Workgroup: Network Working Group
Internet-Draft:
draft-hallambaker-threshold-sigs
Published: 5 January 2020
Intended Status: Informational
Expires: 8 July 2020
Authors: P. M. Hallam-Baker
Venture Cryptography.

Threshold Signatures Using Ed25519 and Ed448

Abstract

A Threshold signature scheme is described. The signatures created are computationally indistinguishable from those produced using the Ed25519 and Ed448 curves as specified in RFC8032 except in that they are non-deterministic. Threshold signatures are a form of digital signature whose creation requires two or more parties to interact but does not disclose the number or identities of the parties involved.

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 8 July 2020.

Table of Contents

1. Introduction

2. Definitions

2.1. Requirements Language

2.2. Defined Terms

2.3. Related Specifications

2.4. Implementation Status

3. Threshold Signature Construction

3.1. Threshold signature

3.2. Ed2519 Signature

3.3. Ed448 Signature

3.4. Security Analysis

3.4.1. Replay Attack

3.4.2. Malicious Contribution Attack

3.4.3. Rogue Key Attack

4. Unanimous Signature

4.1. Using threshold key generation

4.2. Using key splitting

5. Quorate Signature

5.1. Key Generation

5.2. Calculating the secret scalar value

6. Test Vectors

6.1. Unanimous Threshold Signature Ed25519

6.2. Unanimous Threshold Signature Ed448

6.3. Quorate Threshold Signature Ed25519

6.4. Quorate Threshold Signature Ed448

7. Security Considerations

8. IANA Considerations

9. Acknowledgements

10. Normative References

11. Informative References

1. Introduction

Threshold encryption and key generation provide compelling advantages over single private key approaches because splitting the private key permits the use of that key to be divided between two or more roles.

All existing digital signatures allow the signer role to be divided between multiple parties by attaching multiple signatures to the signed document. This approach, known as multi-signatures is distinguished from a threshold signature scheme in that the identity and roles of the individual signers is exposed. In a threshold signature scheme, the creation of a single signature requires the participation of multiple signers and the signature itself does not reveal the means by which it was constructed.

Rather than considering multi-signatures or threshold signatures to be inherently superior, it is more useful to regard both as two points on a continuum of choices:

Multi-signatures Multiple digital signatures on the same document. Multi-signatures are simple to create and provide the verifier with more information but require the acceptance criteria to be specified independently of the signature itself. This requires that the application logic or PKI provide some means of describing the criteria to be applied.

Multi-party key release A single signature created using a single private key stored in an encrypted form whose use requires participation of multiple key decryption shares.

Threshold signatures A single signature created using multiple signature key shares. Signature creation may be subject to complex criteria such as requiring an (n,t) quorum of signers but these criteria are fixed at the time the signature is created

Aggregate Signatures A single signature created using multiple signature key shares such that validation of the aggregate signature serves to validate the participation of each of the individual signers.

This document describes a scheme that creates threshold signatures that are computationally indistinguishable from those produced according to the algorithm specified in [RFC8032]. The scheme does not support the creation of aggregate signatures.

Two versions of the algorithm are presented. Both versions allow a creation of a signature to be divided between a set of n signers such that a minimum of t signers are required to create a signature. The first version of the algorithm (unanimous signature) requires that every member of the set of signers participate in the signing process (i.e. n=t). The second version (quorate signature) allows a signature to be created by a subset of the authorized signers.

The unanimous signature scheme allows key shares to be generated by either dividing a master key or using threshold key generation to construct the master key from two or more key contributions as described in [draft-hallambaker-threshold].

The quorate signature scheme requires that the signature key shares be derived from a master signature key using Shamir secret sharing. The process of key share generation is thus fundamentally one of dividing a master key.

If required, division of the signing key administration roles may be achieved by using threshold key generation to construct the master key and performing the key splitting process separately for each contribution.

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

See [draft-hallambaker-threshold].

2.3. Related Specifications

This document extends the approach described in [draft-hallambaker-threshold] to support threshold signatures. The use of Lagrange polynomials to resolve Shamir Secret Shares is currently described in the Uniform Data Fingerprint specification [draft-hallambaker-

mesh-udf]. It is expected that these three documents will eventually form the basis for two CFRG proposals.

2.4. Implementation Status

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

3. Threshold Signature Construction

The threshold signatures created according to the algorithms described in this document are compatible with but not identical to the signatures created according to the scheme described in [RFC8032]. In particular:

*The signature verification algorithm is unchanged.

*The unanimous threshold scheme produces values of R and S that are deterministic but different from the values that would be obtained by using the aggregate private key to sign the same document.

*The deterministic quorate threshold scheme produces values of R and S that are deterministic for a given set of signers but will change for a different set of signers or if the aggregate private key was used to sign the same document.

*?The non-deterministic quorate threshold scheme produces values of R and S that will be different each time the document is signed.

