Workgroup: Network Working Group
Internet-Draft:
draft-hallambaker-threshold-sigs
Published: 13 January 2021
Intended Status: Informational
Expires: 17 July 2021
Authors: P. M. Hallam-Baker
ThresholdSecrets.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 17 July 2021.

## 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
  1.1. Applications
    1.1.1. HSM Binding
    1.1.2. Code Signing
    1.1.3. Signing by Redundant Services
2. Definitions
  2.1. Requirements Language
  2.2. Defined Terms
  2.3. Related Specifications
  2.4. Implementation Status
3. Principles
  3.1. Direct shared threshold signature
  3.2. Shamir shared threshold signature
  3.3. Stateless computation of final share
    3.3.1. Side channel resistance
  3.4. Security Analysis
    3.4.1. Calculation of r values
    3.4.2. Replay Attack
    3.4.3. Malicious Contribution Attack
    3.4.4. Rogue Key Attack
4. Ed2519 Signature
5. Ed448 Signature
6. Test Vectors
  6.1. Direct Threshold Signature Ed25519
  6.2. Direct Threshold Signature Ed448
  6.3. Shamir Threshold Signature Ed25519
  6.4. Shamir Threshold Signature Ed448
7. Security Considerations
  7.1. Rogue Key attack
  7.2. Disclosure or reuse of the value r
  7.3. Resource exhaustion attack
  7.4. Signature Uniqueness
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 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 $s = (S\text{-}r).k^{-1} \bmod L$. 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 $S_B = rB + k(cBsB + x) \bmod L$.

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:         31219130380639437694769688896227611542048535900l
        34467943432016761653342335248
    Encoded Private
  10 AE C0 C2  16 65 9B 4F  7C 9D DE 82  3E 49 7F D4
  9B 14 BB F8  2D 9F 0C 11  24 D7 15 E3  43 79 57 20
    X:              -13697699435406080999251131063344049965140553452
        75230535371481910664691934716006479350632763595434271914428 9
        23055666860885869803952842897464955304098899 30
    Y:              27879387561061608084416280018586439962550393815 7
        56937417470041484575847933129442414739377683176726648757909 8
        76753757770435041133875539167695159113101935 58
    Encoded Public
  45 16 53 7C  26 50 CF DA  F1 A4 DF 4C  45 DC 3D 95
  4E B6 8E EB  A6 5A 27 D6  CD 5B 43 C5  F4 06 53 ED
ED25519Bob's Key (ED25519)
    UDF:            ZAAA-GTSI-G2ED-255X-XBOB-XSXK-EY
    Scalar:         56721284389150941480030862015889172068550899562 0
        72140666211075925337851277632
    Encoded Private
  E5 CD 34 01  FD 8C 0E 27  81 4B 11 DD  12 68 50 A1
  4B 5A D5 E1  E1 41 D7 68  5F 51 ED B4  3A 84 58 5C
    X:              -13809282472298084436735987888897423507149580966
        95279176144667088404443396397517848239814465756456522327058 8
        53224596424709463475705754755341414062853232 57
    Y:              26368422634287198470631741176042309594706808836 6
        39354679860237843743270748208980665375588139959296306875175 9
        96453625258663082831712843279319704043214586 77
    Encoded Public
  F1 5F C0 78  F8 32 49 2C  D9 64 CC 2B  CF 90 5C 4F
  23 EA BB F8  38 99 C5 FE  F3 AA 67 BE  AB EC D2 5E
```

The composite Signature Key $A = A_a + A_b$

```
Aggregate Key = Alice + Bob ()
    UDF:            TBS
    Scalar:         109634784180323260712231215560085272031403914964
        771733761968142756574260l012
    Encoded Private
  34 33 AB 10  9A 09 A9 61  65 8B 3A EC  58 21 FB 2D
  0D 45 74 49  45 BA E2 CF  A8 98 C2 94  C9 82 6C 02
    X:              -838376752943008528429011216134455942963523723 47
        711317409367737761568353629718805151940195325485285476438422
        923698718220652243749390297055882388709313280
    Y:              160553422944358144751060009820735322036903773802
        361117046457476895165059738086663330972263850675453249990301
        039847381126319665322544612416002508214476153 4
    Encoded Public
  48 1A 27 66  06 AF 4E 3C  20 A4 02 CD  8A 13 46 99
  02 B7 75 F8  AC D4 7E 89  68 FB 68 EB  D8 EF 4A C7
```

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:            505210734621497595393270784745614175113191664157
        417742560010579848211437778 5
R_a =
  DF A3 D5 CC  9F 94 63 67  BB 3E C3 F7  88 4A 0D 52
  00 20 A2 90  13 27 4E 47  03 19 DA EC  BF 74 CB 14
```

