.4. Threshold Group Administration	32

4.7.5.	Synchronous Messaging	32
4.7.6.	Social Media	33
5.	Mesh Cryptography	33
5.1.	Best Practice by Default	34
5.2.	Multi-Level Security	35
5.3.	Threshold Decryption	35
5.4.	Threshold Key Generation	36
5.5.	Threshold Signature	36
5.6.	Data At Rest Encryption	36
5.6.1.	DARE Envelope	37
5.6.2.	Dare Sequence	37
5.7.	Uniform Data Fingerprints.	38
5.7.1.	Friendly Names	38
5.7.2.	Encrypted Authenticated Resource Locators	39
5.7.3.	Secure Internet Names	40
5.8.	Personal Key Escrow	40
6.	Mesh Architecture	42
6.1.	Actors	42
6.1.1.	Account	43
6.1.2.	Device	44
6.1.3.	Service	45
6.2.	Stores	46
6.2.1.	Catalogs	46
6.2.2.	Spools	47
6.3.	Mesh Service Protocol	48
6.3.1.	Protocol Interactions	49
6.4.	The Access Catalog	49
6.5.	Mesh Messaging Protocol	50
6.6.	Using the Mesh with Applications	52
6.6.1.	Future Applications	53
7.	Security Considerations	53
8.	IANA Considerations	53
9.	Acknowledgements	53
10.	Normative References	54
11.	Informative References	56

1. Introduction

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

In 1977, Public Key cryptography laid out a powerful proposition: If Alice and Bob have private keys on their devices and each knows the public key of the other, Alice and Bob can communicate with confidentiality and integrity. The realization of this proposition at Internet scale was vested in a technology called Public Key

Infrastructure (PKI) whose principal function is to provide a trustworthy means by which Alice and Bob can discover each other's public key.

Yet despite the power of PKI, Internet security remains a work in progress. While PKI has proved an effective means of authenticating services to users, attempts to apply PKI to the equally important task of authenticating users to services and securing data at rest have been confined to the margins. One critical reason for that failure is that Public Key Infrastructure has only provided effective tools for managing public keys. If we are to achieve comprehensive Internet security, we must provide every user with the ability to manage private keys across their devices with zero effort on their part.

Threshold cryptography is a sub-field of public key cryptography that defines operations on cryptographic keys, including operations on private keys. Threshold cryptography allows Key generation and key use operations may be split between multiple devices. These tools make zero effort management of private keys practical.

The Mesh is a TKI that addresses the three principal 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

A core principle of the design of the Mesh is autonomy. That is each user has full control over their digital environment and is their own source of authority. They may choose to delegate that

authority to another to act on their behalf (i.e. a Trusted Third Party) and they may choose to surrender parts of that authority to others (e.g. an employer) without surrendering their autonomy. Delegation of authority is always for limited times and limited purposes.

Thus, from the user's point of view, the Mesh is divided into two parts: The part of the Mesh that belongs to them and everything else. As with the Internet, which is a network of networks, a Mesh of Meshes has certain properties that are similar to those of its constituent parts and some that are quite different.

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 [draft-hallambaker-mesh-udf]. 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 [draft-hallambaker-mesh-dare]. Describes the cryptographic message and append-only sequence formats used in Mesh applications and the Mesh Service protocol.
- IV. Schema Reference [draft-hallambaker-mesh-schema]. Describes the syntax and semantics of Mesh Profiles, Catalog and Spool Entries and Mesh Messages and their use in Mesh Applications.
- V. Protocol Reference [draft-hallambaker-mesh-protocol]. Describes the Mesh Service Protocol.
- VI. Reliable User Datagram [draft-hallambaker-mesh-rud]. Describes the Mesh presentation and transport layer.
- VII Mesh Callsign Service [draft-hallambaker-mesh-callsign]. Describes

es the Mesh Callsign Service that supports mapping of Mesh callsigns to the corresponding Mesh Service Provider.

VIII. Security Considerations [draft-hallambaker-mesh-security] Describes the recommended and required algorithm suites and the security considerations for the Mesh protocol suite.

IX. Cryptographic Algorithms [draft-hallambaker-mesh-security] 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.

The following documents describe technologies that are used in the Mesh but do not form part of the Mesh specification suite:

X. The Trust Mesh [draft-hallambaker-mesh-trust]. 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.

JSON-BCD Encoding [draft-hallambaker-jsonbcd]. 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 [draft-hallambaker-web-service-discovery]. Describes the means by which prefixed DNS SRV and TXT records are used to perform discovery of Web Services.

Threshold Modes in Elliptic Curves [draft-hallambaker-threshold]. Describes threshold key generation and key agreement operations for the Ed25519, Ed448, X25519 and X448 elliptic curves.

The following documents describe aspects of the Mesh Reference implementation:

Mesh Developer [draft-hallambaker-mesh-developer]. Describes the reference code distribution license terms, implementation status and currently supported functions.

Mesh Platform [draft-hallambaker-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 require support for the cryptographic algorithm suites relevant to 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 20-Apr-22 4:18:25 PM.
Out of 249 examples, 52 failed.

3. Requirements

The Mathematical Mesh (Mesh) is a Threshold Key Infrastructure that uses cryptography to make computers easier to use.

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, secure use of applications must become effortless. Rather than beginning the design process by imagining all the possible modes of attack and working out how to address these as best as possible with least inconvenience to the user, we must reverse the question and ask how much security can be provided without requiring any effort whatsoever from the user. This principle is called*Zero Effort Security*.

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 not compromise the user's autonomy. Delegation of 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. In 1949, Edward A. Murphy Jr. proposed his now eponymous law which states, 'Anything that can go wrong will go wrong'. We must now apply a similar principle to Internet security: 'Anything that can be made to go wrong is already being made to go wrong and will only get worse until something is done to stop it.'

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 to the design of the Mesh:

- * Cryptographic keys installed in a product during manufacture are only used for the limited purpose of putting that device under control of the user.
- * 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.
- * 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.
- * Mesh Services do not have access to the plaintext of any Mesh Messages or Mesh Catalog data except for the threshold catalog used by the service as the source of access control policy.
- * 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 every possible 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. It does not demand that we immediately address every issue with perfection or delay deployment of technologies that are capable of controlling many risks until we have achieved the control of every risk.

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. The Device Management Challenge

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. In these circumstances, it arguably sufficed to provision a user with a single key pair on the single device they were likely to use. Creating keys was a time-consuming business that might take several minutes, an architecture in which an end user might create and use hundreds of key pairs was beyond the capabilities of the available hardware.

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 were unencrypted. A cryptographic security control that impedes the user is a control that is not going to be used.

Each personal Mesh contains a device catalog in which the cryptographic credentials and device specific application configurations for each connected device are stored. A device granted administration rights can use the device catalog to configure each connected device with the precise cryptographic credentials it needs to perform its functions.

3.2. 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. Here there are two different challenges.

Developing an infrastructure that provides a mapping to a cryptographic key from a name that serves no other purpose than identifying the key is relatively easy. Developing an infrastructure that maps existing names with semantics that are already established is considerably harder.

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 provides an internal trust infrastructure that makes use of a `_direct trust_` model similar to that of PGP fingerprints to which external names may be mapped using whatever validation criteria users consider are appropriate to the purpose for which they intend to use them.

The principles of providing extended trust management in the Mesh are further described in [draft-hallambaker-mesh-trust].

3.3. Application configuration management

Configuration of cryptographic applications is typically an afterthought (or worse). 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.

3.4. Mesh Service Provider

As might be expected of a Key _Infrastructure_, use of the Mesh typically requires the use of an online service to provide an always present point of service through which devices which are not permanently connected can exchange messages. This online service is called a *Mesh Service Provider* (*MSP*).

An MSP performs similar functions to an SMTP mail provider but with one important distinction: MSPs do not issue or own Mesh accounts. A Mesh account is always created by and belongs to its user who can change their MSP at any time without changing their account. Further, a user MAY choose to act as their own MSP.

3.5. 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, the Mesh Catalogs used to maintain collections of device descriptions, bookmarks, credentials, etc. might be used in an electronic records infrastructure to maintain chain of custody of digital evidence.

3.6. Security

The Mesh is designed to provide the greatest practical level of security that does not detract from the user experience. The usual CIA triad is considered:

Confidentiality The confidentiality of user content should be protected at all times and against all unauthorized parties including their MSP.

Reasonable efforts should be taken to protect user data against traffic analysis and metadata attacks. It is not necessary to consider disclosure of this information to MSPs. Metadata must be shielded from external parties but controls to prevent traffic analysis may be left to implementers.

Integrity The design should consider unauthorized modification of data to be at least as serious as disclosure.

Availability The design should consider loss of data likely to be at least as serious as disclosure.

In addition to protecting the user's data, the Mesh is designed to protect the user's autonomy. While the use of any electronic device or service entails a degree of trust, the user should have the right to decide which devices and which service providers to trust and to have the practical ability to revoke that trust at any time they choose.

3.7. Enterprise Deployment

Development of PKI has traditionally focused on the needs of large enterprises. The Mesh is focused on the individual user. While this change of focus is in part a recognition of the need to reverse the traditional bias, it is also a recognition of the fact that we must understand the needs of the individual user before attempting to understand the additional needs of an enterprise IT department serving a large number of users.

4. 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 further details and additional examples.

For clarity and compactness of exposition, these use cases are illustrated using the command line tool `_meshman_`, a tool that makes the cryptographic operations explicit. This does not represent the ideal user experience in which Zero-effort security is achieved. Such a user experience requires that the Mesh operations be seamlessly integrated into the user's applications so that instead of using the meshman tool to encrypt or decrypt document, the word processor application itself would be extended to read and write documents encrypted in the DARE format.

4.1. Creating a Mesh Account

From the user's perspective, their personal Mesh consists of a collection of devices that communicate seamlessly and securely through a Mesh account serviced by an MSP.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 1: Alice's personal Mesh.

As with an email service provider, the user is only likely to be aware of their interactions with their MSP in the case of a service interruption. As far as the user is concerned the data is replicated across their devices automatically unless there is a problem.

While the term 'account' is used because it is the term a user is familiar with that most closely describes its functions, Mesh accounts are different from traditional Internet accounts in one important respect: In order to realize the principle of 'autonomy', Mesh accounts are created by and belong to the user and not the service provider. Should a serious problem occur, a user may opt to change their MSP. But unlike a changing an SMTP email provider, this change is made seamless and cost free.

Another important difference between the Mesh and SMTP is that all Mesh data is encrypted end to end. The MSP does not have access to any user content and does not have access to any user meta-data except that which is strictly necessary to service the account.

The only Mesh catalogs associated with a Mesh account that can be read by an MSP are the Access Catalog which serves as the basis for specifying and enforcing access control policy on the resources associated with the account and the Publications Catalog which is an index of encrypted data published through the account.

To create a Mesh account, the user need only specify the account name and the initial MSP:

The user specifies the initial account address to be used (alice@example.com). Use of this address is of course dependent on authorization by the Mesh Service Provider (example.com) and is likely to require authentication and possibly payment.

```
Alice> meshman account create alice@example.com
Account=alice@example.com
UDF=MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
```

The command returns the value of Alice's Mesh Account fingerprint . This value is used as a unique identifier that is cryptographically bound to the signature key used to authenticate the account profile.

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.

4.1.1. Encrypting and Decrypting files.

Having created an account, Alice can use it to encrypt files and decrypt them on the same machine.

Alice encrypts the text file plaintext.txt to create an encrypted version readable only by Alice:

```
Alice> meshman type plaintext.txt
This is a test
Alice> meshman dare encode plaintext.txt ciphertext.dare /encrypt ^
alice@example.com
Alice> meshman dare verify ciphertext.dare
File: ciphertext.dare
  Bytes: 16
  Encryption Algorithm: A256CBC
    Recipient: MDPR-FJVV-GK5Z-2LJA-LMYV-XSCH-HE2C
  Digest Algorithm: S512
  Payload Digest: 508336AAD1C39BD104DC6D7BC92DEBA0D2CF71351E28029F6
298F80EEF2DEFD19F707FC250DC6D89B1D0ADCF39D5DE89583A45DF026895403BBE99
259AB04B8D
```

Alice can recover the file at any time using the decryption command:

```
Alice> meshman dare decode ciphertext.dare plaintext1.txt
Alice> meshman type plaintext1.txt
This is a test
```

Although the encrypted file can be accessed by Alice with precisely the same ease as the plaintext version, the contents of the encrypted file are not readable by any other user of the machine unless Alice explicitly grants access. The encrypted file may be stored on a shared drive, cloud file system or removable storage without disclosing the contents.

While encrypting and decrypting files using a tool provides the desired functionality, it does not meet our objectives for usability. These capabilities should be integrated into applications or the platform itself.

4.1.2. Catalogs

Every Mesh account is created with a set of catalogs and spools. For example, the bookmarks catalog maintains a list of the user's Web bookmarks. The credentials catalog maintains a list of the user's usernames and passwords for the various network services they use. As with the file encryption example, these capabilities are clearly going to be most effective when incorporated into the user's

applications, (i.e. their Web browser).

Alice adds the username and password she uses to access her weather service account to her credentials catalog:

```
Alice> meshman password add ftp.example.com alicel password
alicel@ftp.example.com = [password]
```

```
Alice> meshman password add www.example.com alice@example.com ^
newpassword
alice@example.com@www.example.com = [newpassword]
```

As with all Mesh Catalogs, the catalog data is encrypted and cannot be accessed by any unauthorized party including the Mesh Service Provider.

If needed, she can retrieve the credentials from the catalog by specifying the network resource to which access is required:

```
Alice> meshman password get ftp.example.com
alicel@ftp.example.com = [password]
```

This capability provides a means of preventing one of the most common causes of enterprise password breach in which a system administrator encodes the access credentials for a service into a script used to access the service. A script containing a command to extract the credentials from a Mesh catalog will only work for a user authorized to access the credentials in the Mesh.

4.2. Adding devices

Computers have become ubiquitous and inexpensive. Most people living in affluent countries interact with several dozen computer systems every day. Every household appliance from the television to the coffee pot has become or is in the process of becoming a computer. It is this circumstance that has exposed the critical flaw in traditional PKI: The lack of practical means of managing private keys across multiple devices.

The Mesh allows users to connect all their devices together so that they may be considered part of a single entity whose component parts communicate and interact seamlessly and securely.

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. The Mesh allows the principle of least privilege when connecting a device granting precisely the set of capabilities required to perform its intended function.

Multiple connection mechanisms are specified, each of which provides strong mutual authentication. In each case, the connection request must be approved by a device provisioned with the Mesh administration privilege:

Direct The connection request is initiated on the device being requested and approved on the administration device. Authentication of the connection request is performed by comparing witness values presented on the connecting device and the administration device.

PIN A PIN code is generated on an administration device and passed to the connecting device out of band. The connecting device provides proof of knowledge of this PIN code when making the connection request allowing an administration device to approve the request automatically without further user interaction.

Dynamic QR This connection mechanism is a variation of the PIN connection mechanism in which administration device presents the PIN code value to the connecting device in the form of a QR code. This allows a connecting device with a camera to connect with minimal user effort.

Static QR This connection method is designed to support connection of constrained IoT devices that lack a camera or display capability but requires that the device be pre-provisioned during manufacture or distribution.

An administration device equipped with a camera reads a static QR code printed on the device that provides the information used to enable the administration device to establish a local network connection (e.g. WiFi, Bluetooth, strobe, IR) that can be used to complete the connection.

These connection mechanisms are described in detail in the Mesh Protocol Reference [draft-hallambaker-mesh-protocol].

4.2.1. Direct Connection

For example, Alice connects a second device using the direct connection mechanism:

The connection request is initiated on the device being connected:

```
Alice2> meshman device request alice@example.com
Device UDF = MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS
Witness value = 6WH6-YJIZ-OKA7-I54P-WZPA-VXTV-PHKJ
```

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 the request, granting the new device the messaging and web roles:

```
Alice> meshman device pending
MessageID: 6WH6-YJIZ-OKA7-I54P-WZPA-VXTV-PHKJ
Connection Request::
MessageID: 6WH6-YJIZ-OKA7-I54P-WZPA-VXTV-PHKJ
To: From:
Device: MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS
Witness: 6WH6-YJIZ-OKA7-I54P-WZPA-VXTV-PHKJ
Alice> meshman device accept 6WH6-YJIZ-OKA7-I54P-WZPA-VXTV-PHKJ ^
/message /web
```

Alice can now synchronize her newly connected device to her account:

```
Alice2> meshman device complete
Device UDF = MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS
Account = alice@example.com
Account UDF = MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
```

4.2.2. PIN Connection

Alice connects a third device using the PIN code connection mechanism:

The Alice begins the connection process by creating a one time use PIN authentication code on her administration device. The PIN creation request specifies the rights to be granted to the connecting device:

```
Alice> meshman account pin /threshold
PIN=ADFR-TEQU-3HJD-IRND-P4TS-CRBD-NI
(Expires=2022-04-21T16:17:50Z)
```

A connection request is made on the connecting device as before except that this time the PIN is specified. This time, only the 'threshold' right is granted.

```
Alice3> meshman device request alice@example.com /pin ^
ADFR-TEQU-3HJD-IRND-P4TS-CRBD-NI
Device UDF = MAA3-BQPZ-WWO4-7Q5B-P7AH-FY5C-ATMD
Witness value = HS22-VO5M-JAG4-RQT4-ROHX-PERK-YYCW
```

Since the connection request is pre-authenticated by the PIN, it is not necessary for Alice to review the connection request. The connection request is accepted automatically when the administration device is synchronized:

```
Alice> meshman message pending
MessageID: HS22-VO5M-JAG4-RQT4-ROHX-PERK-YYCW
  Connection Request::
    MessageID: HS22-VO5M-JAG4-RQT4-ROHX-PERK-YYCW
    To: From:
    Device: MAA3-BQPZ-WWO4-7Q5B-P7AH-FY5C-ATMD
    Witness: HS22-VO5M-JAG4-RQT4-ROHX-PERK-YYCW
MessageID: NDBB-CHFG-OWNI-2WWK-RJI2-KMF7-6AW7
  Confirmation Request::
    MessageID: NDBB-CHFG-OWNI-2WWK-RJI2-KMF7-6AW7
    To: alice@example.com From: console@example.com
    Text: start
Alice> meshman account sync /auto
```

Alice can now synchronize her newly connected device to her account:

```
Alice3> meshman device complete
  Device UDF = MAA3-BQPZ-WWO4-7Q5B-P7AH-FY5C-ATMD
  Account = alice@example.com
  Account UDF = MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
Alice3> meshman account sync
```

The Dynamic QRCode connection scheme uses exactly the same mechanism except that instead of the PIN being presented to Alice in the form of an alphanumeric string, the connection information is encoded as a URI and presented to the connecting device as a QR code.

The URI corresponding to the connection PIN is:

```
mcu://alice@example.com/ADFR-TEQU-3HJD-IRND-P4TS-CRBD-NI
```

4.2.3. Making use of the new device

Having connected a second device and granted it web access rights, Alice can use it to decrypt files and access her bookmark and password catalogs in exactly the same fashion as the first. If a password is changed on one device, all her connected devices receive the update.

For example, because Alice granted the device the Web role, she can now access her credential catalog and decrypt the file she encrypted on her first device from the new device:

```
Alice2> meshman password get ftp.example.com
alice1@ftp.example.com = [password]
```

```
Alice2> meshman dare decode ciphertext.dare plaintext2.txt
Alice2> meshman type plaintext2.txt
This is a test
```

By default, devices connected with the web access right can access all the catalogs connected to a personal Mesh. Devices MAY be connected with greater or lesser access rights according to their intended use. A coffee pot does not require access to the password catalog or the ability to send messages to other Mesh users. A software development station is likely to require the ability to sign commits to a source code repository.

Limiting the access rights granted to a device when it is connected mitigates the consequences of the device being lost, stolen or infected by malware before the compromise occurs. Disconnecting the device from the user's personal Mesh as described in a later section provides further mitigation.

4.2.4. Applications

Connected devices can also make use of connected applications for which they are granted the necessary rights.

Alice creates an SSH profile within her Mesh on the administrative device making the private key information available to devices she has connected to her Mesh with the 'web' access right.

```
Alice> meshman ssh create /web /threshold /id=ssh
UDF: MCXP-WQVY-RTKQ-ZU6P-VOM4-7U6K-FHXXH
```

She can extract the private key to configure her SSH clients:

```
Alice> meshman ssh get ssh /private /file=alice1_ssh_prv.pem
```

She can also extract her public key to configure her SSH server to allow access to the machine:

```
Alice> meshman ssh get ssh /file=alice1_ssh_pub.pem
```

Ideally these steps would be performed on Alice's behalf by an automated script that detects the applications Alice has installed on her device and performs the necessary configuration on her behalf.

The SSH keys created on one device are available to every device connected by the 'web' access right:


```
Alice2> meshman account sync  
Alice2> meshman ssh get ssh /private /file=alice2_ssh_prv.pem
```

At present the Mesh only supports an SSH configuration in which a single client key is shared across multiple devices. The Mesh is in principle capable of supporting more sophisticated configurations in which each device has its own individual client key. Consideration of these configuration modes is currently outside the scope of work for the Mesh and is probably more usefully considered as part of an effort to integrate Mesh functionality into the SSH system. Such an effort would also describe the means by which SSH server key fingerprints are recorded in the Mesh Contacts catalog.

Support for SMTP mail with OpenPGP and S/MIME end to end security is supported in a similar fashion.

4.2.5. Threshold Key Devices

As is to be expected of a Threshold Key Infrastructure, Mesh devices MAY be configured to use a threshold key share for decryption, the other key share being held by the Mesh service servicing the account.

Connecting a device with the threshold access right grants it the same range of functionality as the web access right for as long as it is connected to the user's personal Mesh. Instead of being provisioned with the account decryption key, the device is provisioned with a key share. To decrypt documents encrypted to the account key, the device requires the active participation of the service holding the other half of the shared key. This allows the service to prevent further use of the decryption capability by the device after it has been disconnected from that personal Mesh.

Provisioning devices with threshold access rights has the advantage of allowing greater control of the decryption capability at the cost of requiring an interaction with the network service for each decryption operation. There is thus a tradeoff between performance and security.

While the Mesh architecture permits any decryption key to be threshold shared, it is RECOMMENDED that implementations use this capability sparingly and develop mechanisms that avoid the need for a device to preform repeated threshold operations to decrypt the same data. For example, rather than decrypting every entry in the password catalog each time it is used, a device should encrypt a primary secret under the account threshold key that can be decrypted each time the device is activated and re-encrypt threshold encrypted data to that secret each time a threshold decryption is performed. This provides the same security properties with considerably less load on the service.

The current version of the Mesh protocols require that the administration device used to provision threshold keys to a device have access to the original key. As described in [draft-hallambaker-mesh-schema], this requirement will be lifted in a future edition of the protocol.

4.3. Mesh Messaging

The Mesh Messaging system is a push messaging system analogous to SMTP, but its purpose is limited to secure exchange of control plane messages. This leads to some important differences:

- * Every message is signed and end-to-end encrypted
- * The only communication pattern supported is a four-corner model in which users exchange messages through their respective MSPs.
- * Every message is subject to access control at the inbound and outbound MSP.
- * Message content is limited to 32KB.

This size restriction ensures that exchange of Mesh Messages does not impose an undue burden on the inbound and outbound MSP. While users can and do send much larger messages, 32KB should be more than sufficient to demonstrate to the recipient that the message should be accepted. It is not necessary for a sender to transfer multiple MB message before the receiver decides to refuse it. Connected devices may efficiently synchronize their message spools even over limited bandwidth connections. A short message is never blocked by a larger one.

For exchange of longer messages, a pull model is employed. A short Mesh message sent a message advising the recipient's client of the location from which the full content may be obtained. This approach has many benefits over the SMTP push model. There is no longer a

need for any limitation on message size. The same messaging platform can be used to send a short text message, a spreadsheet or raw video file archives spanning multiple TB.

Exchange of certain content types naturally leads to security concerns. These concerns are mitigated in the Mesh by performing access control on every message. When accepting Bob as a partner, Alice can choose the types of Mesh Message and the types of content she is willing to accept from him. Thus, Alice might be willing to accept a spreadsheet containing macro code from Bob but not from Carol or Mallet. And she might not want to accept anything at all from Susan because of past abuse.

While there are important technical differences between Mesh Messaging and SMTP, these are not visible to Alice or Bob except insofar as there is no restriction on message size other than the storage capacity of the machine they wish to receive the messages on, there is very little scope for messaging abuse and (unless the Mesh becomes ubiquitous) they can only use Mesh Messaging to communicate with other Mesh users. Thus, while Mesh messaging has been designed to enable replacement of SMTP in the long term, it is not currently a focus for the client implementations. Use of Mesh messaging is thus currently limited to support for applications built on the Mesh platform. One of those applications is the device connection protocol describe earlier. Another is the contact exchange protocol used to acquire contact information from other Mesh users.

4.3.1. Contact exchange

Besides management of private keys across devices, the biggest obstacle to effective use of existing security protocols such as SSH, OpenPGP and S/MIME is the difficulty of obtaining the authentic public keys of the counterparties.

The question of issue and validation of credentials is a complex and difficult one that does not have a single answer that is valid for every use case. For certain applications credentials issued by a Trusted Third Party are appropriate. For others, the Web of Trust proposed in OpenPGP provides a better fit to the requirements and constraints. These issues are discussed in [draft-hallambaker-mesh-trust].

Rather than imposing a single trust model for credential acquisition, the Mesh allows the use of whatever model is best for validating a credential for a particular use. It is unlikely Alice would have the same security concerns for communication with her employer, her friends, her bank, etc.

For many applications, Trust After First Use provides an adequate basis for credential acquisition.

Alice wants to exchange Mesh messages with Bob. Although Alice knows Bob's Mesh address (bob@example.com), she does not (yet) have permission to send any message to Bob excepting a request to exchange contact information.

Bob sends Alice a contact exchange request:

```
Bob> meshman contact request alice@example.com
Envelope ID: MCB5-SD6X-OUXX-JAQI-6ZBM-G4FS-TYAE
Message ID: NBBX-LUP5-63JW-AJ6G-5UFG-TYWA-Y6IY
Response ID: MAPM-XKGB-KZ4A-ZAST-JLFX-N4WD-RMIT
```

Alice checks his Mesh messages and approves Bob's request:

```
Alice> meshman account sync
Alice> meshman message pending
MessageID: NAUE-PMNN-4RNJ-E3AW-J4KO-QIVG-PRO6
  Contact Request::
    MessageID: NAUE-PMNN-4RNJ-E3AW-J4KO-QIVG-PRO6
    To: alice@example.com From: mallet@example.com
    PIN: ADCN-FXJI-Q27K-AI5W-O4P7-KU6H-AIGA
MessageID: NBBX-LUP5-63JW-AJ6G-5UFG-TYWA-Y6IY
  Contact Request::
    MessageID: NBBX-LUP5-63JW-AJ6G-5UFG-TYWA-Y6IY
    To: alice@example.com From: bob@example.com
    PIN: ADFZ-RDXJ-IICY-KX57-X6LH-ABQY-IBKQ
Alice> meshman message accept NBBX-LUP5-63JW-AJ6G-5UFG-TYWA-Y6IY
Alice> meshman contact list
Entry: MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
  Person MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
  Anchor MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
  Address alice@example.com

Entry: NA2N-NMA3-3OLA-B65Y-JSYR-WDIO-DGBE
  Person
  Anchor MDRS-IKMP-S6SZ-MR5M-GOIJ-SIHS-W5SJ
  Address bob@example.com
```

Bob can now collect Alice's contact:

```
Bob> meshman account sync /auto
Bob> meshman contact list
Entry: MDRS-IKMP-S6SZ-MR5M-GOIJ-SIHS-W5SJ
      Person MDRS-IKMP-S6SZ-MR5M-GOIJ-SIHS-W5SJ
      Anchor MDRS-IKMP-S6SZ-MR5M-GOIJ-SIHS-W5SJ
      Address bob@example.com

Entry: NAUR-M73V-JMIE-ZQV4-OIIT-ICJV-ZZSQ
      Person
      Anchor MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
      Address alice@example.com
```

At this point Alice and Bob can exchange Mesh messages of any type with seamless end to end security. Every Mesh message is signed and encrypted without exception. If Alice and Bob have used the Mesh to configure their email accounts for OpenPGP or S/MIME, they can use these to exchange end-to-end secure SMTP mail.

Alternatively, Bob might have opted to grant Alice only specific messaging access. Bob might choose to restrict synchronous messaging modalities such as instant messaging or voice that interrupt his workflow to specific colleagues. The fact that Alice wants to speak to Bob does not necessarily mean she is interested in what he might say in reply. Thus, messaging access need not be reciprocated.

As with device connection, multiple contact exchange methods are supported including the use of a QR code printed on a business card or presented on a mobile device. These methods are also described in [draft-hallambaker-mesh-protocol].

As a respected figure within the cryptographic community, Alice might employ a curation service for credential requests advising her that Bob's credentials appear to be in order while Mallet's are suspicious. Such services might be offered by her MSP or another provider. Alice might be willing to accept contact requests from members of professional associations she is a member of or who have attended certain conferences in her field. A variety of approaches might be followed for curation of other requests including Machine Learning approaches.

4.3.2. Confirmation service

The Mesh confirmation service is an improvement of traditional second factor authentication techniques offering offers far greater usability and security.

Instead of being asked to present a meaningless numeric code, Alice is presented a request from a named, authenticated source to confirm a specific action. Alice's response will be signed using a signature key that is unique to the particular confirmation device she uses, thus providing a non-repudiable record of her decision.

Alice attempts to log into a secure console in the control room. The secure console recognizes Alice but a second factor is required. The console issues a challenge to Alice at her registered account asking if she would like to log into the secure console:

```
Console> meshman message confirm alice@example.com start
Envelope ID: MAWU-5FMM-ZN6O-FXE5-TVC4-LO6I-RJ4D
Message ID: NDBB-CHFG-OWNI-2WWK-RJI2-KMF7-6AW7
Response ID: MBO5-GGWR-XOSQ-M6AO-WRP7-CJWT-V6LN
```

Alice checks her pending messages and accepts the request:

```
Alice> meshman message accept NDBB-CHFG-OWNI-2WWK-RJI2-KMF7-6AW7
```

The secure console verifies the response and grants access:

```
Console> meshman message status MBO5-GGWR-XOSQ-M6AO-WRP7-CJWT-V6LN
Accept
```

In an enterprise environment, tying the confirmation process to a specific source, a specific action and specific device allows for confirmation interactions to be used to implement business processes with attribution and thus accountability.

Using traditional second factor approaches, a system administrator presents their credentials to authenticate access to the machine at which point they can perform any action permitted by their current privileges. This typically includes modification of any access logs that might be kept. Using the confirmation approach the individual actions of the system administrator may be authenticated, traced and logged. If a user account is added to the system, it is known which administrator is responsible and the device that was used. This information may then be used if it becomes necessary to unwind the consequences of a breach or an insider threat.

4.4. Encryption Groups

As seen earlier, the Mesh allows encrypted files to be shared with other named users. While this capability is sufficient for simple messaging type use cases, decades of experience prove that it is inadequate to meet the needs of protecting data at rest. In the simple messaging case the list of recipients is known to the sender at the time a message is sent. In the general case the party encrypting the data cannot know the list of intended readers because that will change over time.

Even in the smallest organization, employees join and leave. A new employee must be granted access to all the information they need for their work. The access rights of a terminated employee must also terminate.

Traditional 'Digital Rights Management' product employ key management techniques originating in the field of copyright enforcement to control access to content by controlling disclosure of symmetric decryption keys. This provides the necessary flexibility to control access to the data but leaves the decryption keys vulnerable to a server breach. Such systems do not provide 'end-to-end' security in any useful sense.

Use of threshold techniques allows a threshold service to control decryption of the data without having the ability to decrypt. Sharing data through a Mesh group allows access to be controlled without loss of end-to-end encryption.

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

```
Alice> meshman group create groupw@example.com /web
Account=groupw@example.com
UDF=MASC-RP6Y-4AQ5-HYVY-IOMY-HSXT-FJU5
```

Alice encrypts a test file but she can't decrypt it because she hasn't added herself to the group yet.

```
Alice> meshman type grouptext.txt
The group secret handshake
Alice> meshman dare encode grouptext.txt groupsecret.dare /encrypt ^
groupw@example.com
Alice> meshman dare decode groupsecret.dare grouptext_alice.dare
ERROR - No decryption key is available
```

Alice adds herself to the group, now she can decrypt:

```
Alice> meshman group add groupw@example.com alice@example.com  
alice@example.com [MA3U-EQIV-5G6I-SK6H-2MSE-HEN3-SDVK]
```

```
Alice> meshman account sync /auto  
Alice> meshman dare decode groupsecret.dare grouptext_alice.dare  
Alice> meshman type grouptext_alice.dare  
The group secret handshake
```

At this point, Bob can't encrypt or decrypt messages because he doesn't know the public key and he isn't in the group. Alice could allow Bob to encrypt but not decrypt by sending him the group contact information without a decryption share. Instead she adds Bob to the group as a member:

```
Alice> meshman group add groupw@example.com bob@example.com  
bob@example.com [MA3U-EQIV-5G6I-SK6H-2MSE-HEN3-SDVK]
```

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:

```
Bob> meshman account sync /auto  
Bob> meshman dare decode groupsecret.dare grouptext_bob.dare  
Bob> meshman type grouptext_bob.dare  
The group secret handshake
```

Removing Bob from the group immediately withdraws his access.

```
Alice> meshman group delete groupw@example.com bob@example.com  
bob@example.com [MA3U-EQIV-5G6I-SK6H-2MSE-HEN3-SDVK]
```

Bob cannot decrypt files encrypted under the group key any more. But he still has access to the file grouptext_bob.dare he decrypted earlier.

```
Bob> meshman dare decode groupsecret.dare grouptext_bob2.dare  
ERROR - A cryptographic operation was refused.  
Bob> meshman type grouptext_bob.dare  
The group secret handshake
```

The threshold key service acts as a policy enforcement point and can impose additional accounting and authorization controls on the use of the decryption service.

For example, the threshold key service might be configured to alert a supervisor and/or deny decryption requests if a group member made an unusual volume of requests in a short period.