Recall that a digital signature as specified by [RFC8032] consists of a pair of values S, R calculated as follows:

R = r.B

S = r + k.s mod L

Where B is the base point of the elliptic curve.

r is an unique, unpredictable integer value such that 0 r L

k is the result of applying a message digest function determined by the curve (Ed25519, Ed448) to a set of parameters known to the verifier which include the values R, A and PH(M).

A is the public key of the signer, A = s.B

PH(M) is the prehash function of the message value.

s is the secret scalar value

L is the order of the elliptic curve group.

To verify the signature, the verifier calculates:

$S.B = R + k.A$

Therefore:

$S.B = (r + k.s).B = r.B + k.(s.B) = R + k.A$

The value r plays a critical role in the signature scheme as it serves to prevent disclosure of the secret scalar. If the value r is known, s can be calculated as $s = (S-r).k^{-1} \bmod L$. It is therefore essential that the value r be unguessable.

Furthermore, if the same value of r is used to sign two different documents, this results two signatures with the same value R and different values of k and S. Thus

$S_1 = r + k_1.s \bmod L$

$S_2 = r + k2.s \bmod L$

$s = (S_1 - S_2)(k_1 - k_2)^{-1} \bmod L$

The method of constructing r specified in [RFC8032] ensures that the value r is unique and unguessable provided that the private key from which the secret scalar value is derived is kept secret.

The use of a deterministic means of generating the value r permits auditing of hardware and software implementations to determine if the correct means of constructing r is used by importing a known private key and verifying that the expected values of R are produced when a set of documents is signed.

The threshold schemes described in this document have been designed to preserve the ability to audit the means of constructing the values of r for each individual key share even though the sum of these values and hence the value R will vary according to the number of signature shares used.

3.1. Threshold signature

A threshold signature R, S is constructed by summing a set of signature contributions from two or more signers. Each signer *i* provides a contribution as follows:

$A_i = s_i.B$

$R_i = r_i.B$

$S_i = r_i + k.s_i \bmod L$

Where s_i and r_i are the secret scalar and unguessable value for the individual signer.

The contributions of signers {1, 2, ... n} are then combined as follows:

$R = R_1 + R_2 + ... + R_n$

$S = S_1 + S_2 + ... + S_n$

$A = s.B$

Where $s = (s_1 + s_2 + ... + s_n) \bmod L$

The threshold signature is verified in the same manner as before:

$S.B = R + k.A$

Substituting for S.B we get:

$= (S_1 + S_2 + ... + S_n).B$

$= S_1.B + S_2.B + ... + S_n.B$

$= (r_1 + k.s_1).B + (r_2 + k.s_2).B + ... + (r_n + k.s_n).B$

$= (r_1.B + k.s_1.B) + (r_2.B + k.s_2.B) + ... + (r_n.B + k.s_n.B)$

$= (R1 + k.A1) + (R1 + k.A1) + ... + (Rn + k.An)$

Substituting for R + k.A we get:

$= R_1 + R_2 + ... + R_n + k.(A_1 + A_2 + ... + A_n)$

$= R_1 + R_2 + ... + R_n + k.A_1 + k.A_2 + ... + k.A_n$

$= (R_1 + k.A_1) + (R_1 + k.A_1) + ... + (R_n + k.A_n)$

As expected, the operation of threshold signature makes use of the same approach as threshold key generation and threshold decryption as described in [draft-hallambaker-threshold]. As with threshold decryption it is not necessary for each key share holder to have a public key corresponding to their key share. All that is required is that the sum of the secret scalar values used in calculation of the signature modulo the group order be the value of the aggregate secret scalar corresponding to the aggregate secret key.

While verification of [RFC8032] signatures is unchanged, the use of threshold signatures requires a different approach to signing. In particular, the fact that the value k is bound to the value R means that the participants in the threshold signature scheme must agree on the value R before the value k can be calculated. Since k is required to calculate the signature contributions S_i can be calculated, it is thus necessary to calculate the values R_i and S_i in separate phases. The process of using a threshold signature to sign a document thus has the following stages orchestrated by a coordinator as follows:

0. The coordinator determines the values F, C and PH(M) as specified in [RFC8032] and transmits them to the signers {1, 2, ... n}.

1. Each signer generates a random value r_i such that 1 r_i L, calculates the value $R_i = r_i.B$ and returns R to the coordinator.

2. The coordinator calculates the value $R = R_1 + R_2 + ... + R_n$ and transmits R and A to the signers {1, 2, ... n}.

3. Each signer uses the suppled data to determine the value k and hence $S_i = r_i + k.s_i$ mod L and transmits it to the coordinator.

4. The coordinator calculates the value $S = S_1 + S_2 + ... + S_n$ and verifies that the resulting signature R, S verifies according to the mechanism specified in [RFC8032]. If the signature is correct, the coordinator publishes it. Otherwise, the coordinator **MAY** identify the signer(s) that provided incorrect contributions by verifying the values R_i and S_i for each.

