Workgroup: Network Working Group
Internet-Draft:
draft-hallambaker-threshold-sigs
Published: 9 March 2020
Intended Status: Informational
Expires: 10 September 2020
Authors: P. M. Hallam-Baker
ThresholdShare.com

Threshold Signatures in Elliptic Curves

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.

https://mailarchive.ietf.org/arch/browse/cfrg/Discussion of this draft should take place on the CFRG mailing list (cfrg@irtf.org), which is archived at .

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 10 September 2020.

Copyright Notice

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

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 builds on the approach described in [draft-hallambaker-threshold] to define 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.

The approach used is based on that developed in FROST [Komlo]. This document describes the signature scheme itself. The techniques used to generate keys are described separately in [draft-hallambaker-threshold].

As in the base document, we first describe signature generation for the case that $n = t$ using 'direct' coefficients, that is the secret scalar is the sum of the secret shares. We then show how the scheme is modified using Shamir secret sharing [Shamir79] and Lagrange coefficients for the case that $n > t$.

1.1. Applications

Threshold signatures have application in any situation where it is desired to have finer grain control of signing operations without this control structure being visible to external applications. It is of particular interest in situations where legacy applications do not support multi-signatures.

1.1.1. HSM Binding

Hardware Security Modules (HSMs) prevent accidental disclosures of signature keys by binding private keys to a hardware device from which it cannot be extracted without substantial effort. This provides effective mitigation of the chief causes of key disclosure

but requires the signer to rely on the trustworthiness of a device that represents a black box they have no means of auditing.

Threshold signatures allow the signer to take advantage of the key binding control provided by an HSM without trusting it. The HSM only contributes one of the key shares used to create the signature. The other is provided by the application code (or possibly an additional HSM).

1.1.2. Code Signing

Code signing is an important security control used to enable rapid detection of malware by demonstrating the source of authorized code distributions but places a critical reliance on the security of the signer's private key. Inadvertent disclosure of code signing keys is commonplace as they are typically stored in a form that allows them to be used in automatic build processes. Popular source code repositories are regularly scanned by attackers seeking to discover private signature keys and passwords embedded in scripts.

Threshold signatures allow the code signing operation to be divided between a developer key and an HSM held locally or by a signature service. The threshold shares required to create the signature can be mapped onto the process roles and personnel responsible for authorizing code release. This last concern might be of particular advantage in open source projects where the concentration of control embodied in a single code signing key has proved to be difficult to reconcile with community principles.

1.1.3. Signing by Redundant Services

Redundancy is as desirable for trusted services as for any other service. But in the case that multiple hosts are tasked with compiling a data set and signing the result, there is a risk of different hosts obtaining a different view of the data set due to timing or other concerns. This presents the risk of the hosts signing inconsistent views of the data set.

Use of threshold signatures allows the criteria for agreeing on the data set to be signed to be mapped directly onto the requirement for creating a signature. So if there are three hosts and two must agree to create a signature, three signature shares are created and with a threshold of two.

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 deterministic mechanism described in specification [draft-hallambaker-mesh-udf] is used to generate the private keys used in the test vectors.

2.4. Implementation Status

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

3. Principles

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:

$R = r.B$

$S = r + k.s \bmod L$

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 checks that:

$S.B = R + k.A$

This equality must hold for a valid signature since:

$\boldsymbol{S.B} = (r + k.s).B$

$= r.B + k.(s.B)$

$= R + k.A$

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

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

$S_2 = r + k_2.s \bmod L$

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

The method of constructing r **MUST** ensure that it is unique and unguessable.

3.1. Direct shared threshold signature

A threshold signature R, S is constructed by summing a set of signature contributions from two or more signers. For the case that the composite private key is the sum of the key shares ($n = t$), 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 dealer as follows:

0. The dealer 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 dealer .

2. The dealer 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 dealer .

4. The dealer 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 dealer publishes it. Otherwise, the dealer **MAY** identify the signer(s) that provided incorrect contributions by verifying the values R_i and S_i for each.