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:            677880217486034074720202546367410174561950677574
        530990007035432307188622786 7
R_b =
  DD C8 79 2A  BB D8 72 D5  9D F5 13 22  C2 F1 58 62
  47 DC 19 39  C5 CE 02 FB  24 0B FA 64  D1 55 BC 3E
```

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

```
R =
  5A D0 1C 17  95 ED 9B 99  B8 CD CE 7B  EE 47 6E A5
  0E A6 CF 51  DE DA 89 CB  B5 F4 4C E2  D5 0D 58 FA
    k:              625005044347993004605907480401547053627770740065
        204060245057160070342870275
```

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

```
Bob:
    S_b:            136373130884201209719904273113512997386754201427
        873707075718429302441345086 6
```

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

```
Alice:
    S_a:            694422500722053719583170959521207108671468233956
        308982139355755735703127183 7
    S:              107095073873028707905756576330420681972510799446
        191928614879091209599047171 4
```

Alice checks to see that the signature verifies:

```
S.B = R + kA =
    X:              499652471325922372829034886924764341503336793855
        862151300712776712411804546 24
    Y:              465061436809499600324596437786395684290405421559
        114992621358629287884998854 58
```

### 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:         67228647733113098351303974335061622786434675392496278786072975722251199961844351356940379318639809671792494585484654439698408834482326 4
    Encoded Private
  6F 85 B1 91  9A 37 06 A6  B2 15 79 AD  5B 69 16 6A
  5A CD C8 17  D4 14 1F 68  DA 97 C5 B4  44 79 CE EA
  3C 17 7B E1  29 44 70 DF  41 C8 98 38  1E 7C 9B 3B
  03 63 6F 85  E8 39 31 91
    X:              52604601965504363286847095228694752949228309234412247607715142364524364897451218254840570287356053384667326276706401936547083086110604 9
    Y:              14537455078538085081293442475798686667348523704793855454449269494660806098645949580705545504820871399191947720250115717234689256856152
    Encoded Public
  59 55 F4 7A  66 08 91 35  F8 15 63 F4  90 91 7F 38
  12 E3 49 22  51 F8 BC 4A  41 C9 44 59  5A 64 9B 40
  0B C5 7E 53  48 0F 32 12  90 32 69 38  47 28 94 BB
  99 D1 16 6F  2D D5 3D 4F  80
ED448Bob's Key (ED448)
    UDF:            ZAAA-ITSI-G2ED-44XB-OBXS-XKEY
    Scalar:         45505262669826238539773654772715942394152079290490861260354285090916721598771390232261993392940445574180684806429494528311379968326121 2
    Encoded Private
  CA 15 22 BD  F4 0F 9E 0A  EC A7 61 79  BE 9E E3 38
  BF 93 D3 5B  B3 E6 FC F0  A7 5B 7C F0  E7 B5 89 F6
  2E F6 D1 0E  72 49 4D DF  34 5E 2F 7C  9E 42 1D 85
  AB AB 30 BD  68 C6 3E 35
    X:              75202410820027271083218753555716445507868973459517118999338325989260725302750087854343990875052576388066123217132205985485252278226 5
    Y:              61932987310215967679132614207316679059468311140972938358419983344102848452558369942118142216819085607478632402049221487379649557005651 1
    Encoded Public
  76 2B FC F8  AC 96 79 DE  1C 72 07 65  DD 49 5B 28
  C7 04 CB A8  A5 96 3D D9  9E 23 FA 05  83 15 33 95
  85 82 F8 CF  A3 7A 2F 24  F8 EB D6 AE  20 0A 25 D0
  44 1A F9 C0  86 D7 87 B7  00
```