For clarity, the coordinator role is presented here as being implemented by a single party.

3.2. Ed2519 Signature

The process for creating an Ed25519 signature contribution is as follows

0. Determine the value of the secret scalar value s_i according to the means used to construct the secret shares.

1. Generate a random integer r_i such that 1 r_i L

2. Compute the point $R_i = r_iB$. For efficiency, do this by first reducing r_i modulo L, the group order of B. Let the string R_i be the encoding of this point.

3. Transmit the value R_i to the coordinator

4. Note that the construction of prefix is chosen so as to guarantee that the use of a different message value M or secret scalar value s_i will result in a different value of prefix and thus ensure a different choice of the value r_i.

5. At some later point, the coordinator **MAY** complete the signature by returning the values F, C, A and R as specified in [RFC8032]. The signer **MAY** then complete the signature contribution as follows:

6. Compute SHA512(dom2(F, C) || R || A || PH(M)), and interpret the 64-octet digest as a little-endian integer k.

7. Compute $S_i = (r_i + k * s_i) \bmod L$. For efficiency, again reduce k modulo L first.

8. Return the values R_i, S_i to the coordinator.

9. The coordinator assembles the signature values $R = S = S_1 + S_2 + \ldots + S_n$

10. The coordinator verifies that the signature R, S is valid.

3.3. Ed448 Signature

The process for creating an Ed448 signature contribution is as follows

0. Determine the value of the secret scalar value s_i according to the means used to construct the secret shares.

1. Generate a random integer r_i such that 1 r_i L

2. Compute the point $R_i = r_iB$. For efficiency, do this by first reducing r_i modulo L, the group order of B. Let the string R_i be the encoding of this point.

Transmit the value R_i to the coordinator

At some later point, the coordinator **MAY** complete the signature by returning the values F, C, A and R as specified in [RFC8032]. The signer **MAY** then complete the signature contribution as follows:

0. Compute SHAKE256(dom4(F, C) || R || A || PH(M), 114), and interpret the 114-octet digest as a little-endian integer k.

1. Compute $S_i = (r_i + k * s_i) \bmod L$. For efficiency, again reduce k modulo L first.

2. Return the values R_i, S_i to the coordinator.

3. The coordinator assembles the signature values $R = S = S_1 + S_2 + ... + S_n$

4. The coordinator verifies that the signature R, S is valid.

Note that the process of generating the signature contribution is deterministic for a given secret scalar value but the signature computed **MAY** be deterministic or non-deterministic according to the desired signature properties.

3.4. Security Analysis

We consider a successful breach of the threshold signature scheme to be any attack that allows the attacker to create a valid signature for any message without the participation of the required threshold of signers.

Potential breaches include:

*Disclosure of the signature key or signature key share.

*Modification of signature data relating to message M to allow creation of a signature for message M'.

*Ability of one of the signers to choose the value of the aggregate public key.

*Access control attacks inducing a signer to create a signature contribution that was not properly authenticated or authorized.

We regard attacks on the access control channel to be out of scope for the threshold signature algorithm, though they are certainly a concern for any system in which a threshold signature algorithm is employed.

We do not consider the ability of a signer to cause creation of an invalid signature to represent a breach.

3.4.1. Replay Attack

The most serious concern in the implementation of any Schnorr type signature scheme is the need to ensure that the value r_i is never revealed to any other party and is never used to create signatures for two different values of $k.s_i$.

Ensuring this does not occur imposes significant design constraints as creating a correct signature contribution requires that the signer use the same value of r_i to construct its value or R_i and S_i.

For example, a HSM device may be required to perform multiple signature operations simultaneously. Since the storage capabilities of an HSM device are typically constrained, it is tempting to attempt to avoid the need to track the value of r_i within the device itself using an appropriately authenticated and encrypted opaque state token. Such mechanisms provide the HSM with the value of r_i but do not and cannot provide protection against a replay attack in which the same state token is presented with a request to sign different values of k.

3.4.2. Malicious Contribution Attack

In a malicious contribution attack, one or more parties present a signature contribution that does not meet the criteria $R_i = r_i.B$ and $S_i = r_i + ks_i$.

Such an attack is not considered to be a breach as it merely causes the signature process to fail.

3.4.3. Rogue Key Attack

A threshold signature scheme that allows the participants to 'bring their own key' may be vulnerable to a rogue key attack in which a signer is able to select the value of the aggregate public signature key by selecting a malicious public signature key value.

The scheme described in this document is a threshold signature scheme and does not support this feature. Consequently, this attack is not relevant. It is described here for illustrative purposes only.

This particular attack only applies when the individual signers create their own signature shares. It is not a concern when the signature shares are created by splitting a master signature private key.

