Threshold Signatures in Elliptic Curves

Threshold Signatures in Elliptic Curves

<series>draft-hallambaker-threshold-sigs

<status>informational

<stream>independent

<ipr>trust200902

<author>Phillip Hallam-Baker

<surname>Hallam-Baker

<initials>P. M.

<firstname>Phillip

<email>phill@hallambaker.com

<organization>ThresholdSecrets.com

<keyword>Threshold Cryptography

<keyword>Elliptic Curve

<keyword>Threshold Signature

<keyword>Digital Signature

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.

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

# 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 <info="draft-hallambaker-threshold"/> to define a scheme that creates threshold signatures that are computationally indistinguishable from those produced according to the algorithm specified in <norm="RFC8032"/>. The scheme does not support the creation of aggregate signatures.

The approach used is based on that developed in FROST <info="Komlo"/>. This document describes the signature scheme itself. The techniques used to generate keys are described separately in <info="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 <info="Shamir79"/> and Lagrange coefficients for the case that *n* > *t*.

## 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.

### 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).

### 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.

### 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.

# 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.

## 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 <norm="RFC2119"/>.

## Defined Terms

See <norm="draft-hallambaker-threshold"/>.

## Related Specifications

This document extends the approach described in <norm="draft-hallambaker-threshold"/> to support threshold signatures. The deterministic mechanism described in specification <norm="draft-hallambaker-mesh-udf"/> is used to generate the private keys used in the test vectors.

## Implementation Status

The implementation status of the reference code base is described in the companion document <info="draft-hallambaker-mesh-developer"/>.

# 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 <norm="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 <norm="RFC8032"/> consists of a pair of values *S*, *R* calculated as follows:

*R* = *r.B*

*S* = *r* + *k.s* mod *L*

Where

*B* is the base point of the elliptic curve.

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

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

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

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

*s* is the secret scalar value

*L* is the order of the elliptic curve group.

To verify the signature, the verifier checks that:

*S.B* = *R* + *k.A*

This equality must hold for a valid signature since:

*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 mod *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

*S1* = *r* + *k1*.*s* mod *L*

*S2* = *r* + *k2.s* mod *L*

*s* = (*S1* - *S2*)(*k1* - *k2*)-1 mod *L*

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

## 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:

Ai = si.B

Ri = ri.B

Si = ri + k.si mod L

Where si and ri are the secret scalar and unguessable value for the individual signer.

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

R = R1 + R2 + … + Rn

S = S1 + S2 + … + Sn

A = s.B

Where s = (s1 + s2 + … + sn) mod L

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

S.B = R + k.A

Substituting for S.B we get:

= (S1 + S2 + … + Sn).B

= S1.B + S2.B + … + Sn.B

= (r1 + k.s1).B + (r2 + k.s2).B + … + (rn + k.sn).B

= (r1.B + k.s1.B) + (r2.B + k.s2.B) + … + (rn.B + k.sn.B)

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

Substituting for R + k.A we get:

= R1 + R2 + … + Rn + k.(A1 + A2 + … + An)

= R1 + R2 + … + Rn + k.A1 + k.A2 + … + k.An

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

As expected, the operation of threshold signature makes use of the same approach as threshold key generation and threshold decryption as described in <norm="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 <norm="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 Si can be calculated, it is thus necessary to calculate the values Ri and Si 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:

1. The dealer determines the values F, C and PH(M) as specified in <norm="RFC8032"/> and transmits them to the signers {1, 2, … n}.
2. Each signer generates a random value ri such that 1 < ri < L, calculates the value Ri = ri.B and returns R to the dealer .
3. The dealer calculates the value R = R1 + R2 + … + Rn and transmits R and A to the signers {1, 2, … n}.
4. Each signer uses the suppled data to determine the value k and hence Si = ri + k.si mod L and transmits it to the dealer .
5. The dealer calculates the value S = S1 + S2 + … + Sn and verifies that the resulting signature R, S verifies according to the mechanism specified in <norm="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 Ri and Si for each.

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

## Shamir shared threshold signature

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

Ai = sili.B

Ri = ri.B

*Si* = *ri* + *klisi* mod *L*

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

*Si* = *ri* + *kcisi* mod *L*

Where

*ci* = 1 for the additive shared threshold signature

*ci* = *li* for the Shamir shared threshold signature

## 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 *ri*. 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:

1. All the signers except the final signer *F* generate their value *ri* and submit the corresponding value *Ri* to the dealer
2. Dealer calculates the value *R* - *RF* 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 *cF*.
3. The final signer generates its value *rF*
4. The final signer calculates the value *RF* from which the values *R* and *k* can now be determined.
5. The final signer calculates its key share contribution *SF* = *rF* + *kcFsF* mod *L*.
6. The final signer returns the values *SF* and *R* to the dealer.
7. 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 *rF* deterministically provided that the parameters *R* - *RF* and *cF* are used in its determination. Other signers MUST NOT use a deterministic means of generating their value *ri* since the information known to them at the time this parameter is generated is not sufficient to fix the value of *R*.