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

3.2. Shamir shared threshold signature

To construct a threshold signature using shares created using Shamir Secret Sharing, each private key value s_i is multiplied by the Lagrange coefficient l_i corresponding to the set of shares used to construct the signature:

$A_i = s_i l_i.B$

$R_i = r_i.B$

$S_i = ri + klisi \bmod L$

It is convenient to combine the derivation of S_i for the additive and Shamir shared threshold signatures by introducing a key multiplier coefficient c_i:

$S_i = ri + kcisi \bmod L$

Where $c_i = 1$ for the additive shared threshold signature

$c_i = l_i$ for the Shamir shared threshold signature

3.3. Stateless computation of final share

One of the chief drawbacks to the algorithm described above is that it requires signers to perform two steps with state carried over from the first to the second to avoid reuse of the value r_i. This raises particular concern for implementations such as signature services or HSMs where maintaining state imposes a significant cost.

Fortunately, it is possible to modify the algorithm so that the final signer does not need to maintain state between steps:

0. All the signers except the final signer F generate their value r_i and submit the corresponding value R_i to the dealer

1. Dealer calculates the value $R - R_F$ and sends it to the final signer together with the all the other parameters required to calculate k and the final signer's key multiplier coefficient c_F.

2. The final signer generates its value r_F

3. The final signer calculates the value R_F from which the values R and k can now be determined.

4. The final signer calculates its key share contribution $S_F = rF + kcFsF \bmod L.$

5. The final signer returns the values S_F and R to the dealer.

6. The dealer reports the value R to the other signers and continues the signature process as before.

While this approach to stateless computation of the signature contributions is limited to the final share, this is sufficient to cover the overwhelming majority of real-world applications where $n = t = 2$.

Note that the final signer **MAY** calculate its value r_F deterministically provided that the parameters R - R_F and c_F are used in its determination. Other signers **MUST NOT** use a deterministic means of generating their value r_i since the information known to them at the time this parameter is generated is not sufficient to fix the value of R.

3.3.1. Side channel resistance

The use of Kocher side channel resistance as described in [draft-hallambaker-threshold] entails randomly splitting the private key into two shares and performing the private key operation separately on each share to avoid repeated operations using the same private key value at the cost of performing each operation twice.

This additional overhead **MAY** be eliminated when threshold approaches are used by applying blinding factors whose sum is zero to each of the threshold shares.

For example, if generation of the threshold signature is divided between an application program A and an HSM B using the final share approach to avoid maintaining state in the HSM, we might generate a blinding factor thus:

0. A generates a random nonce n_A and sends it to B with the other parameters required to generate the signature.

1. B generates a random nonce n_B

2. B calculates the blinding factor x by calculating $H(n_A, nB)$ *where H is a strong cryptographic digest function and converting the result to an integer in the range 1 x L.*

3. B calculates the signature parameters as before except that the threshold signature contribution is now $S_B = rB + k(cBsB + x) \mod L$.

4. B returns the nonce n_B to A with the other parameters.

5. A calculates the blinding factor x using the same approach as B

6. A calculates the signature parameters as before except that the threshold signature contribution is now $S_A = rA + k(cAsA - x) \mod L$.

This approach **MAY** be extended to the case that $t > 2$ by substituting a Key Derivation Function (e.g. [RFC5860]) for the digest function.

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. Calculation of r values

The method of constructing the values r_i **MUST** ensure that each is unique and unguessable both to external parties, the signers and the dealer. The deterministic method specified in [RFC8032] cannot be applied to generation of the values r_i as it allows the dealer to cause signers to reveal their key shares by requesting multiple signature contributions for the same message but with different values of k. In particular, requesting signature contributions for the same message:

With different Lagrange coefficients.

With a false value of R

To avoid these attacks, the value r_i is generated using a secure random number generator. This approach requires the signer to ensure that values are never reused requiring that the signing API maintain state between the first and second rounds of the algorithm.

While there are many approaches to deterministic generation of r_i that appear to be sound, closer inspection has demonstrated these to be vulnerable to rogue key and rogue contribution attacks.

3.4.2. 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.

3.4.3. 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.

3.4.4. 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. Ed2519 Signature

The means by which threshold shares are created is described in [draft-hallambaker-threshold].

The dealer selects the signers who are to construct the signature. Each signer then computes the value R_i:

0. Randomly generate an integer r_i such that 1 r_i L.

1. 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.