Consider the case where the aggregate public key signature is calculated from the sum of public signature key share values presented by the signers:

$$A = A_1 + A_2 + ... + A_n$$

If the public key values are presented in turn, the last signer presenting their key share can force the selection of any value of A that they choose by selecting $A_n = A_m - (A_1 + A_2 + ... + A_{n-1})$

The attacker can thus gain control of the aggregate signature key by choosing $A_m = s_m.B$ where s_m is a secret scalar known only to the attacker. But does so at the cost of not knowing the value s_n and so the signer cannot participate in the signature protocol.

This attack allows the attacker and the attacker alone to create signatures which are validated under the aggregate signature key.

The attack is a consequence of the mistaken assumption that a signature created under the signature key $A_1 + A_2 + ... + A_n$ provides evidence of the individual participation of the corresponding key holders without separate validation of the aggregate key.

Enabling the use of threshold signature techniques by ad-hoc groups of signers using their existing signature keys as signature key shares presents serious technical challenges that are outside the scope of this specification.

4. Unanimous Signature

A unanimous threshold signature is a signature created by a set of two or more signers that requires the participation of every member of the set. The key shares used by the signers **MAY** be generated by either:

* Each key share holder generating their own unique key share and combining them.

* Generating a master signature key and dividing it.

Both approaches are described in [draft-hallambaker-threshold].

4.1. Using threshold key generation

To create a threshold signature key share using threshold key generation, each signer generates a pair $\{A_i, s_i\}$ and transmits the public component to the coordinator.

The coordinator calculates the aggregate public key $A = A_1 + A_2 + ... + A_n$

$= s_1.B + s_2.B + ... + s_n.B$

$= ((s_1 + s_2 + ... + s_n) \bmod L).B$

$= s.B$

Where $s = s_1 + s_2 + ... + s_n) \bmod L$

The coordinator **MUST** require and verify proof of possession of the private key component s_i to defeat rogue key attacks of the type described in Section 3.4.3.

For example, the coordinator **MAY** require each signer to create a test signature for their individual signature key share or to

participate in creating a test signature under the aggregate signature key. Such a test signature **MAY** be a Certificate Signing Request as specified in [RFC2314].

Relying parties **MUST NOT** accept signatures purporting to be aggregate signatures under ad-hoc collections of public keys as proof of the involvement of the signers.

4.2. Using key splitting

To divide a keypair {A, p} into n parts, we begin by generating n-1 key pairs according to the method described in [RFC8032].

To calculate the final secret scalar share s_n we begin by calculating the secret scalar value s corresponding to the master private key p and the key shares s_1, s_2, ..., s_{n-1}. The final secret scalar is given by:

$s_n = s - s_1, s_2, ..., s_{n-1}$

Since the final secret scalar is not derived from a private key by means of a digest, it is not possible to use [RFC8032] format to store keys. The format described in [draft-hallambaker-threshold] is used instead.

5. Quorate Signature

Constructing the set of secret scalar key shares using the Shamir secret sharing scheme allows a signature scheme in which the threshold of signers t required to create a signature is smaller than the number of key shares n.

5.1. Key Generation

The key generation process begins with the generation of a master signature key {A, p}. The private key p is used to calculate the secret scalar value s according to the procedure specified in Section 4.1 above.

The secret scalar is then divided as follows.

First, we construct a polynomial of degree t in the modular field L, where L is the order of the curve sub-group:

$f(x) = a_0 + a_1.x + a_2.x^2 + ... a_t.x^t \bmod L$

where $a_0 = s$

$a_1 ... a_t$ are randomly chosen integers in the range 1 a_i L

The values of the key shares are the values $f(x_1)$, $f(x_2)$, ... , $f(x_n)$. That is

$p_i = f(x_i)$

where p_1 ... p_n are the private key shares

x_1 ... x_n are distinct integers in the range 1 x_i L

The means of constructing distinct values x_1 ... x_n is left to the implementation. It is not necessary for these values to be secret.

As with the final secret scalar value above, it is not possible to use [RFC8032] format to store keys. The format described in [draft-hallambaker-threshold] is used instead.

5.2. Calculating the secret scalar value

The value of the secret scalar value used to create a signature depends on the specific choice of signature shares used to construct it. Since the numbering of the key shares is arbitrary, we choose the set of signers {1, 2, ... , t}.

Lagrange polynomials are used to compute a set of coeficients l_1, l_2, ... l_t such that

$s = a_0 = l_1p_1 + l_2p_2 + ... + l_tp_t$

The value of the secret scalars $s_1 + s_2 + ... + s_n$ are thus given by

$s_i = l_ip_i$

The coordinator identifies the signers to participate in the signature event and uses the set of x coordinates corresponding to their key shares to calculate the corresponding Lagrange coefficient li for each as described in [draft-hallambaker-mesh-udf].