4.5. Deleting Devices

If a connected device is lost, stolen or simply broken, Alice can limit further use of the device by disconnecting it from her Mesh:

Alice disconnects the new device:

```
Alice> meshman device delete MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS
ERROR - Cannot access a closed file.
```

Disconnecting a device will always prevent the device receiving further services from the account service and thus the ability to receive encrypted catalog updates. But a device connected with direct key access rights (e.g. web) is still capable of decrypting documents encrypted under the account key unless the device application discovers that it has been disconnected and deletes the corresponding keys:

The device can no longer access the password catalog, but it can still decrypt files:

```
Alice2> meshman account sync
ERROR - The server returned an invalid response.
Alice2> meshman dare decode ciphertext.dare plaintext3.txt
Alice2> meshman type plaintext3.txt
This is a test
```

A device connected with the threshold access right loses the ability to decrypt immediately:

The third device was connected with threshold rights, it is disconnected in the same way as before.

```
Alice> meshman device delete MAA3-BQPZ-WW04-7Q5B-P7AH-FY5C-ATMD
ERROR - Cannot access a closed file.
```

The device can no longer access the password catalog or decrypt files:

```
Alice3> meshman account sync
ERROR - The server returned an invalid response.
Alice3> meshman dare decode ciphertext.dare plaintext3.txt
ERROR - A cryptographic operation was refused.
```

4.6. Escrow and Recovery

While disclosure of sensitive data might cause serious harm to its owner it is very rarely the case that the consequences of disclosure are greater than the consequences of loss. Thus, whenever static data is to be encrypted, the question of key recovery must be considered.

Alice decides to create a recovery key set. To do this, she specifies the number of key shares to be created and the number required for recovery:

```
Alice> meshman account escrow
Share: SAQO-MD74-F00I-VYSU-4IKS-IW6Q-WSPK-HB4C-L5S4-WOVL-KFL6-QQAW-X6
FF-FIP5-O5FA
Share: SAQQ-IHXW-BPO5-IYZY-LT4R-F53O-G3KW-IS64-6IKT-XQQD-OYHJ-XCTA-76
H6-RAQQ-7ZKA
Share: SARC-ELPP-5QPR-3ZA3-27OQ-DEYL-XEGC-KEBX-QTCK-YSK3-TLCU-5VFL-H6
KX-4YRE-QWFQ
```

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

```
>>>> Unfinished ArchitectureRecovery
```

```
Alice2> meshman account recover /verify
ERROR - No account specified
```

4.7. Future Directions

The Mesh is a Threshold Key Infrastructure and as with any infrastructure, it is designed as a platform to support as wide a range of future developments as possible. Selection of the initial feature set was determined by the need to achieve zero-effort security.

Further development of the Mesh will require careful consideration of costs and benefits. The range of security features that could be added is infinite, the range of features that should be added is small. Real world deployment and use is required before extensive addition of new features.

4.7.1. Device Disconnection

While 'single sign on' gets much attention in the Enterprise computing space, it is actually 'single sign off' that provides the real security value. Future enhancements of the Mesh will consider:

?A notification mechanism to allow a disconnected device to advise the user that it has completely disconnected from the user's Mesh and deleted relevant data.

?Use of trustworthy hardware to prevent use of confidential data accessed through a personal Mesh after the device is disconnected.

?Use of threshold timed key release to limit the period during which a device has access to confidential data.

4.7.2. Service Transition

Allowing users to transfer their accounts from one service provider to another without switching costs is an important goal of the Mesh project. Enabling such transfers is the principal purpose of the Callsign registry [draft-hallambaker-mesh-callsign].

Support for account transition is currently limited and untested. In theory a user MAY transfer their account from one service provider to another by opening an account at the new service provider and uploading the data provided that they have access to the profile primary secret and all threshold key shares granted to the devices are regenerated.

A different application of threshold cryptography and in particular the use of threshold techniques to generate key shares would allow a user to transfer their account without the need to reconstitute the primary secret.

4.7.3. Threshold User Account

At present the administration device has direct access to the administrator key and it is not possible to determine which device performed a particular administration action. Similarly, the administration device requires access to the full decryption keys to create threshold key shares for devices.

As with the service transition issue described earlier, meeting this particular requirement using threshold cryptography is straightforward in itself. The challenge lies in meeting the requirements simultaneously. Or to be more precise, the challenge lies in understanding what the precise requirements are.

A sketch has been developed of an approach addressing both sets of requirements as currently understood. Instead of using an additive key splitting approach for creating key shares, the additive approach is combined with a Shamir/Lagrange approach such that:

- * A single administration device can connect devices using key shares mediated by the current service.
- * Two administration devices can create the necessary signatures and key shares to transfer the account to a different service provider.
- * Two administration devices can add a third administration device with the same capabilities as the original two.

Achieving the last requirement makes use of the fact that Lagrange interpolation can be used to generate additional shares without reconstructing the original secret. The x coordinate of each share holder is determined from the fingerprint of the device profile signature key.

4.7.4. Threshold Group Administration

The current group encryption architecture requires that the group administrator have access to the decryption key. The administrator is thus able to decrypt any documents they chose and bypass the accounting restrictions of the service.

Further application of threshold techniques would allow the administrator role to be split between two or more parties, one of which might be a service enforcing additional controls.

4.7.5. Synchronous Messaging

Addition of a presence service capability to the MSP would allow Mesh Messaging to be used to support the full range of synchronous messaging services from text chat (e.g. xmpp) to video and VOIP. The chief security benefit to the end user for such a scheme would be that every communication request is mediated by access control. While it is impossible to absolutely guarantee that every possible form of abuse is prevented, stopping the organized crime ring that just called me purporting to be my credit card company is much more straightforward.

The main technical issue to be addressed to enable such a service is specifying a means of layering Mesh Messages direct over UDP transport. This is currently at the concept phase. While the precise means of layering audio and video formats onto a network connection is a complex problem, it is one that has already been solved by existing standards.

4.7.6. Social Media

One of the chief distinctions between messaging and 'social media' and is that the former is typically used to describe a synchronous interaction between a closed group of users while most social media consists of asynchronous interactions which are frequently (but not always) public.

The Data At Rest Envelope technology used in the Mesh was originally designed to support asynchronous social media interactions with full end-to-end confidentiality. The service hosting a forum or discussion board need not have access to the content of the messages to support the complete range of user interactions.

5. 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. Since threshold cryptography is not currently part of the 'canon' from which designers of cryptographic security protocols work, much of the cryptography used in the Mesh has been designed for the Mesh. Despite this fact, it is properly regarded as part of the Internet platform on which the Mesh is built rather than a part of the Mesh itself.

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.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 2: Constrained Devices connected through a Mesh Hub.

5.1. 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.

Industry best practices change over time. In the 1990s it was generally believed that supporting the widest possible range of cryptographic algorithms allowed the greatest highest level of security. This view has been decisively rejected in the light of the objection that a successful downgrade attack results in the security afforded by the weakest algorithm supported.

5.2. 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.

5.3. Threshold 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 $_z_ = (_x+y_) \bmod _g_$ (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:

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-architecture-20.html](#) for artwork.)

Figure 3: Two key decryption.

The approach to threshold 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 which is in turn one form of threshold cryptography.

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.

These techniques are described in detail in [\[draft-hallambaker-threshold\]](#).

5.4. Threshold Key Generation

The mathematics that support threshold 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 .

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-architecture-20.html](#) for artwork.)

Figure 4: Two party key pair generation.

Threshold 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.

These techniques are also described in detail in [\[draft-hallambaker-threshold\]](#).

5.5. Threshold Signature

The techniques that support threshold decryption and key generation are also applicable to signature albeit with some very important constraints. Incorrect implementation of the techniques used to create ECDSA signatures can result in disclosure of the private key. It is therefore essential that a threshold signature algorithm is rigorously reviewed.

This technique is used in the mesh to partition the use of administration keys so that the consequences of losing an administrative device can be mitigated.

These techniques are currently being developed by the CFRG. This work is currently described in [\[draft-irtf-cfrg-frost\]](#).

5.6. 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.

5.6.1. 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.

5.6.2. Dare Sequence

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

Sequences 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 Sequence 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 each time it starts to establish a new shared secret for that session. The shared secret is then used to create fresh symmetric keying material for each entry in the log using a unique nonce per entry.

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-architecture-20.html](#) for artwork.)

Figure 5: DARE Sequence 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.

Four types of DARE Sequence are used in the mesh

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

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

Logs A DARE Sequence recording a sequence of events.

Archives A DARE Sequence containing a collection of files, each encoded in a DARE envelope. DARE archives perform a similar function to ZIP archives except that the primary objective is to support confidentiality and integrity rather than data compression.

5.7. 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:

ND5L-BTJE-W372-4TUS-E23N-FIXU-6ARA
UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ
SAQF-K5LV-2HDO-PSY5-YQBN-CPMS-7MXM-K
MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN
ABIR-Y75D-WV63-P5WC-KR3F-VA57-YZ6I

UDF content digests are used to support a direct trust model similar to that of OpenPGP. Every Mesh Profile is authenticated by the UDF fingerprint of its signature key. Mesh Friendly Names and UDF Fingerprints thus serve analogous functions to DNS names and IP Addresses. Like DNS names, Friendly Names provide the basis for application-layer interactions while the UDF Fingerprints are used as to provide the foundation for security.

5.7.1. 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 an instruction to any of her devices using the same friendly name and expect consistent results.

5.7.2. 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:

```
mcu://maker@example.com/EBKG-ED3O-HBHK-ZQGS-EX4H-X22S-X4
```

The EARL may be expressed as a QR code:

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-architecture-20.html](#) for artwork.)

Figure 6: 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.

5.7.3. 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--mamq-etea-jbl3-6uke-lrnt-dgc3-oidf.example.com A strong email address for Alice that can be used by a regular email client.

alice@example.com.mm--mamq-etea-jbl3-6uke-lrnt-dgc3-oidf 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.

5.8. 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/Lagrange secret sharing and recovery 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.

To facilitate escrow and recovery, all the public key pairs and key shares associated with a Mesh profile are generated from a seed value using a deterministic algorithm. Thus, escrow of the seed value is sufficient to permit recovery of the private key data.

For example, Alice escrows her Mesh Profile creating three recovery shares, two of which are required to recover the master secret:

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 7: Use of Shamir/Lagrange 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:

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 8: Use of Shamir/Lagrange 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.

6. Mesh Architecture

The Mesh infrastructure is supported by a compact set of structures and protocols. These are discussed in detail in the Schema Reference [draft-hallambaker-mesh-schema] and Protocol Reference [draft-hallambaker-mesh-protocol] documents.

The JSON object model and JSON or JSON-B serialization [draft-hallambaker-jsonbcd] are used for all Mesh data structures. These include:

Assertions A DARE envelope containing a signed data object. Assertions include Profiles and Connections.

Profile A self-signed assertion describing a Mesh principal. Principals include Accounts, Devices and Services.

Every profile specifies a profile signature key under which it is signed. The UDF fingerprint of the profile signature key is used as a unique identifier for the profile. Thus, all attributes declared in the profile such as authentication keys and encryption keys are bound to the unique identifier for the profile.

Connection An assertion signed by one principal delegating rights to another. Connection assertions are used to bind devices to accounts and hosts to a service.

Activation A DARE envelope encrypted under the encryption key of a principal that grant rights or capabilities to that principal. This is typically but not always achieved through use of threshold key techniques.

Message A DARE envelope signed by the sender that is encrypted under the encryption key of the intended recipient(s) whose content is a Mesh messaging object.

Entry An object stored in a catalog that carries an identifier that is unique for that catalog.

6.1. Actors

Three Mesh actors are defined: Accounts, Devices and Services. Each of these is described by a specific type of profile.

6.1.1. Account

Two types of Mesh account are currently specified: personal accounts and group accounts. For concise exposition, this document is limited to the description of personal accounts. Group accounts are specified in the Schema Reference [draft-hallambaker-mesh-schema].

A Mesh account is an abstraction which may be loosely regarded as the thing to which a collection of devices (in the case of a user account) or a collection of members (in the case of a group account) belong.

Each personal account profile specifies:

Profile Signature Key Used to authenticate the profile. Updates to the profile require use of the Profile Signature Key. The Profile Signature Key cannot be changed but a profile may be replaced by a new profile.

Uniform Data Fingerprint The UDF fingerprint of the Profile Signature Key. This is used as a unique identifier for the account.

Account Name The account name through which the profile is serviced.

Administration Keys UDF fingerprint of keys that are authorized to sign device connection assertions and update the Device Catalog.

Signature Key Public parameters of the account signature key. This is the key that counterparties will use to verify messages sent by the account holder.

Encryption Key Public parameters of the account encryption key. This is the key that counterparties will use to encrypt messages sent to the account holder.

Authentication Key Public parameters of the account authentication key. This is the key that counterparties will use to establish authenticated exchanges with the account holder.

The public keys for encryption, authentication and signature specified in the account profile are the only keys that will (in normal circumstances) be visible to other Mesh accounts.

6.1.2. Device

A Mesh Device is any device that is connected to a Mesh Account through a Device profile. A given physical device may have multiple device profiles associated with it but for the purposes of the Mesh, these are considered to be separate devices. A given device profile may be connected to more than one account

The device profile specifies:

Profile Signature Key Used to authenticate the profile. Updates to the profile require use of the Profile Signature Key. The Profile Signature Key cannot be changed but a profile may be replaced by a new profile.

Uniform Data Fingerprint The UDF fingerprint of the Profile Signature Key. This is used as a unique identifier for the device.

Signature Key Public parameters of the device signature key share. This key share is used as a contribution to the signature key the device will use in the context of the account and to authenticate device connection requests.

Encryption Key Public parameters of the account encryption key share. This key share is used as a contribution to the encryption key the device will use in the context of the account and to decrypt activation records sent in response to device connection requests.

Authentication Key Public parameters of the account authentication key share. This key share is used as a contribution to the authentication key the device will use in the context of the account.

Description Optional information describing the device provided by the manufacturer. E.g. model, serial number, date of manufacture etc.

A Mesh Device is connected to an account through the creation of an activation record and a connection record.

The activation record contains key shares that are overlaid on the corresponding shares specified in the device profile to create the set of encryption, authentication and signature keys the device will use in the context of the account. Since the private keys corresponding to the device profile keys are only used to enable the connection of the device to an account, these keys are only trusted to a minimal degree.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 9: Activation of an account key set.

In the ideal case, the device profile keys are fixed to the device such that they may be used to perform private key operations without the ability to extract the private key data from the device. Since the device profile is only trusted for the limited purpose of connecting the device to an account, the device profile may be created during manufacture without undue concern for either disclosure of the private key on the part of the account holder or a reputation attack alleging disclosure of the private key on the part of the manufacturer.

The device connection record is functionally a certificate that the device may use to interact with the Mesh Service or to other devices connected to the same account. Note however that use of threshold cryptography means that Mesh devices would not normally present their device connection record to any other party since all communication with external parties takes place through the keys published in the account profile.

6.1.3. Service

A Mesh Service is an abstract network service that is provided by one or more hosts. The properties of the service are described by the service profile.

The service profile specifies:

Profile Signature Key Used to authenticate the profile. Updates to the profile require use of the Profile Signature Key. The Profile Signature Key cannot be changed but a profile may be replaced by a new profile.

Uniform Data Fingerprint The UDF fingerprint of the Profile Signature Key. This is used as a unique identifier for the device.

Signature Key Public parameters of the service signature key. This is the key that counterparties will use to verify messages sent by the service.

Encryption Key Public parameters of the service encryption key. This is the key that counterparties will use to encrypt messages sent to the service.

Authentication Key Public parameters of the account authentication

key. This is the key that counterparties will use to establish authenticated exchanges with the service.

Hosts are Mesh Devices that have been granted a Host Activation and Host Connection by a service administrator. These are used in the same fashion as the device activation and connection records.

6.2. Stores

Mesh Stores are append-only sequences that are used to represent collections of objects, messages and data. All Mesh stores are implemented as DARE Sequences authenticated by means of a Merkle tree. The payload of each envelope in the sequence is usually encrypted.

Two types of Mesh store are currently defined:

Catalog A set of Mesh objects, each of which has an identifier that is unique in the scope of the catalog. Objects may be added, updated, and deleted.

Spool A sequence of Mesh Messages.

All the state represented within a Mesh account is contained in Mesh stores bound to the account. Thus, to synchronize a device to the state of the account, it is sufficient to synchronize the collection of stores the device is permitted to read. Since every store is an append-only sequence, it is sufficient for the Mesh service to return the envelopes added to each of the stores since the device was last synchronized.

Rapid synchronization of catalogs and spools is ensured by limiting the size of entries to each. Implementations may further improve performance by redacting stores to remove obsolete entries that have been updated or deleted. Alternatively, a device may maintain a complete record of the state of the store to allow erroneous changes to the store to be unwound.

6.2.1. Catalogs

Mesh Catalogs track a collection of entries. Every Mesh account contains a Threshold Catalog that is used by Mesh services as the source of access control policy. The Threshold Catalog is unique in that it is the only catalog whose contents can be read by the Mesh Service. Every other Mesh Catalog connected to a Mesh account is end-to-end encrypted so that it can only be read by devices connected to the account.

The Mesh specifies various catalogs that are used to track information relevant to a Mesh Account:

Device The devices connected to the corresponding Mesh profile.

Contact Logical and physical contact information for people and organizations.

Bookmark Web bookmarks and citations.

Credential Username and password information for network resources.

Calendar Appointments and tasks.

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

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

Access A catalog that is readable by the Mesh Service that is used to determine access to account resources including device access rights, threshold keys and message delivery.

Each catalog connected to an account has a unique identifier of the form `mmm_<name>`. Applications may specify additional catalogs without risk of collision with future Mesh catalogs by using an appropriate IANA assigned protocol label.

6.2.2. Spools

Spools are used to track inbound and outbound messages. Three spools are currently defined:

Inbound Messages that have been received by the service and accepted for delivery to the account.

Outbound Messages that have been sent from the account through the service.

Local A spool used to exchange messages with devices connecting to the device.

6.3. Mesh Service Protocol

Mesh services communicate with Mesh devices and other Mesh Services through the Mesh Service Protocol. Despite the wide range of Mesh functionality, the Mesh protocol is remarkably compact. The bulk of the semantics associated with the Mesh are expressed in the schemas describing Mesh Messages and Catalogs. The objective of reducing the degree of trust in the Mesh service to the absolute minimum by necessity requires that the Mesh Service be extremely simple.

Mesh Service Protocol transactions are divided into the following groups:

Service Description The Hello transaction returns a description of the service including information used to authenticate future interactions with the service.

Account Management The Create and Delete transactions are used to bind an account to a service.

Device Connection The Connect and Complete transactions are used to connect devices to an account

Synchronization The Status, Download and Transact transactions are used to update stores connected to an account.

Messaging The Post transaction is used by one Mesh Service to transfer a message from one of its users to a different Mesh Service serving one of the recipients.

Publication The Publish, Claim and PollClaim transactions are used to publish and retrieve data objects through an account.

Cryptographic The Operate transaction requests that the service performs a cryptographic operation on behalf of the account. This is used to provide execution of threshold operations on behalf of the account holder for both internal and external users.

Future versions of the Mesh Service Protocol may support additional transactions to support features such as providing DNS resolution.

6.3.1. Protocol Interactions

Every Mesh Service Protocol transaction consists of a single request from a Mesh client followed by a single response. Requests and responses are authenticated and encrypted under a key established between the client and the service. This application layer enhancement is in addition to any transport layer enhancement that may be employed (e.g. TLS).

Mesh Service Protocol messages may be exchanged through any binding advertised by the service by means of the Hello transaction. Currently only one binding is defined, mapping Mesh requests and responses to the content data of HTTP POST requests and responses layers over a TLS transport.

While the use of up to three layers of encryption may be regarded as excessive, each layer provides separate protections:

Transport Layer Provides confidentiality for metadata and limited traffic analysis protections.

Application Layer Encryption and authentication of requests and responses using keys bound to the specific device and service performing the interaction provides the basis for access control.

Data Layer Encryption of stored data (catalog data, device activations, etc.) provides end to end security between the devices connected to the account.

6.4. The Access Catalog

The Access Catalog of a Mesh account is the only catalog whose payload contents are readable by the service. This is necessary because the catalog provides the sole source for the authorization component of the access control policy for the account.

Each entry in the catalog specifies an operation that the service will perform when it receives a request that is authenticated and authorized by the access control policy specified in the entry. Operations include:

Service Provision Account operations such as operations on stores (Status, Download, Transact) require that the client be authenticated by means of an authentication key that has a matching entry in the Access catalog that authorizes the operation.

Inbound Message Filtering Messages received by the service that

match the specified criteria will be appended to the inbound message spool.

Threshold Key Generation Performs a threshold key splitting operation of a private key held by the service and encrypts one part under a key known only to the service and encrypts the other under a public key specified by the party making the request.

Key Agreement Performs a key agreement operation on a private key held by the service. This may be used as a component in a threshold key agreement scheme.

Signature Performs a signature operation on a private key held by the service. This may be used as a component in a threshold signature scheme.

These operations provide the vocabulary from which a Threshold Key Infrastructure is built. Keys that are bound to a service using threshold techniques can only be applied with the co-operation of that service.

6.5. Mesh Messaging Protocol

Mesh devices connected to an account interact with the Mesh Service through the Mesh Service protocol. Mesh devices interact with other Mesh devices through the Mesh Messaging Protocols, each of which provides a distinct application functionality:

- * Connection Protocol
- * Confirmation Protocol
- * Contact Exchange Protocol

Each of these protocols is described in depth in the Mesh Protocol Reference [draft-hallambaker-mesh-protocol].

Mesh Messages provide a means of communication between Mesh Service Accounts with capabilities that are not possible or poorly supported in traditional SMTP mail messaging:

- * End-to-end confidentiality and authentication by default.
- * Abuse mitigation by applying access control to every inbound and outbound message.
- * End-to-end secure group messaging.

- * Transfer of exceptionally large data sets (Terabytes).

Note that although Mesh Messaging is designed to facilitate the transfer of very large data sets, the size of Mesh Messages themselves is severely restricted. The current default maximum size being 64 KB. This approach allows Mesh

In addition, the platform anticipates but does not currently support additional cryptographic security capabilities:

- * Traffic analysis resistance using mix networks (Chaum).
- * Simultaneous contract binding using fair contract signing (Micali).

While these capabilities might in time cause Mesh Messaging to replace SMTP, this is not a near term goal. The short-term goal of Mesh Messaging is to support the Contact Exchange and Confirmation applications.

Two important classes of application that are not currently supported directly are payments and presence. While prototypes of these applications have been considered, it is not clear if these are best implemented as special cases of the Confirmation and Contact Exchange applications or as separate applications in their own right.

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.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 10: Four Corner Messaging Model

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.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 11: 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.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 12: 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 exceptionally 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.

6.6. 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:

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-architecture-20.html for artwork.)

Figure 13: 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.

6.6.1. 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 sequence 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 Sequences 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 sequence synchronization primitives provided by a Mesh Service. This approach could afford a robust, secure, and efficient mechanism for software distribution and update.

7. 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].

8. IANA Considerations

This document does not contain actions for IANA

9. Acknowledgements

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

Appendix D: Outstanding Issues

The following issues need to be addressed.

Issue	Description
Newlines	Make handling of newlines in shell commands consistent, sometimes additional newlines are inserted.
Recovery	Expand explanation and example to show applications being recovered.
Unread Bug	Messages are not being correctly marked as read causing spurious error messages.

Table 1

10. Normative References

[draft-hallambaker-jsonbcd]

Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", Work in Progress, Internet-Draft, draft-hallambaker-jsonbcd-21, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-jsonbcd-21>>.

[draft-hallambaker-mesh-callsign]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part VII: Mesh Callsign Service", Work in Progress, Internet-Draft, draft-hallambaker-mesh-callsign-01, 23 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-callsign-01>>.

[draft-hallambaker-mesh-cryptography]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part VIII: Cryptographic Algorithms", Work in Progress, Internet-Draft, draft-hallambaker-mesh-cryptography-08, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-cryptography-08>>.

[draft-hallambaker-mesh-dare]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part III : Data At Rest Encryption (DARE)", Work in Progress, Internet-Draft, draft-hallambaker-mesh-dare-14, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-dare-14>>.

[draft-hallambaker-mesh-developer]

Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-10, 27 July 2020, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-10>>.

[draft-hallambaker-mesh-platform]

Hallam-Baker, P., "Mathematical Mesh: Platform Configuration", Work in Progress, Internet-Draft, draft-hallambaker-mesh-platform-06, 27 July 2020, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-platform-06>>.

[draft-hallambaker-mesh-protocol]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part V: Protocol Reference", Work in Progress, Internet-Draft, draft-hallambaker-mesh-protocol-12, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-protocol-12>>.

[draft-hallambaker-mesh-rud]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part VI: Reliable User Datagram", Work in Progress, Internet-Draft, draft-hallambaker-mesh-rud-00, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-rud-00>>.

[draft-hallambaker-mesh-schema]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part IV: Schema Reference", Work in Progress, Internet-Draft, draft-hallambaker-mesh-schema-09, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-schema-09>>.

[draft-hallambaker-mesh-security]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>>.

[draft-hallambaker-mesh-udf]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.", Work in Progress, Internet-Draft, draft-hallambaker-mesh-udf-15, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-udf-15>>.

- [draft-hallambaker-threshold]
Hallam-Baker, P., "Threshold Modes in Elliptic Curves",
Work in Progress, Internet-Draft, draft-hallambaker-
threshold-06, 5 August 2021,
<[https://datatracker.ietf.org/doc/html/draft-hallambaker-
threshold-06](https://datatracker.ietf.org/doc/html/draft-hallambaker-threshold-06)>.
- [draft-hallambaker-web-service-discovery]
Hallam-Baker, P., "DNS Web Service Discovery", Work in
Progress, Internet-Draft, draft-hallambaker-web-service-
discovery-06, 5 August 2021,
<[https://datatracker.ietf.org/doc/html/draft-hallambaker-
web-service-discovery-06](https://datatracker.ietf.org/doc/html/draft-hallambaker-web-service-discovery-06)>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246,
DOI 10.17487/RFC5246, August 2008,
<<https://www.rfc-editor.org/rfc/rfc5246>>.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data
Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March
2014, <<https://www.rfc-editor.org/rfc/rfc7159>>.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
(HTTP/1.1): Semantics and Content", RFC 7231,
DOI 10.17487/RFC7231, June 2014,
<<https://www.rfc-editor.org/rfc/rfc7231>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web
Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May
2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",
RFC 7516, DOI 10.17487/RFC7516, May 2015,
<<https://www.rfc-editor.org/rfc/rfc7516>>.

11. Informative References

[draft-hallambaker-mesh-trust]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part X: The Trust Mesh", Work in Progress, Internet-Draft, draft-hallambaker-mesh-trust-09, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-trust-09>>.

[draft-irtf-cfrg-frost]

Connolly, D., Komlo, C., Goldberg, I., and C. A. Wood, "Two-Round Threshold Schnorr Signatures with FROST", Work in Progress, Internet-Draft, draft-irtf-cfrg-frost-04, 29 March 2022, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-frost-04>>.

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
20 April 2022

Mathematical Mesh 3.0 Part VIII: Cryptographic Algorithms
draft-hallambaker-mesh-cryptography-09

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.

[Note to Readers]

Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

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

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 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.

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
6. Multi-Party Decryption	9
6.1. Mechanism	11
6.2. Implementation	12
6.2.1. Group Creation	12
6.2.2. Provisioning a Member	13
6.2.3. Message Encryption and Decryption	13
7. Mutually Authenticated Key Agreement	14
8. Security Considerations	15
9. IANA Considerations	15
10. Acknowledgements	15
11. Examples	15
11.1. Key Combination	15
11.1.1. Ed25519	16
11.1.2. Ed448	16
11.1.3. X25519	16
11.1.4. X448	16
11.2. Group Encryption	16

11.2.1. X25519	16
11.2.2. X448	16
12. Normative References	16
13. Informative References	17

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 Arduinino 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]

- * 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]

Support for the following algorithms is RECOMMENDED:

- * AES-GCM-256 Encryption [AES-GCM]
- * SHA-3-512 [SHA-3]
- * KMAC SHA-3-512 [SHA-3-Derived]

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-2-512 [SHA-2]
- * HMAC-SHA-2-512 [RFC2104]
- * HMAC-based Extract-and-Expand Key Derivation Function [RFC5869]
- * Poly1035 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 {public, private} key pairs.

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

$Z = X \otimes Y$ and $z = (x + y) \bmod o$ (where o is the order of the group)

The Result Combination Law states that we can define an operator \otimes such that:

$$(X \otimes E) \otimes (Y \otimes E) = (Z \otimes E) = (e \otimes Z).$$

4.1. Application to Diffie Hellman (not normative)

For the Diffie Hellman system in a modular field p , $o = p-1$ and $a \otimes b = a \cdot b \bmod p$.

Proof:

By definition, $X = e^x \bmod p$, $Y = e^y \bmod p$, and $Z = e^z \bmod p$.

Therefore,

$$Z = e^z \bmod p = e^{(x+y)} \bmod p = (e^x e^y) \bmod p = e^x \bmod p \cdot e^y \bmod p = X \cdot Y$$

A similar proof may be constructed for the operator \otimes .

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:

0. 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}.
1. Generates a private key `_m_` with the specified number of bytes (32 or 57).
2. Calculates the corresponding public key `_M_`.
3. Calculates the Application public key $A = D + M$ where $+$ is point addition.
4. 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.

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.

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.

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 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 $y = g - x \bmod o$, 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 reryption in the Mesh has four parts:

- * Creation and management of the reryption group.
- * Provisioning of members to a reryption 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 reryption group has its own independent Mesh account. This has many advantages:

- * Administration of the reryption 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 reryption 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 Reryption group requires the steps of:

- * Generating the reryption group key pair
- * Creating the reryption group account
- * Generating administrator record for each administrator.
- * Publishing the administrator records to the reryption 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 reryption group, the administrator requires their current reryption 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 reryption 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 reryption group to compute the reryption key for the member. This value and the encrypted decryption key entry are combined to form the reryption 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 reryption operation MUST specify the reryption group address an not just the key fingerprint. This allows the recipient to determine which reryption service to contact to perform the reryption operation.

To decrypt a message, the recipient makes an authenticated reryption request to the specified reryption service specifying:

- * The recipient entry to be used for decryption
- * The fingerprint of the decryption key(s) the device would like to make use of.
- * Whether or not the encrypted decryption key entry should be returned.

The reryption service searches the catalog for the corresponding reryption 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 reryption key specified in the entry and returns the result to the recipient.

The recipient then decrypts the reryption 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.

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 $_{DH}(a, B) = _{DH}(b, A)$ 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 $A ? N_A$.

The server uses its public key B .

The key agreement value is $_{DH}(a + n_A, B) = _{DH}(b, A ? N_A)$

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

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

The key agreement value is $_{DH}(_a + n_A, B \oplus N_B) = _{DH}(_b + n_B, A \oplus N_A)$

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. Normative References

[AES-GCM] Dworkin, M. J., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", November 2007.

[draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>>.

[draft-hallambaker-mesh-security]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>>.

[FIPS197] NIST, "Advanced Encryption Standard (AES)", November 2001.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/rfc/rfc2104>>.

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

- [RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <<https://www.rfc-editor.org/rfc/rfc3394>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/rfc/rfc5869>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/rfc/rfc7748>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/rfc/rfc8032>>.
- [RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <<https://www.rfc-editor.org/rfc/rfc8439>>.
- [SHA-2] NIST, "Secure Hash Standard", August 2015.
- [SHA-3] Dworkin, M. J., "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", August 2015.
- [SHA-3-Derived]
Kelsey, J. M., Chang, S. H., and R. A. Perlner, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash SHARE", December 2016.

13. Informative References

- [Blaze98] "[Reference Not Found!]".
- [draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-10, 27 July 2020, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-10>>.

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
ThresholdSecrets.com
20 April 2022

Mathematical Mesh 3.0 Part III : Data At Rest Encryption (DARE)
draft-hallambaker-mesh-dare-15

Abstract

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

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

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

[Note to Readers]

Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

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

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 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.

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	6
1.2.1.	Signing Individual Plaintext Envelopes	7
1.2.2.	Signing Individual Encrypted Envelopes	7
1.2.3.	Signing sequences of envelopes	7
1.3.	Sequence	8
1.3.1.	Sequence Format	8
1.3.2.	Write	9
1.3.3.	Encryption and Authentication	9
1.3.4.	Integrity and Signature	10
1.3.5.	Redaction	10
1.3.6.	Alternative approaches	11
1.3.7.	Efficiency	11
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.	Encoded Data Sequence	15
3.3.	Content Metadata and Annotations	16
3.4.	Encryption and Integrity	17
3.4.1.	Key Exchange	18
3.4.2.	Key Identifiers	19
3.4.3.	Salt Derivation	19
3.4.4.	Key Derivation	20
3.5.	Signature	21
3.6.	Algorithms	21
3.6.1.	Field: kwd	21
4.	DARE Sequence Architecture	21
4.1.	Sequence Navigation	21
4.1.1.	Tree	22

4.1.2.	Position Index	23
4.1.3.	Metadata Index	23
4.2.	Integrity Mechanisms	23
4.2.1.	Digest Chain calculation	23
4.2.2.	Binary Merkle tree calculation	24
4.2.3.	Signature	24
5.	DARE Schema	25
5.1.	Envelope Classes	25
5.1.1.	Structure: DareEnvelopeSequence	25
5.2.	Header and Trailer Classes	25
5.2.1.	Structure: DareTrailer	25
5.2.2.	Structure: DareHeader	26
5.2.3.	Structure: ContentMeta	27
5.3.	Cryptographic Data	28
5.3.1.	Structure: DareSignature	28
5.3.2.	Structure: X509Certificate	29
5.3.3.	Structure: DareRecipient	29
5.3.4.	Structure: DarePolicy	29
5.3.5.	Structure: FileEntry	30
6.	DARE Container Schema	30
6.1.	Container Headers	30
6.1.1.	Structure: SequenceInfo	30
6.2.	Index Structures	31
6.2.1.	Structure: SequenceIndex	31
6.2.2.	Structure: IndexPosition	32
6.2.3.	Structure: KeyValue	32
7.	Dare Sequence Applications	32
7.1.	Catalog	32
7.2.	Spool	33
7.3.	Archive	33
8.	Future Work	34
8.1.	Terminal integrity check	34
8.2.	Terminal index record	34
8.3.	Deferred indexing	34
9.	Security Considerations	35
9.1.	Encryption/Signature nesting	35
9.2.	Side channel	35
9.3.	Salt reuse	35
10.	IANA Considerations	35
11.	Acknowledgements	35
12.	Appendix A: DARE Envelope Examples and Test Vectors	35
13.	Test Examples	35
13.1.	Plaintext Message	36
13.2.	Plaintext Message with EDS	36
13.3.	Encrypted Message	36
13.4.	Signed Message	38
13.5.	Signed and Encrypted Message	39
14.	Appendix B: DARE Sequence Examples and Test Vectors	39

14.1. Simple sequence	40
14.2. Payload and chain digests	40
14.3. Merkle Tree	42
14.4. Signed sequence	45
14.5. Encrypted sequence	46
15. Appendix C: Previous Frame Function	50
16. Appendix D: Outstanding Issues	50
17. Normative References	51
18. Informative References	53

1. Introduction

This document describes the Data At Rest Encryption (DARE) Envelope and Sequence Syntax. The DARE Envelope syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content. The DARE Sequence 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 Sequence 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 Sequence 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 Sequence.