2. Transmit the value R_i to the dealer

3. At some later point, the dealer **MAY** complete the signature by returning the values F, C, A and R as specified in [RFC8032] together with the key multiplier coefficient c_i. The signers **MAY** then complete their signature contributions:

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

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

6. Return the values R_i, S_i to the dealer .

The dealer then completes the signature by:

0. Computing the composite value $S = S_1 + S_2 + ... + S_n$

1. Verifying that the signature R, S is valid.

2. Publishing the signature.

5. Ed448 Signature

The means by which threshold shares are created is described in [draft-hallambaker-threshold].

The dealer selects the signers who are to construct the signature. Each signer then computes the value R_i:

0. Randomly generate an integer r_i such that 1 r_i L.

1. 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 dealer

0. At some later point, the dealer **MAY** complete the signature by returning the values F, C, A and R as specified in [RFC8032] together with the key multiplier coefficient c_i. The signers **MAY** then complete the signature contributions:

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

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

3. Return the values R_i, S_i to the dealer.

The dealer then completes the signature by:

0. Computing the composite value $S = S_1 + S_2 + ... + S_n$

1. Verifying that the signature R, S is valid.

2. Publishing the signature.

6. Test Vectors

6.1. Direct Threshold Signature Ed25519

The signers are Alice and Bob's Threshold Signature Service '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: 111167936729704271617902644692802945071890447281405479540710227976454124042406369344932655633664630560242213431409139866940284702002648469365756492647970
    Y: 616554041716113965730218081191086647495742351253436802064542856299141386615046548323087409388548650272224487089895079970526014354411536487887012976125920 0
    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:     549407726701534591461529255641981052629714857308898189867276085732291799020168
    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: 142714950693498382163795401962631409640323935129038422061681825518850827098876289800868735522232908519794251130907125878675634341148406570631356841088001 5
    Y: 280943289480041124281894662237574408863886842912546053558599236240968229706795825282419594219442074647093851302547452470435943851347762997860136672501557 3
    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 composite Signature Key $A = A_a + A_b$

```
Aggregate Key = Alice + Bob ()
    UDF:        TBS
    Scalar:     26569330913556569171916721364983482306308422345436973
        562933121131713846842l3
    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
        868890648681144374473372467599418125802954734698507990149436 7
        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 first generates her value r and multiplies it by the base point to obtain the value R_a:

```
Alice:
r_a =  26964569597283588823958971235480505062310410788972392836560088
    34751424110238
R_a =
  00 76 86 F6  47 15 21 49  5D BC 40 61  C8 64 96 7B
  7A 77 B3 10  60 20 1D 50  C7 61 CF 27  CF 2F D3 E0
```

Alice passes her value R_A to Bob along with the other parameters required to calculate i. Bob then calculates his value R_A and multiplies it by the base point to obtain the value R_b:

```
Bob:
r_b =  11540386535681085531005638949280260705343094809283082001566243
    3246392456462
R_b =
  94 41 83 3D  D2 B1 FE 27  AD 28 EF 95  73 B5 5E E9
  4A D2 25 91  18 70 03 9C  C2 9B E6 F0  12 09 A9 5E
```

Bob can now calculate the composite value $R = R_a + R_b$ and thus the value k.

```
R =
  C2 80 BF DE  1C 51 7D 11  34 74 7E A6  21 B8 65 B7
  1D 61 95 86  5E 21 7E 0A  FF 70 73 ED  1F AA 00 F0
k =  170102912253981601724242951379813871831868455538348877674074093
    828674348023
```

Bob calculates his signature scalar contribution and returns the value to Alice:

```
Bob:
S_b =  225520839568397439464155864406652312967791776898042814042093520
    33590198456755
```

Alice can now calculate her signature scalar contribution and thus the signature scalar S.

