Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 4, 2019

P. Hallam-Baker
Comodo Group Inc.

August 31, 2018

Mathematical Mesh Part III: Advanced Cryptographic Services
draft-hallambaker-mesh-advanced-00

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 advanced encryption services supported by the Mesh.

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 4, 2019.

Table of Contents

1. Introduction

One of the core goals of the Mesh is to move the state of the art in commercial cryptography beyond that achieved in the 1990s when PKIX, S/MIME and OpenPGP were first developed. While each of these infrastructures and protocols has been subject to incremental improvement, none has seen widespread adoption of new cryptographic approaches.

This document describes the application of four technologies which have been discussed in the cryptographic literature for many decades but have not (yet) been applied to standards-based network protocols:

- Secret Splitting
- Recryption
- Key Co-Generation
- Quantum Resistant Signatures.

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. Secret Splitting

The secret sharing mechanism used is based on the method of Shamir [Shamir79] .

The mechanism described allows creation of up to 16 shares with a threshold of between 1 and 16 shares.

To share a secret of L bits with a threshold of n we first construct f(x) a polynomial of degree n in the modular field p:

f(x) = a0 + a1.x + a2.x2 + ? an.xn

where:

L Is the length of the secret in bits rounded up to the nearest multiple of 32.

n Is the threshold, the number of shares required to reconstitute the secret.

a0 Is the integer representation of the secret to be shared.

a1 ? an Are randomly chosen integers less than p

p Is the largest prime that is smaller than 2(L+1).

For L=128, p = 2^129-25.

The values of the key shares are the values f(1), f(2),? f(n).

Conversion of octet sequences to integer representation uses network byte order (i.e. big-endian). The first byte of the octet stream is the most significant 8 bits of the integer representation and the last byte is the least significant 8 bits.

Key shares are encoded as an octet sequence:

- Bits 4-7 of the first byte specify the threshold value.
- Bits 0-3 of the first byte specify the x value
- The remaining bytes specify the key share value in network byte order.

For an explanation of how to recover the master secret from the key shares, look up Lagrange basis polynomials on the Web.

3.1. Example: Securing a recovery record

Alice decides to protect her recovery record using a set of five key shares, three of which will be required to recover the key.

Alice's master secret is

{Alice.RecoveryMaster}

Figure 1

The master secret is converted to an integer applying network byte order conventions. Since the master secret is 128 bits, it is guaranteed to be smaller than the modulus. The resulting value becomes the polynomial value a0.

Since a threshold of three shares is required, we will need a third order polynomial. The co-efficients of the polynomial a1, a2, a3 are random numbers smaller than the modulus:

{Alice.RecoveryPolynomial}

Figure 2

The master secret is the value f(0). The key shares are the values f(1), f(2)...f(5):

{Alice.RecoverySharesHex}

Figure 3

The key shares in user (Base32) encoding are:

{Alice.RecoveryShare0Hex}

Figure 4

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.

We use the Lagrange polynomial basis method:

{Alice.RecoveryLagrange}

Figure 5

4. Key Co-Generation

Public Key Co-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.

Public Key Co-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.

4.1. Mechanism

The 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 ? such that there is a keypair {Z, z} such that:

Z = X ? Y and z = (x + y) mod o (where o is the order of the group)

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

(x ? E) ? (y ? E) = (z ? E) = (e ? Z).

For the Diffie Hellman system in a modular field p, o = p-1 and a ? b = a ? b = a.b

Proof,

By definition, X = ex mod p, Y = ey mod p, Z = ez mod p.

Therefore,

Z = ez mod p = ex+y mod p = (exey) mod p = ex mod p.ey mod p = X.Y

A similar proof may be constructed for the operator ?.

4.1.1. Application to Elliptic Curve systems

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

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 this reason, Ed448 and Ed25519 are the preferred algorithms for key agreement even though this is not their intended purpose.

4.2. Implementation for Ed25519 and Ed 448

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

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

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

1. Obtains the device profile of the device(s) to be provisioned to determine the type of key to perform co-generation for. Let the device {public, private} key be {D, d}.