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 PKCS#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).

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-dare-15.html for artwork.)

Figure 1: 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).

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-dare-15.html](#) for artwork.)

Figure 2: 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).

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-dare-15.html](#) for artwork.)

Figure 3: 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. Sequence

DARE Sequence 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. Sequence Format

A Sequence 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:

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-dare-15.html](#) for artwork.)

Figure 4: 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.

Currently, the Mesh only makes use of the Merkle Tree sequence type.

1.3.2. Write

In normal circumstances, Sequences 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, Sequence 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 specified in each envelope.

Each Envelope and by extension, each Sequence 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 Sequence 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 Sequence 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 Sequence 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 Sequence 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].

The terms "Authentication", "Ciphertext", "Digital Signature", "Encryption", "Initialization Vector (IV)", "Message Authentication Code (MAC)", "Plaintext" and "Salt" are defined by the Internet Security Glossary, Version 2 [RFC4949].

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 [RFC7159] Offers compatibility with applications and libraries that support JSON. Payload data is encoded using Base64 incurring a 33% overhead.

JSON-B [draft-hallambaker-jsonbcd] 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 [draft-hallambaker-jsonbcd] 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. 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
  00
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
[EOF]
```

3.3. 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:

```
[{
  "annotations":["iAEAiBdGcm9tOiBBbGljZUBleGFtcGx1LmNvbQ",
    "iAEBiBNUbzogYm9iQGV4YW1wbGUuY29t",
    "iAECiClTdWJqZWNoOiBUT1AtU0VDUkVUIFBYb2R1Y3QgTGFlbmNoIFRvZG
F5IQ"
  ],
  "ContentMetaData":"ewogICJjdHkiOiAiYXBwbGljYXRpb24vZXhhbXBsZS
1tYWlsIn0"},
  "VGhlIENFTyB0b2xkIGllIHRoZSBwcm9kdWN0IGxhdW5jaCBpcyB0b2RheS4gVG
VsbCBuby1vbWUh"
]
```

This contains the same information as before but the data we might wish to encrypt to protect the confidentiality of the payload is separated from data required for processing.

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",
  "kid": "EBQO-VVRO-PZGV-RRCY-HH2Q-E2SE-G4W6",
  "Salt": "6LmVUEKI8-q72JYJg80NVQ",
  "annotations": ["iAEAiCD7pIbyRm2fJJn_9KraDZihW-F8Fn1iIaVBlq3lZL
4X4A",
    "iAEBiCA9QH-N2ClnxAikU15FFjeHFG9H5qX6zrDujPGUL2s1Ig",
    "iAECiDAHsjD13dra5_1NakdNKi6Rj8P7E6fMB_Y1UpL-CWdY0Yselibgz4
97t1P8nKlMIq0"
  ],
  "recipients": [{
    "kid": "MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
    "epk": {
      "PublicKeyECDH": {
        "crv": "Ed25519",
        "Public": "OYT5iH4doxVrj90NRowmffe200OPLlRGqaCav6b-Xw4"
      }
    },
    "wmk": "N3KQ0jcCztbOMSowcvy_UdGNsLL-PMtd9_ZMuWqT4GzEIXj33a
HlKQ"
  }
  ],
  "ContentMetaData": "ewogICJjdHkiOiAiYXBwbGljYXRpb24vZXhhbXBsZS
1tYWlsIn0",
  "bWhY3cljSPEQfl6k6jzZQRZR53Fbrbz1c9OwNp52Ry4_kbv961FXPO-j28kRms
eSrH6_hzJZTMouse1KQA812w"
}]
```

For efficiency of processing, the ContentMetaData is presented in plaintext. This header could be encrypted as an EDS sequence and presented as a cloaked header.

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:

0. The Master Key and salt value are used to extract the PRK (pseudorandom key)
1. 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 Sequence Architecture

4.1. Sequence 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 Sequences 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

As previously described, the JBCD frame structure allows incremental navigation to the immediately preceding frame. The `TreePosition` parameter specifies the start position of any previous frame in the container, thus allowing rapid navigation to that point.

The `TreePosition` parameter MAY be used to enable any frame in the container to be retrieved in $\log_2(n)$ time by means of a binary search. The `TreePosition` parameter specifies the immediately preceding apex of a binary tree formed from the container entries.

For example, the `TreePosition` of frame 6 in a container gives the location of frame 5, the `TreePosition` of frame 5 gives the location of frame 3, the `TreePosition` of frame 3 gives the location of frame 1, and the `TreePosition` of frame 1 gives the location of frame 0:

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-dare-15.html](#) for artwork.)

Figure 5: Binary search tree.

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:

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-dare-15.html](#) for artwork.)

Figure 6: 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 $O(\log_2(n))$ complexity provided that the container format supports at least the binary tree index. Verifying a chain has $O(\log_2(n))$ 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 envelopes.

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. Envelope Classes

A DARE envelope 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 envelope 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 envelope is functionally an object, it is serialized as an ordered sequence. This ensures that the envelope 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 envelope header. May specify the key exchange data, pre-signature or signature data, cloaked headers and/or encrypted data sequences.

Body: Binary (Optional) The envelope body

Trailer: DareTrailer (Optional) The envelope trailer. If present, this contains the signature.

5.2. Header and Trailer Classes

A DARE sequence MUST contain a (possibly empty) DAREHeader and MAY contain a DARETrailer.

5.2.1. Structure: DareTrailer

A DARE envelope Trailer

Signatures: DareSignature [0..Many] A list of signatures. A envelope 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 Envelope Header. Since any field that is present in a trailer MAY be placed in a header instead, the envelope header inherits from the trailer.

EnvelopeId: String (Optional) Unique identifier

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 envelope payload.

KeyIdIdentifier: String (Optional) Base seed identifier.

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.

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 envelope 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 envelope body and to headers specified within the cloaked header.

Processing of cloaked data is described in...

EDSS: Binary [0..Many] If present, the Annotations field contains a sequence of Encrypted Data Segments encrypted under the envelope base seed. The interpretation of these fields is application specific.

Signers: DareSignature [0..Many] A list of 'presignature'

Recipients: DareRecipient [0..Many] A list of recipient key exchange information blocks.

Policy: DarePolicy (Optional) A DARE security policy governing future additions to the container.

ContentMetaData: Binary (Optional) If present contains a JSON encoded ContentInfo structure which specifies plaintext content metadata and forms one of the inputs to the envelope digest value.

SequenceInfo: SequenceInfo (Optional) Information that describes container information

SequenceIndex: SequenceIndex (Optional) An index of records in the current container up to but not including this one.

Received: DateTime (Optional) Date on which the envelope was received.

5.2.3. Structure: ContentMeta

UniqueId: String (Optional) Unique object identifier

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.

MessageType: String (Optional) The mesh message type

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.

Filename: String (Optional) The original filename under which the data was stored.

Event: String (Optional) Operation on the header

Created: DateTime (Optional) Initial creation date.

Modified: DateTime (Optional) Date of last modification.

Expire: DateTime (Optional) Date at which the associated transaction will expire

First: Integer (Optional) Frame number of the first object instance value.

Previous: Integer (Optional) Frame number of the immediately prior object instance value

FileEntry: FileEntry (Optional) Information describing the file entry on disk.

5.3. Cryptographic Data

DARE envelope uses the same fields as JWE and JWS but with different structure. In particular, DARE envelopes MAY have multiple recipients and multiple signers.

5.3.1. Structure: DareSignature

The signature value

Dig: String (Optional) Digest algorithm hint. Specifying the digest algorithm to be applied to the envelope 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.

Manifest: Binary (Optional) The data description that was signed.

SignatureValue: Binary (Optional) The signature value as an Enhanced Data Sequence under the envelope base seed.

WitnessValue: Binary (Optional) The signature witness value used on an encrypted envelope to demonstrate that the signature was authorized by a party with actual knowledge of the encryption key used to encrypt the envelope.

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: 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.

WrappedBaseSeed: Binary (Optional) The wrapped base seed. The base seed is encrypted under the result of the key exchange.

RecipientKeyData: String (Optional) The per-recipient key exchange data.

5.3.4. Structure: DarePolicy

EncryptionAlgorithm: String (Optional) The encryption algorithm to be used to compute the payload.

DigestAlgorithm: String (Optional) The digest algorithm to be used to compute the payload digest.

Encryption: String (Optional) The encryption policy specifier,

determines how often a key exchange is required. 'Single': All entries are encrypted under the key exchange specified in the entry specifying this policy. 'Isolated': All entries are encrypted under a separate key exchange. 'All': All entries are encrypted. 'None': No entries are encrypted.

Default value is 'None' if EncryptKeys is null, and 'All' otherwise.

Signature: String (Optional) The signature policy 'None': No entries are signed. 'Last': The last entry in the container is signed. 'Isolated': All entries are independently signed. 'Any': Entries may be signed.

Default value is 'None' if SignKeys is null, and 'Any' otherwise.

Sealed: Boolean (Optional) If true the policy is immutable and cannot be changed by a subsequent policy override.

5.3.5. Structure: FileEntry

Path: String (Optional) The file path in canonical form.

CreationTime: DateTime (Optional) The creation time of the file on disk in UTC

LastAccessTime: DateTime (Optional) The last access time of the file on disk in UTC

LastWriteTime: DateTime (Optional) The last write time of the file on disk in UTC

Attributes: Integer (Optional) The file attributes as a bitmapped integer.

6. DARE Container Schema

TBS stuff

6.1. Container Headers

TBS stuff

6.1.1. Structure: SequenceInfo

Information that describes container information

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.

ContainerType: String (Optional) Specifies the container type for the following records. This value is ONLY valid in Frame 0 which MUST have a header encoded in JSON.

Index: Integer (Optional) The record index within the file. This MUST be unique and satisfy any additional requirements determined by the ContainerType.

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.

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

6.2. Index Structures

TBS stuff

6.2.1. Structure: SequenceIndex

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

6.2.2. 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.

UniqueId: String (Optional) Unique object identifier

6.2.3. Structure: KeyValue

Specifies a key/value entry

Key: String (Optional) The key

Value: String (Optional) The value corresponding to the key

7. Dare Sequence Applications

DARE Sequences 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 Sequences 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.

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.

7.3. Archive

A DARE Archive is a DARE Sequence 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 Abdulhaya?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":"YX39DIWEBqxIJOBjRlyPNubhuB4oKmyz_f_PZk2Wft4"}}}
```

Alice's signature private key parameters are:

```
{
  "PrivateKeyECDH":{
    "crv":"Ed25519",
    "Private":"i-OxFUS-3G00ftVIy3TYOAPWBjOnz3ZUNqLEdIx7stA"}}}
```

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":[{}],  
  "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm9lZ2ggdG8gcmVxdWlyZSBtdWx0aXBsZSBibG9ja3M"  
}]
```

13.2. Plaintext Message with EDS

If a plaintext message contains EDS sequences, these are also in plaintext:

```
{  
  "DareEnvelope":[{  
    "annotations":["iAEAiC1TdWJqZWN0OiBNZXNzYWdlIGlldGFkYXRhIHNo  
b3VsZCBiZSBibmNyeXB0ZWQ",  
    "iAEBiAoyMDE4LTAyLTAx"  
  ]},  
  "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm9lZ2ggdG8gcmVxdWlyZSBtdWx0aXBsZSBibG9ja3M"  
}]
```

13.3. Encrypted Message

The creator generates a base seed:

```
E6 71 C9 D3  99 29 57 00  01 C5 DF 54  5C 81 1F C6  
75 BF 99 11  EC F9 92 BA  4B FF 5E B2  0A F4 E8 8A
```

For each recipient of the message:

The creator generates an ephemeral key:

```
{
  "PrivateKeyECDH":{
    "crv":"Ed25519",
    "Private":"v1EY8ZhRS0pkHVz94JNyFA_RPhfcsdxFjYoHWe2LN84"}}}
```

The key agreement value is calculated:

```
29 53 00 38 88 B1 25 C0 4C D6 DC 4F D0 C3 BD EB
B0 B9 97 E7 CD 95 A0 8E AF A6 FE FD 18 A0 65 58
```

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 base seed:

```
6A EA FC 8C 42 63 BD 42 A5 CC 41 6C 3A 5F 45 EA
5C C8 E1 8F CA 41 3B 5B B1 14 33 1D 08 08 38 2D
```

The wrapped base seed is:

```
37 72 90 D2 37 02 CE D6 CE 31 23 B0 72 FC BF 51
D1 8D B0 B2 FE 3C CB 5D F7 F6 4C B9 6A 93 E0 6C
C4 21 78 F7 DD A1 E5 29
```

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:

```
B0 5D 23 F4 DC C8 92 EE CE 02 F7 33 96 5F 23 37
```

The base seed and salt value are used to generate the payload encryption key:

```
09 A5 BE 8B F7 48 35 A6 F0 E3 07 3C 9D 47 B6 1B
1B AD 48 BB 99 E3 2F 92 CC FF C6 3F EC 3B 37 6C
```

Since AES is a block cipher, we also require an initializariion vector:

```
69 85 92 7A 4F F3 A1 C7 7A 04 20 90 9F 43 0E E8
```

The output sequence is the encrypted bytes:

```
92 58 64 D4 AF 6A 3D C1 59 2F ED 2B 02 EE FA 53
9A 21 F1 57 F9 29 B6 5C 9D 71 1F 69 87 3A FD F8
26 28 54 99 12 83 07 AE 73 72 45 73 8D 0F 66 9F
1B F8 B6 A8 F7 F9 10 14 4B C7 23 95 96 9C E7 A2
```

Since the message is not signed, there is no need for a trailer. The completed message is:

```
{
  "DareEnvelope":[{
    "enc":"A256CBC",
    "kid":"EBQK-WRSP-WK6E-AVC5-2ZQI-PZ5K-2K6I",
    "Salt":"sF0j9NzIku70Avcz1l8jNw",
    "recipients":[{
      "kid":"MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
      "epk":{
        "PublicKeyECDH":{
          "crv":"Ed25519",
          "Public":"LE55Fn_dDQFZvz1JvYToyy-GmKcHD4zELw8U6WWZK
qk"}},
      "wmk":"Q6XR1OPMdcNSaJ2Vd-BsdWNkAl5SMm3Hho6uoPmWPFVoWzeg
WF9boQ"}
    ]},
    "klhk1K9qPcFZL-0rAu76U5oh8Vf5KbZcnXEfaYc6_fgmKFSZEoMHRnNyRXON
D2afG_i2qPf5EBRLxyOVlpznog"
  ]}
```

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.

```
{
  "DareEnvelope":[{
    "dig":"S512",
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdWx0aXBs
ZSBibG9ja3M",
    {
      "signatures":[{
        "alg":"S512",
        "kid":"MAR2-RQDD-TLR2-UQP5-YCU5-26Y3-YWCZ",
        "signature":"araghHqBT0RZa5qlXgtBS0udR3jdtSv1UiqNdsUlzd
mjmbqvt83WKGmcS07JvLufqfF9de1LCK5LT4yuAlO8Cg"}
      ],
      "PayloadDigest":"raim8SV5adPbWWn8FMM4mrRAQCO9A2jZ0NZAnFXWlG
0xF6sWGJbnKSdtIJMmMU_hjarlIPEoY3vy9UdVlH5KAg"}
    ]}
```

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.

```
{
  "DareEnvelope": [{
    "enc": "A256CBC",
    "dig": "S512",
    "kid": "EBQN-KLCB-QE3Z-S4IC-Y46H-MN7G-FVSC",
    "Salt": "9_yqbGfRxVimraQfhaIFZQ",
    "recipients": [{
      "kid": "MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
      "epk": {
        "PublicKeyECDH": {
          "crv": "Ed25519",
          "Public": "OT13Sphcbw3rDkKJdjt73V2Ji0Wle1QnULdE88Aj
20"}},
      "wmk": "o9WQhJKLmnMG39XgxOMArUKNauSiKudfNvZnlFUUsj5YqTC_
eMpHXg"}
    ]},
  "gYktKBTQrq_lcfBgCJp7qT-BciaMw5WZ4-oRnLiLcu-87pDrOETXN18yWG_p
glzyDeyGkvcYPgHIGzZ3O4zGSg",
  {
    "signatures": [{
      "alg": "S512",
      "kid": "MAR2-RQDD-TLR2-UQP5-YCU5-26Y3-YWCZ",
      "signature": "iI0aJ271ojzRuTln6nr6Q_HGQ015mehYlMZSmXznmZ
iFkwB9tY-J-5ASmNhntFeUBeFJkocfGw225HHG9pdsBw",
      "witness": "G9x6MI7_vEAK9bLN97ZgzzZeXG01A8bGY53CmwSkVwE"}
    ],
    "PayloadDigest": "2Nw9Ydgm-aEoc5i4TwqT8lAuruwEtpfnj3KeLZsqoI
j5SpsR6l3RjlIhDhsd2p2Tb2QS8eXD73csYNnlmA8G1A"}
  ]}
```

14. Appendix B: DARE Sequence 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 sequence

The following example shows a simple sequence with first frame and a single data frame:

```
f4 91
f0 8b
f0 00
f0 00
91 f4
f5 01 73
f0 42
f1 01 2c
73 01 f5
```

Since there is no integrity check, there is no need for trailer entries. The header values are:

Frame 0

```
{
  "DareHeader":{
    "policy":{},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"List",
      "Index":0}}}
[Empty trailer]
```

Frame 1

```
{
  "DareHeader":{
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1}}}
[Empty trailer]
```

14.2. Payload and chain digests

The following example shows a chain sequence 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

```
{
  "DareHeader":{
    "dig":"S512",
    "policy":{},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"Chain",
      "Index":0}}}

```

[Empty trailer]

Frame 1

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1}}}

```

```
{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "ChainDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYV
RVz9tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}}

```

Frame 2

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":2}}}

```

```
{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "ChainDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYV
RVz9tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}}

```

Frame 3

```

{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":3}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "ChainDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYV
RVz9tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}}}

```

14.3. Merkle Tree

The following example shows a chain sequence with a first frame and six data frames. The trailers now contain the TreePosition and TreeDigest values:

Frame 0

```

{
  "DareHeader":{
    "dig":"S512",
    "policy":{},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"Merkle",
      "Index":0}}}

```

[Empty trailer]

Frame 1


```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1,
      "TreePosition":0}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYVR
Vz9tn_KQc6GdA-P4VSRigBygV650Ed2Vv3YDhww"}}
```

Frame 2

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":2,
      "TreePosition":392}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"7fHmkEIsPkN6sDYAOLvpIJn5Dg3PxDDAaq-1l2kh8722kok
kFnZQcYtjuVC71aHNXI18q-lPnfRkmwryG-bhqQ"}}
```

Frame 3

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":3,
      "TreePosition":392}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYVR
Vz9tn_KQc6GdA-P4VSRigBygV650Ed2Vv3YDhww"}}
```

Frame 4

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":4,
      "TreePosition":1676}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"vJ6ngNATvZcXSMALi5IUqz11GBxBnTNVcC87VL_BhMRCbAv
KSj8gs0VFgxxLkZ2myrtaDIwhHoswiTiBMLNWug"}}
```

Frame 5

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":5,
      "TreePosition":1676}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYVR
Vz9tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}}
```

Frame 6

```

{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":6,
      "TreePosition":2963}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"WgHlz3EHczVPqgtpc39Arv7CFIsCbFVsk8wg0j2qLlEfur9
SZ0mdr65Ka-HF0Qx8gg_DAOiJwUrwADDXyxVJOg"}}}

```

14.4. Signed sequence

The following example shows a tree sequence with a signature in the final record. The signing key parameters are:

```

{
  "PrivateKeyECDH":{
    "crv":"Ed25519",
    "Private":"i-OxFUS-3GO0ftVIy3TYOAPWBjOnz3ZUNqLEdIx7stA"}}}

```

The sequence headers and trailers are:

Frame 0

```

{
  "DareHeader":{
    "dig":"S512",
    "policy":{},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"Merkle",
      "Index":0}}}

```

[Empty trailer]

Frame 1

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1,
      "TreePosition":0}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"T7SlFcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjglTD6yMYVR
Vz9tn_KQc6GdA-P4VSRigBygV650Ed2Vv3YDhww"}}
```

Frame 2

```
{
  "DareHeader":{
    "dig":"S512",
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":2,
      "TreePosition":392}}}

{
  "DareTrailer":{
    "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZ
DlZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
    "TreeDigest":"7fHmkEIsPkN6sDYAOLvpIJn5Dg3PxDDAaq-1l2kh8722kok
kFnZQcYtjuVC7laHNXI18q-lPnfRkmwryG-bhqQ"}}
```

14.5. Encrypted sequence

The following example shows a sequence in which all the frame payloads are encrypted under the same base seed established in a key agreement specified in the first frame.

Frame 0

```
{
  "DareHeader":{
    "enc":"A256CBC",
    "kid":"EBQA-6HXW-N2VL-QJGQ-FC2P-IOYT-WPC6",
    "Salt":"6OYJju7Gj8yqWR7mTJRgoQ",
    "recipients":[{
      "kid":"MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
      "epk":{
        "PublicKeyECDH":{
          "crv":"Ed25519",
          "Public":"iSCiXJ6PL_Hk5EI9VI7V2bRrVgdcdf8LL2jLZDJvgVM"}},
      "wmk":"vlqR9frLpk1V4uhWdpibspUHGbWCxdJFgRD8nHQihRdi246AEc
kx2Q"}
    ],
    "policy":{
      "enc":"none",
      "dig":"none",
      "EncryptKeys":[{
        "PublicKeyECDH":{
          "crv":"Ed25519",
          "Public":"e1XlMJ7VH6j7i-tJsSDwwnbWHRU7wnQOnBrJZVHXiWo"}}
      ],
      "Sealed":true},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"List",
      "Index":0}}}

```

[Empty trailer]

Frame 1

```
{
  "DareHeader":{
    "enc":"A256CBC",
    "kid":"EBQM-WPVJ-LD2X-MEKW-XKL5-JUYV-6ES4",
    "Salt":"zdNLitcxA2_6kvUvMnr9EQ",
    "recipients":[{
      "kid":"MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
      "epk":{
        "PublicKeyECDH":{
          "crv":"Ed25519",
          "Public":"hS8D7duh7bP2DBugpDUS-yUs0U40fqBsrS5NKqgGgTk"}},
      "wmk":"BNwEbb9EX_3v0siNWZ5jBM3cZW67R6RI_EhA1P-FE5oy5C7eZ-
YwUQ"}
    ],
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1}}}

```

[Empty trailer]

Frame 2

```
{
  "DareHeader":{
    "enc":"A256CBC",
    "kid":"EBQD-6DCH-W3U5-R7FQ-GVZK-UYXM-GIA4",
    "Salt":"QUg6uXAODywqpNC1wYAjZg",
    "recipients":[{
      "kid":"MBUX-V4NE-VRJS-6NT7-6QKR-DE2W-QQBG",
      "epk":{
        "PublicKeyECDH":{
          "crv":"Ed25519",
          "Public":"I7G-lWAN78mny3E8VuklpznSL7GV-BH70yBQHMDtM4g"}},
      "wmk":"p2k1W_K9MdBNXj6rJ08yZrXdPhQHDo1Jnly8MOGosKLTNdJxb3
f4uQ"}
    ],
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":2}}}

```

[Empty trailer]

Here are the sequence 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).

```
f5 02 ef
f1 02 d8
f0 10
f0 00
ef 02 f5
f5 02 f9
f1 01 c3
f1 01 30
f9 02 f5
f5 02 f9
f1 01 c3
f1 01 30
f9 02 f5
```

The following example shows a sequence in which all the frame payloads are encrypted under separate key agreements specified in the payload frames.

Frame 0

```
{
  "DareHeader":{
    "policy":{
      "enc":"none",
      "dig":"none",
      "Sealed":true},
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "DataEncoding":"JSON",
      "ContainerType":"List",
      "Index":0}}}
```

[Empty trailer]

Frame 1

```
{
  "DareHeader":{
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":1}}}
```

[Empty trailer]

Frame 2

```
{
  "DareHeader":{
    "ContentMetaData":"e30",
    "SequenceInfo":{
      "Index":2}}}
```

[Empty trailer]

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.

Issue	Description
Signature	No examples are given of signing a container. Need individual, chain, tree. Leave notarized for notary draft.
Indexing	No examples are given of indexing a container
Archive	Should include a file archive example
File Path	Mention the file path security issue in the security considerations
Security Considerations	Write Security considerations
AES-GCM	Switch to using AES GCM in the examples
KMAC	Switch to using KMAC for KDF
Witness	Complete handling of witness values.
Schema	Complete the schema documentation

Table 1

17. Normative References

[draft-hallambaker-jsonbcd]

Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", Work in Progress, Internet-Draft, draft-hallambaker-jsonbcd-21, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-jsonbcd-21>>.

[draft-hallambaker-mesh-architecture]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>>.

[draft-hallambaker-mesh-security]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>>.

[draft-hallambaker-mesh-udf]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.", Work in Progress, Internet-Draft, draft-hallambaker-mesh-udf-15, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-udf-15>>.

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

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

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <<https://www.rfc-editor.org/rfc/rfc2315>>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <<https://www.rfc-editor.org/rfc/rfc3394>>.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <<https://www.rfc-editor.org/rfc/rfc4880>>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/rfc/rfc4949>>.

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

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/rfc/rfc6838>>.

- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/rfc/rfc7159>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/rfc/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/rfc/rfc7517>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/rfc/rfc7518>>.

18. Informative References

- [BLOCKCHAIN] Chain.com, "Blockchain Specification".
- [Davis2001] Davis, D., "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML", May 2001.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/rfc/rfc5652>>.
- [ZIPFILE] PKWARE Inc, "APPNOTE.TXT - .ZIP File Format Specification", October 2014.

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
ThresholdSecrets.com
20 April 2022

Mathematical Mesh 3.0 Part IV: Schema Reference
draft-hallambaker-mesh-schema-10

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.

[Note to Readers]

Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

This document is also available online at <http://mathmesh.com/Documents/draft-hallambaker-mesh-schema.html>.

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 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.

Table of Contents

1. Introduction	5
2. Definitions	5
2.1. Requirements Language	6
2.2. Defined Terms	6
2.3. Related Specifications	6
2.4. Implementation Status	6
3. Actors	6
3.1. Accounts	7
3.2. Device	10
3.2.1. Activation	11
3.2.2. Connection Assertion	16
3.3. Service	17
4. Catalogs	19
4.1. Access	21
4.1.1. Access Capability	22
4.1.2. Null Capability	23
4.1.3. Cryptographic Capabilities	24
4.1.4. Publication Capability	25
4.2. Application	26
4.2.1. Mail	26
4.2.2. SSH	33
4.3. Bookmark	36
4.4. Contact	36
4.5. Credential	40
4.6. Device	40
4.7. Network	40
4.8. Publication	41
4.9. Task	41
5. Spools	41
5.1. Outbound	42
5.2. Inbound	43
5.3. Local	43
5.4. Log	44
6. Logs	44
7. Cryptographic Operations	44
7.1. Key Derivation from Seed	44
7.2. Message Envelope and Response Identifiers.	45
7.3. Proof of Knowledge of PIN	46
7.4. EARL	48
8. Mesh Assertions	48

8.1.	Encoding	48
8.2.	Mesh Profiles	49
8.3.	Mesh Connections	50
8.4.	Device Pre-configuration	51
9.	Architecture	54
9.1.	Mesh Account	55
9.1.1.	Account Profile	55
9.2.	Device Management	56
9.2.1.	The Device Catalog	56
9.2.2.	Mesh Devices	57
9.3.	Mesh Services	58
9.4.	Mesh Messaging	59
9.4.1.	Message Status	59
9.4.2.	Four Corner Model	60
9.4.3.	Traffic Analysis	60
10.	Publications	61
10.1.	Profile Device	61
10.2.	Contact Exchange	61
11.	Schema	61
11.1.	Shared Classes	61
11.1.1.	Classes describing keys	61
11.1.2.	Structure: KeyData	61
11.1.3.	Structure: CompositePrivate	62
11.2.	Assertion classes	62
11.2.1.	Structure: Assertion	62
11.2.2.	Structure: Condition	62
11.2.3.	Base Classes	62
11.2.4.	Structure: Connection	63
11.2.5.	Structure: Activation	63
11.2.6.	Structure: ActivationEntry	63
11.2.7.	Mesh Profile Classes	63
11.2.8.	Structure: Profile	63
11.2.9.	Structure: ProfileDevice	63
11.2.10.	Structure: ProfileAccount	64
11.2.11.	Structure: ProfileUser	64
11.2.12.	Structure: ProfileGroup	65
11.2.13.	Structure: ProfileService	65
11.2.14.	Structure: ProfileHost	65
11.2.15.	Connection Assertions	65
11.2.16.	Structure: ConnectionDevice	65
11.2.17.	Structure: ConnectionApplication	66
11.2.18.	Structure: ConnectionGroup	66
11.2.19.	Structure: ConnectionService	66
11.2.20.	Structure: ConnectionHost	66
11.2.21.	Activation Assertions	66
11.2.22.	Structure: ActivationDevice	66
11.2.23.	Structure: ActivationAccount	67
11.2.24.	Structure: ActivationApplication	67

11.3.	Data Structures	67
11.3.1.	Structure: Contact	67
11.3.2.	Structure: Anchor	68
11.3.3.	Structure: TaggedSource	68
11.3.4.	Structure: ContactGroup	68
11.3.5.	Structure: ContactPerson	68
11.3.6.	Structure: ContactOrganization	68
11.3.7.	Structure: OrganizationName	69
11.3.8.	Structure: PersonName	69
11.3.9.	Structure: NetworkAddress	69
11.3.10.	Structure: NetworkProtocol	70
11.3.11.	Structure: Role	70
11.3.12.	Structure: Location	70
11.3.13.	Structure: Bookmark	70
11.3.14.	Structure: Reference	71
11.3.15.	Structure: Task	71
11.4.	Catalog Entries	71
11.4.1.	Structure: CatalogedEntry	71
11.4.2.	Structure: CatalogedDevice	72
11.4.3.	Structure: CatalogedPublication	72
11.4.4.	Structure: CatalogedCredential	73
11.4.5.	Structure: CatalogedNetwork	73
11.4.6.	Structure: CatalogedContact	73
11.4.7.	Structure: CatalogedAccess	73
11.4.8.	Structure: CryptographicCapability	73
11.4.9.	Structure: CapabilityDecrypt	74
11.4.10.	Structure: CapabilityDecryptPartial	74
11.4.11.	Structure: CapabilityDecryptServiced	74
11.4.12.	Structure: CapabilitySign	74
11.4.13.	Structure: CapabilityKeyGenerate	75
11.4.14.	Structure: CapabilityFairExchange	75
11.4.15.	Structure: CatalogedBookmark	75
11.4.16.	Structure: CatalogedTask	75
11.4.17.	Structure: CatalogedApplication	75
11.4.18.	Structure: CatalogedMember	76
11.4.19.	Structure: CatalogedGroup	76
11.4.20.	Structure: CatalogedApplicationSSH	76
11.4.21.	Structure: CatalogedApplicationMail	76
11.4.22.	Structure: CatalogedApplicationNetwork	76
11.5.	Publications	76
11.5.1.	Structure: DevicePreconfiguration	76
11.6.	Messages	77
11.6.1.	Structure: Message	77
11.6.2.	Structure: MessageError	77
11.6.3.	Structure: MessageComplete	77
11.6.4.	Structure: MessagePinValidated	77
11.6.5.	Structure: MessagePin	77
11.6.6.	Structure: RequestConnection	78

11.6.7.	Structure: AcknowledgeConnection	78
11.6.8.	Structure: RespondConnection	78
11.6.9.	Structure: MessageContact	79
11.6.10.	Structure: GroupInvitation	79
11.6.11.	Structure: RequestConfirmation	79
11.6.12.	Structure: ResponseConfirmation	79
11.6.13.	Structure: RequestTask	79
11.6.14.	Structure: MessageClaim	79
11.6.15.	Structure: ProcessResult	80
12.	Security Considerations	80
13.	IANA Considerations	80
14.	Acknowledgements	80
15.	Normative References	80
16.	Informative References	81

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 entries and messages and their 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. Actors

The Mesh mediates interactions between three principal actors: `*Accounts*`, `*Devices*`, and `*Services*`.

Currently two account types are specified, `*user accounts*` which belong to an individual user and `*group accounts*` that are used to share access to confidential information between a group of users. It may prove useful to define new types of account over time or to eliminate the distinction entirely. When active a Mesh account is bound to a Mesh Service. The service to which an account is bound MAY be changed over time but an account can only be bound to a single service at a time.

A Mesh account is an abstract construct that (when active) is instantiated across one or more physical machines called a device. Each device that is connected to an account has a separate set of cryptographic keys that are used to interact with other devices connected to the account and MAY be provisioned with access to the account private keys which MAY or MAY NOT be mediated by the current Mesh Service. A user's Mesh accounts and the devices connected to them constitute that user's Personal Mesh.

A Mesh Service is an abstract construct that is provided by one or more physical machines called Hosts. A Mesh Host is a device that is attached to a service rather than an account.

3.1. Accounts

A Mesh Account is described by a Profile descended from Profile Account and contains a set of Mesh stores. Currently two account profiles are defined:

ProfileUser Describes a user account.

ProfileGroup Describes a group account used to share confidential information between a group of users.

Both types of profile specify the following fields:

ProfileSignature The public signature key used to authenticate the profile itself

AccountAddress The account name to which the account is currently bound. (e.g. `alice@example.com`, `@alice`).

ServiceUdf If the account is active, specifies the fingerprint of the service profile to which the account is currently bound.

AdministratorSignature The public signature key used to verify administrative actions on the account. In particular addition of devices to a user account or members to a group account.

AccountEncryption The public encryption key for the account. All messages sent to the account MUST be encrypted under this key. By definition, all data encrypted under this account is encrypted under this key.

User accounts specify two additional public keys, AccountSignature and AccountAuthentication which allow signature and authentication operations under the account context.

Every account contains a set of catalogs and spools that are managed by the service as directed by the contents of the associated Access catalog.

For example, the personal account profile Alice created in

For example, Alice creates a personal account:

```
Alice> meshman account create alice@example.com  
Account=alice@example.com  
UDF=MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF
```

The account profile created is:

```

{
  "ProfileUser":{
    "ProfileSignature":{
      "Udf":"MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"ni85QjaM8wU5vRoKmwNxD0F9c4SK303Mk0Gad5WlJ8hgB
iYWw9oNzmi32sw8XAMer6UM0SoTc24A" } } },
      "AccountAddress":"alice@example.com",
      "ServiceUdf":"MDSK-EUHS-QXGD-LKOF-AVC7-V2RH-LV6Z",
      "EscrowEncryption":{
        "Udf":"MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"tR85RCqWv8-X5Bk0NU4EVljQFJ585FNE3ZwyWzXSVtJHi
x0FZ7jZQ7xg9uurw8KOKl5M0UW7LLOA" } } },
        "AdministratorSignature":{
          "Udf":"MBDV-XXNH-2RUB-RBMZ-5NG7-L3CD-3THV",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"Ed448",
              "Public":"HUwN4RVhGczFlOm2bDcevvVYyd6gjdq33QqV8Uq39dGas
RzQn9_PVgCBRI_8MjiverTKdaaEI32A" } } },
          "CommonEncryption":{
            "Udf":"MDPR-FJWV-GK5Z-2LJA-LMYV-XSCH-HE2C",
            "PublicParameters":{
              "PublicKeyECDH":{
                "crv":"X448",
                "Public":"55jUkmqn3gwG0b2HzDVu3Hlf5sO6GgVlj_vaYFwAEksDc
My3wyvUwt9ojkeUKT6304Dwfrh-Uw8A" } } },
            "CommonAuthentication":{
              "Udf":"MBVI-EWLO-EI7J-OVAK-GGZH-6YHW-ZJSU",
              "PublicParameters":{
                "PublicKeyECDH":{
                  "crv":"X448",
                  "Public":"fTU3TeB1-7K8SZpo4tQxZPpJAb-_d3NIdJhlkxWaiZogJ
REK9adPf9Kns5mqr11UTToIMhzfdJaA" } } },
              "CommonSignature":{
                "Udf":"MAMP-BX4G-AKK2-YHPA-IXJV-Z2KV-UXBW",
                "PublicParameters":{
                  "PublicKeyECDH":{
                    "crv":"Ed448",
                    "Public":"Y6-D2DbbKlaVXvG5ZQweLd5_kP1ECACR40bDmpg-Y4Ks9
2FNe-uysWUrM_LmQKOIPjJr5L8NOBEA" } } } } }

```

3.2. Device

Every Mesh device has a set of private keys that are unique to that device. These keys MAY be installed during manufacture, installed from an external source after manufacture or generated on the device. If the platform capabilities allow, device private keys SHOULD be bound to the device so that they cannot be extracted or exported without substantial effort.

The public keys corresponding to the device private keys are specified in a ProfileDevice. This MUST contain at least the following fields:

ProfileSignature The public signature key used to authenticate the profile itself.

Encryption Public encryption key used as a share contribution to generation of device encryption keys to be used in the context of an account and to decrypt data during the process of connecting to an account.

Authentication Public authentication key used as a share contribution to generation of device authentication keys to be used in the context of an account and to authenticate the device to a service during the process of connecting to an account.

Signature Public signature key used as a share contribution to generation of device signature keys to be used in the context of an account.

For example, the device profile corresponding to one of the devices belonging to Alice is:

```

{
  "ProfileDevice":{
    "ProfileSignature":{
      "Udf":"MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"pwa2YXUVQCKc31N0BL1_aSo270xT1Qo37IW6HWadhTx-b
wqFEvdbZJ4UnjPjabKFLPs3NeXj77yA"}}},
      "Encryption":{
        "Udf":"MAAB-KTVM-DBAR-H2MO-BR65-7ADT-MLLA",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"pD72qIWSU1Z51BA0C220t-ZgE22uhnBP77VZz4gsiBj_8
8XnpfK33J34WuKorrW32CZe_-SkqviA"}}},
        "Signature":{
          "Udf":"MBQA-M2E2-PZPR-GIM3-JCRJ-QDDC-NJXL",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"Ed448",
              "Public":"HwultsJThxHMvig7PhCBnjgEYY9r7Ima0uYyKkYY5kwB9
iD4K30jiEomSrdWFpOz6I4j_wWsFKsA"}}},
          "Authentication":{
            "Udf":"MDRS-RHS6-4XIE-34VE-2ZLM-GKWL-VMMN",
            "PublicParameters":{
              "PublicKeyECDH":{
                "crv":"X448",
                "Public":"ywFQCwMmLGNTUZh-py5Oef30Dy9j8CwWIVCZAPsuWowfM
EUjROOPFF3q2NAg0PI3Lq87bPaUdrQA"}}}}}}

```

3.2.1. Activation

The device private keys are only used to perform cryptographic operations during the process of connecting a device to an account. During that connection process, a threshold key generation scheme is used to generate a second set of device keys bound to the account by combining the base key held by the device with a second device private key provided by the administration device approving the connection of the device to the account. The resulting key is referred to as the device key. The process of combining the base keys with the contributions to form the device keys is called Activation.

For example, Alice connects the device whose profile is shown above to her account:

```

Alice2> meshman device complete
Device UDF = MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS
Account = alice@example.com
Account UDF = MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF

```

The activation record granting the device rights to operate as a part of the account is:

```

{
  "ActivationAccount":{
    "ActivationKey":"ZAAQ-GK4S-YPOU-UMUP-MVLX-2FO3-QN7Y-UFQK-HEPB-R
Y4L-WSUB-2NYA-VW5G-VFL5",
    "AccountUdf":"MA75-5N5Q-BPQF-5LMP-AN6X-NM4E-U4KS" } }

```

And:

```

{
  "ActivationCommon":{
    "Entries":[{
      "Resource":"MMM_Contact",
      "Key":{
        "Udf":"MDRE-XGSY-3FEQ-7JBD-5CGH-QTC7-TZEV",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"rJfOVOiZXyn_r--rH7gff3rIQFbtYMnhHsKQFYibG
1R9W-RSXUIjHZfBx4F94e7FSe3Qi9wIb1CA" } },
          "PrivateParameters":{
            "PrivateKeyECDH":{
              "crv":"X448",
              "Private":"8yGWHtYjzkzgrOEs13qsqZmS93diaEiFxp2IBMJJ7
M3-LF_SfVv02Wzp3_557yaPh6UOYE7K2r-I" } } } },
      {
        "Resource":"MMM_Publication",
        "Key":{
          "Udf":"MDAV-VMHF-DRT3-BKDL-Y2HL-GLK2-GHEW",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"X448",
              "Public":"Igv5xtCKBflhcoMivJ-sBYXF8-sn4AuTe_lzgHzKq
_8wyiejH7-QEzrOIuxOvnFaTphUM24DXqYA" } },
          "PrivateParameters":{
            "PrivateKeyECDH":{
              "crv":"X448",
              "Private":"EhcNnOUIsxV3XIdx-7Dcy8_bSPHyXv6YN1Sr-OLp
5EPV22v5GRNQ63-4RWPe2AwGowo-JO9LQCU" } } } },
      {
        "Resource":"MMM_Inbound",

```

```
"Key":{
  "Udf":"MCPS-VCZ3-XVV5-PBAI-QN5B-CF6E-A75G",
  "PublicParameters":{
    "PublicKeyECDH":{
      "crv":"X448",
      "Public":"P6u7tVLfiqyvACUQJWiu_P36h38sHXcXbaVqL5nh
wVE7g9w6IAmP22cBm-omewfEdpZN7rR1bqA" }},
    "PrivateParameters":{
      "PrivateKeyECDH":{
        "crv":"X448",
        "Private":"ItPikKDnBdWk1bzKw10zc4H1g9L96MvbVrWS1L2o
PKg-kVtak8idY3jblfetl_WpEK8lf5mi2AI" }}}} ,
{
  "Resource":"MMM_Outbound",
  "Key":{
    "Udf":"MAFH-RDQT-JWOQ-INJX-WEBR-J3HJ-M7TZ",
    "PublicParameters":{
      "PublicKeyECDH":{
        "crv":"X448",
        "Public":"LJi26ccidXyLIetfl5nwtKa1pYOeYhB9XkxUpPYxJ
wleIq06pwYX14PRxdWKvpm4vMx4V_W1Tk4A" }},
      "PrivateParameters":{
        "PrivateKeyECDH":{
          "crv":"X448",
          "Private":"v3Ra_hFcIHgRavFA_BKTw5jaHlFN3RR7GNwX8lXX
BEymY7Jn42Zu2r9RYIxJqbf25pvbmbpN03ME" }}}} ,
{
  "Resource":"MMM_Network",
  "Key":{
    "Udf":"MABR-5WCQ-TNIL-GJGY-JLEH-F477-ABMS",
    "PublicParameters":{
      "PublicKeyECDH":{
        "crv":"X448",
        "Public":"s-RYfBg8hEC4BPRWnDR-DSa3PblqNzcozSSugzBwB
XOVUti4jDJBof5naUr5cbtabSj0EsgLZmkA" }},
      "PrivateParameters":{
        "PrivateKeyECDH":{
          "crv":"X448",
          "Private":"KhYa4lUPXaVe8fQL4fzbch7-yhMhEf7LBacgxMdB
kqJ-8F1arGILPArRAF3Ib4MvphwEafcEsIA" }}}} ,
{
  "Resource":"MMM_Application",
  "Key":{
    "Udf":"MBXU-FZUK-KFH2-72J2-Q4N2-A7AB-25GF",
    "PublicParameters":{
      "PublicKeyECDH":{
        "crv":"X448",
        "Public":"lwu54QyqdNjLWQHIRdZ3_bpv9JKuoJDtyCG0lWghA
```



```

xOt4toLqrdsWrC0qqZnt3edJKJKJ8m6O8CA"}},
  "PrivateParameters":{
    "PrivateKeyECDH":{
      "crv":"X448",
      "Private":"iOUDwM6SAetyLJMC6uG_3CIWFOpWAMy9mZC1lWSL
dzkiSfnwWhz2a0NQ5bwTRLyDAYyccBx_s7c"} } } },
  {
    "Resource":"MMM_Credential",
    "Key":{
      "Udf":"MDG5-EPRO-L3LG-GGFU-WKSG-EXU3-GGAB",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"X448",
          "Public":"INLLEYrLIPzFvcrxknMiC6CBWpZbn8i6PkyYrTWdK
adc8DqCQ1PaW0gayF-Fjyh2nAl0sTJu8nqA"} } },
      "PrivateParameters":{
        "PrivateKeyECDH":{
          "crv":"X448",
          "Private":"csfYsFBXA0qawDzo5kA8lCku_yV-jv9tro0yNvNv
740-gzNM1jaRRNdplxXu0ltgWDa3gEmwNRC"} } } },
    {
      "Resource":"MMM_Task",
      "Key":{
        "Udf":"MAT6-WUMY-SZ3J-ZREZ-RLV2-YJQ5-ASC7",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"_WW7zyQPuEJLZ7oVTd_EccJwZ1Ld5KAIdkW2RBBG1
btoVF1Gpna4yr4qVlgSrR0driZDaZUJIDaA"} } },
        "PrivateParameters":{
          "PrivateKeyECDH":{
            "crv":"X448",
            "Private":"JWzHJH5yBYhltzkWtL2H4N2svYfem3p8oiB129
Mqz59t87R1jctfhl9kwwilllPF54xJmXmTg"} } } },
      {
        "Resource":"MMM_Bookmark",
        "Key":{
          "Udf":"MBQJ-3DZR-GNXB-W3UQ-P620-G4RK-HX20",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"X448",
              "Public":"n27OoqLidzKr9ju-p9jY-0R4vsKyUy_5e6lak4_kC
aG1Mr5jroj-w7y4VrybGm-NfGEyY-UMBkMA"} } },
          "PrivateParameters":{
            "PrivateKeyECDH":{
              "crv":"X448",
              "Private":"1IeSz2X9UOCME9mQF37f_8RziEV3LVBQgDxVZNgb
yh0YWATkwGQM17l2oXYYaWm2zdY6Bu8r7uE"} } } }

```

```

    ],
    "Encryption":{
      "Udf":"MDPR-FJVW-GK5Z-2LJA-LMYV-XSCH-HE2C",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"X448",
          "Public":"55jUkmqn3gwG0b2HzDVu3Hlf5sO6GgVlj_vaYFwAEksDc
My3wyvUwt9ojkeUKT6304Dwfrh-Uw8A"}},
        "PrivateParameters":{
          "PrivateKeyECDH":{
            "crv":"X448",
            "Private":"14xBD_TtMiv4VXLfv53eQqAXkGzDsI5dl5IZekWy4Yi8
uTPw35kXuzIhgNvw1REvfU2JdBVh3wo"}},
          "Authentication":{
            "Udf":"MBVI-EWLO-EI7J-OVAK-GGZH-6YHW-ZJSU",
            "PublicParameters":{
              "PublicKeyECDH":{
                "crv":"X448",
                "Public":"fTU3TeBl-7K8SZpo4tQxZPpJAb-_d3NIdJhlkxWaiZogJ
REK9adPf9Kns5mqr1lUTToIMhzfdJaA"}},
              "PrivateParameters":{
                "PrivateKeyECDH":{
                  "crv":"X448",
                  "Private":"LqBnHkzzISgjBeoCMjlxXlp_pnrZ8Cdfn0kMTzIUf4tL
IvwRIueHQEYWP5_nvYSmYbMrJCWUA0U"}},
                "Signature":{
                  "Udf":"MAMP-BX4G-AKK2-YHPA-IXJV-Z2KV-UXBW",
                  "PublicParameters":{
                    "PublicKeyECDH":{
                      "crv":"Ed448",
                      "Public":"Y6-D2DbbKlaVXvG5ZQweLd5_kP1ECACR40bDmpg-Y4Ks9
2FNe-uysWUrM_LmQKOIPjjr5L8NOBEA"}},
                    "PrivateParameters":{
                      "PrivateKeyECDH":{
                        "crv":"Ed448",
                        "Private":"IAfy3NjVxhiNYFt16w5A99iy3TqCByxQLb9l5WoWlxN5
pjHzHeH9Ibr3n22suIvvsctPdfPAeeo"}}}}}

```

The Mesh protocols are designed so that there is never a need to export or escrow private keys of any type associated with a device, neither the base key, nor the device key nor the contribution from the administration device.

This approach to device configuration ensures that the keys that are used by the device when operating within the context of the account are entirely separate from those originally provided by the device manufacturer or generated on the device, provided only that the key contributions from the administration device are sufficiently random and unguessable.

3.2.2. Connection Assertion

The administration device combines the public keys specified in the device profile with the public components of the keys specified in the activation record to calculate the public keys of the device operating in the context of the account. These public keys are then used to create a ConnectionDevice and a ConnectionService assertion signed by the account administration signature key.

The ConnectionDevice assertion is used by the device to authenticate it to other devices connected to the account. This connection assertion specifies the Encryption, Authentication, and Signature keys the device is to use in the context of the account and the list of roles that have been authorized for the device..

```
{
  "ConnectionDevice":{
    "Authentication":{
      "Udf":"MAVT-XX2Y-B6D2-7SJ3-VVTW-MQ6C-S2MU",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"X448",
          "Public":"yrFrem_3XKqaQAvnlTxaZ2msYD-dBceF8NOssaE7BS5bh
BD_ViasKtPXFncsZ-4LdAjpHE2bWKIA" } } },
      "Roles":["message",
        "web"
      ],
      "Signature":{
        "Udf":"MCJE-YAQI-I4OU-EXTX-CQ5W-IORV-HKH4",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"Ed448",
            "Public":"ayCD-NfNvsgHfxy4lyDEysG8PD36zgLq1AZmh86_R4qY2
IpzPiymPiunbLL-pRZy8pPKDiwHPG0A" } } },
      "Encryption":{
        "Udf":"MDCW-SXW2-ROVU-4R3G-E5R3-2JGI-YBPF",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"Un-_awwiGjXgaO99A66zrVJwUilnUaYAuftP4HTmsnZg_
fhq3Z0rcja-z-er-BbJ9MHAqf3TxfyA" } } } } }
```

The ConnectionService assertion is used to authenticate the device to the Mesh service. In order to allow the assertion to fit in a single packet, it is important that this assertion be as small as possible. Only the Authentication key is specified.

The corresponding ConnectionService assertion is:

```
{
  "ConnectionService":{
    "Authentication":{
      "Udf":"MAVT-XX2Y-B6D2-7SJ3-VVTW-MQ6C-S2MU",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"X448",
          "Public":"yrFrem_3XKqaQAvnlTxaZ2msYD-dBceF8NOssaE7BS5bh
BD_ViasKtPXFncsZ-4LdAjpHE2bWKIA"}}}}}
```

The ConnectionDeviceassertion MAY be used in the same fashion as an X.509v3/PKIX certificate to mediate interactions between devices connected to the same account without the need for interaction with the Mesh service. Thus, a coffee pot device connected to the account can receive and authenticate instructions issued by a voice recognition device connected to that account.

While the ConnectionDeviceassertion MAY be used to mediate external interactions, this approach is typically undesirable as it provides the external parties with visibility to the internal configuration of the account, in particular which connected devices are being used on which occasions. Furthermore, the lack of the need to interact with the service means that the service is necessarily unable to mediate the exchange and enforce authorization policy on the interactions.

Device keys are intended to be used to secure communications between devices connected to the same account. All communication between Mesh accounts SHOULD be mediated by a Mesh service. This enables abuse mitigation by applying access control to every outbound and every inbound message.

3.3. Service

Mesh services are described by a ProfileService. This specifies the encryption, and signature authentication keys used to interact with the abstract service.

```

{
  "ProfileService":{
    "ProfileSignature":{
      "Udf":"MDSK-EUHS-QXGD-LKOF-AVC7-V2RH-LV6Z",
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"Ed448",
          "Public":"UuWD8qxdeqk6pyWkoz63qBpJPCcZOb-hySYQb_Lx5fGfY
OoU4gB7V6VauAfG-uIBDBMqglQmcGQA"}}},
      "ServiceAuthentication":{
        "Udf":"MDAL-ZI5N-4UKZ-H6VL-F25K-PHNF-ZUVA",
        "PublicParameters":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"d3bn_-qEVwBM69Z93Kabn3MqSnc9GQDlFT2_Rcx5tVRme
b_bjy7lvSRsk3ZP04Dj2cUBM4Agr-oA"}}},
        "ServiceEncryption":{
          "Udf":"MA4K-EVCK-36OZ-UHSQ-SHLK-36N3-YW7L",
          "PublicParameters":{
            "PublicKeyECDH":{
              "crv":"X448",
              "Public":"P_owWgt7wdtuvcsGCPfQo8uF5CFXG2RPwcTBlKZqx0VIf
9hpMdeyuAjRMFeE5_3nRm0ywL6tkUQA"}}},
          "ServiceSignature":{
            "Udf":"MAC3-YJSU-42F3-BB4L-T47H-VF6M-4IXM",
            "PublicParameters":{
              "PublicKeyECDH":{
                "crv":"Ed448",
                "Public":"_pT0cmw66uaQbd0QhE15yUtmlUDsdoZ1zLtGrqNnDfTbh
Q8qUqDlpPG4fszIFa9viKYE90CBA2EA"}}}}}}

```

Since Mesh accounts and services are both abstract constructs, they cannot interact directly. A device connected to an account can only interact with a service by interacting with a device authorized to provide services on behalf of one or more accounts connected to the service. Such a device is called a Mesh Host.

Mesh hosts MAY be managed using the same ProfileDevice and device connection mechanism provided for management of user devices or by whatever other management protocols prove convenient. The only part of the Service/Host interaction that is visible to devices connected to a profile and to hosts connected to other services is the ConnectionHost structure that describes the set of device keys to use in interactions with that specific host.

```
{
  "ConnectionService":{
    "Subject":"MBDH-L24Q-ZFNI-RSNS-AQ7Y-WGCQ-HRZ4",
    "Authority":"MDSK-EUHS-QXGD-LKOF-AVC7-V2RH-LV6Z",
    "Authentication":{
      "PublicParameters":{
        "PublicKeyECDH":{
          "crv":"X448",
          "Public":"_fdKvOXPYHKFFb8oljLKA3raIGkamEuL8beeoknQpBZVc
hhCv9QOGm47SBPow59_avyQuKO2fWSA"}},
        "Account":"@example"}}
```

Mesh Services MAY make use of the profile and activation mechanism used to connect devices to accounts to manage the connection of hosts to services. But this is optional. It is never necessary for a device to publish a ProfileHost assertion.

4. 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.

Access: mmm_Access Describes access control policy for performing operations on the account. The Access catalog is the only Mesh catalog whose contents are readable by the Mesh Service under normal circumstances.

Application: mmm_Application Describes configuration information for applications including mail (SMTP, IMAP, OpenPGP, S/MIME, etc) and SSH and for the MeshAccount application itself.

Bookmark: mmm_Bookmark Describes Web bookmarks and other citations allowing them to be shared between devices connected to the profile.

Contact: `mmm_Contact` Describes logical and physical contact information for people and organizations.

Credential: `mmm_Credential` Describes credentials used to access network resources.

Device: `mmm_Device` Describes the set of devices connected to the account and the permissions assigned to them

Network: `mmm_Network` Describes network settings such as WiFi access points, IPSEC and TLS VPN configurations, etc.

Member: `mmm_Member` Describes the set of members connected to a group account.

Publication: `mmm_Publication` Describes data published under the account context. The data MAY be stored in the publication catalog itself or on a separate service (e.g. a Web server).

Task: `mmm_CatalogTask` Describes tasks assigned to the user including calendar entries and to do lists.

The Access, and Publication catalogs are used by the service in certain Mesh Service Protocol interactions. The Device and Member catalogs are used to track the connection of devices to a user account and members to a group for administrative purposes. These interactions are further described below.

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.

4.1. Access

The access catalog `mmm_Access` contains a list of access control entries providing authorization to devices authenticated by a particular credential. The access catalog provides information that is necessary for the Mesh Service to act on behalf of the user. It is therefore necessary for the service to be able to decrypt entries in the catalog.

The entries in the catalog have type `CatalogedAccess` and specify a capability. The following capabilities are defined:

`NullCapability` A capability granting no access rights. May be used to establish a positive statement denying all access.

`AccessCapability` Authorizes a device authenticated by specified means to request privileged account operations. For example, requesting the status of an account catalog. Also used to provision devices with a copy of their `CatalogedDevice` entry encrypted under a key held by the device.

`CryptographicCapability` Specifies a private key encrypted under the encryption key of the service and criteria specifying the parties authorized to request use of the key.

`PublicationCapability` Authorizes a device authenticated by specified means to obtain a data item.

The Access catalog plays a central role in all operations performed by the service on behalf of the user.

Every access capability is gated by a specified set of authentication criteria. The following authentication criteria are currently defined:

`Profile Authentication Key` The account profile authentication key authorizes any account action without the need for an access catalog entry. This capability is normally only used during account binding. Administration devices SHOULD NOT have access to the account profile authentication key after binding is completed.

`Device Authentication Key` The service will only perform the operation if the device making the request presents the specified authentication key.

This form of authentication is necessary to restrict access to account operations so that only connected devices can interact with stores, etc.

Account Profile Identifier The service will only perform the operation if the device making the request presents an authentication key that is credentialed by a connection assertion to the specified account profile.

This form of authentication is necessary to perform administration operations on a group account since it is the account rather than the device that is authorized to perform the operation.

Proof of Knowledge The service will only perform the operation if proof of knowledge of the identified shared secret is provided.

This form of authentication criteria is used to allow device connection and contact exchange by means of static (i.e. printed) QR codes.

Future: Currently, the set of authentication criteria is limited to direct grants of a single capability to a single specified device or account. This approach may prove to be unnecessarily verbose requiring the same information to be repeated multiple times.

4.1.1. Access Capability

The access capability permits a specified service operation on the account. Optionally, an access capability MAY specify a Data entry encrypted to a key held by the device.

The access capability specifies the set of rights granted to the requester and optionally specifies an EnvelopedCatalogedDevice entry containing the CatalogedDevice entry for the device encrypted under the base encryption key or account encryption key of the device.

The CatalogedDeviceDigest value serves as a tag for the cached data.

4.1.1.1. Operation Rights

The reference code does not currently implement operation rights beyond denying all operations to devices that do not have an access capability entry.

Expansion of the rights handling is planned to permit granular expression of access rights.

mmm_o_UnbindAccount UnbindAccount

mmm_o_Connect Connect

mmm_o_Complete Complete

mmm_o_Status Status (of specified catalogs or all catalogs)
mmm_o_Download Download (of specified catalogs or all catalogs)
mmm_o_Transact Transact (of specified catalogs or all catalogs)
mmm_o_Post Post outbound message

4.1.1.2. Messaging

The reference code has limited messaging capabilities at present and messaging rights are not specified. The following is a list of possible rights:

mmm_m_Contact Contact messages from the specified subject.
mmm_m_Confirmation Confirmation messages from the specified subject.
mmm_m_Async Asynchronous delivery messages (e.g. mail)
mmm_m_Sync Synchronous delivery messages (e.g. chat)
mmm_m_Presence Forward presence request.

The following media are defined

mmm_c_Text Text that MUST NOT contain links or external references
mmm_c_Linked Text that MAY contain links or external reference
mmm_c_Audio Audio data (e.g. VOIP, voicemail)
mmm_c_Video Video data
mmm_c_Code Content containing active code including macros, scripts and executables.

4.1.2. Null Capability

The null capability is used to affirmatively deny access to a function. This allows access requests from previously authorized devices whose credentials have been revoked to be handled separately from requests from devices that were never authorized.

4.1.3. Cryptographic Capabilities

A Mesh Service can perform cryptographic operations on a private key according to access criteria specified by the user. This capability is used to support use of threshold cryptography to mitigate compromise of a particular device or individual. The splitting of a cryptographic key into two or more parts allows the use of that key to be split into two or more roles.

Note that this approach limits rather than eliminates trust in the service. As with services presenting themselves as 'zero trust', a Mesh service becomes a trusted service after a sufficient number of breaches in other parts of the system have occurred. And the user trusts the service to provide availability of the service.

A Mesh Service MAY also offer to perform private key operations for other purposes. An embargo agent might offer to decrypt data under a private key but only after a specified date and time. An expiry agent might offer to decrypt data but only before a specified date and time. Such services MAY be reserved to the customers of a specified service or provided to the general public. Users of such services MAY combine key services provided by multiple service providers using threshold techniques to achieve separation of roles.

Since a service might not willingly co-operate with an account transfer request, extension of the Mesh service protocol will be required to enable threshold sharing of the keys required to effect account transfer. This would require one administration device to act as a proxy for threshold signature etc. operations being requested by another administration device. While implementation of such a scheme to support this limited function could be achieved with little difficulty, such a scheme might not support the wider range of peer-to-peer threshold capabilities that might be useful. For example, the confirmation protocol might be modified so that instead of merely providing non-repudiable evidence of the user's response to a request, the confirmation device served as a policy enforcement point through control of a necessary threshold share.

The following service cryptographic operations are specified:

4.1.3.1. Threshold Key Share

A private key share s , held by the service is split into key shares x , y such that $a = x + y$. One key share is encrypted under a decryption key held by the service. The other is encrypted under a public key specified by the party making the request.

This operation is not currently implemented in the Reference code. When implemented, it will allow the functions of the administration device to be threshold shared between the device and the service, thus allowing the administration capability to be revoked if the device is lost, stolen or otherwise compromised.

Implementation of this capability is expected to be based on the scheme described in . [draft-komlo-frost]

4.1.3.2. Key Agreement

A private key share s , held by the service is used to calculate the value $(sl + c).P$ where l , c are integers specified by the requestor and P is a point on the curve.

This operation is used

4.1.3.3. Threshold Signature

A private key share s , held by the service is used to calculate a contribution to a threshold signature scheme.

The implementation of the cryptographic operations described above is described in [draft-hallambaker-threshold].

Implementation of signatures is not currently covered pending completion of [draft-irtf-cfrg-frost].

4.1.3.4. Fair Exchange

Perform a Micali Fair Exchange trusted intermediary operation.

On receipt of a signature $SIG_B(Z)$, where $Z=E_k(A, B, M)$, the service decrypts Z and returns the result to B .

4.1.4. Publication Capability

The publication capability is not currently implemented. Implementation would allow the Claim/PollClaim mechanism to be eliminated in favor of a mechanism capable of re-use for other purposes.

4.2. Application

The application catalog `mmm_Application` 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 the device activation record.

Two applications are currently defined:

Mail An SMTP email account and associated encryption and signature keys for S/MIME and OpenPGP.

SSH Secure Shell Client.

Accounts MAY specify multiple instances of each but each application instance is considered as describing a single application account. Thus, if Alice has email accounts `alice@example.com` and `alice@example.net`, she will have application entries for each. Accounts connected to Alice's Mesh account may be authorized to use either, both or none of the email accounts.

***Note*:** The implementation of these features in the current specification is considered to be a 'proof of concept' rather than a proposed final form. There are many issues that need to be considered when integrating a legacy protocol with extensive deployment into a new platform.

4.2.1. 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.

```

{
  "CatalogedApplicationMail": {
    "Key": "mailto:alice@example.net",
    "Grant": [ "web"
    ],
    "EnvelopedEscrow": [ [ {
      "enc": "A256CBC",
      "kid": "EBQL-UZXE-NDYQ-4ZWU-MD2J-ZRVB-VKMJ",
      "Salt": "YpZfSceDyFABMtX0EaezWQ",
      "recipients": [ {
        "kid": "MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
        "epk": {
          "PublicKeyECDH": {
            "crv": "X448",
            "Public": "-G2w5cKrAlMlTWkcds8EdD_Q9yXkkmVrroiG-
S7oupxqwWa8lj4D5lUzOSXYw4ppzS8Wivahq1SA" } },
          "wmk": "mNAKX_Hqp6ceS_sGCCmPrEUl9f-OlS_yP9NjePwvsbtC
BFewaljXHQ" }
        ] },
        "c2049fWM29aBRW-li7JCSch34qT9yLKa6-lt3bpQWXGr7P6iXbokuOje
bMaQy3hYRU72pXJGOUkUadfyMayRzHKoiQwOlFyCni6JpVwBvjsHZ41nXQKzZWh5c
nmIAMyB6Uspq8R_FKpKBf8QfzvDJcGTzOxTbfxRHxXMlD2WGsgZ4a4vWPedSOEX
MXHZ5C2KVYhvCOD5M0LDeinDO6RUQKaJrmIxe2OUP34N3d2wr_J9KaWyByAUOLRut
JvhloVa8EqIFcQ8jDoTI48le5fttEuHRPuaR8IB_ypsn3t8udgPDbIyZ_k3gNQV8_
LHflxeGls6GfeVU4WdHPylDFpU6pz0gGU_H06wbMMwhemwYWnOKD7zHs3pwxLp9k
t3Ez22s6-Gt8eVcAyTUoIv6wfeqZhkBO-K7sRlPgRKnL3MQDBwtX_s5bkJns3WBuL
1s8XExqt02j6zC2fa5Pnevvyq0xEfQpUMhYuQHi0trPuLzHWS82x7cfCIomZIIaOK
jgOxeA8f8lUgwf-PyFG5XZ3RPiVGBPIuhkj3Jkrall4sYP9p0byq3rByDwuXVeDhJ
CdoRiRfoKoDnEE72axBKDFHDusAum_RvKjVIOfvSj1K8u8hPz1IPfrUGX8Ont03yF
x3a7Vvw0Kngjw_3fAgmnJBAt4tNALKv6K650byAeyA7ePGST9aXRpmgZqeSHuDTD4
7LbpW9772R8IPrMPPOfB-z559sIuaKdO-Va_qKF0bDsdb-_DR_UFYc6j5kaz_CcQy
h-z7ENT1-5Cn8SZn6YJ_ppWGCv9BzY-zKRbMvY040Pm7kgHis6Ypz5uWNEuwUR_Xq
_F20ojdhPBD_V52quh8kZ5mr9Yhq5APnsNX8xe4qoBjbl2D1zeYwGPq2ycU5-vzpl
cA2LAUGJ9kAFBLimpn0uevlnJqKJmG9v0KglhByQJzyIVSn1Lb9F1CJfPMKrLZv-h
kukW5eoZuHdMUfKWBIP0UfHOuWaoLdOEdHo06FewH7CUGF1AkpGYCwSnI5cB4N0qF
QWDw6RMq390c46kaTeAUW6zVpMmQ5gIIt1sBR2yxpiS4m0bgfugTdEDSAKwvHQ_ix
ZE5pPw-D7nmvx7ubtBHLp3KSBm8qHdroiQRjPQHkBTsnoVExgJQBhcHAe18Ef5kDp
33TcRSoCW56K-CJBW8uGOp3MgeaEwWcR9psSdIWpfweIWI3uQLPfTk1VGBLrcNoTw
q2cDtrlWTTXuFmUNNyxyMoFfJ95yasEKFscz2HzPKqYN4eK9MWh3qs5fIj0Lmr-30
HhEbaky88mvszrXglZrNb3TJINSHSSVw1MH7M7o-jWdImMUowqSzdgvXyTVEPg1f
NTpIN3cWPD2WS_281j1CBZXo0K6cj-SkNxzzEvR5qg7pslatW_5ucehZXRN_MSARo
ffWDKFjF1rXi8Z0o5gqaKPniJ2rxIKIjbiGulE2ayeKB9A5PErAEJ44LKkC34fTiH
L5Xe_jiDDPiJfT8YUcWDRULvv8PcjQexAP9AOji-hLYB2pv6z36pl2z8JxqrjskiM
EtipMIJserHYFj3TqqNlv7PUiR5ifgSHB1B4Be20lT95T3W5TqZ0xEMW9LJKfw6YC
xp53HqqW7Fjc2r3mWmbldkqKunrr1ZnXKwszWMV97gzjTgc1y4iT0TndETW8ucT1T
DdeLae9IKFimsKgLOxkxBHT16ECSYZLOztW2E5uDF23l75bZocsnLGj7JW4TMkoJq
SODI7q2L46yhJCnBkctPABOFhi8HV_qz1Jl3Lep6VGCUXdeFDWSMwZKvbwHAFf01M
YSxhNavVycaGnIfcld4w_nLiJw2r-hLuH0zk_dy3H0sPD4h-dDyu7aCnD77Y0dFdK

```

