Threshold Cryptography in Elliptic Curves

Threshold Cryptography in Elliptic Curves

<series>draft-hallambaker-threshold

<status>informational

<stream>independent

<ipr>trust200902

<author>Phillip Hallam-Baker

<surname>Hallam-Baker

<initials>P. M.

<firstname>Phillip

<email>phill@hallambaker.com

<organization>Venture Cryptography.

<keyword>Threshold Cryptography

<keyword>Elliptic Curve

<keyword>Threshold Encryption

<keyword>Threshold Key Generation

<keyword>Side Channel Resistance

Threshold cryptography schemes are described with application to the Ed25519, Ed448, X25519 and X448 Elliptic Curves. Threshold key generation allows generation of keypairs to be divided between two or more parties with verifiable security guaranties. Threshold decryption allows elliptic curve key agreement to be divided between two or more parties such that all the parties must co-operate to complete a private key agreement operation. The same primitives may be applied to improve resistance to side channel attacks.

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

Public key cryptography provides greater functionality than symmetric key cryptography by introducing separate keys for separate roles. Knowledge of the public encryption key does not provide the ability to decrypt. Knowledge of the public signature verification key does not provide the ability to sign. Threshold cryptography extends the scope of traditional public key cryptography with further separation of roles.

This document describes the implementation of threshold key generation and threshold decryption to .

Threshold Key Generation

Threshold Decryption

Threshold Key Agreement

Multiple Signatures

Threshold Signatures

This document describes the implementation of threshold key generation and threshold decryption for the Elliptic Curve Diffie Hellman schemes described in <norm="RFC8032"/> and <norm="RFC7748"/>.

# 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

TBS.

## Related Specifications

TBS.

## Implementation Status

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

# Principles

The threshold cryptography mechanisms described in this specification are made possible by the fact that Diffie Hellman key agreement and elliptic curve variants thereof support properties we call the Key Combination Law and the Result Combination Law.

Let {*X*, *x*}, {*Y*, *y*}, {*A*, *a*} be {public, private} key pairs and r [.] S represent the Diffie Hellman operation applying the private key r to the public key S.

The Key Combination law states that we can define an operator [x] such that there is a keypair {*Z*, *z*} such that:

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

The Result Combination Law states that we can define an operator [+] such that:

(*x* [.] *A*) [+] (*y* [.] *A*) = (*z* [.] *A*) = (*a* [.] *Z*).

## Application to Diffie Hellman (not normative)

The key combination and result combination laws are concisely demonstrated for Diffie Hellman in a modular field <info="RFC2631"/>. The realization of these laws is outside the scope of this document.

For the Diffie Hellman system in a modular field p, with exponent e:

* r [.] S = Sr mod p
* o = p-1
* *A* [x] *B* = *A* [.] *B* = *AB* mod *p*.

*Proof:*

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

Therefore,

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

Moreover, A = ea mod p,

Therefore,

(Ax mod p).(Ay mod p) = (AxAy) mod p) = (Ax+y) mod p) = Az mod p

= eaz mod p = (ez)a mod p = Za mod p

## Multi-Party Key Generation

The Key Combination Law provides the basis for Key Co-Generation. This technique allows party A to ensure that a private key used by party B incorporates a provided contribution without gaining any additional information on that private key.

and has not been compromised by malware or other 'backdoor' compromise to the machine during or after manufacture.

For the Diffie Hellman system, the Key Combination law provides all the mechanism needed to implement a Key Co-Generation mechanism. If the Device key is {*X*, *x*}, the administration device can generate a Co-Generation Key Pair {*Y*, *y*} and generate a Device Connection Assertion for the final public key E calculated from knowledge of X and Y alone. Passing the value *y* to the device (using a secure channel) allows it to calculate the corresponding private key *e* required to make use of the Device Connection Assertion.

This approach ensures that a party with knowledge of either *x* or *y* but not both obtains no knowledge of *z*.

## Multi-Party Decryption

The Result Combination Law provides the basis for Multi-Party Decryption.

## Mutually Authenticated Key Exchange.

The Key Combination Law provides the basis

The Result Combination Law is used to provide a Key Exchange mechanism that provides mutual authentication of the parties while preserving forward secrecy.

# Application to Elliptic Curves

For elliptic curve cryptosystems, the operators [x] and [.] are point addition.

Implementing a robust Key Co-Generation for the Elliptic Curve Cryptography schemes described in <norm="RFC7748"/> and <norm="RFC8032"/> requires some additional considerations to be addressed.