```
Alice:
S_a =  640490359018679929396246162396925933144774981306129108832670270
    32679305425610
S =  142310640853851147463083370499278822026855122266181162274568702700
    984049631376
```

Alice checks to see that the signature verifies:

```
S.B = R + kA =
    X: 870701950246721079701356833237097346414927654603615468597696790
        39147985870824
    Y: 518372260008422157448163243093901187267784474460247840693581070
        69430908552186
```

6.2. Direct Threshold Signature Ed448

The signers are Alice and Bob's Threshold Signature Service '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 composite 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 first generates her value r and multiplies it by the base point to obtain the value R_a:

```
Alice:
r_a =  625655274937139198578651935196023659310121016735625773118381415405476623256718067873688890373064694135364043742109186927500074962552 78
R_a =
  45 C6 4E FC  B3 CA 83 D6  12 9E 6C 9C  15 6A 63 E7
  50 B1 64 8A  F7 4C 45 BC  60 00 89 07  04 D3 6E ED
  55 15 E4 5D  BC 4F 61 A2  02 BD 37 FA  30 57 41 90
  CE 13 95 49  35 50 77 2D  00
```

Alice passes her value R_A to Bob along with the other parameters required to calculate i. Bob then calculates his value R_A and multiplies it by the base point to obtain the value R_b:

Bob:
r_b = 685535816920212635765454098468725349450882264665170458200744545356372013002531041190738811820576800111218528926352540342021878109825957

R_b =

21 27 22 F0	00 88 E8 D0	97 06 50 01	03 C8 37 24
A2 59 35 40	2B 3A 6D F9	90 4F 05 22	0C 43 71 D2
2C 0B 7D B0	4D 1D 95 7C	8C 0B 4D 19	19 6F 41 FF
A4 6F BF 9C	D8 47 58 9C	00	

Bob can now calculate the composite value $R = R_a + R_b$ and thus the value k.

R =

44 E0 13 83	9E 90 CE 1B	8B F2 71 04	6C E9 AE 16
F8 BE 72 22	3E 29 BE C4	FB 41 DF 47	31 CC E3 9C
93 2A 71 6B	B1 32 3E EE	8D 00 98 B5	DE E9 FF 33
1B 61 A9 F8	13 1E E2 5F	80	

k = 855935932252697801738534690989305842959140890035644024382291026009124689192398260008095709124990846658648602608610131291879967363 36428

Bob calculates his signature scalar contribution and returns the value to Alice:

Bob:
S_b = 232253084730683179523179694341329037316272285063087687722071784914930944184070988830794027539874172628160344923079214317925789340736 81

Alice can now calculate her signature scalar contribution and thus the signature scalar S.

Alice:
S_a = 158225210955474676918761484287784841602547168360770712857577167181048545744661477835857402287508345521742231691864803690645070692022407

S = 181450519428542994871079453721917745334174396867079481629784345672541640163068576718936805041495762784558266184172725122437649626096088

Alice checks to see that the signature verifies:

```
S.B = R + kA =
    X: 202379584762181379837764781419500978468728870833639482813100436
        52612307164298
    Y: 420246678721605797231957483067827769701178182687570423951670336
        071351675224163
```

6.3. Shamir Threshold Signature Ed25519

The administrator creates the composite key pair

```
ED25519Aggregate Key (ED25519)
    UDF:        ZAAA-GTSI-GQED-255X-XAGG-REGA-TEXK-EY
    Scalar:     39348647608109113656999806950437958090469802387424444
        589375066079861075223816
    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
        99049697167784279959512347660021645117385875752575305387583747
        8249060472500577218550685239 70
    Y: 20211025649802071998810413948266748565975140520947927724517956
        20676255050777515980186295517468245337267098109901934556623856
        15273611630344103185130545804 0
    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
```

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 = 39348647608109113656999806950437958090469802387424444589375066079
    861075223816
a1 = 27841156338804313049318255952510639422909238911749728184666239070
    091375688
```

The key share values for the participants are

```
xa = 1
ya = 319146087778660690018319239117549752560712982943665628754997762
    7503895344559

xb = 2
yb = 321930203412541121323251064712800816503003906834840601573464386
    6573986720247

xc = 3
yc = 324714319046421552628182890308051880445294830726015574391931010
    5644078095935
```

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

The Lagrange coefficients are:

```
la = 361850278866613110698659328152149712042855817968995380300097546
    9142727125496
lc = 361850278866613110698659328152149712042855817968995380300097546
    9142727125494
```