```

yFWPHi41DOGrkN3XsNDihu72RNo1kEEqqbyRPxmFURJP8DnlxgJiaPdMBiDFpkJSS
olwLL1sOV4j6w8m1Dqox91rrZnpnBzANjS_BAGuLM4f7KyQ-JTIFPcybCgfbxnlaQ
cxejBDERoePNaddM_IeCQM9RRN0CTwJQZWw17at7nxafleQw3IKva2_3U525CA9M
oFHRSJ756jcZlkLrCoJOK0iuL0-ZLOmTRPsEFFAskMuDtD6t889dA3lueToKcClBJ
xurcvol_8LDl-oe7KyOyTEI3-wlviSiorJGtleVlnCyN5ZylH4rRE-6Tayz_23UR3
-JTNjky4mDnDCEsZINpiZKsZd3DyoNkThwV08mv4IsTYQj_sBzEjnPkaCZMZ2CNj
u8aqHmVqLyW3JWIkTxwIPihlZ0feM6faVhh1SmhubnBneAeJWb5vJw24dlh1I2U1T
odUEMFfNjthO6rUZjZtVzIhbi4ma3G_-eTJGCOg__7z12-V-a2wk7OH1eJvGB5HNg
O6vP55GTvnkSV8NL40kmABOY5Q33ePYxFK7TXnTlsYj3uqvHZw_-kDgy_U2MW3VTX
Fc8N5a7jFF1pwv2SWmjYcG7VXB6gqsTTBEcUMtVxc8w44RJdduHBeClk1UY36BHB3
UXt4eNctHsY6UNynzRzuMhw5cVU4vhmCinDdx_xPTbptWQzqtQd2ARWEDWl6BEze7
tft2upt70ol8hlRg0BQRH1yfwbmiGd35lwykRIKL2l4MAalrbCgPLz41lU5FMNBq
Jyn8irtjeNDTPm7BG6J4ehBz4pcDffIKsYtVbMT68iEl15unalolituFkyXV4UCHr
Dj4cWWjDa2eWgr8AKunYcTMZREfLsFxrTbmm6gQfz6gc6cxuDHLc0ItfjqmTp-Q5d
Bi17Wp2o18QQX7kgahxa6gNnTdV73nKroiWP2UhXgRctA-r1Dpal1QrMseVUZOs5W
k_9S0TW7EwnRG_ECGW8LhpJ6letVkSDP3mTV9XXCBfga71LSV2VPJKH4YvuVURD3U
f4oM0XCAY8kU1mP8sEXnmI4eJHmnWpH7LPOxTzst6PVyDXVRccF1Yjbc66Fxm07JR
qXYaJojdTgrsXvz9fKr_Mj0oRgNCab0GdNpPHD-hNpD3P-0lxxOUEbNFyp_A9jgg7
vfmBuKWWpezONsvCv40mU5JtQbMMOytVdny4jDZHkLdqBh6N6x1P6XGJt_qrDKmz2
_CvqcuUh93Ep4Hh9wWqOsJZvnjyP13d4OsOWbIPHZDKBauBR1ovbT_qY52IHZsPf0
3Cb8QxJt6qrTJ79ay59OKOR-HvKD0k9Tqo1svfBYc8z4WA"
  },
  [{
    "enc": "A256CBC",
    "kid": "EBQF-XDKF-XDI6-5LNG-AHLY-CH3I-KI2F",
    "salt": "e1J8nMW9M6gT_hNYx-UMZQ",
    "recipients": [{
      "kid": "MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
      "epk": {
        "PublicKeyECDH": {
          "crv": "X448",
          "Public": "8VUU5j0JTwCYCCieL003KrgKkvTb0L4jEpcTa
08SiDQiGvdDfr0rYMjI8QC1A4u40oJQ7Dy8guyA"
        },
        "wmk": "bg-KGIaWlB6GqEb4V1HPhE3fOed9qOsfkjY003UT1-a
5QXSBM3WDQ"
      }
    ]
  },
  "_4_1NBgGmLj1Zs1fhK8ZfqkDcV-qCcQAIjCx4ExUFkmhEyz0QEppi5Wo
Z4ovGnxXfML8xbxFUFd47_gcUvvDac1oK75t4PKf1xDUukZUQZ3Qn6zFIOI5u5JQN
0mOyRgCgZ6d-KCFpY_VOdntC2Lq1TtTb7yJ2Q-OgJ-eBi8yWVJcr0sG2zSXRnbz17
Hv_2QotUcC8TTzxduUIRuXGsc2Zhz4d84XCd0Pu8f4IfEYWAYxvK9S5hK7hiuP4Y
b-o7uDwCiNmGkNKHfG3vfyCsgNdHbaDoHszXOdArhDm-qDF4f1IpFW_snXTiK0KpF
vT1URzRmwfzdkQt-J9AOx__hoR4oMH7B2YAF7IPih94Bp2ORZm2NPgyalu2Gb4zqk
6VrRTjYwzhnuuDhpX59ik3SdyjimjOKjHy5GNoRVX9L2MZS4-1guMoJoSvjWIQtV
aEa8TXtfMyfR-EgKQRd05aAAEWIOKj500gBfIE840Js535TMb14w6PWquZnqyrAOo
wwkAWTibtMx7gcNrxkqU_buQWZHIV-1j5HQLRGAviYlbyKCjuTZ2Ktp5w0641P1l6
db1G6SDgf1OsfJdBhG61qLL9S0tfw0thiV2XuQq8uDwi2-A6qe80EQpC0RY1UEDlC
1ebjZk2I1bpMbJHe2aBQB4MroiOwgqRyzEcaxBqd8tji6bsIGie4r_7VvVmP01Z2a
gC78Fw_ToX7v8gwcM7f6ucSepWv165HgJYBD1lwjGoI5ANP2Zc12hPDpS96NKknpZ

```

PKuRqoL0l12r6cfXimPBjRUCvDF5EwTrqIk40h9R8U0fCfSk3VVPfm4-q41SUFPP
 TldlHQrgUSxLGvg64qzzJpEUHPH1xdhxh2xpFJ3PvzPkkM_X0ux_e_MmqNFzfktzo
 NXySStKv85ufWCIO-4-zX0SBkkXsVUNkStr6W_oUay2G_NNVNKQjLwtn4WfBEO0TV
 8DesfVP0J_K1k6mtLMmY0DvBm9BotqXnuSBHkL_7jsR-Qp6LXpM-N5jamWx4oCivr
 6d5sW0A2dXbZaNRKBGUUAac6CQLqn1TOIwLhmi08iYb7k_iEqnj1UxZr_ppAkk1F8
 vuSnOVv4-jfkhgADSmMmRx6XM6G9UVx0hVY43rxUv86cf1ZmcRyX-R-QPUOSY25K
 3nsIG2wpK93UvHyqlqoDX6LmH2WMI7biKJrMbKOPnTON49DDZBXp4ziFhcWif7iIO
 1K-QduVF1Xc-hbKYnq8haw9_Nb16SI5CB6ompq1wLUM5FxxUJmQp8KB3BQMOC3_R3j
 pKrPCUijY7Rxx3pdHN9TsdKDXyqHwR0wvAriN1A43La4Nkg9DlbvfiwR6LzdhP7gL
 oGqxBDot8g-hN9xEsE80iMugQDsJ8lF6D9J3SDadFuZAZ0FCsBwWj6_zUheedmo1L
 IE4wlNm55_4e93gwKZEou5DBu6z20YcrQVcR1EFPwZypXMH3AYDDGk-S58Xpk2xL_
 -H20LlRSx6dvs_M_xwoSZYZW9ntX5qaeydH2zZg3sCCOIamJT-xbfsVx2020ig08g
 gI6tgOnWZuHitRDibX0hzw2OXrx3ClLcqnft1aT9oVjMhjYwvlzwhKxJKBft9Xr1T
 gnwFjWuDW28jXN-A19QwnH1z55ZPS9pQSNwzfqlmBEBOTao0f7F4oN6DhM83NafUD
 RU9tAnLwHNPQZC-j3-UgOAbaY7G6Z6s8wMUX8uWYocQ2kUxVxh8vqrg20cKM2G3gU
 K-3JwL8dJrDKsWTXulacD7o1JDC-GrC7oZI_gzLixPABjOqdOzN0HFys5OqThUlhg
 nb4ATW05BORODzDY3cQHrFyscAuQs19300K0sBBNT7zD3xsC1whLuJkDxDNWJK67w
 FcipcRx_di4CtCRSNff9VHEvG_y3okPgXFPD7HX-KRdilSXTbC387tq9MOQ_a8xZD
 SCzkH6432_gwFYxctOISauSITFggcaR7iSdKyDMEwDNPcHFkHsrtrtQ91MBcc1wpo-
 jI5rm-Kma3pHJAhnOQ141fnKY00NliuYBhCWPvawTeafwa15pWvRak1SfiAT2iiBT
 mfH3TszxIH8McjWe4ySkQ9OWRBFwF9-CJxa3LlIWAAtvTytOS5DYPs94KprYbImam4
 SmmYOSgFfy77EZ7tevGWaihu-qdnS8Ue9t7Gcwt2EKvGOqlKkw_z16b702fHgfiic
 KsvDkzckVjDyWau3NX3bkmxNQLLS-SGXvXN4oOCsg3j-nvHgaQ_NBelDM-66Fw82M
 318soeZMXCnPokD0SfJ3LSJ8df_Wy3_lpCuzfFFVv-aC2EMbljaP933BmQ0zrBsJR
 UugGdzUFqGfw1wbDQ3hVKfghKrQQLQPRip5YbLhN_i13ipkthcWZGile1Rs8JCKN
 4xoyZDt84tPU1BzmKdpWsr0uxay0mSejgRrO7XZvBYL4-kWDuCnyfAZeiWl_nGnwK
 Np5VuxQ-x1lTlLZGKLohcZfWneWowI4FdB6ATAfQKHbDTbvbn5jJmky4elqVLL2zS
 P9nExiFoRp59AkjfiECyfox_aHWTU3dqyRtr4uSsqo7qNa1SyS92Qh8f33fFCGo6OA
 uq6yYc0h2qheJNqnYyGjV8TLdIVJ5pTmZvZjAJkSNGRpNS3BSK0mil83gWOjNF7rM
 MCR72aPOsJKjvNyX61If3B0XwaCVAL0PnwonpRaO9r4s1ntHG4rlyf2G15iYub1HK
 chpvXQFyfKhHtKvfyEV4_ggGz65teinIm1m4wzZ8L0521m71ECvIKo-xYF9t66_IV
 DqRG5kiNklmKwHMx2aTz79H_ioHg-hxfulou90N_IFIBawahDFVZEPmRxtNkp7zVd
 x_XKTgoY3XU60deTCxQdB0dSbd6mxpF3pEdwSguqO9khjmKKBgHy05LxaZSMGSbiR
 9fBj_7l0TinFl97zDTNz-dNjRTOMo8bwtLDJCZMCtqR97k1zJVs0SKhDwjMrTvFN9
 WMR5Q08bqo8MJAIDgR1wBCtHb_tm6cbm5tKxcwruArWggq7CxHTTgtP5xfu9JTGqQ
 RC251rHlZXGDizWDfuTDyv42K6OSgX9zkDV2Rmjfo8y6A0eBm45I5RJQ3UGd5u6kA
 xW-elPaX6n-MGgUhPu2OG9Oxx-TwxhbrwJ9YA1vG1KQC6ZCEg-TjUXUmbaDikP_4V
 SZ1131wS7LH5K0xD1vo4WqE4gS7Y9CN30diBTyviewJN69Q"

},

[{

```
"enc": "A256CBC",
"kid": "EBQJ-7FWU-YBJF-EK3B-DOJS-HVBX-RPLT",
"Salt": "e5O6pSFdqyKUJUcedkyblQ",
"recipients": [{
  "kid": "MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
  "epk": {
    "PublicKeyECDH": {
      "crv": "X448",
```



```
"Public": "NfjQmmZNM5i41TbIliFUwGsX9uTT9ngMqxprP
JVjOevSYfgtvOe9XeMN_n04z8KObjZg-CXOPuaA"}},
"wmk": "tPdFA95AShcouY4SKCPFltBeHnlo_nUzpuHuAeri015c
meM_1JlNoQ"}
}},
"chbmpCsHHclLNKgMICThaBnUq-dr2d_sVkc4CZqM6zVlRuNGEbjjeFOkb
0ToVNM2jPxhulK70x2jDpbMqSeG21ysv-UQv5PkwIDtbNp7OSPCBlhgghnyzlo1pj
6RccaM8OBLnEnWKb1T5yWwYeHWbTbHsIEeGA8t4_TVP8GFhPECtO1GduNKtso5iQl
bo83BEB507yWRzJIFBdWbvmTw6-jIWdx9lfwnU8oYwd-3URcVF07H5MIYdQhyKwwt
eNVoG0lmgdhsNDQfeEv4v6bqBckiwns54ta7pLo0zANAh6w3kyTX2ZiTs_WlPupDC
OBDKNKqSHttZXU-tCQzmlziXBtV2Z82CM0XyFibTYsF4GL-fzQhr6gQNIV4IWoxco
sYovhP2azlUmVaH86IWTzJd_6HdvXhv75obkMzbRi0WI_cXoJbh-NBDLJf7t83JGI
5DOAYYAib9y2Zd2bb-bptKQ4efM7zZx_PqzjwKckQAWil6yTU09mzsJ3HsckgRG_S
v5GOEXeR2eQsIxExxOPlbRziTLI87Kts0iL5exFfgT9RYzI-IOJY03B8xI0Tc-IFA
mWOjF092Wtgg4FU3B43zUyRI09hVSagCp03lEZIz7kIhNBf8-VnN6enM0-L8rtV00
-f8FDRDKPYBFgEpr6bUZk1aopJ4RspB_A64K4ukj0HXE9JVe-Bgfu2Vtzc0PjYiP3
xCWDc_6Sk5ezKYzgvFMuKkRSRzmwivCLHDCp4vUDdLMawhKXMnlkbGQAdiJZdTkg-
FQhd-KWB467kRbwmJn7ArXCMjt2cGmegOPBXPL7YhX7WP_FN2kcUUVBTuNj9VshMK
lvTieRLevlrlurpAjs0uQupphdhVA7tJljoPIXC8_tOr9ML0XTE5NVuYBA9PXg7Dh
i9_E9ywpTNEemzFI4ZHI1IbKhCx1-VMgs_wextW0X_otxAL94CrVl3sNNBLWJT018
xFkeVqeEoITEq2oKMVhmB3gWktuc5cIUyHB-a8KZGqVNmMctndRdmuX27G4eVnzsa
owXkZA5VpCoEIqUi-t5Z-DipsxZ_ts3kkh0E73cjDzXKxCTjjLhMGS3Kb86D_AKHP
MjRLlZsmDfI8Xh2S6LFxnIdmGwUdVSHGi_RQFEw9xdWAuf1tZM-Fg_CCBucP2Wkfo
zpvY_uk7QvCE7dlfznyIx-iC_uaIpmbt0eXZuy364cLA5IlgLEJPbRCfhiBEWNTJz
-n_QMvDskan5B5XJSpJHtwHIGF1WWJlWl17ELfLf3yZ7QD4i7qbLkw92t32v7zIgK
kzW745Ai10XjuzI4ANJk1SXk3zCvjgsTxxhM124DugCHdhaEs09U1F4U8UNPDj9US_
qy-JCm5LQWTyi3DUnJiKmIaThI9VcOeJenw8go57IRVqNbFrL8EyRfCXQnEH52SUP
T_LQNOP44pqLx-KL1FPvSMRH4XQGSQaqPkGW70m59huT7WhnPMqV8sbUM2ldc4E1X
MrqiPME1UFybC34BtklogIfJ-fvTQ_0UtMuneZjtnxe5tS7bmf295voyjHhWJLSNP
eSfdXRjfgnGoDjeCsBc4CqBk3104sX-HII_J9bKzJ6eqCpC55idVrIGdq-omwYZvq
dCybvjyoQk90KNykTF10pLhx0sMqbpMw8MwhaOnlMwYiLr_Ax37rB3iUBzEGFUSgI
X3x9GqTth15Sgk-WXEJ-_m9LPrJENwSgbcYkh3usKVBKRNjsVQfOsyU4iWGTP20bj
UyY_duwXdLt38aGVovQHs_N8caERZDqHx5lQrbjAmPmZj4ajsyVwa6IFIy-YA1imR
x4DwAWaz5lN8JSKNZaOjU8ncL5q5xHc4JEtDcgZV2BvreE_HudyyiEf_7BUc_gmay
KC-kc3IJYyZsZAIFC3QSpLi5PacXUczT3qwmTFRvjwCPQ_tB5tfSiIh70IFRw7jPc
69zri77VMWx7QzWwkFc0a18ZvtgslYHgCTU_Gk4atELJF1DspQDI00WPUrZxk3-oD
Dhe64Vut8zEYiDCSuFBzzpjy_QEAS29i5VeR7WoiridKwk3OPjjhsaHEyTGxyQXlk
Oyd4dD8XBDTIvtCeboXoy9L29sjCUPUSJLcupvFLORNy-1aLqDF3HImbZvZaJvtBB
dautoRohBx8ru2eBx-4AXooeT_BrmBdudgSym0030-Yns3gqUhfZSUiBb2jjTvVNo
xt2jc_78SXVI7F82HAEKnTebaDzXA7UWv7akN0YpgHv_QYEJHCyEVP1tdYQqmju7f
9A2AqboZA_RHKucaotOqL9xNOzfQCjFLQxptFV2JHCnqTQBi_wroCy09R9zdbS3J
E1EEVUKIUDI_LBi9vRAVUtGKDVRsisEp5D3uwZgBV_Ebe86g2CwCXuT2Hxh7KjC59
YNhm09bqIqaETbJWkUa_QMQTizhDTdDTRRkOC46lFADpibvYZT5ca4QJnRmKYEdiq
sssF02bZpngmyqz-_g9hUAuu-S26kByY13M118RaPe1ZnaspNdgvefFW7jSI5_cNG
IKsdVaqyF1hOqla5G7ZUcKgorulG8wfv0IGJ3YCEBaSFkce5SbPy_b3sFdV5cEICt
YUSgvfWQPHqP_N9XQFb-PgJikuY2W_Q1ABSzb2jDvIyn4ukpm2MvCN7o82DTyQdlY
4-iElJOvbcSM_RltmBuSTDqorytO2GsdgWobnqb5MnN7hm4uiY6zznbMXNAdtyh_
_oKfRQvWPzwZEymgQUTXWNVu_XD-qraAwixuJZ74hWae2_QSX8jBDjzpz1TUUs2b1
```

```
q4nfo6qzHx8I5sad02TrxfmRV3KJRD20TlBhW574uKHN9PZgWqQBqDiDfwAXifDlB
Y91sdgsobTFF5-1zuDYrPQu1uHit7c5v_rfhvn2QFkYkDRovVj0EympYC8_YRjPg1
LL0vG-2VEtTN190-vOBSFpmj3Hv5Vx00eDkbP1QOCId-w0NwHU4-uf8UDlRjGIqF6
YsmZc2ZsADZJA3FJK4YQVf6EjphUuGG5UQMhmoCnj-7mhdIiiG-xQkYxV5kQ3fksQ
HeAyHyPILVcB2hzwIvttgDPRD02mjk-2vh4euskUt4yu5uZFkUvYx1PnI3_iw2hkL
Mgyzmo0m-ex6kyOotpUcBdAfjwIoccWdX-2kak_HblruY-ptHbVULSaJvPFQaUbmC
Jig7zS-fcAlSogctaSGqLRJyyQJF_jsbPVC8RT3a1GYNUA"
  ],
  [{
    "enc": "A256CBC",
    "kid": "EBQC-5ZNE-DTJU-J43T-XAAM-GLKE-TRNW",
    "Salt": "lFwvWx2QoCeEwcP5nSF4-g",
    "recipients": [{
      "kid": "MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
      "epk": {
        "PublicKeyECDH": {
          "crv": "X448",
          "Public": "uPd5bw7lmtOtKJrUeQ0YkKRz1CX2W1HEDJ4Ej
uXqvUgdWtip-Q1T3vg51aRGS7EmpXwZCgyJ3vwA" } },
        "wmk": "yikmln0p4FgWoVDC_LnLe4zYS753-ZSD3N61RVv5D7tG
9NzEC2-n5A" }
      ]
    },
    "2U26mTitfIVITGp4p4tV3zIv1WT63ugOLHSfF6xTCOSgf4mplIHVReJJ
6bLyWVZ4QpeqEL413psn3448AYqcUwzIDaBA5JvYvwR7Bh2-1007rSOnRzBZ3rpme
j_5G0FOFjRf5ZCuxLYbapP0_yaooTLIs2EkDB6y1MpLvug_jdSfEgbFC8buUxXf5V
MA2jfTqbJVgP35twvoYNpJg1ABFOYUJoCW7OVmD7m2YqfHUSWa_0zN8KEtOziXYfS
Q8IZKzowpn4M5CPSTpezAISxuTGPP15q9zp0szpgJwNqvjPW-0qcFTJBznmXjI4H4
xYSREwJOZweTlbYKwgRCIdNrtEZwq8yPVzeHgRYrHEUjd_rGwBhMNaSugOcrgrvE-1
7EkQsSr0WvxpOUkJE10JUCoDYyo5s2KRPOsbLzW1WiJGRCFsRfpaL9-Nla-SGTq80
hdqhWiNWzCpyISIQZEKjLCrcdK4vEd6DlTKZaMr5B7GIRdSlkC_tprq8iDfK_J1h2
bh-viV3DAU7dz3_yVkyHlCATpPfIRb3elBIxmZr1MchqTlT_WXP85PoJzKcYivA9V
A6m5luo1s2vzQSeaKu83w3UPfVcL4L2hcAHW0xfMERfGEVf0AzJvhfRipWxbDOem8
3H7yzX-MdzAnqJdJlwemSDTdDS9ZeYoCE76zBYSfPooELQh_0xaIp65oQYN_geySG
D5_cAW64bRezaFtpQU0XLCcZBCt1Fzwc13yiaazg4PBr0kGPhtX9py0hK6061Gjle
URh9wdvvyKp3_6xOTDy6TEp4YkBy7bDCpWkadZukcyT-yTyWEgUG_y3fQ8yezFCfvs
V_3vr3qcuIxrnyb9Zl1xNi3opt5qyGGZ9ebcdmWnJZa9Ckbe2uvBBUAu5kwi00j7v
OdqHTphg5Kwdtj9wSfmZuWfmcNkv4e2lbe_SvEZz7IK1ZtbKQ8NzClKEZ5Ys1I8vZ
YPS2uigXXIqtF2MBXWoQr-wKK320X_wlWEzSd0APiJdspzbfpQP-56eYwdyK10n4Bf
A61gL6ZqX-oGiNlUi8srS00HlaxaJPqXzO2_kzu6GwowGu9X33yNACTU8kLFkuK2T
zPIcnbuWb8f6gqkDnjh_Qi1B2Zcj6U4RpmiVL-xMHzzaqaKDGnxBchb2K_i-HT1T-
bxUYqCpvcw7eJx1-OkUY-ucWyOykVgpc2j-6dWzhrbrCUVYqIoJYN0RUGTPQrcSJ3
NXaLM6AwMIUmbKLv1nxIZHyOYG6xIQrHUoXmHES7Q5dNp_LJ5nYIKG58CXEqauDat
ABPCu_BQ88S2UP46US-5xB0u1GZaRTl0V6Jsrl9A7pGul-mhpvpzLg8MdhvV4v9_4
rh03NiOovR5AXRraiWuBSN7bav-0X14_D2_V2zYoNrIuClT00-b1Zvkm_xcY7J8DX
42slBpIrSQB4labT9dwm_wmaVSJaQYorNE7DpCaYS51BUftwrixicxnN04aaPzZ20
xxcQ1lRvk5zFMnqwEbAKkbMJ0i5BG6I_gmJiUP7tCob43T7SofefMwBjM7V5y5R7
mzTnFiBudEg0Djcg0_2d_FOchbH2yRRzpG8Wxcd5kJBhi7DTgFvrDvvdI5XoB4v3
dkS7QDM2CvatSPsrrEaeIvWkcQXXm0eNBIIH1o4TFjwQ9GJ3FczQygun_LCofl0Se
```

```

dnXRfjgaRdt6WHHUElMlh694IQ3PfIA2MasCus26RB9PyGjLNHo2KWdI4Ehs6FVDE
xnflIPE2UmRfZhU3X7CBUEiKldIaps1NWhizk8BoATi3stUTXpQx5rqUMsOrBdIwJ
6je_C2SehyarO2pI6FsBwZJ7emur_pAmkGsa0j8er4kNe7Vbp9TN3jVqu5KXd2_kL
OZcQJsXpXwi_bBPY22qO_hv3sPp5gkBPNYXw9lXJiyBjd9boyQvXV43s-x2P00zAD
SnhTUbZ-K45sc7MOrmFqOPmrNz0QYp0dSA6w0MhGeGkHGZrNdJngfvm3y04nM_21k
hd8-7OXxAbtq7jLfhmtQqjpmzlbceoJWJ3wFHEuqg-ELI0pSlCOVm_xGK95K2GRvz
P0chNrZxNaDT5tDL-5lvhzhN3beYKq-BTsXQ-p76xAP1RcEgBs7Y-14M0tJgU2pa5
p9eiqVCYaNf6B72gHsNXY5sM7xyRD1BSDFLyR6sgqs5UVCHap-t4PVasn107vdQn-
Hci_q8KROFfawkrvHxDmkE_D0Mn1aokf3uElprEW3alrDaFd6AN2rvsFj2et17M40
LbiJoYNqh2M_DUJ-d9Z89grHP4xi64QUNuwo4Y57v3KUSotKDECaJuZ73Ux0070Hf
nF6sv4RbIjI8OWzxn0R3Ur9EsrfG1RyYAESeM48AN4xqfrpGT48bSCJBiwIXBK4k4
Q8cfy7draXuk6VZmIlB--bEd8U5b6QgoFgzqmXKtIr6Reh6H04cqppby8RL_rSRkJ
18SBvv0aQPm1CdJx3m_fxDaRsYF-I3Sk4VpuiZ62YS532bsvtcVo521D7U9A1Gj-X
jrxC61bnHrQ_4FQlk5vRs1cK5yK7HFFMERkomBmD1E3IQW0BLF774R6NJMD8Mbxw
JHiOiHqQLFnmNRvjPomjd_bc28yFncGdRpqW3g6upZr-8GX82MM4EKKdwPamr5QsX
tONeIJxNDEVTTXnyHM3IMekbwXA-aMhRHMCK5mXGMnXL3jWogPwEdEON48dyu53y9
osbusDhzhRINDTD6uhCjsXuGlGnhv_YQV9S-OxF9FVB2Ofjciwwg8J7SaGILqU_20
t6IP5N4ciN9zoBd2bFqIikl9Esp6HLnMLxboq-rpq6qJzABptj1LNywhMMzuFxf8
RmOvUD9F7x3EhblzoaGvAsItI-cuZFzxpEUCUQFRFTxvxx6iKNZDwiYTBdSHIEjXKU
L2Xqg5yNPJ23-B5q3px2wsUVyQRi336A7s6mQJuH7OWOTDLK8Ppcb_gBzwt-ndEVC
ZPE-RSpUZMDFZ6tbtKY14P9cq9Kb0Uby2MLMLQao8vRjECTrEA5-q48rgN5kYdkCp
oI6uLdCGIqVzrIy8oHUDDzw3G4ROBJ1Tk1YVxn29fOvBKX20YCuxe8bz6kktX8vSd
3vf30A0-kWOSEwSDL3YIhbXIgVTeB7vqo-Sam-nX_IVvu0F5JD6tIK_usSkAbIrEU
flu0fv7eCQN9exXc3cVQUWRLjKh1JkQRl-f_jdKb2xMNCQ"
]
1,
"AccountAddress":"alice@example.net",
"InboundConnect":"imap://alice@imap.example.net",
"OutboundConnect":"submit://alice@submit.example.net",
"SmimeSign":{
  "Udf":"MBFI-KY4H-RDBR-TZAS-ZZUP-GRQD-VGDK",
  "PublicParameters":{
    "PublicKeyRSA":{
      "kid":"MBFI-KY4H-RDBR-TZAS-ZZUP-GRQD-VGDK",
      "n":"1tp65TuDE-Bg1ALU15QM1bK-78H6oMMYZcjdCnVjjynM5wYIdvb
ZGlpPexxnkjWyHx55qAS-CldNAQ-rqCwezpk3klfwIwrFVbOnVP9fZrdFPnWLZOC
y3lmU1VGhO55TjoZrjc8g7uxc-Ea5aw9sA0Im0H5nGwtinolHsHYO5aZq_pYG0D3S
LdXkHzyyVfbrQV85iE9_szKN7OGAv1A-JxBJ1M5dLrEmUvBo40fiZvVgv1H2IJ8mL
HYJC_5fSUL5-0suIzEGrCgEoYpHLVF2YcxbHSKi2huplGyWqau80F9R6wmSCZKIjN
gTPfNeceOcN4bNkiP8FinNVcd-TnVEIQ",
      "e":"AQAB"
    }
  },
  "SmimeEncrypt":{
    "Udf":"MA4K-FLCZ-MITB-NDNH-UUVK-IBRT-P3MC",
    "PublicParameters":{
      "PublicKeyRSA":{
        "kid":"MA4K-FLCZ-MITB-NDNH-UUVK-IBRT-P3MC",
        "n":"2bUq7peCou6gvvFFSgqGs6eLvSfcSLy11sgZ3zKWb3vQd2K6HO
Ia1R9qht7lsypsbVfY1VXNN_Oku2t-dfmlq0G6vkvIgz5tpB4zCcQudum9MKNavbd

```

```

ieWHAFI6iVctK6ugbPCMX7yZJwAnI0ghOTj1ICZIZ_oG9NXnlL3RAgclp-Qtw8t1v
jE_yTn1iBEUuOX0MLumQ1QbPwj_-oOMv5cU1y9RJhQDk0X66gcDOoFdInRHZX60Yh
_ojYrtVMlY66-As3sbRpJGCg69tNnHQOxoAAZYa2nuJVVoQoV4Rs4zK-fWvbvXWfvZ
dcW9Ni8gqslU13_2shC_f-wKCbMQwjEQ",
  "e": "AQAB" } } },
  "OpenpgpSign": {
    "Udf": "MBWE-RBKQ-2FVU-4YYB-E23N-ZRXC-CEOI",
    "PublicParameters": {
      "PublicKeyRSA": {
        "kid": "MBWE-RBKQ-2FVU-4YYB-E23N-ZRXC-CEOI",
        "n": "qCGk27z6pWkMB3JTTz_VNJsp2iTion1lDThZpD66zPIweV573L
FQdziNyUt3LfZ0g3gNNRGaYu80cU8YAq4hLDggWF1Vcbh4vDhMNgnPy3Mx41lF62x
s8nbxJSqoZwboBtp_KZoGF4yeaDuDW2Mn3DMYfJI4iFm6WjHIPxP6LFUg3hYO6Edx
uesvxs80fnc_xmH9RgMhxf4JGF1EFxBBXz6SJ4wZLYHuFfx985tdEFmQdDEvZi11g
03s5B-3S8SL15uEr945aai9-zo6IbLuuVfRlr2ycWc2fAadv4K-P76IfpigCQfdls
dVG2Q23LFw5mzHWZscQ6nZsWoeEWVL-Q",
        "e": "AQAB" } } },
    "OpenpgpEncrypt": {
      "Udf": "MDNE-BRJE-2RCO-T3BN-2KTU-NU6J-WSPU",
      "PublicParameters": {
        "PublicKeyRSA": {
          "kid": "MDNE-BRJE-2RCO-T3BN-2KTU-NU6J-WSPU",
          "n": "4qQ0ipjyNkIgg3xWU1e20tFamndalvqluPa6KSQTCmHUNxHegV
GHHBU9yyL3I0SFca7Tla20bs5KLMvx4ITz-pxebDIhs1hs6pTdzicWSuk8zFUHM65
P1VyiHXZn630Rlc6MzMZT_WoGsSFTf0cMhbsOk0Z5-mRtWPJX88cAT3hXxeWouOTc
_3PZUWIYhwo57txefvNqpMVjfcxCOF9gFJhT-uylltYYQ46cOcGoczKTdO2gkziE_
P-xhS5sQVnvJJUxqvH7XnvZ5O_3BqlLpaxalceSmC3DkaQslvDpWaCnb9VfABAAQg
ynowqslbPRBzuFwldlFbiWnxnF2XnAQQ",
          "e": "AQAB" } } } } }

```

Note that the inbound and outbound server configuration does not specify the access credentials to be used to access the service. These are specified in the Credential catalog.

Future: The mail application should support automated means of credentialling the public key including obtaining an X.509v3 certificate or uploading the key to a key service.

4.2.2. SSH

SSH configuration profiles are described by entries in multiple catalogs

CatalogedApplicationSsh entries in the Applications catalog. Specify an SSH client credential or certificate signing credential

CatalogedCredential entries in the Credential catalog. Specify SSH host keys (i.e. contents of the known hosts file)

CatalogedContact entries in the Contacts catalog. Specify SSH client keys (i.e. material from which an authorized_key file entry might be constructed).

Future: Client and Host certificates are not currently supported. This is clearly desirable but requires additional implementation considerations.

Future: Provisioning of SSH host private keys is currently out of scope. This is best considered as part of the device provisioning and authorization flow and will lead to entries being created/updated in the device catalog.

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.