2. Generates a private key m with the specified number of bytes (32 or 57].

3. Calculates the corresponding public key M.

4. Calculates the Application public key A = D+M where + is point addition.

5. Constructs the application device entry containing the private key value m and encrypts under the device encryption key d.

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

4.3. Example: Provisioning the Confirmation Service

For example, Alice provisions the confirmation service to her watch. The device profile of the watch specifies an Ed448 signature key:

{Alice.CogenDeviceProfile}

Figure 6

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

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

{Alice.CoGenerationPrivateSeed}

Figure 7

A key derrivation function (HKDF) is used to derrive a 256 bit key.

{Alice.CoGenerationPrivate2}

Figure 8

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

{Alice.CoGenerationPublicComp}

Figure 9

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

{Alice.CoGenerationPrivateEncrypted}

Figure 10

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

5. Recryption

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.

Traditional End-to-End Encryption is static.

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

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

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

Proxy re-encryption provides a technical capability that meets the needs of such communication patterns. Conventional symmetric key cryptography uses a single key to encrypt and decrypt data. Public key cryptography uses two keys, the key used to encrypt data is separate from the key used to decrypt. Proxy re-encryption

introduces a third key (the recryption key) that allows a party to permit an encrypted data packet to be decrypted using a different key without permitting the data to be decrypted.

The introduction of a recryption key permits end-to-end confidentiality to be preserved when a communication pattern requires that some part of the communication be supported by a service.

The introduction of a third type of key, the recryption key permits two new roles to be established, that of an administrator and recryption service. There are thus four parties:

Administrator

Holder of Decryption Key, Creator of Recryption Keys

Sender

Holder of Encryption Key

Recryption Service

Holder of Recryption keys

Receiver

Holder of personal decryption key

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

Mesh/Recrypt Parties

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

5.1. Mechanism

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

Recall that the key combination law for Diffie Hellman crypto-systems states that we can add two private keys to get a third. It follows

that we can split the private key portion of a keypair {G, g} into two parts by choosing a random number that is less than the order of the Diffie-Hellman group to be our first key x. Our second key is y = g - r mod o, where o is the order of the group.

Having generated x, y, we can use these to perform private key agreement operations on a public key E and then use the result combination law to obtain the same result that we would have obtained using g.

One means of applying this mechanism to recryption would be to generate a different random value x for each member of the group and store it at the recryption service and communicate the value y to the member via a secure channel. Applying this approach we can clearly see that the recryption service gains no information about the value of the private key since the only information it holds is a random number which could have been generated without any knowledge of the group private key.

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

5.2. Implementation

Implementation of recryption in the Mesh has four parts:

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

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

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

- Administration of the recryption group may be spread across multiple Mesh users or transferred from one user to another without requiring specification of a separate management protocol to support these operations.

- The recryption account address can be used by Mesh applications such as group messaging, conferencing, etc. as a contact address.

- The contact request service can be used to notify members that they have been granted membership in the group.

5.2.1. Group Creation

Creation of a Recryption group requires the steps of:

- Generating the recryption group key pair

- Creating the recryption group account

- Generating administrator record for each administrator.

- Publishing the administrator records to the recryption catalog.

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

5.2.2. Provisioning a Member

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

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

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

5.2.3. Message Encryption and Decryption

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

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

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

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

5.3. Example: Messaging group

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

{Alice.RecryptGroup}

Figure 11

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

```
{Alice.RecryptMessagePlaintext}
{Alice.RecryptMessageCiphertext}
```

Figure 12

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

```
{Bob.RecryptGroup}
```

Figure 13

Alice generates a recryption/decryption key entry. The recryption key is a random value smaller than the modulus:

```
{bob.RecryptRecryptionKey}
```

Figure 14

The decryption key is group private encryption key minus the recryption key (mod p):

```
{bob.RecryptDecryptionKey}
```

Figure 15

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

```
{bob.RecryptRecryptionEntry}
```

Figure 16

Note that the only information available to the server is a random number and a encrypted value only Bob can read. It is therefore impossible for compromise of the service to cause disclosure of any information encrypted under the group key unless Bob's private key is also compromised.