* The secret scalar used in the EdDSA algorithm is calculated from the private key using a digest function. It is therefore necessary to specify the Key Co-Generation mechanism by reference to operations on the secret scalar values rather than operations on the private keys.
* The Montgomery Ladder traditionally used to perform X25519 and X448 point multiplication does not require implementation of a function to add two arbitrary points. While the steps required to create such a function are fully constrained by <norm="RFC7748"/>, the means of performing point addition is not.

## Implementation for Ed25519 and Ed448

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

Note that the 'private key' described in <norm="RFC8032"/> is in fact a seed used to generate a 'secret scalar' value that is the value that has the function of being the private key in the ECDH algorithm.

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

1. Obtains the device profile of the device(s) to be provisioned to determine the type of key to perform co-generation for. Let the device {public, private} key be {D, d}.
2. Generates a private key *m* with the specified number of bytes (32 or 57].
3. Calculates the corresponding public key *M*.
4. Calculates the Application public key A = D+M where + is point addition.
5. Constructs the application device entry containing the private key value m and encrypts under the device encryption key d.

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

## Implementation for X25519 and X448

While the point addition function can be defined for any elliptic curve system, it is not necessary to implement point addition to support ECDH key agreement.

In particular, point multiplication using the Montgomery ladder technique over Montgomery curves only operate on the u co-ordinate and only require point doubling operations.

The notation of <norm="RFC7748"> is followed using {u, v} to represent the coordinates on the Montgomery curve and {x, y} for coordinates on the corresponding Edwards curve.

### Coordinate Recovery

The relationship between the u and v coordinates is specified by the Montgomery Curve formula itself:

v2 = u3 + Au2 + u

Since v2 has a positive (v) and a negative solution (-v), it follows that v2 mod p will have the solutions v, p-v. And furthermore if p is a large prime it must be an odd number. Consequently, if v is odd, p-v must be even and vice versa. It is thus sufficient to record whether v is odd or even to enable recovery of the v coordinate from u.

The Montgomery curve is constructed so that the coordinate u2 of P2 = {u2, v2} = k. {u1, v1} can be calculated using only the value u1. Since the curve is symmetric about the u axis, U(k. {u1, v1}) = U(k. {u1, -v1}). But to recover the correct value of v2 it is of course essential to begin with the correct value of v1.

### Point Addition

The point addition formula for the Montgomery curve is defined as follows:

Let P1 = {u1, v1}, P2 = {u2, v2}, P3 = {u3, v3} = P1 + P2

By definition:

u3 = B(v2 - v1)2 / (u2 - u1)2 - A - u1 - u2

= B((u2v1 - u1v2)2 ) / u1u2 (u2 - u1)2

v3 = ((2u1 + u2 + A)(v2 - v1) / (u2 - u1)) - B (v2 - v1)3 / (u2 -u1)3 - v1

For curves X25519 and X448, B = 1 and so:

u3 = ((v2 - v1).(u2 - u1)-1)2 - A - u1 - u2

v3 = ((2u1 + u2 + A)(v2 - v1).(u2 - u1)-1) - ((v2 - v1).(u2 -u1)-1)3 - v1

This may be implemented using the following code:

<CODE BEGINS>

B = v2 - v1

C = u2 - u1

CINV = C^(p - 2)

D = B \* CINV

DD = D \* D

DDD = DD \* D

u3 = DD - A - u1 - u2

v3 = ((u1 + u1 + u2 + A) \* B \* CINV) - DDD - v1

<CODE ENDS>

Performing point addition thus requires that we have sufficient knowledge of the values v1, v2. At minimum whether one is odd and the other even or if both are the same.

### Montgomery Ladder with Coordinate Recovery

As originally described, the Montgomery Ladder only provides the u coordinate as output. López and Dahab <info="Lopez"> provided a formula for recovery of the v coordinate of the result for curves over binary fields. This result was then extended by Okeya and Sakurai <info="Okeya"> to prime field Montgomery curves such as X25519 and X448. The realization of this result described by Costello and Smith <info="Costello"> is applied here.

The scalar multiplication function specified in <norm="RFC7748"> takes as input the scalar value k and the coordinate u1 of the point P1 = {u1, v1} to be multiplied. The return value in this case is u2 where P2 = {u2, v2} = k.P1.