Alice and Carol select their values ra, rc

```
ra = 543693216284346229326653900607887891883036755440162091264891427
    8387841243984
Ra =
  8D 87 BC 1E  A9 84 92 05  56 93 31 35  B4 76 C0 98
  EA F4 60 12  A5 5F D6 9B  0A 69 41 1A  36 45 7A 83

rc = 153880008066324440639430181684773955917306056033011390782993111
    7643929735504
Rc =
  4B 65 A9 E9  87 7B 90 37  04 1F A1 64  16 69 F0 C3
  0E 45 1B F4  5A 55 5C F1  99 A4 71 2C  B0 00 EB 15
```

The composite value $R = R_a + R_c$

```
R =
  3B C4 76 D4  33 AC 43 21  D7 7E 4F 2D  58 E9 4B 92
  70 F2 36 37  2C D9 28 EA  4E 91 57 80  2F D4 21 E3
```

The value k is

```
k = 650222059714442346590675788610639100157400099522246794714311417349939649783
```

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 = 116868852801377924328819530524174916798213656446503062832399097211311589134;
sc = 19949311934340233438456788299812377182020840260598759310413204163206880775 27
```

The signature contributions can now be calulated:

```
Sa = 164208815151832593482346705486826659125235517101029538623941306306504724 38
Sc = 256712231802739170846941079387902592658021656570472227584300769478345437 9696
```

The dealer calculates the composite value S = S_a + S_b

```
S =  258354319954257496781764546442770859249274011741482522970540182541410505213 4
```

The dealer checks to see that the signature verifies:

```
S.B = R + kA =
    X: 383057553965331832443391617262841832660090906571768342660050390161348107847 85
    Y: 294076998104900021910560065205253292137609857079802521374774853526436 76612632
```

6.4. Shamir Threshold Signature Ed448

The administrator creates the composite key pair

ED448Aggregate Key (ED448)
    UDF: ZAAA-ITSI-GQED-44XA-GGRE-GATE-XKEY
    Scalar: 508904606564197215312735879582840960158109827605415754207268050539683337837216003977228732536078674802149039736292653681850024283019712
    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: 441091733552781426694844383707249146851763689335471762398096297503768465595321590690311221269514682222687386378631457535068446135118173
    Y: 532194027185357212124609812001044341800778251886758682940700795084662920552823356888138706016038637934794839496624474125511419755284720
    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 = 508904606564197215312735879582840960158109827605415754207268050539683337837216003977228732536078674802149039736292653681850024283019712
a1 = 160292239260782229231778370338449107741928257574589648924176843765938863574713864292893351873520049039965264366659426161520599832630033

The key share values for the participants are

```
xa = 1
ya = 124067802603274276632521394005286667134807064664516857920327244920184189527311111121534818534521595952183422975677092929049356536700408

xb = 2
yb = 102650360790154783226968812371734641288324982067276991774131538890977049140485389698232415116349269028838393633244856785796200509680662

xc = 3
yc = 81232918977035289821416230738182615441842899470037125627935832861769908753659668274930011698176942105493364290812620642543044482660916
```

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 = 271964880837215269530332481300488641150918952203711977134827073201317204344949815636374307412539793080438754521149038747587752804050241
Ra =
  5F 55 7D 9B  FF 93 7E 29  29 17 86 F3  5B 86 2F 7E
  00 2A 10 A4  C4 1B 5C 62  62 D8 C8 CD  50 AD 5D E0
  F6 6A 79 63  A4 E3 59 E7  3B 62 83 5B  6B 39 94 42
  2A C5 F1 3D  8C A2 EC 91  00

rc = 90500060032114252948992049036269519789042141541472728148010341208363468368576349995442027468094400827122020880533451096364827401441912
Rc =
  57 94 25 6D  A6 B2 E3 AB  59 5D 6F 3D  F8 2E C6 9E
  3D 7C F8 04  C2 D6 98 5D  CF 3E 15 60  08 AB 8B 66
  19 AB 0A 6D  1A E6 B1 45  D1 A6 0B 21  23 CA 59 A7
  F3 A3 DE 35  8E 04 1D 02  80
```