[Note: The discussion of Shamir Secret Sharing should probably be moved from its current location in [draft-hallambaker-mesh-udf] to either this document or the base document [draft-hallambaker-threshold]. Which document is better suited will depend on whether it is decided to implement (n,k) threshold signatures and/or key escrow of elliptic curve scalar secrets.]

6. Test Vectors

6.1. Unanimous Threshold Signature Ed25519

The signers are Alice and Bob. Each creates a key pair:

```
ED25519Alice's Key (ED25519)
    UDF:        ZAAA-GTSI-GXED-255X-XALI-CEXS-XKEY
    Scalar:     562712440811861309806365450179451565805161018943524924595949676142233994288880
    Encoded Private
  33 40 0E 22  D8 67 17 F4  8A 9F 6A 46  61 B4 0E AD
  8C D0 DD C3  79 CD 85 BD  95 5C 90 B9  6C CB 8C 23
    X: 1111679367297042716179026446928029450718904472814054795407102279764541240424063693449326556336646305602422134314091398669402847020026484693657564926479700
    Y: 6165540417161139657302180811910866474957423512534368020645428562991413866150465483230874093885486502722244870898950799705260143544115364878870129761259200
    Encoded Public
  E2 AB 8F 37  62 C8 7B F9  E9 BC 59 0C  2E 99 A5 58
  0C C3 19 D5  CD DA 53 DF  3E C1 F0 C0  FE D3 55 5E
ED25519Bob's Key (ED25519)
    UDF:        ZAAA-GTSI-G2ED-255X-XBOB-XSXK-EY
    Scalar:     549407726701534591461529255641981052629714857308898189867276085732299790201680
    Encoded Private
  68 9A 68 92  8A 06 17 84  35 3C B7 08  F8 56 00 3F
  BA 31 8C 42  B0 42 FE 2D  18 F2 7F AB  CD 10 49 F1
    X: 1427149506934983821637954019626314096403239351290384220616818255188508270988762898008687355222329085197942511309071258786756343411484065706313568410880015
    Y: 2809432894800411242818946622375744088638868429125460535585992362409682297067958252824195942194420746470938513025474524704359438513477629978601366725015573
    Encoded Public
  32 E5 8D 5E  66 B2 F9 E9  14 79 08 71  96 3B 9A 75
  A2 31 59 4B  8E ED 18 EF  BD FF 11 D4  47 2A 8C F4
```

The Aggregate Signature Key $A = A_a + A_b$

```
Aggregate Key = Alice + Bob ()
    UDF:        TBS
    Scalar:     26569330913556569171916721364983482306308422345436973
        5629331211317138468421 3
    Encoded Private
  B5 CE 0E B3  9C CF 18 99  CF 8D 4C BB  AE 81 79 1F
  CE 13 AA 3E  63 59 5B AC  8D 2C EB A4  55 C5 DF 05
    X: 67872685043898469012456949773240814121645904736114813455820339
        8688906486811443744733724675994181258029547346985079901494367
        752381127781166234556148580090
    Y: 364817400583696454844201809760049320620853759415223440529075 94
        011855279215855119710748489220456229080281065525351030244845 5
        412854899211810141579790925095 4
    Encoded Public
  29 65 63 86  4F FB 10 8D  BA 7A 0A 68  04 6D 00 DA
  9B 1D C3 A4  AF BA 95 B4  5D 27 B4 35  00 2F DF 32
```

To sign the text "This is a test", Alice and Bob first generate their values r which they multiply by the base point to obtain the value R_i:

```
Alice:
r_a =  63604859197442869293148706887422539627692219221403898700132668
    2121229168067
R_a =  System.Byte[]

Bob:
r_a =  90588289213224300609691011367428583037027596393781713986313426
    4309321564224
R_a =  System.Byte[]
```

The aggregate value $R = R_a + R_b$

```
R =
  FB 4A B1 B4  53 C4 90 54  C4 F3 FD 00  87 AE EF 3D
  33 CE 2B 5B  F2 88 DB 13  FD F7 28 67  26 EE 07 1F
```

The value k is calculated

```
k =  40335796207451342355816419009827689385454638927672676076 07223155591
    591374371726
```

Alice and Bob both calculate their signature scalar contribution:

Alice:
S_a = 665554071446397329885082547053639648297876977567546310949144578662199168944

Bob:
S_b = 147554812970368889293253219411887656127817230118539305710045112504633964811 4

The coordinator calculates the aggregate value S = S_a + S_b

S = 214110220115008622281761474117251620957604927875293936804959570370853881705 8

The coordinator checks to see that the signature verifies:

S.B = R + kA =
X: 20168784599297644264752251999842912626695351875278949808545814238064903547119
Y: 17798208503807446200181400098695224792791795831502230059016274558554804974562