To recover the coordinate v2 we require the values x\_2, z\_2, x\_3, z\_3 calculated in the penultimate step:

<CODE BEGINS>

x\_1 = u

x\_2 = 1

z\_2 = 0

x\_3 = u

z\_3 = 1

swap = 0

For t = bits-1 down to 0:

k\_t = (k >> t) & 1

swap ^= k\_t

// Conditional swap as specified in RFC 7748

(x\_2, x\_3) = cswap(swap, x\_2, x\_3)

(z\_2, z\_3) = cswap(swap, z\_2, z\_3)

swap = k\_t

A = x\_2 + z\_2

AA = A^2

B = x\_2 - z\_2

BB = B^2

E = AA - BB

C = x\_3 + z\_3

D = x\_3 - z\_3

DA = D \* A

CB = C \* B

x\_3 = (DA + CB)^2

z\_3 = x\_1 \* (DA - CB)^2

x\_2 = AA \* BB

z\_2 = E \* (AA + a24 \* E)

(x\_2, x\_3) = cswap(swap, x\_2, x\_3)

(z\_2, z\_3) = cswap(swap, z\_2, z\_3)

Return x\_2, z\_2, x\_3, z\_3

<CODE ENDS>

The values x\_2, z\_2 give the projective form of the u coordinate of the point P2 = {u2, v2} = k.P1 and the values x\_3, z\_3 give the projective form of the u coordinate of the point P3 = {u3, v3} = (k+1).P1 = P1 + k.P1 = P1 + P2.

Given the coordinates {u1, v1} of the point P1 and the u coordinates of the points P2, P1 + P2, the coordinate v2 MAY be recovered by trial and error as follows:

<CODE BEGINS>

v\_test = SQRT (u3 + Au2 + u)

u\_test = ADD\_X (u, v, u\_2, v\_test)

if (u\_test == u\_3)

return u\_test

else

return u\_test +p

<CODE ENDS>

Alternatively, the following MAY be used to recover {u2, v2} without the need to extract the square root and using a single modular exponentiation operation to convert from the projective coordinates used in the calculation. As with the Montgomery ladder algorithm above, the expression has been modified to be consistent with the approach used in <norm="RFC7748"> but any correct formula may be used.

<CODE BEGINS>

x\_p = u

y\_p = v

B = x\_p \* z\_2 //v1

C = x\_2 + B //v2

D = X\_2 - B // v3

DD = D^2 // v3

E = DD. X\_3 // v3

F = 2 \* A \* z\_2 // v1

G = C + F // v2

H = x\_p \* x\_2 // v4

I = H + z\_2 // v4

J = G \* I // v2

K = F \* z\_2 //v1

L = J - K //v2

M = L \* z\_3 //v2

yy\_2 = M - E // Y'

N = 2 \* y\_p // v1

O = N \* z\_2 // v1

P = O \* z\_3 // v1

xx\_2 = P \* x\_q // X'

zz\_2 = P \* z\_ q // Z'

ZINV = (zz\_2^(p - 2))

u2 = xx\_2 \* ZINV

v2 = yy\_2 \* ZINV

return u2, v2

<CODE ENDS>

# Threshold Key Generation

Threshold Key Generation is a capability that is used in the Mesh to enable provisioning of application specific private key pairs to connected devices without revealing any information concerning the application private key of the device.

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

Threshold Key Generation allows an administration device to provision a connected device with an application specific private key that is specific to that application and no other such that the application can determine the public key of the device but has no knowledge of the private key.

Provisioning an application private key to a device requires the administration device to:

* Generate a new application public key for the device.
* Construct and publish whatever application specific credentials required for use of the device with the application.
* Providing the information required to make use of the private key to the device.

Note that while the administration device needs to know the device application public key, it does not require knowledge of the device application private key.

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

# Threshold Decryption

A key limitation of most deployed messaging systems is that true end-to-end confidentiality is only achieved for a limited set of communication patterns. Specifically, bilateral communications (Alice sends a message to Bob) or broadcast communications to a known set of recipients (Alice sends a message to Bob, Carol and Doug). These capabilities do not support communication patterns where the set of recipients changes over time or is confidential. Yet such requirements commonly occur in situations such as sending a message to a mailing list whose membership isn’t known to the sender, or creating a spreadsheet whose readership is to be limited to authorized members of the ‘accounting’ team.

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

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

Proxy re-encryption provides a technical capability that meets the needs of such communication patterns. Conventional symmetric key cryptography uses a single key to encrypt and decrypt data. Public key cryptography uses two keys, the key used to encrypt data is separate from the key used to decrypt. Proxy re-encryption introduces a third key (the recryption key) that allows a party to permit an encrypted data packet to be decrypted using a different key without permitting the data to be decrypted.

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

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

Administrator

Holder of Decryption Key, Creator of Recryption Keys

Sender

Holder of Encryption Key

Recryption Service

Holder of Recryption keys

Receiver

Holder of personal decryption key

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

## Mechanism

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

Recall that the key combination law for Diffie Hellman crypto-systems states that we can add two private keys to get a third. It follows that we can split the private key portion of a keypair {*G*, *g*} into two parts by choosing a random number that is less than the order of the Diffie-Hellman group to be our first key *x*. Our second key is *y* = *g* - *r* mod *o*, where *o* is the order of the group.