```
{
  "CatalogedApplicationSsh":{
    "LocalName":"ssh",
    "Key":"MCXP-WQVY-RTKQ-ZU6P-VOM4-7U6K-FHXXH",
    "Grant":["web",
      "threshold"
    ],
    "EnvelopedEscrow":[[{
      "enc":"A256CBC",
      "kid":"EBQG-TSDD-KPUM-Y3KS-TSIF-OGQ2-UIBE",
      "Salt":"KO-vj1hCiJn_L7gETkIiew",
      "recipients":[{
        "kid":"MBZP-WZAZ-B6KQ-MYYP-H7KD-VVBA-7T6U",
        "epk":{
          "PublicKeyECDH":{
            "crv":"X448",
            "Public":"G5PYCVsNi99zjwXBuxbzxS-yOeBeYWApIrHvM
xPSOttBQS5wLqj6Q7x8xP-7B0c_Cbk8qwShNE-A"}},
          "wmk":"-K10vu8TcHok8Wo9BAHoLwaDUkBxhMDJ6FpS8vvvhQSf
v-VEjLyw6A"}
        ]},
        "4eG28l0r23lJLShpB1X4NcOjxVUp8X-LBaNaAEROX9Ngk9A-8u1ONoWr
KCbJlRBG8orgqdSYwHEj5SruwRXO9uAkb8rulxvg2toik5TlodlYaImeD2gY9mSTD
iWEeUSBl_E909W1OOULjd58zjWWlhS8vIiNh1zWoQSJXMD78gyUtZhjfk0J2mT_It
_zCVpPmT6uAWPYlX3n33wH8Hw5b34VyF1NLIJh6Yho_6bVO8wR7kAGoOYJCs6N3V4
JFPmnhpZyCEFlqJ4X3quCPZchpnQsoRmTF10XsWbuaIT7sYxdh53Tf1JAnvEgrZY
keVRORTDQlqtNJtOS1lHmBusg7NSIbv7vZOXFExOT8fQrze3Ls5QFS1HSglQN-qUR
e1zfMz6CBIOiZ_q-ctvkQtKMBTxWR5ABoZjGZg0aSSct_o7JwUoDnm14hyX9Ptzw7
0hbyTXJE1_JX2V4dIJ4YpdH8HtdUIKfB5c-_TCu-ex1B850UI7LEqqo07FTuNeWZ0
```

```

OjHEfl1t4gos2wjStHnNfCn6TY4XxXrp4KSa7Uw9060gFrYqIYDvkiGs8XabMr_Afb
n3-9xxrHWDgzvDn3n51kEEf-omH8goD45m-UzgVi_1fJTrNQePcJ1Js6Jb4xqPw0J
UC2Rp-zB6nA-MzdFLnbhOVaF610oX-nQxVNhiVml4ABifIiDz0hDIK39aC9EzYESN
vYJU5OaDZ_2yIfc9ADC2WabkeRgYP7-imVcBFKcARTIgcj6--DTDnFtFc4hoS_UZc
hnuKW1PMc-AH4pej1VjnEYMG2Ch4-UDvWDu5yJLiR2asFxn1R84bcrCJf6qCZs-nX
6xG6nzOiHol-cDDOTqB3pvm6Hauvo4RRFtqqjy1Tg-VlY9V6kD4TfhgQKLkLfTHqe
MRFZiVjVS_d6n4oFnPE85y54As3XEHu0P06bT47GNZJ352XFZiXK477F_5gzmRVbc
kcHLjbmddqDKCAzKzGp0ah3VyC12TidCEq4_qKveEMcXLehB1kPrfEzeF5DtWkzRM1
ZvahMgW3uAtNzp_7po9BrWeuBWqmrTWbvWYMDuzQktlYi6b06uN1vPV6msQCRs_f4
fPoBcOItMS1bjQgfgRSuLr76qK43TzoFMbCldHcZv6gZGUpHiQS5BqGgWqFneJzu0
hZJVKbPxgNZ18Xe6kOdeJNKK2TbAkQ9HfdMZ3QcAcFaGU8WjhUqWnCwMA-GEPJat5
tO_BtovwCY_phpbbQVYDhJvhAHYp43zcwNTNbss81FVNJPfv-bibLumK6w2oT9yk
pLm7pHWYY__TaM13w5zeSL7Dxbuknfiv5-SY-3o6_5s_p8_57H13TAhub0cP303DT
uZf10XexPGRv3zrloeXgb4tDKFXMDihE1qwdBvY00Z18Y9-Ku3mW9M1p6nBuHcOR
HSbzBLjgDBMS7jz1lesUvr08wLNQ1g38o0Ja2EbtRE8ghPOE3SiL_QH15V4fNct17Q
BZ_yld261eyOjYBQWApYtsEoV058Su-IpfWsC166p500UoeZ-GmlYoGoVodjDr002
sBKbcFZStEM8a0p50EevhtMrPxd6sQaf7HDc422mYI2649dibVxWDgaJnZc7NSE2m
j5F2zyjKpEt3yqSSqFY9eRlwOInNtr1PG3d0bwdNqAECznAZqIBCrr6SVrn3bggyaF
RTpDMQRJt-vIKLFRBbJw8GV6NRNovv24VCPYCIhfKfHoAV0rvBZr-qR3MTWEbWkTA
AUPneGZKgSVDUphKjd8vMZwnrT7xauAcXLAE1_K7bvGbu05aFgkQM1EICn42-VSfs
UDhYv4ZsOdOPEymKrgzr3Pb6N6pKz8YuznB6RmmBhkNgz8_DHGbfVgaovMLpk9ZL7
0wMVh7hiX5XcgAvk8b3ZarwbniOdERXE4-zRw_j7Rnt7twmfFSDVINPhFPiCiFixg
R4hnG1Ecn8s82Q-3QJ9BrMBE5xWScoJh_BxeAk1LE2Epb95UkQBv6b9Xp5gz-6xOJ
dBYsJCWVBnWXH19OQNeta9TQOem-7k7Xc5n3HVkn0q040SsL2FNTxaFUgWZW4dSCp
XZPzaGFRdDG4nr710EkBBQD_Lpv7UzSdAQcluSxwft7DdxnhktoSx8yMFR2KIDtVI
UmDN4Efu1AKiD_fZJQ11NRk4GndN-ePAZGgZr3mHZqTguCmmfC8y4qPQIR30HoFJx
rWHU-uzZDKN2Kq3TydR0GQxLkL-fCW3ejehtbU_q2xDDL1KUpVzCYEeP8PuwYawCak
adYsTFJrVzPSyboQrholdGk-PyLTQ1vY45Xp3I4tiGnKWREBM2CJAmT_vLEa77ru9
rM3fY-Z2WrBOILIL943PxPHFx2aqQ2s0W6AAf1grIIi8sTLL7GEyhsqTO_Xzui-q9
5ZBbrz-mpylMMphpNAgvv9hz49vyEvmQZY8GdlM7IPO0DGal14tNPn3gcmCiQ8CCZ
9NxYLCazfaqIUMZU9BukkAG9p04LY3amI_cOlFInKbSAmqcTFLcKJFPOFhskxqubG
dr8VNH5MdZUM5bmoiQZWvMDt-az39M_MZYAfWvy9opM0-oe1nI4Bw2Kh4aoteIOi
mEi7kbucpth03r7VN46n5Sxf1GrbKR4LsDAWyROBURvRLDthbKP9a2pt3MWuGvgFa
W9ntaSx51LKf471vyvtFkmX_eJslRZGyhDt21Pf5iJ3RO2bqYGpkxofDFp0iJYTVs
Soz8MiF5KrGFB213k7aXkzhQOqnSuIpzeIzKW6SJjqf7O_mGugetW5CCZq73H39zZ
BqzkeQt10ZVmIsnvcOSRS0Nc7nRxocka6W9HexdE_HokPaJl_fND8B1oM5H13zQ
raZuxfV_K3-yNgltDBMEFPtAgVWgE28Pvame14HDFfmDMoLVjmgyjhVv5JBcPTeCd
Tph99ZFh4285NzyOo4PUPUIOBO-XzRn6MmsaDh7ySmtDNEDYdJTIJEQDpHWTfAXoi
we0Ijd0anDgDg55LuGhyhafR1E57pZYuIEclgioFY_uA_xm6g4HSTVkcN99r-M7x0
t1214SIBF241pUiW-wMMLpRBWHQPfAG-HeK85oBGrnE4kMV1Pb7ax7nQnpdlhQd7m
_dW9x4Je5-nZGInlS7WC4iL4_hu0RPPUcsHaBUAM4wjLsGpPftg8YW-RrmLOVHToi
MY6HhB61bObwQvSQgXjA3DMEYBCfZ52wtc5OKQd8R8aVrw"

```

```

    ],
    "ClientKey": {
      "Udf": "MCXP-WQVY-RTKQ-ZU6P-VOM4-7U6K-FHXH",
      "PublicParameters": {
        "PublicKeyRSA": {

```

```

      "kid": "MCXP-WQVY-RTKQ-ZU6P-VOM4-7U6K-FHXH",
      "n": "v0EWseYtsQP3dC_eBaDEK76z7Sg_fMmYaMiq_WrR_tJJvcxxrV
3rHFLAugg4NAH4evuCjq99W07T4PLNNR3Dee6HrFpf9ktKplHina37_ZqvOUbpLSY
DGCnV_4ghAun1qYcyREcZ-x88NuXbHSni09k2KAc5HxSfKQPuhUOnTBcK8xR83psR
u4jpYTM31Djga8iFVJQRaC9t0Q1aD3BXHKtak3mMMV0GGYBX55xLcYTsIggXLEmOx
ZhJqLgY3pNE77jIqmyWL8aryPBVrdYIYne8uNSCaDa-mE-ao_9jsjGYseOeTrkJ6g
1Ne1CpL4iiNzpJmP4kAI_3Si4jJk8xyQ",
      "e": "AQAB" } } } }

```

4.3. Bookmark

The bookmark catalog `mmm_bookmark` 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": {
    "LocalName": "Sites-1",
    "Uid": "NDU5-XXSS-6KLM-MO6Q-S3F5-SJ7P-FO73",
    "Uri": "http://www.example.com",
    "Title": "site1" } }

```

4.4. Contact

The contact catalog `mmm_contact` contains `CatalogEntryContact` entries which describe the person, organization or location described.

The fields of the contact catalog provide a superset of the capabilities of vCard [RFC2426].

```

{
  "CatalogedContact": {
    "Key": "MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF",
    "Self": true,
    "Contact": {
      "ContactPerson": {
        "Id": "MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF",
        "Anchors": [ {
          "Udf": "MAMQ-ETEA-JBL3-6UKE-LRNT-DGC3-OIDF",
          "Validation": "Self" }
        ],
        "NetworkAddresses": [ {
          "Address": "alice@example.com",

```

[illegible]

[illegible]

```

RmtiV2x1YVhOMGNTRjBiM0pUYVdkdVlYUfjBjvVpT2lCN0NpQwQJQ0EKEICBnSUNKV
lpHWWlPaUFpVfVVKRVZpMVlXRTVJTFRKUlZVSXRva0pOV2kwMVRrYzNMVXd6UTBRDe
0xUklWaUlzQwogIGlBZ0lDQWdJQ0pRZFdKc2FXTlFZWEPoYldWMFpYsnpJam9nZXdv
vZ0lDQWdJQ0FnSUNKUWRXSnNhV05MWlHsCiAgRlEWUklJam9nZXdvZ0lDQWdJQ0Fn
SUNBZ0ltTnlkaUk2SUNKRlPpEUTBPQ0lzQ2lBZ0lDQWdJQ0FnSUNBaVUKICBIVmliR
2xqSWpvZ0lraFzKmdQwVWxab1IyTjZSbXhQYlRKA vJHTmxkblpXV1hsa05tZHFaSE
V6TTFGeFZqaAogIFZjVE0lWkVkaGMxsjZVVzQlWDFBS0lDQldame5DVWtsZk9FMXF
hWFpsY2xSTFpHRmhSVWt6TWtFaWZYMtLMCIaGQW9nSUNBZ0lrTnZiVzF2YmtWdVkz
SjVjSFJwYji0aU9pQjddAuFnsUNBZ0lDslZaRl1pT2lBaVRVU1FVaTEKICBU2xa
ExVZExOVm90TWt4S1FTMUlUVmxXTFZovFEwZ3RTRVv5UXlJc0NpQwQJQ0FnSUNKU
RXSnNhV05RWQogIFhKaGJXVjBaWEp6SWPwZ2V3b2dJQ0FnsUNBZ0lDslFkV0pzYvd
OTFpYbEZRMFJJSWpvZ2V3b2dJQ0FnsUNBCiAgZ0lDQWdJbU55ZG1JNk1DSl1ORFE0
SWl3S0lDQWdJQ0FnSUNBZ0lDslFkV0pzYVdNaU9pQWlOVFZxVld0dGMKICBXNHpam
2RITUdJeVNicEVWblV6U0d4bU5YTlBoa2RuVm14cVgzWmhXVVoZUVVWcmMwUmpUWG
t6ZDNsMlZRbwogIGdJSGQwT1c5cWEyVlZTMVEyTXpBMFJIZG1jbWd0VlhjNFFTsjl
mWDBzQ2lBZ0lDQWlRMjl0Yl1c5dVFYVjBhCiAgRlZlZEdsallYUnBiMjRpT2lCN0Np
QWdJQ0FnSUNKVlpHWWlPaUFpVfVVKVlNTMUZWMHhQTFVWsk4wb3RUMVoKICBCU3kxs
FIxcElMVfpaU0ZjdFdrCFRWU0lzQ2lBZ0lDQWdJQ0pRZFdKc2FXTlFZWEPoYldWMF
pYsnpJam9nZQogIHdvZ0lDQWdJQ0FnSUNKUWRXSnNhV05MWlHsRlEWUklJam9nZXdv
vZ0lDQWdJQ0FnSUNBZ0ltTnlkaUk2SUNKCIaGWU5EUTRJaXdLSUNBZ0lDQWdJQ0Fn
SUNKUWRXSnNhV01pT2lBaVpsUlZnMfJsvUWpFde4wcZVMXB3YnpSMFUKICBYaGFVS
EJLUVdJdFgYUXpUa2xrU2l1oc2EzafVZ2xhYjJkSl1rVkwXPV0ZrVufvZ0l1HWTVTmj
V6Tl1cxeGnqRQogIhHwVlJYVjBsTmFIcGlARXBoUVNKKWZYMhNDaUFnsUNBaVEyOXR
ivZl1VTJsbmJtRjRjBkWEpsSWpvZ2V3b2dJCIaGQ0FnSUNBaVZXUm1Jam9nSWsxQlRW
QXRRbGcwUnkxQlMwc3lMVmxJUVUvDFNwaEtWaTfHTwT0V0xWVl1RbGMKICBpTEFvZ
0lDQWdJQ0FpVUHwaWJHbGpVR0Z5WvcxbGRHVnljeUk2SUhzS0lDQWdJQ0FnSUNBaV
VlVmlir2xqUwogIDJWNvJVTktVTQ0k2SUhzS0lDQWdJQ0FnSUNBZ0lDSmpjbl1pT2l
BaVJXUTBORGdpTEFvZ0lDQWdJQ0FnSUNBCiAgZ0lsQjFZbXhWwXlJNk1DSl1pOaTFF
TWtSaVlrdHNZVlpZ2GtjMVdsRjNaVXhrTlY5clVERKZRMZEZVWpRd1kKICBrUnRjR
2NOV1RSTGN6a3lSazVsTFhWNUNpQWdJmWRWY2sxZlRHMVJTMDlKVUdwcWNqVklPRT
VQUWtWQklumQogIDlmWDE5IiwKICAgICAgICAgIHsKICAgICAgICAgICAgInNpZ25
hdHVyZXMiOiBibewogICAgICAgICAgICAgICAgImFsZyI6ICJTNTeyIiwKICAgICAg
ICAgICAgICAgICJraWQiOiAiIUFNUS1FEVB1UpCTDMtN1VLRs1Muk5ULURHqZMtT
0lEriIsCiAgICAgICAgICAgICAgICaIC2lbnmF0dXJlIjogIkZpCUdTN3NkLWwtaV
hlVzB0bldPSVViBUp4dzBTTEJIA19GNFZZeWE4QU11mJNkVksKICBlYmdisC1NdFN
BS18tMEZwdVh5V2NSVVRUOEfzSGVhbGpzR2U3WTl0tJrXx05UOHRJQVNzOVpzWmE0
SFhVeQogIEFCM3ZPek1lU082d2k1YkhlaGMteldoa0VQWmh2ZG1CTWNpemtPRFlBI
nldLAogICAgICAgICAgICAIUGF5bG9hZERpZ2VzdCI6ICJwYm54M0ZHZVd1WldPck
FOUkQldm8zVVlualpScEhHbXBMD1NXVkpuc05aNFMKICBGZTRxVm4taGZOclo1NTd
obkpocDRhRdDFTjJwNkI3SVZOTW1lS185dyJ9XSwKICAgICAgICAIUHVjdG9jb2xz
IjogW3sKICAgICAgICAgICAgIlByb3RvY29sIjogIm1tbSJ9XX1dfX0",
{
    "signatures": [{
        "alg": "S512",
        "kid": "MAMP-BX4G-AKK2-YHPA-IXJV-Z2KV-UXBW",
        "signature": "P5Zhrm_5gMxQ2QlEQKXSDr03F6xjL1TR
CjS568xsRv_o13mr84x80mEVOUWwBVlltpa5ezjLEGAYyJupBS1qtRVxWLLyY8w-
Vje3zocM-kn_wQqxBjWE6GwrLoSj1KICFDO8Brq1SkZMtgpw97FzEA"}

```

```

    ],
    "PayloadDigest":"aRSD7Lw6GWggbqxAhn77PNOe2ekZNQR1
bCVj-ESSgdDH836wVdwzFXwkMe63uvysVSdtoR4mAYojoG2LU5j_nA"}
  }}
}]]]]

```

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.

4.5. Credential

The credential catalog `mmm_credential` contains `CatalogEntryCredential` entries which describe credentials used to access network resources.

```

{
  "CatalogedCredential":{
    "Service":"ftp.example.com",
    "Username":"alice1",
    "Password":"password"}}

```

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.

4.6. Device

The device catalog `mmm_Device` contains `CatalogEntryDevice` entries which describe the devices connected to the account and the permissions assigned to them.

Each device connected to a Mesh Account has an associated `CatalogEntryDevice` entry that includes the activation and connection records for the account. These records are described in further detail in section ???.

4.7. Network

The network catalog contains `CatalogEntryNetwork` entries which describe network settings, IPSEC and TLS VPN configurations, etc.

```

{
  "CatalogedNetwork":{
    "Service":"myWiFi",
    "Password":"securePassword"}}

```

4.8. Publication

[Note, this catalog is obsolete, the functions provided by this catalog are being merged with the Access catalog]

The publication catalog `mmm_Publication` contains `CatalogEntryPublication` entries which describe content published through the account.

If the data being published is small, it MAY be specified in the `CatalogEntryPublication` entry itself as enveloped data. Otherwise a link to the external content is required.

The Publication catalog is currently used to publish two types of data:

`Contact` Used in the Static QR Code Contact Exchange interaction.

`Profile Device` Used in the Preconfigured Device Connection interaction.

The interactions using this published data are described in [draft-hallambaker-mesh-protocol].

>>>> Unfinished `SchemaEntryPublication`

Missing example 11

4.9. Task

The Task catalog `mmm_Task` 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":{
    "Title":"SomeItem",
    "Key":"NC4X-EQN6-S6RF-NJKY-PTPW-2SI7-QELL"}}}
```

5. Spools

Spools are DARE Sequences containing an append only list of messages sent or received by an account. Three spools are currently defined:

Inbound Messages sent to the account. These are encrypted under the account encryption keys of the sender and receiver that were current at the time the message was sent.

Outbound Messages sent from the account. These are encrypted under the account encryption keys of the sender and receiver that were current at the time the message was sent.

Local Messages sent from the account for internal use. These are encrypted under the encryption key of the intended recipient alone. This is either the account administration encryption key or a device encryption key.

Every Mesh Message has a unique message identifier. Messages created at the beginning of a new messaging protocol interaction are assigned a random message identifier. Responses to previous messages are assigned message identifiers formed from the message identifier to which they respond by means of a message digest function.

Every Mesh Message stored in a spool is encapsulated in an envelope which bears a unique identifier that is formed by applying a message digest function to the message identifier. Each stored message has an associated state which is initially set to the state Initial and MAY be subsequently altered by one or more MessageComplete messages subsequently appended to the spool. The allowable message states depending upon the spool in question.

5.1. Outbound

The outbound spool stores messages that are to be or have been sent and MessageComplete messages reporting changes to the status of the messages stored on the spool.

Messages posted to the outbound spool have the state Initial, Sent, Received or Refused:

Initial The initial state of a message posted to the spool.

Sent The Mesh Service of the sender has delivered the message to the Mesh Service of the recipient which accepted it.

Received The Mesh Service of the sender has delivered the message to the Mesh Service of the recipient and the recipient has acknowledged receipt.

Refused The Mesh Service of the sender has delivered the message to the Mesh Service of the recipient which refused to accept it.

MessageComplete messages are only valid when posted to the spool by the service.

5.2. Inbound

The inbound spool stores messages that have been received by the Mesh service servicing the account and MessageComplete messages reporting changes to the status of the messages stored on the spool.

Messages posted to the outbound spool have the state Initial, Read:

Initial The initial state of a message posted to the spool.

Read The message has been read.

A message previously marked as read MAY be returned to the unread state by marking it as being in the Initial state.

5.3. Local

The local spool stores messages that are used for administrative functions. In normal circumstances, only administrator devices and the Mesh Service require access to the local spool.

The local spool is used to store MessagePin messages used to notify administration devices that a PIN code has been registered for some purpose and RespondConnection messages used to inform a device of the result of a connection request.

The local spool is used in a device connection operation to provide a device with the activation and connection records required to access the service as an authorized client. Servicing these requests requires that the service be able to access messages stored in the spool by envelope id.

Messages posted to the outbound spool have the states Initial, Closed:

Initial The initial state of a message posted to the spool.

Closed The action associated with the message has been completed.

Future: Redefining the role of the Local spool would allow the Claim/PollClaim operations used in device connection to be eliminated and greater consistency achieved between the device connection interactions.

5.4. Log

The log spo

6. Logs

The logging functions are not currently implemented.

Logs are records of events. Mesh logs SHOULD be encrypted and notarized.

The following logs are specified:

Service A log written by the Mesh Service containing a list of all actions performed on the account

Exception A log written by the Mesh Service containing a list of all exception events such as requests for access that were refused.

Notary A log written by administration devices connected to the account containing a sequence of status entries and cross notarization receipts.

The notary log will perform a particularly important role in future Mesh versions as it provides the ultimate root of trust for the account itself through cross notarization with the account holder's MSP which in turn achieves mutual cross notarization with every other MSP by cross notarizing with the Callsign registry. Thus every Mesh user is cross notarized with every other Mesh user making use of the Callsign registry through a graph with a diameter of 4.

7. Cryptographic Operations

The Mesh makes use of various cryptographic operations including threshold operations. For convenience, these are gathered here and specified as functions that are referenced by other parts of the specification.

7.1. Key Derivation from Seed

Mesh Keys that derived from a seed value use the mechanism described in [draft-hallambaker-mesh-udf]. Use of the keyname parameter allows multiple keys for different uses to be derived from a single key. Thus escrow of a single seed value permits recovery of all the private keys associated with the profile.

The keyname parameter is a string formed by concatenating identifiers specifying the key type, the actor that will use the key and the key operation:

7.2. Message Envelope and Response Identifiers.

Every Mesh message has a unique Message Identifier MessageId. The MakeID() function is used to calculate the value of Envelope Identifier and Response identifier from the message identifier as follows:

```
static string MakeID(string udf, string content) {
    var (code, bds) = UDF.Parse(udf);
    return code switch
    {
        UdfTypeIdIdentifier.Digest_SHA_3_512 =>
            UDF.ContentDigestOfDataString(
                bds, content, cryptoAlgorithmId:
                    CryptoAlgorithmId.SHA_3_512),
        _ => UDF.ContentDigestOfDataString(
            bds, content, cryptoAlgorithmId:
                CryptoAlgorithmId.SHA_2_512),
    };
}
```

Where the values of content are given as follows:

application/mmm/envelopeid The proposed IANA content identifier for the Mesh message type.

application/mmm/responseid The proposed IANA content identifier for the Mesh message type.

For example:

MessageID
= NCAA-7UYA-TG2C-6XUC-UG3B-4XGT-OBIE

EnvelopeID
= MBHZ-QYVP-T5DQ-FQAP-AWD4-FLMO-ZZJT

ResponseID
= MB2Z-JQXS-7IEO-K5OJ-YI3P-FZC2-OGFU

7.3. Proof of Knowledge of PIN

Mesh Message classes that are subclasses of MessagePinValidated MAY be authenticated by means of a PIN. Currently two such messages are defined: MessageContact used in contact exchange and RequestConnection message used in device connection.

The PIN codes used to authenticate MessagePinValidated messages are UDF Authenticator strings. The type code of the identifier specifies the algorithm to be used to authenticate the PIN code and the Binary Data Sequence value specifies the key.

The inputs to the PIN proof of knowledge functions are:

PIN: string A UDF Authenticator. The type code of the identifier specifies the algorithm to be used to authenticate the PIN code and the Binary Data Sequence value specifies the key.

Action: string A code determining the specific action that the PIN code MAY be used to authenticate. By convention this is the name of the Mesh message type used to perform the action.

Account: string The account for which the PIN code is issued.

ClientNonce: binary Nonce value generated by the client using the PIN code to authenticate its message.

PayloadDigest: binary The PayloadDigest of a DARE Envelope that contains the message to be authenticated. Note that if the envelope is encrypted, this value is calculated over the ciphertext and does not provide proof of knowledge of the plaintext.

The following values of Action are currently defined:

Device Action info for device PIN

Contact Action info for contact PIN

These inputs are used to derive values as follows:

```
alg =          UdfAlg (PIN)
pinData =      UdfBDS (PIN)
saltedPINData = MAC (Action, pinData)
saltedPIN =    UDFPresent (HMAC_SHA_2_512 + saltedPINData)
PinId =        UDFPresent (MAC (Account, saltedPINData))
```

The issuer of the PIN code stores the value saltedPIN for retrieval using the key PinId.

The witness value for a Dare Envelope with payload digest PayloadDigest authenticated by a PIN code whose salted value is saltedPINData, issued by account Account is given by PinWitness() as follows:

```
witnessData = Account.ToUTF8() + ClientNonce + PayloadDigest
witnessValue = MAC (witnessData , saltedPINData)
```

For example, to generate saltedPIN for the pin ADFR-TEQU-3HJD-IRND-P4TS-CRBD-NI used to authenticate a an action of type Device:

```
pin = ADFR-TEQU-3HJD-IRND-P4TS-CRBD-NI
action = message.
```

```
alg = UdfAlg (PIN)
     = Authenticator_HMAC_SHA_2_512
```

```
hashalg = default (alg, HMAC_SHA_2_512)
```

```
pinData = UdfBDS (PIN)
         = System.Byte[]
```

```
saltedPINData
  = hashalg(pinData, hashalg);
  = System.Byte[]
```

```
saltedPIN = UDFPresent (hashalg + saltedPINData)
           = AAV6-EBKF-JIUO-B2UV-UQX7-OKHB-OAAX
```

The PinId binding the pin to the account alice@example.com is

```
Account = alice@example.com
```

```
PinId = UDFPresent (MAC (Account, saltedPINData))
       = ADDU-7BE6-DN7R-U2BB-VST6-DYZL-YEZR
```

Where MAC(data, key) is the message authentication code algorithm specified by the value of alg.

When an administrative device issues a PIN code, a Message PIN is appended to the local spool. This has the MessageId PinId and specifies the value saltedPIN in the field of that name.

When PIN code authentication is used, a message of type `MessagePinValidated` specifies the values `ClientNonce`, `PinWitness` and `PinId` in the fields of those names. These values are used to authenticate the inner message data specified by the `AuthenticatedData` field.

7.4. EARL

The UDF Encrypted Authenticated Resource Locator mechanism is used to publish data and provide means of authentication and access through a static identifier such as a QR code.

This mechanism is used to allow contact exchange by means of a QR code printed on a business card and to connect a device to an account using a static identifier printed on the device in the form of a QR code.

In both cases, the information is passed using the EARL format described in [draft-hallambaker-mesh-udf].

8. 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 a device to an account.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-schema-10.html for artwork.)

Figure 1: Profiles And Connections

8.1. 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-callsign] 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.

8.2. 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.

8.3. Mesh Connections

A Mesh connection is an assertion describing the connection of a device or a member to an account.

Mesh connections provide similar functionality to 'end-entity' certificates in PKIX but with the important proviso that they are only used to provide trust between a device connected to an account and the service to which that account is bound and between the devices connected to an account.

A connection is valid with respect to an account with profile _P_ if and only if:

- * The profile _P_ is valid
- * The AuthorityUdf field of the connection is consistent with the UDF of _P_
- * The profile is signed under the key specified in the AdministrationKey field of _P_.
- * Any conditions specified in the profile are met

A connection has the status current with respect to an account with profile `P` if and only if:

- * The connection is valid with respect to the account with profile `P`.
- * The profile `P` is current.

A device is authenticated with respect to an account with profile `P` if and only if:

- * The connection is valid with respect to the account with profile `P`.
- * The device has presented an appropriate proof of knowledge of the `DeviceAuthentication` key specified in the connection.

8.4. Device Pre-configuration

The `DevicePreconfiguration` record provides a means of bundling all the information used to preconfigure a device for use in the Mesh. This comprises:

- * The Enveloped `ProfileDevice`.
- * A `ConnectionDevice` assertion credentialing the device to the configuration provider Mesh Service.
- * A `ConnectionService` assertion credentialing the device to the configuration provider Mesh Service.
- * The secret seed used to create the `ProfileDevice` data.

The `DevicePreconfiguration` record MAY be used as the means of preconfiguring devices to allow connection to a user's account profile using the Preconfigured/Static QR Code device connection interaction.

For example, Alice's coffee pot was preconfigured for connection to a Mesh account at the factory and the following `DevicePreconfiguration` record created:

```
{
  "DevicePreconfigurationPrivate":{
    "EnvelopedProfileDevice":[{
      "EnvelopeId":"MBOB-5GVY-Q43B-KODG-UJ3E-LY7V-36UV",
      "dig":"S512",
      "ContentMetaData":"ewogICJYbmlxdWVJZCI6ICJNQk9CLTVHVlktUT
```

[illegible]

[illegible]