6.2. Unanimous Threshold Signature Ed448

The signers are Alice and Bob. Each creates a key pair:

```
ED448Alice's Key (ED448)
    UDF:        ZAAA-ITSI-GXED-44XA-LICE-XSXK-EY
    Scalar:     6349580358365881768811044631478607697634723636135403 5
        5597788771064742993095132758589292255654895141583596922516472
        738879360490167934280
    Encoded Private
  A0 53 4C 93  3C 34 00 76  AE 5D B5 4A  C2 71 5F 43
  E1 D6 63 2C  5C 56 53 C8  98 A0 8F 03  FF F5 22 96
  91 45 8C 2B  CF E3 FD 7E  2A 9E 0B D6  F4 CC 66 61
  43 62 72 7B  34 B4 79 92
    X: 247431975092678332621114495565272851208681677122099195708384263
        466168536901525943558973091346360088759980994772668117646359
        614426660579
    Y: 213428991205767705376644620496852583908537297883034283490511308
        752175233505795318243164692156369495328007220135137156078814
        081547431302
    Encoded Public
  0A 3B F3 27  E7 E1 67 63  2C 59 E2 1C  D1 84 C7 83
  E8 1E D1 68  9F 32 A1 16  99 00 5C DA  29 B9 6C 08
  E4 15 57 7E  E5 63 C2 32  08 23 41 68  5F 49 1F FF
  BC 4D CD 3A  4E A6 85 49  00
ED448Bob's Key (ED448)
    UDF:        ZAAA-ITSI-G2ED-44XB-OBXS-XKEY
    Scalar:     7264980377319975156499854389189890483971840931291078 0
        026204194116098964372733198765813218218197005424558732207053 5
        846720571414845714224
    Encoded Private
  BC 53 B4 74  3E A7 A7 FA  9F 05 9A BC  8C 22 26 15
  A1 4E BB 10  0E B5 59 6B  DE 9C 1B E9  F2 3C 65 42
  E7 B4 47 18  60 AC 18 A6  D2 78 B8 BC  CE F5 F4 28
  B2 3A FF 08  61 EF 6B 7C
    X: 582358519348086406219208168729590591726924111876409504322030 9
        811674899775013446023140669809131700806303040879853663428420 7
        667468558264
    Y: 343907676979092838924957612591865386321204224589392131201372282
        605645565659182621638118563408068571815485272672562417899582 7
        091591132128
    Encoded Public
  93 63 5A 45  2D 4C 94 32  45 23 CD E2  A8 46 E4 78
  A0 80 59 DA  36 CB 6B 0C  06 64 6F BE  51 AB C0 BF
  1E DB A8 3F  2B 3B 80 0F  BF 00 E6 78  DD E0 83 E9
  AC 20 02 55  87 07 39 38  00
```

The Aggregate Signature Key $A = A_a + A_b$

```
Aggregate Key = Alice + Bob ()
    UDF:        TBS
    Scalar:     894883060512736340697732382628418830417840755398415503672228636597106090916876462340541507950185640860121886233669494966515613996100051
    Encoded Private
  D3 29 DD AB  F6 0D 99 8B  75 65 B8 06  36 C9 3A 2C
  D4 08 C3 9B  7C F9 77 8C  68 29 0E 3D  5D C7 3E 00
  92 8B DC AE  26 FB 16 39  CD 25 1B 23  4A 5A 05 61
  1D 5C C4 70  0A C9 84 1F
    X: 17985659098670117617173315763082238685735647626871251468000984208031711109169618360730761417172696057630877497574224926053219916057099
    Y: 31506323224859159594386181995639405170623657273945727288760063162440669468261733472504018128790535106676341465854382862384150916197586
    Encoded Public
  9B 3E DF 49  55 40 9F 7B  EA 0B AA 40  B7 3D 15 82
  60 9F 7C 40  CF 67 DE 56  56 0D 03 87  63 3B 15 F2
  45 33 FE 48  BD 2D A0 A2  8B CC 74 DA  94 0F 39 00
  AC 39 CB 0A  9F A4 EB B0  00
```

To sign the text "This is a test", Alice and Bob first generate their values r which they multiply by the base point to obtain the value R_i:

```
Alice:
r_a =  737594182900547948613735728585662887816193581293424017507745078622804885800058848191945477492881706276894168576141800759328053029999804
R_a =  System.Byte[]

Bob:
r_a =  462379632785132153223961480500348655869468897041220423485368326258117837235219419683096432484508107476011679279942494583193881896018677
R_a =  System.Byte[]
```

The aggregate value $R = R_a + R_b$

```
R =
  A0 FF BC 4A  92 40 39 FE  A3 E2 98 19  8A CB 60 54
  65 E1 EF C5  3E E2 AA 54  8D 27 78 CB  F8 34 DC E2
  9C DF A0 D7  F1 D2 7F EA  33 A2 A7 07  76 29 A1 7C
  EB D4 97 F1  A3 CE 0B 61  80
```

The value k is calculated

```
k =  42533023164616794810824342821933400854697147280826319967495091713652886334116113018933782740782184044709302956924210923803512329390659
```