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

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

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

## Implementation

Implementation of recryption in the Mesh has four parts:

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

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

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

* Administration of the recryption group may be spread across multiple Mesh users or transferred from one user to another without requiring specification of a separate management protocol to support these operations.
* The recryption account address can be used by Mesh applications such as group messaging, conferencing, etc. as a contact address.
* The contact request service can be used to notify members that they have been granted membership in the group.

### Group Creation

Creation of a Recryption group requires the steps of:

* Generating the recryption group key pair
* Creating the recryption group account
* Generating administrator record for each administrator.
* Publishing the administrator records to the recryption catalog.

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

### Provisioning a Member

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

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

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

### Message Encryption and Decryption

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

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

* The recipient entry to be used for decryption
* The fingerprint of the decryption key(s) the device would like to make use of.
* Whether or not the encrypted decryption key entry should be returned.

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

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

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

# Mutually Authenticated Key Agreement

Diffie Hellman key agreement using the authenticated public keys of the principals provides mutual authentication of those principals.

For example, if Alice's key pair is {*a*, *A*} and Bob's key pair is {*b*, *B*}, the Diffie Hellman key agreement value *DH* (*a*, *B*) = *DH* (*b*, *A*) can only be generated from the public information if *a* or *b* is known.

The chief disadvantage of this approach is that it only allows Alice and Bob to establish a single shared secret that will never vary and does not provide forward secrecy. To avoid this, cryptographic protocols usually perform the key agreement against an ephemeral key and either accept that the client key is not authenticated or perform multiple key agreements and combine the results.

Using the Result Combination Law allows a key agreement mechanism to combine the benefits of mutual authentication with the use of ephemeral keys without the need for multiple private key operations or additional round trips.

In its simplest form, the key exchange has two parties which we refer to as the client and the server. The client being the party that initiates the protocol exchange and the server being the party that responds. Let the public key pair of the client be {*a*, *A*} and that of the server {*b*, *B*}.

Two versions of the key agreement mechanism are specified:

Client ephemeral

The client contributes an ephemeral key pair {*nA*, *NA*}. The effective public key of the client is *A* ⊗ *NA*.

The server uses its public key *B*.

The key agreement value is *DH* (*a* + *nA*, B) = *DH* (*b*, *A* ⊗ *NA*)

Dual ephemeral

The client contributes an ephemeral key pair {*nA*, *NA*}. The effective public key of the client is *A* ⊗ *NA*.

The server contributes an ephemeral key pair {*nB*, *NB*}. The effective public key of the client is *B* ⊗ *NB*.

The key agreement value is *DH* (*a* + *nA*, *B* ⊗ *NB*) = *DH* (*b* + *nB*, *A* ⊗ *NA*)

The function of the ephemeral key is effectively that of a nonce but it is shared with the counter-party as a public key value.

The dual ephemeral approach has the advantage that it limits the scope for side channel attacks as both sides have contributed unknown information to the key agreement value. The disadvantage of this approach is that the key agreement value can only be calculated after the server has provided its ephemeral.

Implementations MAY take advantage of the result combination law to enable private key operations involving the authenticated key (or a contribution to it) to be performed in trustworthy hardware.

An advantage of this key exchange mechanism over the traditional TLS key exchange approach is that no signature operation is involved, thus ensuring that either party can repudiate the exchange and thus the claim that they were in communication.

The master secret is calculated from the key agreement value in the usual fashion. For ECDH algorithms, this comprises the steps of converting the key agreement value to an octet string which forms the input to a Key Derivation Function.

# Security Considerations

# IANA Considerations

This document requires no IANA actions.

# Acknowledgements

.

# Examples

## Key Combination

### Ed25519

### Ed448

### X25519

### X448

## Group Encryption

### X25519

### X448

# References (to process)

Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, Advances in Cryptology - CRYPTO ’96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, volume 1109 of Lecture Notes in Computer Science, pages 104–113. Springer, 1996

Julio L´opez and Ricardo Dahab. Fast multiplication on elliptic curves over GF(2m) without precomputation. In C¸ etin Kaya Ko¸c and Christof Paar, editors, Cryptographic Hardware and Embedded Systems, First International Workshop, CHES’99, Worcester, MA, USA, August 12-13, 1999, Proceedings, volume 1717 of Lecture Notes in Computer Science, pages 316–327. Springer, 1999.

Katsuyuki Okeya and Kouichi Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomeryform elliptic curve. In C¸ etin Kaya Ko¸c, David Naccache, and Christof Paar, editors, Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings, volume 2162 of Lecture Notes in Computer Science, pages 126–141. Springer, 2001.