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

```
{bob.RecryptRecryptionCreateEntry}
```

Figure 17

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

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

{bob.RecryptRecryptionRequest}

Figure 18

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

{bob.RecryptRecryptionResponse}

Figure 19

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

{bob.RecryptDecryptionValue}

Figure 20

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

{bob.RecryptKeyAgreementValue}

Figure 21

This value allows the test message to be decrypted.

6. Quantum Resistant Signatures.

Quantum computing has made considerable advances over the past decade and the field has now reached the point where a machine weighing many tons can apply Shor's algorithm to factor numbers as large as 35 before decoherence occurs.

Should construction of a large-scale device prove practical, it will in principle be possible to break all of the public key cryptosystems currently in use. While public key cryptosystems that resist quantum cryptanalysis are currently in development, none has yet reached a

sufficient state of maturity for the field to reach consensus that they are resistant to ordinary cryptanalysis, let alone offer a replacement.

The consequence of successful quantum cryptanalysis for encryption systems is that all material encrypted under existing public key systems could be decrypted by a quantum capable attacker. Nor is mitigation of this consequence practical since it is not the adoption of new cryptographic algorithms that make a system more secure, it is the elimination of weak options that provides improvement.

The Mesh does not currently provide an infrastructure that is Quantum Resistant but could in principle be used as the basis for deploying a Needham-Schroeder style symmetric key infrastructure or a future PKI based on an as yet undecided quantum cryptanalysis resistant public key algorithm.

Mesh profiles MAY include a Quantum Resistant Signature Fingerprint (QRSF). This contains the UDF fingerprint of an XMSS signature public key [RFC8391] together with the parameters used to derive the private key set for the public key from a 256 bit master secret.

Should it ever become necessary to make use of the QRSF, the user first recovers the master secret from whatever archival mechanism was used to protect it. The use of secret sharing to protect the secret is RECOMMENDED. The master secret is then used to reconstruct the set of private keys from which the public key set is reconstructed. The profile owner can now authenticate themselves by means of their XMSS public key.

6.1. Example: Creating a Quantum Resistant Signature Fingerprint

Alice decides to add a QRSF to her Mesh Profile. She creates a 256 bit master secret.

{Alice.QuantumMaster}

Figure 22

To enable recovery of the master key, Alice creates five keyshares with a quorum of three:

{Alice.QuantumShares}

Figure 23

Alice uses the master secret to derrive her private key values:

{Alice.QuantumXMSSPrivate}

Figure 24

These values are used to generate the public key value:

{Alice.QuantumXMSSPublic}

Figure 25

The QRSF contains the UDF fingerprint of the public key value plus the XMSS parameters:

{Alice.QuantumXMSSUDF}

Figure 26

Alice adds the QRSF to her profile and publishes it to a Mesh Service that is enrolled in at least one multi-party notary scheme.

7. Security Considerations

TBS

8. IANA Considerations

All the IANA considerations for the Mesh documents are specified in this document

9. Acknowledgements

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.

10. 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 applying the formula:

2^(n+1) - (1 SHL (n+1)) - B(1 SHL (n+1))

Number of bits	Prime
32	2^33 - 9
64	2^65 - 49
96	2^97 - 141
128	2^129 - 25
160	2^161 - 159
192	2^193 - 31
224	2^225 - 49
256	2^257 - 93
288	2^289 - 493
320	2^321 - 9
352	2^353 - 139
384	2^385 - 265
416	2^417 - 1029
448	2^449 - 241
480	2^481 - 273
512	2^513 - 445

Table 1

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017.

11.2. Informative References

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

[draft-hallambaker-mesh-architecture] Hallam-Baker, P., "Mathematical Mesh: Architecture", draft-hallambaker-mesh-architecture-05 (work in progress), August 2018.

[draft-hallambaker-mesh-developer] Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", draft-hallambaker-mesh-developer-07 (work in progress), April 2018.

[RFC8391] Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, DOI 10.17487/RFC8391, May 2018.

[Shamir79] "[Reference Not Found!]".

11.3. URIs

[1] http://mathmesh.com/Documents/draft-hallambaker-mesh-advanced.html

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com