Alice and Bob both calculate their signature scalar contribution:

```
Alice:
S_a =  98531765382438321188310203252700171214078910507928709174117939037199188119568765810761809446720891557832102898183598559283112963570131

Bob:
S_b =  10826185029558642533243398893637670455024435365016467182849367928536029945920208532683123154260212263130421123048975504051476260323231952
```

The coordinator calculates the aggregate value $S = S_a + S_b$

```
S =  25083934604123023883413240217075742175912923986263865932239068527413483617231265421397285697630638225826020419581691295024119707152304
```

The coordinator checks to see that the signature verifies:

```
S.B = R + kA =
    X: 872293175429970138182526349977779650987637127472354447683753752526319556986 3
    Y: 484267195493689880400472071857746758710895444480348020402439841774187792882 37
```

6.3. Quorate Threshold Signature Ed25519

The administrator creates the aggregate key pair

ED25519Aggregate Key (ED25519)
    UDF: ZAAA-GTSI-GQED-255X-XAGG-REGA-TEXK-EY
    Scalar: 39348647608109113656999806950437958090469802387424444
        5893750660798610752238 16
    Encoded Private
  37 39 5E 7A 8B A5 A0 19 46 4B 58 22 EA 24 A5 71
  45 2C 2A AC 7A 3E FB CA CE 3F D4 12 9A BA EB 70
    X: 14198837758377867455716504277518729070915183249890461230792115
        99049697167784279959512347660021645117385875752575303887583 74
        78249060472500577218550685239 70
    Y: 20211025649802071998810413948266748565975140520947927724517956
        20676255050777515980186295517468245337267098109901934556623 85
        61527361163034410318513054580 40
    Encoded Public
  6E 13 79 B4 39 DA 97 9C 5A 34 CE 79 CD 1B 50 DF
  A0 76 AD 49 81 6D 52 59 A4 2C DB CE 44 FF 3E F5

3

   Three key shares are required for Alice, Bob and Carol with a
   threshold of two. The parameters of the Shamir Secret Sharing
   polynomial are:

a0 = 39348647608109113656999806950437958090469802387424444589375066 07
    9861075223816
a1 = 64782350749366692329225467090628535268007477232844358935603799 98
    498854036401

4

   The key share values for the participants are

xa = 1
ya = 24048492190522096060832342812428461721278519544294348469237404 48
    647203754283

xb = 2
yb = 16460787166566166250325944272627054580714833183339631344821695 08
    860603539695

xc = 3
yc = 88730821426102364398195457328256474401511468223849142204059856 90
    74003325107

5

   Alice and Carol are selected to sign the message "This is another
   test"

   The Lagrange coefficients are:

```
la = 36185027886661311069865932815214971204285581796899538030009754690
    142727125496
lc = 36185027886661311069865932815214971204285581796899538030009754690
    142727125494
```

Alice and Carol select their values ra, rc

```
ra = 54679902911852315297226422922829106771809281866730908982867253510
    703148584114
Ra = System.Byte[]

rc = 38422951798749789103496445624075627158693379174379766201413719390
    137906271828
Rc = System.Byte[]
```

The aggregate value $R = R_a + R_c$

```
R =
  B1 23 76 E0  38 C5 15 2A  B4 0F AB E2  FA 0C AC 81
  D0 34 28 B7  13 14 75 A2  29 08 3C C0  62 3B 97 2D
```

The value k is

```
k = 24387631527158770388984723649123275410144206977564237560989138440
    76357243053
```

The values R and k (or the document to be signed) and the Lagrange coefficients are passed to Alice and Carol who use them to calculate their secret scalar values:

```
sa = 72257766172444455161114447033857663786203361113341060733865861420
    113532756919;
sc = 31748486815356192849956159948802147484210008385707080919806761840
    605725462941
```

The signature contributions can now be calulated:

```
Sa = 525850937279813825804833906239866887178739547491539450682695840
    7016143211994
Sc = 236908710295121963585805412178511303671624552418318998419111826
    8317926650670
```

The coordinator calculates the aggregate value $S = S_a + S_b$

```
S =  390590898417095679933206621140787667646524639718676885016125737
    048615611675
```

The coordinator checks to see that the signature verifies:

```
S.B = R + kA =
    X: 44137199296357774791154193859950611817340233105447413062332736
        902705770844448
    Y: 29688538012678578878269116100526571773188243259702429043049505
        096610475369173
```