The composite Signature Key $A = A_a + A_b$

```
Aggregate Key = Alice + Bob ()
    UDF:            TBS
    Scalar:         37081017585983033086790579245768850275405505798
        8
        9431004203730936080319183691990159484919536564829667987003
        16
        6459151585145535287018580
2
    Encoded Private
  4A AB 7A BB  2D 95 72 75  B1 3A 1D 22  24 17 76 2D
  A1 D5 55 94  67 35 8C E7  A1 A0 ED 0C  E7 88 FF 9F
  6E 2F 70 80  89 F5 01 2A  C0 AD 4C 4E  7B 90 68 6C
  F4 53 BA 32  9B 70 0F 0D
    X:              58324955340769999284154112964835446252412293188
        85705805155251963990666340677631698415401706202386907579053
        6
        30514579317017660114474427
    Y:              51804043756281118116941374071829093835126916888
        25712410716468924572185200107775886440641278975614969911163
        3
        05182323456988626099626934
1
    Encoded Public
  34 70 8D 08  DE 63 0B A6  49 2A 33 D8  B7 15 A9 84
  A4 87 F6 B6  C7 4B 1C AE  5A 1F 7C 4B  12 70 FB CF
  5A A9 3C 20  31 BA 9A 53  A0 FE 2A 43  24 97 06 F8
  DA 40 0D 88  E3 D9 DE 2E  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:            15480181626724046454683444651545640665184531440
        1
        00297726490569350044666985787991118909012690364306009869590
        2
        15990266846595204366520172
9
R_a =
  BF 60 68 8C  92 23 91 A7  92 65 D7 A9  3A 11 B6 25
  91 CC 72 0D  83 F7 80 06  4C 7F 7B FA  F5 60 CF FC
  43 DA 5E 9F  71 09 6C 51  6E 28 E7 8D  50 2D 7A 4A
  1F 00 17 FF  18 F5 65 F0  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: 151741242222551333693536358753113477279079323953405968709541531009609312639878485678278493044984250865569658971735381320787025215934551

R_b =

E2 20 7A 34 5E E2 BE B0 EE DC 3D 7E 98 AB 00 5B
7E B5 4A 6D 9D 6B AE 00 C3 61 3C 0E BF 85 44 84
2D C2 46 BD 6A EB CF 60 52 A6 22 7F 3E 6D 52 D7
1B B5 A8 FB A2 6E D9 19 00

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

R =

7D E7 D1 AC 39 91 2D A1 64 82 A2 12 11 FD 48 2A
E4 C1 69 4F F1 DB 8C F4 B0 41 44 DB 81 9A 99 93
28 80 BD FC 4E 30 9A 0D 24 7C 2E 97 36 EB DA E9
78 83 08 B9 A5 1A 9F AF 80

k: 152478129684675943479409248843466240733035903267926235089418642613018543821412858874657453613785631671228639879208851203344161958472626

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

Bob:

S_b: 48308025717910676096709611259971167259530693934996497663692684612726013852291320682694383487154034336746467404823679970210640505379249

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

Alice:

S_a: 92976538608972950053953380267864497012076619552159254513604782454606452130056976085487612522524691939995877943127262307963446226438525

S: 14128456432688362615066299152783566427160731348715575217729746706733246598234829676818199600967872627674234534795094227817408673181777 4

Alice checks to see that the signature verifies:

```
S.B = R + kA =
    X:              438553256512884225923994157378894696848243269381
        587867100004786255910808966 86
    Y:              100086885282402628787474925500974806696629978712
        714426597958576720943534380 94
```

### 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:         367238470592488326468789252109412889361910680229
        030897606928447791655888795 04
    Encoded Private
  FE 48 94 1F  EB 3D 28 E1  61 81 E2 1E  E1 CF F2 1E
  1E 70 91 30  DF 98 9F 1C  34 EB BB 74  C5 C8 07 EB
    X:              143576564277195758046684172284175869008525477709
        640743490221115123376609940386394888392330104965579307772627
        313244177612005636942740116142030215202393600
    Y:              844838272625277895849027219595751726665225134917
        547580682441821283235675507225396641352769322822815561632929
        543097074319051436285787045255908364074589900
    Encoded Public
  DF E8 0A 2B  E9 6C 53 C0  AB 9B BC BC  39 95 9A 61
  9C 33 2E 22  24 A7 F7 F2  21 06 AC 6D  01 5D 0B E2
```