The composite value $R = R_a + R_c$

```
R =
  94 87 5A 3A  CE 75 7C 97  C0 24 93 5D  00 4C 0A AD
  8D 5B 74 9A  2F 1A 16 EE  85 8C EC EE  C3 17 91 68
  B2 80 4B 73  63 AC A1 04  65 82 16 80  B0 E1 48 3A
  02 63 18 A4  ED 88 15 3F  00
```

The value k is

```
k = 285973197663720232889198742536063535281163142479553045546289086995
    14741698440791352191812187685479498956177113463735604193977889389
    5471
```

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 = 439202283100969231145113903592886711380025682494577181011831758513
    13028032942708096610647251009001796496484075442397708880027894540
    0833;
sc = 141093221585384077726622836602909825867488890436810952256404633364
    64261049584709751578730749442603904910563611563685351983502233618
    319321
```

The signature contributions can now be calulated:

```
Sa = 123791119332859846109872057684293861983384732496375794738113504560
    73449603770713498263282818815719425632776761697840567160491262299
    7176
Sc = 435229461017060259660314451107954032760927490640384168236123746438
    75702906580890909475911316949238322551514292057263950813229229464
    825
```

The dealer calculates the composite value $S = S_a + S_b$

```
S =  16731406543456587207590350279508926525947748156041421156172587920
    46101989442880258921087395051064325788792819090356696224181418524
    62001
```

The dealer checks to see that the signature verifies:

```
S.B = R + kA =
    X: 31370116705528265987661207265970557523081230091184475728092100
       801345054140398
    Y: 14983649831297656181541916312764508612330840642761296389133763
       572240307037592
```

7. Security Considerations

All the security considerations of [RFC7748], [RFC8032] and [draft-hallambaker-threshold] apply and are hereby incorporated by reference.

7.1. Rogue Key attack

The rogue key attack described in [draft-hallambaker-threshold] is of particular concern to generation of threshold signatures.

If *A* and *B* are public keys, the intrinsic degree of trust in the composite keypair *A* + *B* is that of the lesser of *A* and *B*.

7.2. Disclosure or reuse of the value r

As in any Schnorr signature scheme, compromise of the value *r* results in compromise of the private key. The base signature specification [RFC8032] describes a deterministic construction of *r* that ensures confidentiality and uniqueness for a given value of *k*.

As described above, this approach is not applicable to the generation of values of r_i to compute threshold signature contributions. Accordingly the requirements of [RFC4086] regarding requirements for randomness **MUST** be observed.

Implementations **MUST NOT** use a deterministic generation of the value r_i for any threshold contribution except for calculating the final contribution when all the other parameters required to calculate *k* are known.

7.3. Resource exhaustion attack

Implementation of the general two stage signing algorithm requires that signers track generation and use of the values r_i to avoid reuse for different values of R_i. Implementations **MUST** ensure that exhaustion of this resource by one party does not cause other parties to be denied service.

7.4. Signature Uniqueness

Signatures generated in strict conformance with [RFC8032] are guaranteed to be unique such that signing the same document with the same key will always result in the same signature value.

The signature modes described in this document are computationally indistinguishable from those created in accordance with [RFC8032] but are not unique.

Implementations **MUST** not use threshold signatures in applications where signature values are used in place of cryptographic digests as unique content identifiers.

8. IANA Considerations

This document requires no IANA actions.

9. Acknowledgements

[TBS]

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-08, 6 January 2020, <https://tools.ietf.org/html/draft-hallambaker-mesh-udf-08>.

[draft-hallambaker-threshold]
Hallam-Baker, P., "Threshold Key Generation and Decryption in Ed25519 and Ed448", Work in Progress, Internet-Draft, draft-hallambaker-threshold-00, 5 January 2020, <https://tools.ietf.org/html/draft-hallambaker-threshold-00>.

[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>.

[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>.

[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>.

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>.

[Komlo] Komlo, C. and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures", 2020.

[RFC5860] Vigoureux, M., Ward, D., and M. Betts, "Requirements for Operations, Administration, and Maintenance (OAM) in MPLS Transport Networks", RFC 5860, DOI 10.17487/RFC5860, May 2010, <https://www.rfc-editor.org/rfc/rfc5860>.

[Shamir79] Shamir, A., "How to share a secret.", 1979.