### Side channel resistance

The use of Kocher side channel resistance as described in <info="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:

1. A generates a random nonce *nA* and sends it to B with the other parameters required to generate the signature.
2. B generates a random nonce *nB*
3. B calculates the blinding factor *x* by calculating *H*(*nA, nB*) where *H* is a strong cryptographic digest function and converting the result to an integer in the range 1 < *x* < *L*.
4. B calculates the signature parameters as before except that the threshold signature contribution is now *SB* = *rB* + *k*(*cBsB* + *x*) mod *L*.
5. B returns the nonce *nB* to A with the other parameters.
6. A calculates the blinding factor *x* using the same approach as B
7. A calculates the signature parameters as before except that the threshold signature contribution is now *SA* = *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. <info="RFC5860"/>) for the digest function.

## 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.

### Calculation of r values<anchor="rvalue">

The method of constructing the values *ri* MUST ensure that each is unique and unguessable both to external parties, the signers and the dealer. The deterministic method specified in <norm="RFC8032"/> cannot be applied to generation of the values ri 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 ri 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 ri that appear to be sound, closer inspection has demonstrated these to be vulnerable to rogue key and rogue contribution attacks.

### Replay Attack

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

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

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 ri within the device itself using an appropriately authenticated and encrypted opaque state token. Such mechanisms provide the HSM with the value of ri 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.

### Malicious Contribution Attack

In a malicious contribution attack, one or more parties present a signature contribution that does not meet the criteria Ri = ri.B and Si = ri + ksi.

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

### Rogue Key Attack<anchor="roguekey"/>

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 = A1 + A2 + … + An

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 An = Am - (A1 + A2 + … + An-1)

The attacker can thus gain control of the aggregate signature key by choosing Am = sm.B where sm is a secret scalar known only to the attacker. But does so at the cost of not knowing the value sn 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 A1 + A2 + … + An 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.

# Ed2519 Signature

The means by which threshold shares are created is described in <info="draft-hallambaker-threshold"/>.

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

1. Randomly generate an integer ri such that 1 < ri < L.
2. Compute the point Ri = riB. For efficiency, do this by first reducing ri modulo L, the group order of B. Let the string Ri be the encoding of this point.
3. Transmit the value Ri to the dealer

At some later point, the dealer MAY complete the signature by returning the values F, C, A and R as specified in <norm="RFC8032"/> together with the key multiplier coefficient ci. The signers MAY then complete their signature contributions:

1. Compute SHA512(dom2(F, C) || R || A || PH(M)), and interpret the 64-octet digest as a little-endian integer k.
2. Compute Si = (ri + kcisi) mod L. For efficiency, again reduce k modulo L first.
3. Return the values Ri, Si to the dealer .

The dealer then completes the signature by:

1. Computing the composite value S = S1 + S2 + … + Sn
2. Verifying that the signature R, S is valid.
3. Publishing the signature.

# Ed448 Signature

The means by which threshold shares are created is described in <info="draft-hallambaker-threshold"/>.

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

1. Randomly generate an integer ri such that 1 < ri < L.
2. Compute the point Ri = riB. For efficiency, do this by first reducing ri modulo L, the group order of B. Let the string Ri be the encoding of this point.
3. Transmit the value Ri to the dealer

At some later point, the dealer MAY complete the signature by returning the values F, C, A and R as specified in <norm="RFC8032"/> together with the key multiplier coefficient ci. 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 Si = (ri + kcisi) mod L. For efficiency, again reduce k modulo L first.
3. Return the values Ri, Si to the dealer.

The dealer then completes the signature by:

1. Computing the composite value S = S1 + S2 + … + Sn
2. Verifying that the signature R, S is valid.
3. Publishing the signature.

# Test Vectors

<include=..\Examples\ExamplesThresholdSig.md>

# Security Considerations

All the security considerations of <norm="RFC7748"/>, <norm="RFC8032"/> and <info="draft-hallambaker-threshold"/> apply and are hereby incorporated by reference.

## Rogue Key attack

The rogue key attack described in <info="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*.

## 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 <norm="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 *ri* to compute threshold signature contributions. Accordingly the requirements of <norm="RFC4086"/> regarding requirements for randomness MUST be observed.

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

## Resource exhaustion attack

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

## Signature Uniqueness

Signatures generated in strict conformance with <norm="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 <norm="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.

# IANA Considerations

This document requires no IANA actions.

# Acknowledgements

[TBS]