6.4. Quorate Threshold Signature Ed448

The administrator creates the aggregate key pair

```
ED448Aggregate Key (ED448)
    UDF:        ZAAA-ITSI-GQED-44XA-GGRE-GATE-XKEY
    Scalar:     508904606564197215312735879582840960158109827605415754
        207268050539683337837216003977228732536078674802149039736292
        653681850024283019712
    Encoded Private
  78 22 7E 3B  89 95 80 5D  04 19 DC 27  F1 7F 9B E4
  86 2B 0B DD  55 64 EE 04  19 49 4D DE  B9 04 3B 9E
  8B 7D DC EC  EC 8F DD 1D  E7 88 86 FD  11 FD 78 EF
  1A 8B 84 8F  77 00 73 65
    X: 44109173355278142669484438370724914685176368933547176239809629
        750376846559532159069031122126951468222268738637863145753506 8
        446135118173
    Y: 53219402718535721212460981200104434180077825188675868294070079
        508466292055282335688813870601603863793479483949662447412551 1
        419755284720
    Encoded Public
  43 61 20 A0  B1 DF AA BD  6B 55 00 97  A3 BE CB B8
  09 57 20 88  16 69 E4 B9  E1 7E 9C 13  C0 41 5B CB
  4D 3E E4 99  2E 2D 48 89  1C C0 FB 26  58 C2 DD 5C
  C1 DC 17 82  D7 A0 43 EE  80
```

Three key shares are required for Alice, Bob and Carol with a threshold of two. The parameters of the Shamir Secret Sharing polynomial are:

```
a0 = 0
a1 = 32303104410901325448815025064978420882006905219645253608794072719476574575314165522466194196662039682316916303558691241762314558691569
```

The key share values for the participants are

```
xa = 1
ya = 32303104410901325448815025064978420882006905219645253608794072719476574575314165522466194196662039682316916303558691241762314558691569

xb = 2
yb = 64606208821802650897630050129956841764013810439290507217588145438953149150628331044932388393324079364633832607117382483524629117383138

xc = 3
yc = 96909313232703976346445075194935262646020715658935760826382218158429723725942496567398582589986119046950748910676073725286943676074707
```

Alice and Carol are selected to sign the message "This is another test"

The Lagrange coefficients are:

```
la = 90854840536950861318665475986000566794205170085914757535186274897573001980769792858097877645846187981655146854545831152386877929824891
lc = 90854840536950861318665475986000566794205170085914757535186274897573001980769792858097877645846187981655146854545831152386877929824889
```

Alice and Carol select their values ra, rc

```
ra = 99164093738328554456802254096658192046948902724662437580887236117041266054169276412287108673385940901301330720438850816815324572671888
Ra = System.Byte[]

rc = 10736095117039832084779503235995842819090087342796874167887891342295073597868347842139557994718349208704173355864423042065027676646177 0
Rc = System.Byte[]
```

The aggregate value $R = R_a + R_c$

```
R =
  30 A1 B3 17  69 AB 99 0E  57 5E 0B A5  2E 51 72 C0
  BA 31 35 39  51 7C 83 B8  22 01 9F 85  45 93 9B 54
  19 A2 FA FD  34 7A CA 88  7D 05 52 A6  21 5C E0 B7
  69 62 F4 94  6C 15 C3 11  80
```

The value k is

```
k = 10235373755332719753132453070965244233375545860969817073088585850349843086119970818725465449309010426736084043284792610670057777560797973
```

The values R and k (or the document to be signed) and the Lagrange coefficients are passed to Alice and Carol who use them to calculate their secret scalar values:

```
sa = 139309497153302849491888013583468198117215527915382637948377383976787863843741041141797168940839247505130521309883868015030349767862243;
sc = 42400183920598873145442938388532935471194812256446877121995165818358140117798544574398586350853128458179772399207794289743406091787536
```

The signature contributions can now be calulated:

```
Sa = 11764221449113877578196506946825464528759007339777891241759696110
    960649954558779908875349243926380740989095329523226588667703100750
    667839
Sc = 12616953178780321444037644399362512218738995817915693431752576550
    854633821139078221755632234822272242667761955063098795652421410850
    378620
```

The coordinator calculates the aggregate value $S = S_a + S_b$

```
S =  62102065205040267585010561489878633886569691405106331664750176980
    638237360824622734690239258317292240237543474954087612723869530139
    96680
```

The coordinator checks to see that the signature verifies:

```
S.B = R + kA =
    X: 41458922531347602467333936260245029252955857925674064823925020
        498684283758735
    Y: 46892836594134893138380416498249507660693894634845371281694628
        683354340901899
```

7. Security Considerations

TBS.

8. IANA Considerations

This document requires no IANA actions.

9. Acknowledgements

10. Normative References

[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-07, 18 October 2019, <https://tools.ietf.org/html/draft-hallambaker-mesh-udf-07>.

[draft-hallambaker-threshold] "[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>.

[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>.

11. Informative References

[draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-09, 23 October 2019, <https://tools.ietf.org/html/draft-hallambaker-mesh-developer-09>.

[RFC2314] Kaliski, B., "PKCS #10: Certification Request Syntax Version 1.5", RFC 2314, DOI 10.17487/RFC2314, March 1998, <https://www.rfc-editor.org/rfc/rfc2314>.