```
    "KeyType": "MeshProfileDevice" }},  
    "ConnectUri": "mcu://maker@example.com/EBKG-ED30-HBHK-ZQGS-EX4H-  
X22S-X4" }}
```

The use of the publication mechanism in device connection is discussed further in [draft-hallambaker-mesh-protocol].

9. Architecture

The Mesh architecture has four principal components:

Mesh Account A collection of information (contacts, calendar entries, inbound and outbound messages, etc.) belonging to a user who uses the Mesh to management.

Mesh Device Management The various functions that manage binding of devices to a Mesh to grant access to information and services bound to that account.

Mesh Service Provides network services through which devices and other Mesh users may interact with a Mesh Account.

Mesh Messaging An end-to-end secure messaging service that allows short messages (less than 32KB) to be exchanged between Mesh Accounts and between the Mesh devices connected to a particular account.

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 Mesh Service Provider which the user may change at any time of their choosing.

A Mesh Account May be active or inactive. By definition, an active Mesh account is serviced by exactly one Mesh Service, an inactive Mesh account is not serviced by a Mesh Service. A Mesh Service Provider MAY offer a backup service for accounts hosted by other providers. In this case the backup provider is connected to the account as a Mesh device, thus allowing the backup provider to maintain a copy of the stores contained in the account and facilitating a rapid transfer of responsibility for servicing the account should that be desired. The use of backup providers is described further in [draft-hallambaker-mesh-discovery].

9.1. Mesh Account

Mesh Accounts contains all the stateful information (contacts, calendar entries, inbound and outbound messages, etc.) related to a particular persona used by the owner.

By definition a Mesh Account is active if it is serviced by a Mesh Service and inactive otherwise. A Mesh user MAY change their service provider at any time. An active Mesh Account is serviced by exactly one Mesh Service at once but a user MAY register a 'backup' service provider to their account in the same manner as adding an advice. This ensures that the backup service is pre-populated with all the information required to allow the user to switch to the new provider without interruption of service.

Each Mesh account is described by an Account Profile. Currently separate profile Account Profile are defined for user accounts and group accounts. It is not clear if this distinction is a useful one.

9.1.1. Account Profile

A Mesh account 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.

An Account Profile MUST specify an EscrowEncryption key. This key MAY be used to escrow private keys used for encryption of stored data. They SHOULD NOT be used to escrow authentication keys and MUST NOT be used to escrow signature keys.

A user should not need to replace their account profile unless they intend to establish a separate identity. To minimize the risk of disclosure, the Profile Signature Key is only ever used to sign updates to the account profile itself. This allows the user to secure their Profile 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.

9.1.1.1. Creating a ProfileMaster

Creating a ProfileMaster comprises the steps of:

0. Creating a Master Signature key.
1. Creating an Online Signing Key
2. Signing the ProfileMaster using the Master Signature Key

3. Persisting the ProfileMaster on the administration device to the CatalogHost.
4. (Optional) Connecting at least one Administration Device and granting it the ActivationAdministration activation.

9.1.1.2. Updating a ProfileMaster

Updating a ProfileMaster comprises the steps of:

0. Making the necessary changes.
1. Signing the ProfileMaster using the Master Signature Key
2. Persisting the ProfileMaster on the administration device to the CatalogHost.

9.2. Device Management

Device management allows a collection of devices belonging to a user to function as a single personal Mesh. Two catalogs are used to manage this process:

- * The Access catalog is used to instruct the Mesh Service how to respond to requests from the device.
- * The Device catalog records information for use by administration devices managing the device.

9.2.1. The Device Catalog

Each Mesh Account 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.

9.2.2. 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. When bound to an account, the base keys specified in the Device Profile are combined with the key data provided in the Activation device to construct the keys the device will use in the context of the account.

(Artwork only available as svg: No external link available, see [draft-hallambaker-mesh-schema-10.html](#) for artwork.)

Figure 2: 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.

9.2.2.1. Creating a ProfileDevice

Creating a ProfileDevice comprises the steps of:

0. Creating the necessary key
1. Signing the ProfileDevice using the Master Signature Key
2. Once created, a ProfileDevice is never changed. In the unlikely event that any modification is required, a completely new ProfileDevice MUST be created.

9.2.2.2. Connection to a Mesh Account

Devices are only connected to a personal Mesh by an administration device. This comprises the steps of:

0. Generating the PrivateDevice keys.
1. Creating the ConnectionDevice data from the public components of the ProfileDevice and PrivateDevice keys and signing it using the administration key.
2. Creating the Activations for the device and signing them using the administration key.
3. Creating the CatalogEntryDevice for the device and adding it to the CatalogDevice of the account.
4. Creating an AccessCapability granting the necessary access rights for the device and adding that to the CatalogAccess of the account.

These steps are usually performed through use of the Mesh Protocol Connection mechanism. However, Mesh clients MAY support additional mechanisms as circumstances require provided that the appropriate authentication and private key protection controls are provided.

9.3. Mesh Services

A Mesh Service provides one or more Mesh Hosts that support Mesh Accounts through the Mesh Web Service Protocol.

Mesh Services and Hosts are described by Service Profiles and Host Profiles. The means by which services manage the hosts through which they provide service is outside the scope of this document.

As with a Device connected to a Mesh Account, the binding of a Host to the service it supports is described by a connection record:

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-schema-10.html for artwork.)

Figure 3: 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.

9.4. 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).

All communications between Mesh accounts takes 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].

9.4.1. Message Status

As previously described in section ###, every message stored in a spool has a specified state. The range of allowable states is defined by the message type. New message states MAY be defined for new message types as they are defined.

By default, messages are appended to a spool in the Initial state, but a spool entry MAY specify any state that is valid for that message type.

The state of a message is changed by appending a completion message to the spool as described in [draft-hallambaker-mesh-protocol].

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 message will be removed if the sequence is subsequently purged.

9.4.2. Four Corner Model

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

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-schema-10.html for artwork.)

Figure 4: Four Corner Messaging Model

The outbound Mesh Service checks to see that the request to send a message 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. Services that fail to implement appropriate controls are likely to be subject to sanctions from either their users or from other services.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-schema-10.html for artwork.)

Figure 5: Performing Access Control on Outbound Messages

The inbound Mesh Service also checks to see that messages received are consistent with the service Acceptable Use Policy and the user's personal access control settings.

Mesh Services that fail to police abuse by their account holders SHOULD be subject to consequences in the same fashion as account holders.

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-schema-10.html for artwork.)

Figure 6: Performing Access Control on Inbound Messages

9.4.3. 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 naive 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.

10. Publications

Static QR codes MAY be used to allow contact exchange or device connection. In either case, the QR code contains an EARL providing the means of locating, decrypting and authenticating the published data.

The use of EARLs as a means of publishing encrypted data and the use of EARLs for location, decryption and authentication is discussed in [draft-hallambaker-mesh-dare] .

10.1. Profile Device

10.2. Contact Exchange

When used for contact exchange, the envelope payload is a CatalogedContact record.

Besides allowing for exchange of contact information on a business card, a user might have their contact information printed on personal property to facilitate return of lost property.

11. Schema

11.1. Shared Classes

The following classes are used as common elements in Mesh profile specifications.

11.1.1. Classes describing keys

11.1.2. Structure: KeyData

The KeyData 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

X509Chain: Binary [0..Many] X.509 Certificate chain.

X509CSR: Binary (Optional) X.509 Certificate Signing Request.

NotBefore: DateTime (Optional) If present specifies a time instant that use of the private key is not valid before.

NotOnOrAfter: DateTime (Optional) If present specifies a time instant that use of the private key is not valid on or after.

11.1.3. Structure: CompositePrivate

Inherits: Key

DeviceKeyUdf: String (Optional) UDF fingerprint of the bound device key (if used).

11.2. Assertion classes

Classes that are derived from an assertion.

11.2.1. 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.

11.2.2. Structure: Condition

Parent class from which all condition classes are derived.

[No fields]

11.2.3. Base Classes

Abstract classes from which the Profile, Activation and Connection classes are derived.

11.2.4. Structure: Connection

Inherits: Assertion

SubjectUdf: String (Optional) UDF of the connection target.

AuthorityUdf: String (Optional) UDF of the connection source.

11.2.5. Structure: Activation

Inherits: Assertion

Contains the private activation information for a Mesh application running on a specific device

ActivationKey: String (Optional) Secret seed used to derive keys that are not explicitly specified.

Entries: ActivationEntry [0..Many] Activation of named resources.

11.2.6. Structure: ActivationEntry

Resource: String (Optional) Name of the activated resource

Key: KeyData (Optional) The activation key or key share

11.2.7. Mesh Profile Classes

Classes describing Mesh Profiles. All Profiles are Assertions derived from Assertion.

11.2.8. Structure: Profile

Inherits: Assertion

Parent class from which all profile classes are derived

ProfileSignature: KeyData (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.

11.2.9. Structure: ProfileDevice

Inherits: Profile

Describes a mesh device.

Description: String (Optional) Description of the device

BaseEncryption: KeyData (Optional) Base key contribution for encryption keys. Also used to decrypt activation data sent to the device during connection to an account.

BaseAuthentication: KeyData (Optional) Base key contribution for authentication keys. Also used to authenticate the device during connection to an account.

BaseSignature: KeyData (Optional) Base key contribution for signature keys.

11.2.10. Structure: ProfileAccount

Base class for the account profiles ProfileUser and ProfileGroup. These subclasses may be merged at some future date.

Inherits: Profile

AccountAddress: String (Optional) The account address. This is either a DNS service address (e.g. alice@example.com) or a Mesh Name (@alice).

ServiceUdf: String (Optional) The fingerprint of the service profile to which the account is currently bound.

EscrowEncryption: KeyData (Optional) Escrow key associated with the account.

AccountEncryption: KeyData (Optional) Key currently used to encrypt data under this profile

AdministratorSignature: KeyData (Optional) Key used to sign connection assertions to the account.

11.2.11. Structure: ProfileUser

Inherits: ProfileAccount

Account assertion. This is signed by the service hosting the account.

AccountAuthentication: KeyData (Optional) Key used to authenticate requests made under this user account.

AccountSignature: KeyData (Optional) Key used to sign data under the account.

11.2.12. Structure: ProfileGroup

Inherits: ProfileAccount

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]

11.2.13. Structure: ProfileService

Inherits: Profile

Profile of a Mesh Service

ServiceAuthentication: KeyData (Optional) Key used to authenticate service connections.

ServiceEncryption: KeyData (Optional) Key used to encrypt data under this profile

ServiceSignature: KeyData (Optional) Key used to sign data under the account.

11.2.14. Structure: ProfileHost

Inherits: Profile

KeyAuthentication: KeyData (Optional) Key used to authenticate service connections.

KeyEncryption: KeyData (Optional) Key used to pass encrypted data to the device such as a

11.2.15. Connection Assertions

Connection assertions are used to authenticate and authorize interactions between devices and the service currently servicing the account. They SHOULD NOT be visible to external parties.

11.2.16. Structure: ConnectionDevice

Inherits: Connection

Connection assertion used to authenticate service requests made by a device.

AccountAddress: String (Optional) The account address

DeviceSignature: KeyData (Optional) The signature key for use of the device under the profile

DeviceEncryption: KeyData (Optional) The encryption key for use of the device under the profile

DeviceAuthentication: KeyData (Optional) The authentication key for use of the device under the profile

11.2.17. Structure: ConnectionApplication

Inherits: Connection

Connection assertion stating that a particular device is

[No fields]

11.2.18. Structure: ConnectionGroup

Describes the connection of a member to a group.

Inherits: Connection

[No fields]

11.2.19. Structure: ConnectionService

Inherits: Connection

[No fields]

11.2.20. Structure: ConnectionHost

Inherits: Connection

[No fields]

11.2.21. Activation Assertions

11.2.22. Structure: ActivationDevice

Contains activation data for device specific keys used in the context of a Mesh account.

Inherits: Activation

AccountUdf: String (Optional) The UDF of the account

11.2.23. Structure: ActivationAccount

Inherits: Activation

ProfileSignature: KeyData (Optional) Grant access to profile online signing key used to sign updates to the profile.

AdministratorSignature: KeyData (Optional) Grant access to Profile administration key used to make changes to administrator catalogs.

AccountEncryption: KeyData (Optional) Grant access to ProfileUser account encryption key

AccountAuthentication: KeyData (Optional) Grant access to ProfileUser account authentication key

AccountSignature: KeyData (Optional) Grant access to ProfileUser account signature key

11.2.24. Structure: ActivationApplication

Inherits: Activation

[No fields]

11.3. Data Structures

Classes describing data used in cataloged data.

11.3.1. Structure: Contact

Inherits: Assertion

Base class for contact entries.

Id: String (Optional) The globally unique contact identifier.

anchors: Anchor [0..Many] Mesh fingerprints associated with the contact.

NetworkAddresses: NetworkAddress [0..Many] Network address entries

Locations: Location [0..Many] The physical locations the contact is associated with.

Roles: Role [0..Many] The roles of the contact

Bookmark: Bookmark [0..Many] The Web sites and other online presences of the contact

Sources: TaggedSource [0..Many] Source(s) from which this contact was constructed.

11.3.2. Structure: Anchor

Trust anchor

Udf: String (Optional) The trust anchor.

Validation: String (Optional) The means of validation.

11.3.3. Structure: TaggedSource

Source from which contact information was obtained.

LocalName: String (Optional) Short name for the contact information.

Validation: String (Optional) The means of validation.

BinarySource: Binary (Optional) The contact data in binary form.

EnvelopedSource: Enveloped (Optional) The contact data in enveloped form. If present, the BinarySource property is ignored.

11.3.4. Structure: ContactGroup

Inherits: Contact

Contact for a group, including encryption groups.

[No fields]

11.3.5. Structure: ContactPerson

Inherits: Contact

CommonNames: PersonName [0..Many] List of person names in order of preference

11.3.6. Structure: ContactOrganization

Inherits: Contact

CommonNames: OrganizationName [0..Many] List of person names in order of preference

11.3.7. Structure: OrganizationName

The name of an organization

Inactive: Boolean (Optional) If true, the name is not in current use.

RegisteredName: String (Optional) The registered name.

DBA: String (Optional) Names that the organization uses including trading names and doing business as names.

11.3.8. Structure: PersonName

The name of a natural person

Inactive: Boolean (Optional) If true, the name is not in current use.

FullName: String (Optional) The preferred presentation of the full name.

Prefix: String (Optional) Honorific or title, E.g. Sir, Lord, Dr., Mr.

First: String (Optional) First name.

Middle: String [0..Many] Middle names or initials.

Last: String (Optional) Last name.

Suffix: String (Optional) Nominal suffix, e.g. Jr., III, etc.

PostNominal: String (Optional) Post nominal letters (if used).

11.3.9. Structure: NetworkAddress

Provides all means of contacting the individual according to a particular network address

Inactive: Boolean (Optional) If true, the name is not in current use.

Address: String (Optional) The network address, e.g. alice@example.com

NetworkCapability: String [0..Many] The capabilities bound to this address.

EnvelopedProfileAccount: Enveloped (Optional) The account profile

Protocols: NetworkProtocol [0..Many] Public keys associated with the network address

11.3.10. Structure: NetworkProtocol

Protocol: String (Optional) The IANA protocol|identifier of the network protocols by which the contact may be reached using the specified Address.

11.3.11. Structure: Role

OrganizationName: String (Optional) The organization at which the role is held

Titles: String [0..Many] The titles held with respect to that organization.

Locations: Location [0..Many] Postal or physical addresses associated with the role.

11.3.12. Structure: Location

Appartment: String (Optional)

Street: String (Optional)

District: String (Optional)

Locality: String (Optional)

County: String (Optional)

Postcode: String (Optional)

Country: String (Optional)

11.3.13. Structure: Bookmark

Uri: String (Optional)

Title: String (Optional)

Role: String [0..Many]

11.3.14. 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.

11.3.15. Structure: Task

Key: String (Optional) Unique key.

Start: DateTime (Optional)

Finish: DateTime (Optional)

StartTravel: String (Optional)

FinishTravel: String (Optional)

TimeZone: String (Optional)

Title: String (Optional)

Description: String (Optional)

Location: String (Optional)

Trigger: String [0..Many]

Conference: String [0..Many]

Repeat: String (Optional)

Busy: Boolean (Optional)

11.4. Catalog Entries

11.4.1. Structure: CatalogedEntry

Base class for cataloged Mesh data.

Labels: String [0..Many] The set of labels describing the entry

11.4.2. Structure: CatalogedDevice

Inherits: CatalogedEntry

Public device entry, indexed under the device ID Hello

Udf: String (Optional) UDF of the signature key of the device in the Mesh

DeviceUdf: String (Optional) UDF of the offline signature key of the device

SignatureUdf: String (Optional) UDF of the account online signature key

EnvelopedProfileUser: Enveloped (Optional) The Mesh profile

EnvelopedProfileDevice: Enveloped (Optional) The device profile

EnvelopedConnectionUser: Enveloped (Optional) The public assertion demonstrating connection of the Device to the Mesh

EnvelopedActivationDevice: Enveloped (Optional) The activation of the device within the Mesh account

EnvelopedActivationAccount: Enveloped (Optional) The activation of the device within the Mesh account

EnvelopedActivationApplication: Enveloped [0..Many] Application activations granted to the device.

11.4.3. Structure: CatalogedPublication

Inherits: CatalogedEntry

A publication.

Id: String (Optional) Unique identifier code

Authenticator: String (Optional) The witness key value to use to request access to the record.

EnvelopedData: DareEnvelope (Optional) Dare Envelope containing the entry data. The data type is specified by the envelope metadata.

NotOnOrAfter: DateTime (Optional) Expiration time (inclusive)

11.4.4. Structure: CatalogedCredential

Inherits: CatalogedEntry
Protocol: String (Optional)
Service: String (Optional)
Username: String (Optional)
Password: String (Optional)

11.4.5. Structure: CatalogedNetwork

Inherits: CatalogedEntry
Protocol: String (Optional)
Service: String (Optional)
Username: String (Optional)
Password: String (Optional)

11.4.6. Structure: CatalogedContact

Inherits: CatalogedEntry
Key: String (Optional) Unique key.
Self: Boolean (Optional) If true, this catalog entry is for the user who created the catalog.

11.4.7. Structure: CatalogedAccess

Inherits: CatalogedEntry
[No fields]

11.4.8. Structure: CryptographicCapability

Id: String (Optional) The identifier of the capability. If this is a user capability, MUST match the KeyData identifier. If this is a serviced capability, MUST match the value of ServiceId on the corresponding service capability.
KeyData: KeyData (Optional) The key that enables the capability

EnvelopedKeyShares: Enveloped [0..Many] One or more enveloped key shares.

SubjectId: String (Optional) The identifier of the resource that is controlled using the key.

SubjectAddress: String (Optional) The address of the resource that is controlled using the key.

11.4.9. Structure: CapabilityDecrypt

Inherits: CryptographicCapability

The corresponding key is a decryption key

[No fields]

11.4.10. Structure: CapabilityDecryptPartial

Inherits: CapabilityDecrypt

The corresponding key is an encryption key

ServiceId: String (Optional) The identifier used to claim the capability from the service. [Only present for a partial capability.]

ServiceAddress: String (Optional) The service account that supports a serviced capability. [Only present for a partial capability.]

11.4.11. Structure: CapabilityDecryptServiced

Inherits: CapabilityDecrypt

The corresponding key is an encryption key

AuthenticationId: String (Optional) UDF of trust root under which request to use a serviced capability must be authorized. [Only present for a serviced capability]

11.4.12. Structure: CapabilitySign

Inherits: CryptographicCapability

The corresponding key is an administration key

[No fields]

11.4.13. Structure: CapabilityKeyGenerate

Inherits: CryptographicCapability

The corresponding key is a key that may be used to generate key shares.

[No fields]

11.4.14. Structure: CapabilityFairExchange

Inherits: CryptographicCapability

The corresponding key is a decryption key to be used in accordance with the Micali Fair Electronic Exchange with Invisible Trusted Parties protocol.

[No fields]

11.4.15. Structure: CatalogedBookmark

Inherits: CatalogedEntry

Uri: String (Optional)

Title: String (Optional)

Path: String (Optional)

11.4.16. Structure: CatalogedTask

Inherits: CatalogedEntry

EnvelopedTask: Enveloped (Optional)

Title: String (Optional)

Key: String (Optional) Unique key.

11.4.17. Structure: CatalogedApplication

Inherits: CatalogedEntry

Key: String (Optional)

EnvelopedCapabilities: DareEnvelope [0..Many] Enveloped keys for use with Application

11.4.18. Structure: CatalogedMember

ContactAddress: String (Optional)

MemberCapabilityId: String (Optional)

ServiceCapabilityId: String (Optional)

Inherits: CatalogedEntry

11.4.19. Structure: CatalogedGroup

Inherits: CatalogedApplication

EnvelopedProfileGroup: Enveloped (Optional) The Mesh profile

EnvelopedActivationAccount: Enveloped (Optional) The activation of the device within the Mesh account

11.4.20. Structure: CatalogedApplicationSSH

Inherits: CatalogedApplication

[No fields]

11.4.21. Structure: CatalogedApplicationMail

Inherits: CatalogedApplication

[No fields]

11.4.22. Structure: CatalogedApplicationNetwork

Inherits: CatalogedApplication

[No fields]

11.5. Publications

11.5.1. Structure: DevicePreconfiguration

A data structure that is passed

EnvelopedProfileDevice: Enveloped (Optional) The device profile

EnvelopedConnectionDevice: Enveloped (Optional) The device connection

ConnectUri: String (Optional) The connection URI. This would normally be printed on the device as a QR code.

11.6. Messages

11.6.1. Structure: Message

MessageId: String (Optional) Unique per-message ID. When encapsulating a Mesh Message in a DARE envelope, the envelope EnvelopeID field MUST be a UDF fingerprint of the MessageId value.

Sender: String (Optional)

Recipient: String (Optional)

11.6.2. Structure: MessageError

Inherits: Message

ErrorCode: String (Optional)

11.6.3. Structure: MessageComplete

Inherits: Message

References: Reference [0..Many]

11.6.4. Structure: MessagePinValidated

Inherits: Message

AuthenticatedData: DareEnvelope (Optional) Enveloped data that is authenticated by means of the PIN

ClientNonce: Binary (Optional) Nonce provided by the client to validate the PIN

PinId: String (Optional) Pin identifier value calculated from the PIN code, action and account address.

PinWitness: Binary (Optional) Witness value calculated as KDF (Device.Udf + AccountAddress, ClientNonce)

11.6.5. Structure: MessagePin

Account: String (Optional)

Inherits: Message

Expires: DateTime (Optional)

Automatic: Boolean (Optional) If true, authentication against the PIN code is sufficient to complete the associated action without further authorization.

SaltedPin: String (Optional) PIN code bound to the specified action.

Action: String (Optional) The action to which this PIN code is bound.

11.6.6. Structure: RequestConnection

Connection request message. This message contains the information

Inherits: MessagePinValidated

AccountAddress: String (Optional)

11.6.7. Structure: AcknowledgeConnection

Connection request message generated by a service on receipt of a valid MessageConnectionRequestClient

Inherits: Message

EnvelopedRequestConnection: Enveloped (Optional) The client connection request.

ServerNonce: Binary (Optional)

Witness: String (Optional)

11.6.8. Structure: RespondConnection

Respond to RequestConnection message to grant or refuse the connection request.

Inherits: Message

Result: String (Optional) The response to the request. One of "Accept", "Reject" or "Pending".

CatalogedDevice: CatalogedDevice (Optional) The device information. MUST be present if the value of Result is "Accept". MUST be absent or null otherwise.

11.6.9. Structure: MessageContact

Inherits: MessagePinValidated

Reply: Boolean (Optional) If true, requests that the recipient return their own contact information in reply.

Subject: String (Optional) Optional explanation of the reason for the request.

PIN: String (Optional) One time authentication code supplied to a recipient to allow authentication of the response.

11.6.10. Structure: GroupInvitation

Inherits: Message

Text: String (Optional)

11.6.11. Structure: RequestConfirmation

Inherits: Message

Text: String (Optional)

11.6.12. Structure: ResponseConfirmation

Inherits: Message

Request: Enveloped (Optional)

Accept: Boolean (Optional)

11.6.13. Structure: RequestTask

Inherits: Message

[No fields]

11.6.14. Structure: MessageClaim

Inherits: Message

PublicationId: String (Optional)

ServiceAuthenticate: String (Optional)

DeviceAuthenticate: String (Optional)

Expires: DateTime (Optional)

11.6.15. Structure: ProcessResult

For future use, allows logging of operations and results

Inherits: Message

Success: Boolean (Optional)

ErrorReport: String (Optional) The error report code.

12. 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].

13. IANA Considerations

All the IANA considerations for the Mesh documents are specified in this document

14. Acknowledgements

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

15. Normative References

[draft-hallambaker-mesh-architecture]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>>.

[draft-hallambaker-mesh-callsign]

Hallam-Baker, P., "Mathematical Mesh 3.0 Part VII: Mesh Callsign Service", Work in Progress, Internet-Draft, draft-hallambaker-mesh-callsign-01, 23 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-callsign-01>>.

- [draft-hallambaker-mesh-dare]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part III : Data At Rest Encryption (DARE)", Work in Progress, Internet-Draft, draft-hallambaker-mesh-dare-14, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-dare-14>>.
- [draft-hallambaker-mesh-discovery]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part VI: Mesh Discovery Service", Work in Progress, Internet-Draft, draft-hallambaker-mesh-discovery-01, 13 January 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-discovery-01>>.
- [draft-hallambaker-mesh-protocol]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part V: Protocol Reference", Work in Progress, Internet-Draft, draft-hallambaker-mesh-protocol-12, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-protocol-12>>.
- [draft-hallambaker-mesh-security]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>>.
- [draft-hallambaker-mesh-udf]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.", Work in Progress, Internet-Draft, draft-hallambaker-mesh-udf-15, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-udf-15>>.
- [draft-hallambaker-threshold]
Hallam-Baker, P., "Threshold Modes in Elliptic Curves", Work in Progress, Internet-Draft, draft-hallambaker-threshold-06, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-threshold-06>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.

16. Informative References

[draft-hallambaker-mesh-developer]

Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-10, 27 July 2020, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-10>>.

[draft-irtf-cfrg-frost]

Connolly, D., Komlo, C., Goldberg, I., and C. A. Wood, "Two-Round Threshold Schnorr Signatures with FROST", Work in Progress, Internet-Draft, draft-irtf-cfrg-frost-04, 29 March 2022, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-frost-04>>.

[draft-komlo-frost]

Komlo, C. and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures", Work in Progress, Internet-Draft, draft-komlo-frost-00, 7 August 2020, <<https://datatracker.ietf.org/doc/html/draft-komlo-frost-00>>.

[RFC2426] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, DOI 10.17487/RFC2426, September 1998, <<https://www.rfc-editor.org/rfc/rfc2426>>.

[RFC5545] Desruisseaux, B., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <<https://www.rfc-editor.org/rfc/rfc5545>>.

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
ThresholdSecrets.com
20 April 2022

Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.
draft-hallambaker-mesh-udf-16

Abstract

This document describes the underlying 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. For convenience, UDFs are typically presented to the user in the form of a Base32 encoded string. 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.

[Note to Readers]

Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

This document is also available online at <http://mathmesh.com/Documents/draft-hallambaker-mesh-udf.html>.

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 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.

Table of Contents

1. Introduction	4
1.1. UDF Types	5
1.1.1. Cryptographic Keys and Nonces	5
1.1.2. Fingerprint type UDFS	6
1.2. Using UDFs in URIs	7
1.2.1. Name Form	7
1.2.2. Locator Form	7
1.3. Secure Internet Names	9
2. Definitions	10
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	12
3.1.1.	Precision Improvement	12
3.2.	Type Identifier	12
3.3.	Content Type Identifier	14
3.4.	Truncation	14
3.4.1.	Compressed presentation	15
3.5.	Presentation	15
3.6.	Alternative Presentations	16
3.6.1.	Word Lists	16
3.6.2.	Image List	16
4.	Fixed Length UDFs	16
4.1.	Nonce Type	16
4.2.	OID Identified Sequence	17
4.3.	Encryption/Authentication Type	18
4.4.	Key Pair Derivation	18
4.4.1.	Extraction step	21
4.4.2.	Elliptic Curve Diffie Hellman Key Pairs type 1-4	22
4.4.3.	Elliptic Curve Diffie Hellman Key Pairs type 5-7	23
4.4.4.	RSA Key Pairs	24
4.4.5.	Any Key Algorithm	26
4.5.	Shamir Shared Secret	27
4.5.1.	Secret Generation	28
4.5.2.	Recovery	29
5.	Variable Length UDFs	30
5.1.	Content Digest	31
5.1.1.	Content Digest Value	31
5.1.2.	Typed Content Digest Value	31
5.1.3.	Content Digest Compression	32
5.1.4.	Content Digest Presentation	32
5.1.5.	Example Encoding	33
5.1.6.	Using SHA-2-512 Digest	33
5.1.7.	Using SHA-3-512 Digest	34
5.1.8.	Using SHA-2-512 Digest with Compression	34
5.1.9.	Using SHA-3-512 Digest with Compression	35
5.2.	Authenticator UDF	35
5.2.1.	Authentication Content Digest Value	36
5.2.2.	Authentication Value	36
5.3.	Content Type Values	38
5.3.1.	PKIX Certificates and Keys	39
5.3.2.	OpenPGP Key	40
5.3.3.	DNSSEC	40
6.	UDF URIs	40
6.1.	Name form URI	41
6.2.	Locator form URI	41
6.2.1.	DNS Web service discovery	41

6.2.2.	Content Identifier	42
6.2.3.	Target URI	42
6.2.4.	Postprocessing	42
6.2.5.	Decryption and Authentication	43
6.2.6.	QR Presentation	43
7.	Strong Internet Names	43
8.	Security Considerations	43
8.1.	Confidentiality	44
8.2.	Availability	44
8.3.	Integrity	44
8.4.	Work Factor and Precision	44
8.5.	Semantic Substitution	45
8.6.	QR Code Scanning	46
9.	IANA Considerations	46
9.1.	Protocol Service Name	46
9.2.	Well Known	47
9.3.	URI Registration	47
9.4.	Media Types Registrations	48
9.4.1.	Media Type: application/pkix-keyinfo	48
9.4.2.	Media Type: application/udf	49
9.5.	Uniform Data Fingerprint Type Identifier Registry	50
9.5.1.	The name of the registry	50
9.5.2.	Required information for registrations	50
9.5.3.	Applicable registration policy	50
9.5.4.	Size, format, and syntax of registry entries	50
9.5.5.	Initial assignments and reservations	51
10.	Acknowledgements	51
11.	Appendix A: Prime Values for Secret Sharing	51
12.	Appendix B: Shamir Shared Secret Recovery Using Lagrange Interpolation	54
13.	Normative References	57
14.	Informative References	59

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 four 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:

```
Nonce:      ND5L-BTJE-W372-4TUS-E23N-FIXU-6ARA
Secret:     UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ
SHA-2 Digest:MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
SHA-3 Digest:KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN
Public Key: OAYC-4MAH-AYBS-WZLQ-AUAA-GIYA-AQQK-XW5Y-YO6L-TIJU-43NU-2
           2J5-VP23-IJ7A-4JCH-LBCC-LFAZ-6QHO-GLUY-FEQ
```

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:
ND5L-BTJE-W372-4TUS-E23N-FIXU-6ARA
```

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.

KeyValue:

DE BB 13 26 28 7A 83 5B 58 65 38 B1 48 8C 7C

Encryption/Authenticator UDF:

UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ

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: UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ

Share 0: SAQF-K5LV-2HDO-PSY5-YQBN-CPMS-7MXM-K

Share 1: SAQQ-KDBQ-SBT2-OG5Y-FSWT-2QTU-VXIQ-4

Share 2: SARL-JIXL-J4EG-M3CS-SVL2-SR2W-MBZY-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: ABIR-Y75D-WV63-P5WC-KR3F-VA57-YZ6I

1.2. Using UDFs in 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://example.com/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:

EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z

One or more electronic forms of the invoice are encrypted under the key EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z 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/EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z

The EARL is then rendered as a QR code:

(Artwork only available as svg: No external link available, see draft-hallambaker-mesh-udf-16.html for artwork.)

Figure 1: 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.

Discover ("example.com", "mmm-udf") =
https://example.com/.well-known/mmm-udf/

Next the fingerprint of the source UDF is obtained.

UDF (EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z) =
MBUC-DFHR-NRPA-ZLHR-CZRT-75CV-VWL2-OSIC-2YUH-QJ7C-3WWA-7CLC-W4X7-YNBO

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:

<https://example.com/.well-known/mmm-udf/MBUC-DFHR-NRPA-ZLHR-CZRT-75CV-VWL2-OSIC-2YUH-QJ7C-3WWA-7CLC-W4X7-YNBO>

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:

Type ID	Initial	Algorithm
0	A	HMAC_SHA_2_512
1	A	HMAC_SHA_3_512
32	E	HKDF_AES_512
33	E	HKDF_AES_512
80	K	SHA_3_512
81	K	SHA_3_512 (20 compressed)
82	K	SHA_3_512 (30 compressed)
83	K	SHA_3_512 (40 compressed)
84	K	SHA_3_512 (50 compressed)
96	M	SHA_2_512
97	M	SHA_2_512 (20 compressed)
98	M	SHA_2_512 (30 compressed)
99	M	SHA_2_512 (40 compressed)
100	M	SHA_2_512 (50 compressed)
104	N	Nonce Data
112	O	OID distinguished sequence (DER encoded)
144	S	Shamir Secret Share
200	Z	Secret seed

Table 1

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.

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 finger print 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. Compressed presentation

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 an identifier of 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 permit 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 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 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. OID Identified Sequence

An OID Identified Sequence Type UDF consists of the type identifier octet 108 followed by the Binary Data Sequence value.

The Binary Data Sequence value is an octet sequence that contains the DER encoding of the following ASN.1 data:

```
OIDInfo ::= SEQUENCE {
    algorithm      AlgorithmIdentifier,
    data           BIT STRING }

AlgorithmIdentifier ::= SEQUENCE {
    algorithm      OBJECT IDENTIFIER,
    parameters    ANY DEFINED BY algorithm OPTIONAL }
```

OID Identified Sequences are intended to allow arbitrary data sequences to be encoded in the UDF format without exhausting the limited type identifier space.

In particular, OID Identified Sequences MAY be used to specify public and private key values.

Given the following Ed25519 public key:

```
AB DB B8 C3  BC B9 A1 34  E6 DB 4D 69  3D AB F5 B4
27 E0 E2 44  75 84 42 59  41 9F 40 EE  32 E9 82 92
```

The equivalent DER encoding is:

```
30 2E 30 07  06 03 2B 65  70 05 00 03  23 00 04 20
AB DB B8 C3  BC B9 A1 34  E6 DB 4D 69  3D AB F5 B4
27 E0 E2 44  75 84 42 59  41 9F 40 EE  32 E9 82 92
```

To encode this key as a UDF OID sequence we prepend the value OID and convert to Base32:

OID: OAYC-4MAH-AYBS-WZLQ-AUAA-GIYA-AQQK-XW5Y-YO6L-TIJU-43NU-2
 2J5-VP23-IJ7A-4JCH-LBCC-LFAZ-6QHO-GLUY-FEQ

The corresponding UDF content digest value is more compact and allows us to identify the key unambiguously but does not provide the value:

MDGT-32T4-MBOH-S5XB-7THD-VD6U-4NOU

4.3. Encryption/Authentication Type

Encryption and Authenticator Type UDFs consists of the type identifier specifying the algorithm to be used on the key data 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 and Authenticator 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 or Authenticator type UDF is a UDF fingerprint of the UDF in Base32 presentation with content type 'application/udf'.

4.4. Key Pair Derivation

The key pair derivation type is used to specify a public key pair value by means of a sufficiently random input to a deterministic key generation function.

A key pair derivation Type UDF consists of the type identifier octet 200 followed by the Binary Data Sequence value.

The first two octets of the Binary Data Sequence value comprise the Key Specifier which specifies the algorithm and key uses for which the private key is to be derived.

- * Bits 6-7 of the first octet specify the key use.
- * Bits 0-5 of the first byte and bits 0-7 of the second specify the key type in network byte order.

In the unlikely event that this code space is ever exhausted, allocation of a new UDF type code will be required.

The following key uses are specified:

Code	Algorithm	Description
0	Any	Any
1	Encryption	Encryption
2	Signature	Signature
3	Authentication	Authentication

Table 2

Two types of key type are defined: Explicit and Generic.

Explicit key types specify a public key cryptographic algorithm and all the parameters required to generate a key pair. Generic key types are used to specify a type of key but not the algorithm which MUST be specified when the key is generated.

Derivation of key pairs for the following algorithms is specified in this document:

Code	Algorithm	Description
0	Any	Seed MAY be used to generate keypairs for any algorithm
1	X25519	X25519 keypair as described in RFC7748
2	X448	X448 keypair as described in RFC7748
3	Ed25519	Ed25519 keypair as described in RFC8032
4	Ed448	Ed448 keypair as described in RFC8032
5	P256	NIST curve P-256
6	P384	NIST curve P-384

7	P521	NIST curve P-521
8	RSA2048	2048 bit RSA keypair
9	RSA3072	3072 bit RSA keypair
10	RSA4096	4096 bit RSA keypair
11-255	ReservedIetf	Reserved for IETF recommended algorithms
256	MeshProfileDevice	Mesh device profile
257	MeshActivationDevice	Mesh device activation
258	MeshProfileAccount	Mesh account account
259	MeshActivationAccount	Mesh account activation
260	MeshProfileService	Mesh service profile
261	MeshActivationService	Mesh host activation
262-511	ReservedMesh	Reserved for future Mesh use

Table 3

The key parameter derivation function takes as inputs, the UDF seed value seed, the parameter identifier param and an optional string specifying a key name keyname.

KeyParam (seed, param, keyname)

The value param is an octet sequence determined by the actual key type generated. The first two octets of parm are always equal to the key identifier for the key algorithm and key usage being generated. If the key derivation algorithm requires multiple inputs, additional octets are specified for each of the different inputs required.

The HKDF function [RFC5869] is used to derive key pairs for all the algorithms specified in this document. Derivation functions for additional key algorithms MAY use a different function for this purpose provided that it meets the applicable security requirements.

The HKDF function is specified as a two-step extract-expand process with an optional non-secret value input at both steps.

4.4.1. Extraction step

The HKDF extraction step calculates a PRK value from a salt and IKM:

```
HKDF-Extract(salt, IKM) -> PRK
```

The IKM value is the binary presentation of the complete Binary Data Sequence as originally specified. The salt value is null.

The output from the extraction step forms the input to the expand step:

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

The info parameter of the HKDF function is the concatenation of alg, param and the UTF8 binary representation of keyname.

```
info = alg + param + keyname.UTF8()
```

An X25519 key may be derived as follows:

```
Fingerprint =
  ZAAA-CDJ5-DHPA-DUUW-WIPQ-UXNC-DSAR-U7A
```

```
IKM =
  00 01 0D 3D 19 DE 01 D2 96 B2 1F 0A 5D A2 1C 81
  1A 7C
```

```
salt =
  00 01
```

```
PRK =
  DA 2E 80 6F 2D B1 54 56 7E 27 B4 91 49 0A 35 3A
  5D 99 92 AA A2 2F 2D 2A 50 4B 13 5B 87 DF 63 67
  62 92 67 9C B3 B8 10 47 31 52 A2 42 FA 04 84 39
  7A 64 15 84 C0 6B 51 F7 19 4A 20 35 BA 2E D1 59
```

```
OKM =
  E7 22 39 E1 AB 77 AC 9C B4 6A A0 12 27 68 9E 28
  14 60 2F A8 76 08 38 5E D5 E6 5D E7 0C C8 42 E8
```

```
Key =
  E7 22 39 E1 AB 77 AC 9C B4 6A A0 12 27 68 9E 28
  14 60 2F A8 76 08 38 5E D5 E6 5D E7 0C C8 42 E8
```

Derivation of an X448 key:

Fingerprint =
 ZAAA-FFQA-3LE5-SAHG-E6K6-HOTN-TVLB-K4A

Key =
 AE 6A 6D 0B CC 48 C3 31 E7 55 0F 52 F9 96 83 C5
 15 7C 8A 74 80 36 B7 E9 17 24 D7 DD A1 56 76 3C
 15 00 68 B7 23 F5 DB 32 48 1B 72 C0 2E B0 22 45
 A3 B8 80 67 B3 88 06 9F

Derivation of an Ed25519 key:

Fingerprint =
 ZAAA-GZ5N-PSNF-7LMS-QJZN-3O2X-GJXV-X6I

Key =
 3A 36 00 56 2E EC 2F 24 A7 8C 22 F3 A9 A2 EF 1B
 6E AF 07 D4 99 28 53 A5 5B 0A CC EE 4C 3B 7D 30

Derivation of an Ed448 key:

Fingerprint =
 ZAAA-ILZB-KTQV-YWUK-FO7E-MQVV-EWPR-UPA

Key =
 DF 5A 89 B8 1D 56 92 41 32 D1 B2 C9 4F 74 69 E3
 C9 E5 5F 23 33 A1 CE 22 54 08 EE 53 46 0F 9B 13
 9D 54 95 2B F9 D9 77 2A FA 07 3C 9D 89 CC C5 0E
 7E 86 7E 84 7C 58 5D 89

4.4.2. Elliptic Curve Diffie Hellman Key Pairs type 1-4

The generation of key pairs for X25519, X448, Ed25519 and Ed448 is specified in [RFC7748] and [RFC8032]. In each case, the public and private key parameters are generated from a string of random octets whose transformation to the secret scalar function is described in the document.