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 =    367238470592488326468789252109412889361910680229030897606928
            44779165588879504
a1 =    699266283035359788689002485914571600271382111380710376847895
            2287632180176739
```

The key share values for the participants are

```
xa =    1
ya =    294476425608857249929830691829039493762190980430747893160091
            437085043550309

xb =    2
yb =    501336786301929228466689879317612556188957348579440556370927
            86431769476059

xc =    3
yc =    704279650898379080973669384707747725833271684866504782411604
            5074063949652798
```

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

The Lagrange coefficients are:

```
la =    361850278866613110698659328152149712042855817968995380300097
            5469142727125496
lc =    361850278866613110698659328152149712042855817968995380300097
            5469142727125494
```

Alice and Carol select their values ra, rc

```
ra =    456116926701492705315133938623040527696276882295617965847376
            7682545245216294
Ra =
  D4 45 96 7B  EC 72 EF EB  CE 64 45 4B  F1 04 BE 89
  82 76 38 A9  C7 CD 49 D5  AC 89 89 15  A1 2C F9 ED

rc =    482074679100753533345731495679776832764315286485235535312553
            5253541881347149
Rc =
  84 2F BA 3B  E3 BB 6B FD  1E A7 4A 9A  F7 69 CB F2
  42 E0 40 37  72 CB 44 76  91 F3 78 4C  38 6A 55 70
```

The composite value $R = R_a + R_c$

```
R =
  86 D3 74 FB  11 A5 B0 02  0E C8 D8 47  81 F6 D3 0B
  2F 98 1A 78  A4 B6 29 8E  CF 8F 1F BA  C6 DF 9C CE
```

The value k is

```
k =     1085717265856137458707104721211825439059660721763252401194296512368686397102
```

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 =    4060217427079416981881339319265056361071844650336075642741112624770292450958
sc =    3715607322840366809104839639504255611690757935047383693943928401253479424590
```

The signature contributions can now be calulated:

```
Sa =    3928954189689632035122668360462914023178183698289426892984772946200577969243
Sc =    2537522373324191456496014333212082192353479531853952253693561893073151349028
```

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

```
S =     6466476563013823491618682693674996215531663230143379146678334839273729318271
```

The dealer checks to see that the signature verifies:

```
S.B = R + kA =
    X:              22642771465710202002560483838014829063703190202361838906492538114789522304796
    X:              10613093543154701158645711080916412421174392144729537260912052744073378658652
```

### 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: 723088510822916843359337925516642493307623385482
113107480846498794254549074097051759295396782499503452909258
97846850655305536698954745 6

Encoded Private

59 DC 8A 5F 5E AF 8C FA 96 19 F8 EE 78 13 00 12
33 0E 12 80 2D 25 E6 EF E8 E2 56 B5 83 6A 0C CF
DC 11 96 A5 A5 D1 39 AA 34 25 0B 52 ED 9F 38 92
5D 9F 7B BC B9 BC 86 45

X: 600163199260212879671026282440221570752543874569
276531213297382365938924845597497264583528185273760383031589
25167107013312482098672476

Y: 568007995844826855892481230051783440873263817862
016100095069663100696528804467952219402043387612562057320585
56186506804622665544312258 2

Encoded Public

ED C3 90 99 38 0B 8F CD 60 29 24 04 6C DE 52 33
A2 07 3E 56 8D 27 B5 B9 21 60 CF E9 E7 9D D6 4A
11 47 20 E6 9D FE 75 C7 04 14 70 18 B4 52 10 83
D0 EC 98 BD F5 E6 E3 D5 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 = 723088510822916843359337925516642493307623385482113107480846
498794254549074097051759295396782499503452909258978468506553055
3669895474 56

a1 = 165663618071837435927824367225611232537435726800694979220111
025223864535406082007744102129685821489679922265075700364540659
70071315644 4

The key share values for the participants are

```
xa =    1
ya =    161913404599147388737838484854249191491417751595490026419467
            324837535068634020716638614505301551489279590349217802222
            1874620044264104784

xb =    2
yb =    145867341597083102028331900107859290440443138224355490569205
            800266253600078563138666520879695680602996202320584410922
            4110505989117611449

xc =    3
yc =    129821278595018815318825315361469389389468524853220954718944
            275694972131523105560694427254089809716712814291951019622
            6346391933971118114
```

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

The Lagrange coefficients are:

```
la =    908548405369508613186654759860005667942051700859147575351862
            748975730019807697928580978776458461879816551468545458311
            152386877929824891
lc =    908548405369508613186654759860005667942051700859147575351862
            748975730019807697928580978776458461879816551468545458311
            152386877929824889
```

Alice and Carol select their values ra, rc

```
ra =    103517366944050550717591081348710241163469949228538856371118
            478339700605492489484997392756754471608330724190410413471
            1187333803802632789
Ra =
  86 8B B6 BF  E1 FA 18 BB  5A D6 79 D2  6F 60 E9 7A
  B9 76 58 AA  96 3B 5E FD  83 E7 79 09  53 A2 AE 7B
  89 C6 30 72  31 13 C3 97  9D 0C 75 BB  F2 DC 87 72
  46 CD F8 BF  6F 08 27 FF  00

rc =    129544664690317775866810455605532383977152960638027121327286
            304624135059473711611035445709263959068338756559785616511
            9513959988103594270
Rc =
  D6 0E 7A 4B  C1 D1 A4 A4  09 A9 4E 2C  0C 11 E8 31
  E3 F7 0D C0  AD 7E 90 6D  53 63 6B D0  D0 5D 5F BD
  44 34 4F B9  1D 5C 05 7B  A8 52 5D 39  00 8B 47 30
  46 15 B7 39  00 35 A6 8D  00
```

The composite value $R = R_a + R_c$

```
R =
  84 76 AE 71  96 E4 5B 2C  32 7A CE 8C  62 4E C5 C7
  56 90 58 7B  46 C1 99 87  95 72 E0 39  14 59 50 3A
  53 63 60 8A  2B 14 DD C2  99 AF 57 5D  7F 28 6C DC
  73 4E 72 6A  0A 67 B9 F0  80
```

The value k is

```
k =     84671549231286167578687763742759373166446032444366791613772 4
            03306404008461876775118415847980023413626823976377915241
            076018146143898434
```

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 =    61160425824819360469426775309372653648716287221405524558828 4
            37461156598989491489241726002660634857956075230117611670
            507156310536507397
sc =    11679904177639231497791829429126643889367607774521903771090 0
            41194765993819998680536898212824678751760690314773358184
            9131577788874090722
```

The signature contributions can now be calulated:

```
Sa =    67439849117239258231539144796184259291662766614171153373298 6
            40302585673169645319736684479821824290746431281389532198
            884328704969312923
Sc =    32350102109480081620830217218772278945098539110879011931614 6
            83012739259001534721728765613699056985895227578447593375
            275645256932691561
```

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

```
S =     99789951226719339852369362014956538236761305725050165304913 3
            23315324932171180041465450093520881276641658859837125574
            159973961902004484
```

The dealer checks to see that the signature verifies:

```
S.B = R + kA =
    X:              310176585478125150718252258963045651393161473743
        314418941766523710610402459 8
    X:              546975372341826393522134872750971962955374107574
        98782836263975525610781195547
```

## 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-11, 2 November 2020, <https://tools.ietf.org/html/draft-hallambaker-mesh-udf-11>.

**[draft-hallambaker-threshold]**
Hallam-Baker, P., "Threshold Modes in Elliptic Curves", Work in Progress, Internet-Draft, draft-hallambaker-threshold-04, 2 November 2020, <https://tools.ietf.org/html/draft-hallambaker-threshold-04>.

**[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-10, 27 July 2020, <https://tools.ietf.org/html/draft-hallambaker-mesh-developer-10>.

**[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.