Thus, info is the null string and the value L is specified as follows:

Algorithm	L
X25519	256
X448	448
Ed25519	256
Ed448	448

Table 4

4.4.3. Elliptic Curve Diffie Hellman Key Pairs type 5-7

The generation of key pairs for the curves P-256, P-384 and P-521 described in [RFC5903] is not mandated by the specification. FIPS 186-4 specifies two approaches. A modified form of the mechanism Key Pair Generation Using Extra Random Bits specified in B.4.1 is used as follows:

The number of random bits L is given by the following table:

Algorithm	L
P-256	320
P-384	448
P-521	592

Table 5

Note that this rounds up the number of random bits required to the nearest integer multiple of 8.

The OKM value is interpreted as an integer in Network Byte Order, that is the first byte contains the most significant bits, to yield the parameter c.

The parameter c is reduced modulo the value of the prime field n to yield the secret scalar value d:

$$d = (c \bmod (n-1)) + 1.$$

A P-256 key may be derived as follows:

Fingerprint =

ZAAA-LLBO-4A4E-LFMH-EJ73-XVFG-7PZ5-V7Y

IKM =

00 05 AC 2E E0 38 45 95 87 22 7F BB D4 A6 FB F3
DA FF

salt =

00 05

PRK =

0F 48 0F 0C 93 30 AE EE 41 FD 8F A2 1C C2 C6 CA
3A E1 4B 54 E7 83 C0 25 85 F0 CD 2A 65 3F 18 A7
9F 2A 5A ED 6A E3 64 6A 05 7D 1A 1A B8 68 B3 F3
4F A9 10 9A 05 E1 A4 9A 2C CC 40 43 36 8A 24 C0

OKM =

E2 00 EC 22 63 17 D5 E5 52 F9 CD B6 45 23 A9 8B
EF 32 26 E0 24 A0 E7 2B 7F CB C2 0B CB FA 0F 5C
59 D1 7C 4A D8 12 2E 4C

Key = 823521039787465146191678159095729811571036184098859836027994
10986678676075099

Derivation of a P-384 key:

Fingerprint =

ZAAA-NPLI-G7Z3-WFD2-GBJ6-OONN-ELTO-MHA

Key = 369049211431889063087900251703207474490953076630513949620729
23012683284321458397574918591433311657724460124046828583

Derivation of a P-521 key:

Fingerprint =

ZAAA-PQCC-YFVT-LRWP-7MUZ-GJV3-HLX2-JPQ

Key = 634654264002940134552342747000178673315150389242882127875899
96717989335708073735991026483008684752699986204269344550
4370476919922072068801363203357706689700

4.4.4. RSA Key Pairs

Generation of RSA key pairs requires two parameters, p , q which are prime.

The value of the param input used to calculate info is the value of the key identifier value with one of the following tag values concatenated to the end.

Parameter	Tag	UTF8 equivalent string
p	[112]	p
q	[113]	q

Table 6

The value of L is the same for generating the OKM values from which q are derived and is determined by the algorithm identifier:

Algorithm	L
RSA-2048	1024
RSA-3072	1536
RSA-4096	2048

Table 7

The RSA parameter p is the smallest prime integer that is greater than the OKM value corresponding to the info value "p" interpreted as an integer in Network Byte Order.

The RSA parameter q is the smallest prime integer that is greater than the OKM value corresponding to the info value "q" interpreted as an integer in Network Byte Order.

Note that this algorithm does not mandate a particular method of primality testing nor does it impose any additional testing on the values p or q. If an application requires the use of primes with conditions it will be necessary to attempt multiple key derivations with different Binary Data Sequence values until parameters with the desired properties are found.

An RSA-2048 may be derived as follows:

Fingerprint =
ZAAA-RJ5I-OSMI-X2KH-MBHX-KUPB-OC54-NQI

IKM =
00 08 A7 A8 74 98 8B E9 47 60 4F 75 51 E1 70 BB
C6 C1

salt =
00 08

[Generation of the PRK as before]

Info(p) =
70

OKM(p) =
92 D4 DA FA C4 22 DB 17 B0 04 93 C6 F1 D2 7A AF
34 6F 69 98 54 1A F5 F3 E3 ED DA 98 F5 64 EE 6A

Info(q) =
71

OKM(q) =
01 50 07 9F B3 53 70 5A 7E 95 63 BD 19 8D 52 59
2F EE 38 E7 8F D4 46 D9 4C 55 E6 DD 39 CA DB 36

Key P = 664137588122357253348380132353218815863396125741622195396345
89986848279686793

Key Q = 593713231506709718978311683387355253795918273379156509909895
725618914057069

4.4.5. Any Key Algorithm

The Any key algorithm allows a single UDF value to be used to derive key pairs for multiple algorithms. The IKM value is the same for each key pair derived. The salt value is changed according to the algorithm for which the key is to be derived.

```
Fingerprint =  
  ZAAA-A6WP-XMGW-FUOF-2T5L-AHNL-FBPY-RSY
```

To generate an RSA-2048 key

```
salt =  
  00 08
```

```
Key P = 184377705562733023433840697873299239654937691662139401284561  
        64676903354830137
```

```
Key Q = 741016989403010251265552685128898155357242513700210607224783  
        50575007811846521
```

To generate an X25519 key

```
salt =  
  00 08
```

```
Key =  
  System.Byte[]
```

4.5. 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 $(x, f(x))$, 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.

4.5.1. 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 :

$$f(x) = a_0 + a_1.x + a_2.x^2 + \dots a_n.x^n$$

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.

a_0 Is the integer representation of the secret to be shared.

$a_1 \dots a_n$ 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 $f(1), f(2), \dots, f(n)$.

The most straightforward approach to generation of Shamir secrets is to generate the set of polynomial coefficients, a_0, a_1, \dots, a_n and use these to generate the share values $f(1), f(2), \dots, f(n)$.

Note that if this approach is adopted, there is a small probability that one or more of the values $f(1), f(2), \dots, f(n)$ 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.

4.5.2. 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

12 0A C3 1B FF 09 CD 86 CD 3E 6B 4B CF BA 91 8D

This has the UDF representation:

CIFM-GG77-BHGY-NTJ6-NNF4-7OUR-RU

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 a_0 .

Since a threshold of three shares is required, we will need a second order polynomial. The co-efficients of the polynomial a_1 , a_2 are random numbers smaller than the modulus:

$a_0 = 23981984180677462358025211329449202061$

$a_1 = 217449633820028444820075594055263988236$

$a_2 = 299283543253615188179136358544176182525$

The master secret is the value $f(0) = a_0$. The key shares are the values $f(1)$, $f(2) \dots f(5)$:

$f(1) = 200432794333382631893862556497121161315$

$f(2) = 294885957151441250861223403889609062605$

$f(3) = 307341472634853319260107753506912905931$

$f(4) = 237799340783618837090515605349032691293$

$f(5) = 86259561597737804352446959415968418691$

The first byte of each share specifies the recovery information (quorum, x value), the remaining bytes specify the share value in network byte order:

```

f(1) =
  30 96 C9 F3  B9 2E 52 75  AE C9 C5 75  B1 93 D0 B0
  63
f(2) =
  31 DD D8 F8  31 86 F7 0B  B9 E2 74 52  07 3E 2A B8
  CD
f(3) =
  32 E7 37 D0  85 08 F7 8F  A8 17 4B 00  4C CE C8 AA
  CB
f(4) =
  33 B2 E6 7C  B3 B4 54 01  79 68 49 80  82 45 AA 86
  5D
f(5) =
  34 40 E4 FC  BD 89 0C 61  2D D5 6F D2  A7 A2 D0 4B
  83

```

The UDF presentation of the key shares is thus:

```

f(1) = SAYJ-NSPT-XEXF-E5NO-ZHCX-LMMT-2CYG-G
f(2) = SAY5-3WHY-GGDP-OC5Z-4J2F-EBZ6-FK4M-2
f(3) = SAZO-ON6Q-QUEP-PD5I-C5FQ-ATGO-ZCVM-W
f(4) = SAZ3-FZT4-WO2F-IALZ-NBEY-BASF-VKDF-2
f(5) = SA2E-BZH4-XWEQ-YYJN-2VX5-FJ5C-2BFY-G

```

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:

0. Applying the digest algorithm to determine the Content Digest Value
1. Applying the digest algorithm to determine the Typed Content Digest Value
2. Determining the compression level from bytes 0-3 of the Typed Content Digest Value.
3. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.
4. 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:

$$\text{CDV} = \text{H}(\text{<Data>})$$

Where

H(x) is the cryptographic digest function

<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:

$$\text{TCDV} = \text{H}(\text{<Content-ID> + } ?? + \text{CDV})$$

Where

A + B represents concatenation of the binary sequences A and B.

<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. Content Digest 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 trailing zero bits Compression factor = 0

29 or fewer trailing zero bits Compression factor = 20

39 or fewer trailing zero bits Compression factor = 30

49 or fewer trailing zero bits Compression factor = 40

50 or more trailing 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. Content Digest Presentation

The type identifier is determined by the algorithm and compression factor as specified above.

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 06 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 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 06 DC 67 2A 67
 71 4F FF 8F 83 C4 55 38 36 38 AE 42 7A 82 9C 85

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-XI6O-ZSLU-2VOA

260 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF-XI6O-ZSLU-2VOA-
TZQ6-JMHP-TSXP

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

<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(<Content-ID> + ':' + H(<Data>)) =
 8A 86 8A 06 1C 54 6E 7E 3F 75 5F 39 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 0B 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 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 0B B2 1A 8D B9 F5 8D FC EC BD 01 F4 37

The 125 bit fingerprint value is KCFI-NCQG-DRKG-47R7-OVPT-TCHZ-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:

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 04 E6 56 3F F3 C0 C7
C4 1D 3F 43 DD DC F1 A5 67 A7 E0 67 9A B0 C6 B7

```

<Content-ID> + ':' + H(<Data>) =

```

74 65 78 74 2F 70 6C 61 69 6E 3A 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 04 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>)) =

```

8E 14 D9 19 4E D6 02 12 C3 30 A7 BB 5F C7 17 6D
AE 9A 56 7C A8 2A 23 1F 96 75 ED 53 10 EC E8 F2
60 14 24 D0 C8 BC 55 3D C0 70 F7 5E 86 38 1A 0B
CB 55 9C B2 87 81 27 FF 3C EC E2 F0 90 A0 00 00

```

The prefixed Binary Data Sequence is thus

```

61 8E 14 D9 19 4E D6 02 12 C3 30 A7 BB 5F C7 17
6D AE 9A 56 7C A8 2A 23 1F 96 75 ED 53 10 EC E8
F2 60 14 24 D0 C8 BC 55 3D C0 70 F7 5E 86 38 1A
0B CB 55 9C B2 87 81 27 FF 3C EC E2 F0 90 A0 00

```

The 125 bit fingerprint value is MGHBJWIZ-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:

0. Applying the digest algorithm to determine the Content Digest Value
1. Applying the MAC algorithm to determine the Authentication value
2. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.
3. 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. Authentication 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:

$$CDV = H(\langle Data \rangle)$$

Where

$H(x)$ is the cryptographic digest function

$\langle Data \rangle$ 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:

$$AV = MAC(\langle OKM \rangle, (\langle Content-ID \rangle + \text{??} + CDV))$$

Where

$\langle OKM \rangle$ is the authentication key as specified below

$MAC(\langle Key \rangle, \langle data \rangle)$ is the result of applying the Message Authentication Code algorithm to with Key $\langle Key \rangle$ and data $\langle data \rangle$

The value $\langle OKM \rangle$ 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
```

```
ContentType =
  74 65 78 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 69 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 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

```

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 quite different.

application/pkix-cert A PKIX Certificate

application/pkix-crl A PKIX CRL

application/pkix-keyinfo Content type identifier for PKIX KeyInfo data type

application/pgp-keys Content type identifier for OpenPGP Key

application/dns A DNS resource record in binary format

application/udf-encryption UDF Fingerprint list

application/udf-lock UDF Fingerprint list

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 CLRs 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 SubjectPublicKeyInfo structure defined in the PKIX certificate specification encoded using DER encoding rules.

The SubjectPublicKeyInfo structure is defined in [RFC5280] as follows:

```
SubjectPublicKeyInfo ::= SEQUENCE {  
    algorithm      AlgorithmIdentifier,  
    subjectPublicKey BIT STRING }
```

This schema results in an identical DER encoding to the OIDInfo sequence specified in section XXX. The distinction between these productions is that the OIDInfo schema is intended to be used to encode arbitrary data while the application/pkix-keyinfo content type is only intended to be used to describe public keys.

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 of 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 URI

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 URI

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.
- * Append the content identifier to the Web Service Endpoint as a suffix to form the target URI.
- * Retrieve content from the Web Service Endpoint by means of a GET method.
- * 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 of 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: Binary

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

Type name: application

Subtype name: udf

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:

Code	Description	Reference
0	HMAC_SHA_2_512	[This document]
1	HMAC_SHA_3_512	[This document]
32	HKDF_AES_512	[This document]
33	HKDF_AES_512	[This document]
80	SHA_3_512	[This document]
81	SHA_3_512 (20 compressed)	[This document]
82	SHA_3_512 (30 compressed)	[This document]
83	SHA_3_512 (40 compressed)	[This document]
84	SHA_3_512 (50 compressed)	[This document]
96	SHA_2_512	[This document]
97	SHA_2_512 (20 compressed)	[This document]
98	SHA_2_512 (30 compressed)	[This document]
99	SHA_2_512 (40 compressed)	[This document]
100	SHA_2_512 (50 compressed)	[This document]
104	Nonce Data	[This document]
112	OID distinguished sequence (DER encoded)	[This document]
144	Shamir Secret Share	[This document]
200	Secret seed	[This document]

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 $(2^8)^n$ that is larger than the secret to be encoded and determining the next largest number that is prime.

Bytes	Bits	Prime
1	8	2^8+1
2	16	$2^{16}+1$
3	24	$2^{24}+43$
4	32	$2^{32}+15$
5	40	$2^{40}+15$
6	48	$2^{48}+21$
7	56	$2^{56}+81$
8	64	$2^{64}+13$
9	72	$2^{72}+15$
10	80	$2^{80}+13$
11	88	$2^{88}+7$
12	96	$2^{96}+61$
13	104	$2^{104}+111$
14	112	$2^{112}+25$
15	120	$2^{120}+451$
16	128	$2^{128}+51$
17	136	$2^{136}+85$
18	144	$2^{144}+175$
19	152	$2^{152}+253$
20	160	$2^{160}+7$
21	168	$2^{168}+87$
22	176	$2^{176}+427$
23	184	$2^{184}+27$

24	192	$2^{192+133}$
25	200	$2^{200+235}$
26	208	$2^{208+375}$
27	216	$2^{216+423}$
28	224	$2^{224+735}$
29	232	$2^{232+357}$
30	240	$2^{240+115}$
31	248	2^{248+81}
32	256	$2^{256+297}$
33	264	$2^{264+175}$
34	272	2^{272+57}
35	280	2^{280+45}
36	288	$2^{288+127}$
37	296	2^{296+61}
38	304	2^{304+37}
39	312	2^{312+91}
40	320	2^{320+27}
41	328	2^{328+15}
42	336	$2^{336+241}$
43	344	$2^{344+231}$
44	352	2^{352+55}
45	360	$2^{360+105}$
46	368	$2^{368+127}$
47	376	$2^{376+115}$

48	384	$2^{384+231}$
49	392	$2^{392+207}$
50	400	$2^{400+181}$
51	408	2^{408+37}
52	416	$2^{416+235}$
53	424	$2^{424+163}$
54	432	$2^{432+1093}$
55	440	$2^{440+187}$
56	448	$2^{448+211}$
57	456	2^{456+21}
58	464	$2^{464+841}$
59	472	$2^{472+445}$
60	480	$2^{480+165}$
61	488	$2^{488+777}$
62	496	$2^{496+583}$
63	504	$2^{504+133}$
64	512	2^{512+75}

Table 8

For example, the prime to be used to share a 128 bit value is $2^{128} + 51$.

12. Appendix B: Shamir Shared Secret Recovery Using Lagrange Interpolation

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 :

$$f(x) = a_0 + a_1.x + a_2.x^2 + \dots a_n.x^n$$

The shared secret is the binary representation of the value a_0

Given n shares $(x_0, y_0), (x_1, y_1), \dots (x_{(n-1)}, y_{(n-1)})$, The corresponding the Lagrange basis polynomials $l_0, l_1, \dots l_{(n-1)}$ are given by:

$$l_m = ((x - x(m_0)) / (x(m) - x(m_0))) \cdot ((x - x(m_1)) / (x(m) - x(m_1))) \cdot \dots \cdot ((x - x(m_{(n-2)})) / (x(m) - x(m_{(n-2)})))$$

Where the values $m_0, m_1, \dots m_{(n-2)}$, are the integers 0, 1, .. $n-1$, excluding the value m .

These can be used to compute $f(x)$ as follows:

$$f(x) = y_0.l_0 + y_1.l_1 + \dots y_{(n-1)}.l_{(n-1)}$$

Since it is only the value of $f(0)$ that we are interested in, we compute the Lagrange basis for the value $x = 0$:

$$l_z_m = ((x(m_1)) / (x(m) - x(m_1))) \cdot ((x(m_2)) / (x(m) - x(m_2)))$$

Hence,

$$a_0 = f(0) = y_0.l_z_0 + y_1.l_z_1 + \dots y_{(n-1)}.l_{(n-1)}$$

The following C# code recovers the values.

```
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;
        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. Normative References

- [draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>>.
- [draft-hallambaker-mesh-dare]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part III : Data At Rest Encryption (DARE)", Work in Progress, Internet-Draft, draft-hallambaker-mesh-dare-14, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-dare-14>>.
- [draft-hallambaker-mesh-security]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>>.

- [draft-hallambaker-web-service-discovery]
Hallam-Baker, P., "DNS Web Service Discovery", Work in Progress, Internet-Draft, draft-hallambaker-web-service-discovery-06, 5 August 2021,
<<https://datatracker.ietf.org/doc/html/draft-hallambaker-web-service-discovery-06>>.
- [RFC2014] Weinrib, A. and J. Postel, "IRTF Research Group Guidelines and Procedures", BCP 8, RFC 2014, DOI 10.17487/RFC2014, October 1996, <<https://www.rfc-editor.org/rfc/rfc2014>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/rfc/rfc5280>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/rfc/rfc5869>>.
- [RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <<https://www.rfc-editor.org/rfc/rfc5903>>.
- [RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, DOI 10.17487/RFC6031, December 2010, <<https://www.rfc-editor.org/rfc/rfc6031>>.

- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/rfc/rfc7748>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/rfc/rfc8032>>.
- [SHA-2] NIST, "Secure Hash Standard", August 2015.
- [SHA-3] Dworkin, M. J., "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", August 2015.

14. Informative References

- [draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-10, 27 July 2020, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-10>>.
- [draft-hallambaker-mesh-trust]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part X: The Trust Mesh", Work in Progress, Internet-Draft, draft-hallambaker-mesh-trust-09, 5 August 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-trust-09>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/rfc/rfc4086>>.
- [RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <<https://www.rfc-editor.org/rfc/rfc4880>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/rfc/rfc5785>>.

- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/rfc/rfc5890>>.
- [RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, DOI 10.17487/RFC6355, August 2011, <<https://www.rfc-editor.org/rfc/rfc6355>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/rfc/rfc6763>>.
- [RFC7595] Thaler, D., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/rfc/rfc7595>>.
- [Shamir79] Shamir, A., "How to share a secret.", 1979.
- [XMLSchema] Gao, S., Sperberg-McQueen, C. M., Thompson, H. S., Mendelsohn, N., Beech, D., and M. Maloney, "W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures", 5 April 2012.

LAMPS Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2019

M. Richardson
Sandelman Software Works
T. Werner
Siemens
W. Pan
Huawei Technologies
June 18, 2019

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 additionally 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.

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
2. Terminology	3
3. Requirements Language	3
4. Changes to EST endpoint processing	3
5. Clarification of ASN.1 for Certificate Attribute set.	4
6. Clarification of error messages for certificate enrollment operations	4
7. Privacy Considerations	4
8. Security Considerations	4
9. IANA Considerations	4
10. Acknowledgements	4
11. References	4
11.1. Normative References	4
11.2. Informative References	5
Authors' Addresses	5

1. Introduction

[RFC7030] defines the Enrollment over Secure Transport, or EST protocol.

This specification defines a number of HTTP end points 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, /csrattrs) specify the use of base64 encoding with a Content-Transssfer-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]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-21 (work in progress), June 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<https://www.rfc-editor.org/info/rfc7030>>.

11.2. Informative References

- [errata4384]
"EST errata 4384: ASN.1 encoding error", n.d.,
<<https://www.rfc-editor.org/errata/eid4384>>.
- [errata5107]
"EST errata 5107: use Content-Transfer-Encoding", n.d.,
<<https://www.rfc-editor.org/errata/eid5107>>.
- [errata5108]
"EST errata 5108: use of Content-Type for error message",
n.d., <<https://www.rfc-editor.org/errata/eid5108>>.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part One: Format of Internet Message
Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996,
<<https://www.rfc-editor.org/info/rfc2045>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616,
DOI 10.17487/RFC2616, June 1999,
<<https://www.rfc-editor.org/info/rfc2616>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
Protocol (HTTP/1.1): Message Syntax and Routing",
RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
DOI 10.17487/RFC7231, June 2014,
<<https://www.rfc-editor.org/info/rfc7231>>.

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Thomas Werner
Siemens

Email: thomas.werner@siemens.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

BESS Workgroup
INTERNET-DRAFT
Intended Status: Standards Track

A. Sajassi, Ed.
A. Banerjee
S. Thoria
D. Carrel
Cisco
B. Weis
Individual
J. Drake
Juniper

Expires: January 8, 2020

July 8, 2019

Secure EVPN
draft-sajassi-bess-secure-evpn-02

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/lid-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

4.2.2.2 Multiple or Single ESP SA policies	12
4.3 Initial IPsec SAs Generation	13
4.4 Re-Keying	13
4.5 IPsec Databases	13
5 Encapsulation	13
5.1 Standard ESP Encapsulation	14
5.2 ESP Encapsulation within UDP packet	15
6 BGP Encoding	16
6.1 The Base (Minimal Set) DIM Sub-TLV	16
6.2 Key Exchange Sub-TLV	17
6.3 ESP SA Proposals Sub-TLV	18
6.3.1 Transform Substructure	19
7 Applicability to other VPN types	19
8 Acknowledgements	20
9 Security Considerations	20
10 IANA Considerations	20
10 References	20
11.1 Normative References	20
11.2 Informative References	21
Authors' Addresses	22

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 signaling 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:

ID	Tunnel ID
N	Nonce

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?

"tenant"?] 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.

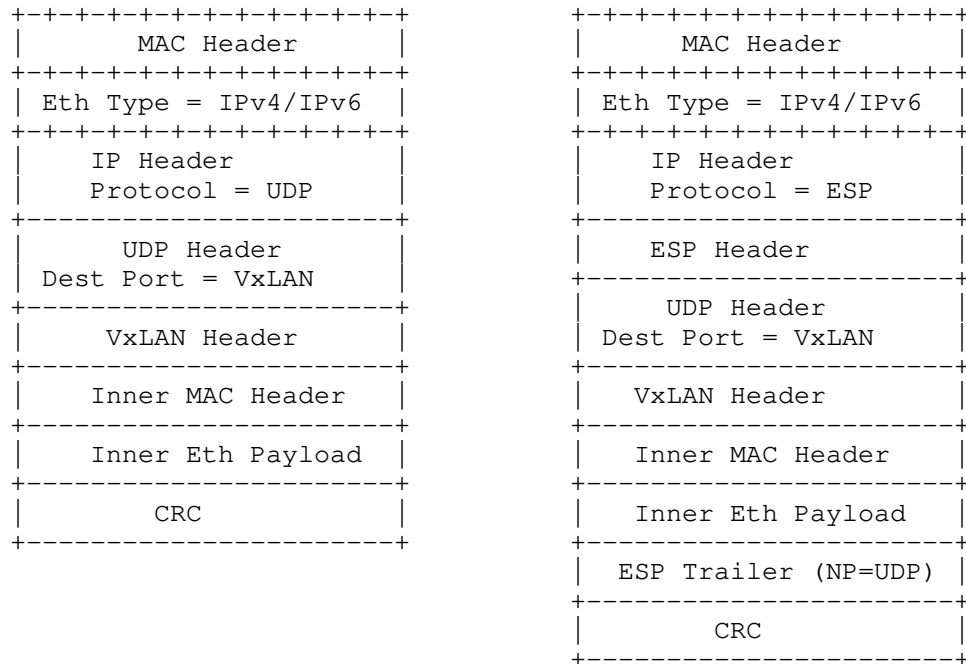


Figure 3: VxLAN Encapsulation within ESP

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]

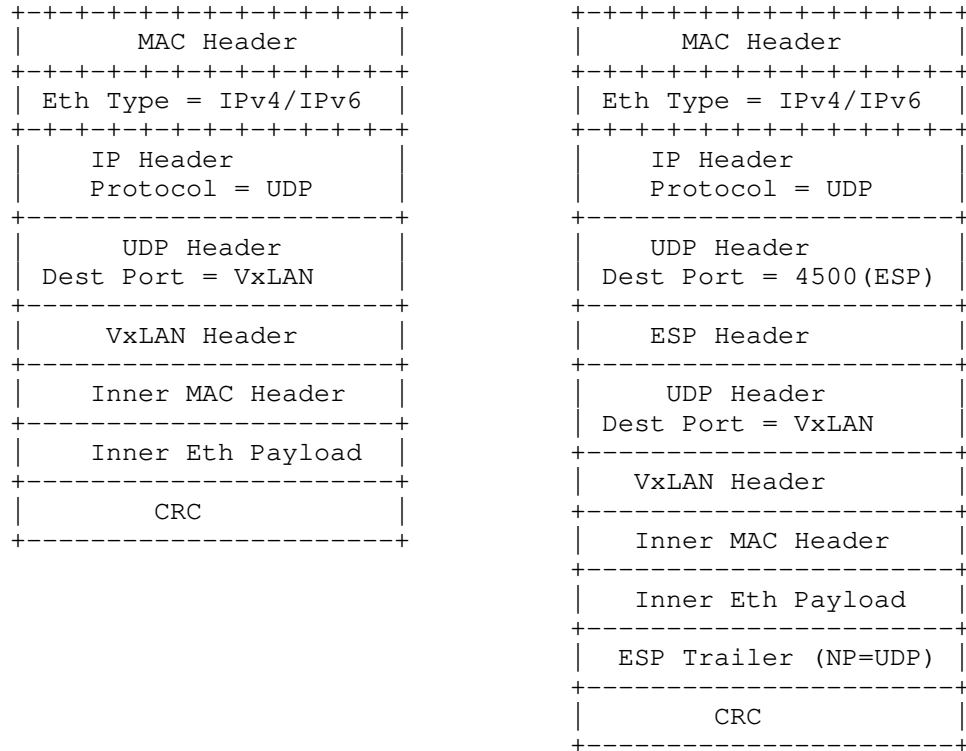


Figure 4: VxLAN Encapsulation within ESP Within UDP

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.

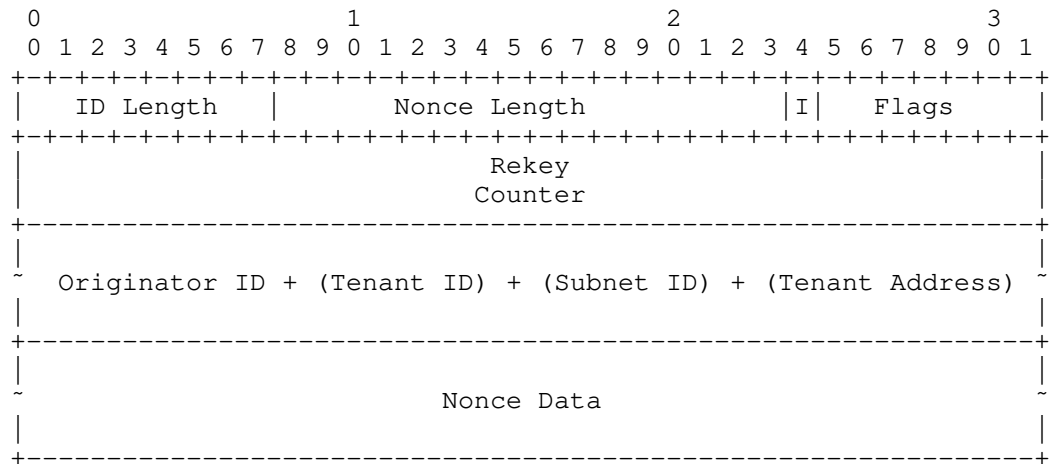


Figure 5: The Base DIM Sub-TLV

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.

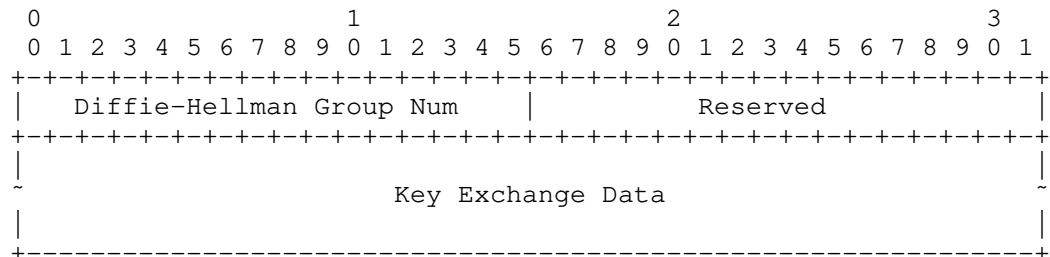


Figure 6: Key Exchange Sub-TLV

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.

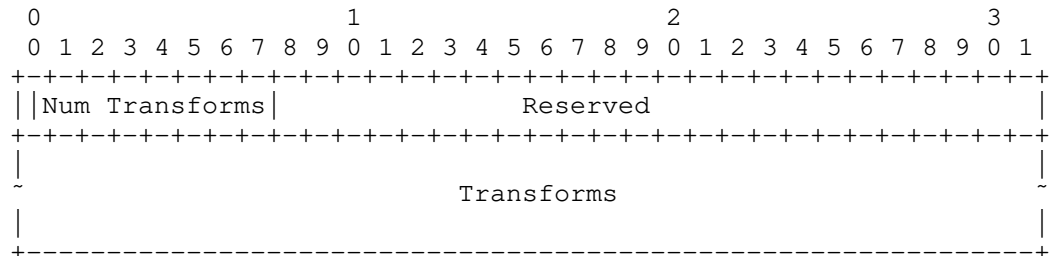


Figure 8: ESP SA Proposals Sub-TLV

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

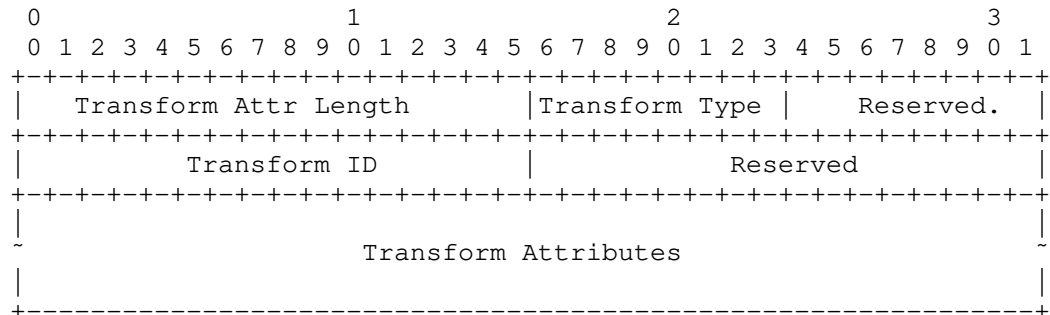


Figure 9: Transform Substructure Sub-TLV

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.

Functionality	EVPN	IP-VPN	MVPN	VPLS
per PE	IPv4/v6 route	IPv4/v6 route	IPv4/v6 rte	IPv4/v6
per tenant	IMET (or new)	lpbk (or new)	I-PMSI	N/A
per subnet	IMET	N/A	N/A	VPLS AD
per IP	EVPN RT2/RT5	VPN IP rt	*,G or S,G	N/A
per MAC	EVPN RT2	N/A	N/A	N/A

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC2119

Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

[RFC7432] Sajassi et al., "BGP MPLS Based Ethernet VPN", RFC 7432, February, 2015.

[RFC8365] Sajassi et al., "A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN)", RFC 8365, March, 2018.

[TUNNEL-ENCAP] Rosen et al., "The BGP Tunnel Encapsulation Attribute", draft-ietf-idr-tunnel-encaps-03, November 2016.

[CONTROLLER-IKE] Carrel et al., "IPsec Key Exchange using a Controller", draft-carrel-ipsecme-controller-ike-00, July, 2018.

[IKEV2IANA] IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters", <<http://www.iana.org/assignments/ikev2-parameters/>>.

[RFC3948] Huttunen et al., "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[IKEV2-IANA] IANA, "Internet Key Exchange Version 2 (IKEv2) Parameters", February 2016, www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005.

11.2 Informative References

[RFC4364] Rosen, E., et. al., "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, February 2006.

[RFC4761] Kompella, K., et. al., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, January 2007.

[RFC4762] Kompella, K., et. al., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, January 2007.

[RFC6513] Rosen, E., et. al., "Multicast in MPLS/BGP IP VPNs", RFC

6513, February 2012.

[RFC6514] Rosen, E., et. al., "BGP Encodings and Procedures for Multicast in MPLS/BGP IP VPNs", RFC 6514, February 2012.

[RFC7606] Chen, E., Scudder, J., Mohapatra, P., and K. Patel, "Revised Error Handling for BGP UPDATE Messages", RFC 7606, August 2015, <<http://www.rfc-editor.org/info/rfc7606>>.

[802.1Q] "IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks", IEEE Std 802.1Q(tm), 2014 Edition, November 2014.

[RFC7348] Mahalingam, M., et al., "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014.

[GENEVE] Gross, J., et al., "Geneve: Generic Network Virtualization Encapsulation", Work in Progress, draft-ietf-nvo3-geneve-06, March 2018.

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

Email: bew.stds@gmail.com

John Drake
Juniper
Email: jdrake@juniper.net