

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: June 11, 2016

I. Liusvaara  
Independent  
December 9, 2015

CFRG curves and signatures in JOSE  
draft-liusvaara-jose-cfrg-curves-00

## Abstract

This document defines how to use curves and algorithms from IRTF CFRG elliptic curves work (Diffie-Hellman and signatures) in JOSE.

## 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 <http://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 June 11, 2016.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">1.1.</a>	<a href="#">Requirements Terminology</a>	<a href="#">2</a>
<a href="#">1.2.</a>	<a href="#">Notation</a>	<a href="#">3</a>
<a href="#">2.</a>	<a href="#">Key type 'OKP'</a>	<a href="#">3</a>
<a href="#">3.</a>	<a href="#">Algorithms</a>	<a href="#">3</a>
<a href="#">3.1.</a>	<a href="#">Signatures</a>	<a href="#">3</a>
<a href="#">3.1.1.</a>	<a href="#">Algorithms</a>	<a href="#">3</a>
<a href="#">3.1.2.</a>	<a href="#">Signing</a>	<a href="#">4</a>
<a href="#">3.1.3.</a>	<a href="#">Verification</a>	<a href="#">4</a>
<a href="#">3.2.</a>	<a href="#">ECDH-ES</a>	<a href="#">4</a>
<a href="#">3.2.1.</a>	<a href="#">Performing the ECDH operation</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">Security considerations</a>	<a href="#">5</a>
<a href="#">5.</a>	<a href="#">Acknowledgements</a>	<a href="#">5</a>
<a href="#">6.</a>	<a href="#">IANA considerations</a>	<a href="#">5</a>
<a href="#">7.</a>	<a href="#">References</a>	<a href="#">8</a>
<a href="#">7.1.</a>	<a href="#">Normative References</a>	<a href="#">8</a>
<a href="#">7.2.</a>	<a href="#">Informative References</a>	<a href="#">8</a>
<a href="#">Appendix A.</a>	<a href="#">Examples</a>	<a href="#">8</a>
<a href="#">A.1.</a>	<a href="#">Ed25519 private key</a>	<a href="#">9</a>
<a href="#">A.2.</a>	<a href="#">Ed25519 public key</a>	<a href="#">9</a>
<a href="#">A.3.</a>	<a href="#">JWK thumbprint canonicalization</a>	<a href="#">9</a>
<a href="#">A.4.</a>	<a href="#">Ed25519 Signing</a>	<a href="#">9</a>
<a href="#">A.5.</a>	<a href="#">Ed25519 Validation</a>	<a href="#">10</a>
<a href="#">A.6.</a>	<a href="#">ECDH-ES with X25519</a>	<a href="#">11</a>
<a href="#">A.7.</a>	<a href="#">ECDH-ES with X448</a>	<a href="#">12</a>
	<a href="#">Author's Address</a>	<a href="#">13</a>

[1.](#) Introduction

Internet Research Task Force (IRTF) Crypto Forum Research Group (CFRG) selected new elliptic curves and signature algorithms for asymmetric key cryptography. This document defines how those curves and algorithms are to be used in JOSE in interoperable manner.

This extends [\[RFC7517\]](#) and [\[RFC7518\]](#)

[1.1.](#) Requirements Terminology

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 [\[RFC2119\]](#).

## [1.2.](#) Notation

All inputs to and outputs from the the ECDH and signature functions are defined to be octet strings, with the exception of output of verification function, which is a boolean.

## [2.](#) Key type 'OKP'

A new key type (kty) value "OKP" (Octet Key Pair) is defined for public key algorithms that use octet strings as private and public keys. It has the following parameters:

- o The parameter "kty" MUST be "OKP".
- o The parameter "crv" MUST be present, and contain the subtype of the key (from "JSON Web Elliptic curve" registry).
- o The parameter "x" MUST be present, and contain the public key encoded using base64url [[RFC4648](#)] encoding.
- o The parameter "d" MUST be present for private keys, and contain the private key encoded using base64url encoding. This parameter MUST NOT be present for public keys.

Note: Do not assume that there is an underlying elliptic curve, despite the existence of the "crv" and "x" parameters.

When calculating thumbprints [[RFC7638](#)], the three public key fields are included in the hash. That is, in lexographic order: "crv", "kty" and "x".

## [3.](#) Algorithms

### [3.1.](#) Signatures

#### [3.1.1.](#) Algorithms

The following signature algorithms are defined here (to be applied as values of "alg" parameter). All these have keys with subtype of the same name:

alg value:	subtype:	The algorithm:
Ed25519	Ed25519	Ed25519
Ed25519ph	Ed25519ph	Ed25519ph
Ed448	Ed448	Ed448
Ed448ph	Ed448ph	Ed448ph

The key type for these keys is "OKP" and key subtype for these algorithms MUST be the same as the algorithm name.

The keys of these subtypes MUST NOT be used for ECDH-ES.

[TBD: Merge the alg values into a single one that can perform signing with any signature-capable OKP subtype? That would remove a source of possible errors, since then the message and key could not mismatch in algorithm.]

### [3.1.2.](#) Signing

Signing for these is preformed by applying the signing algorithm defined in [[I-D.irtf-cfrg-eddsa](#)] to the private key (as private key), public key (as public key) and the JWS Signing Input (as message). The resulting signature is the JWS Signature value. All inputs and outputs are octet strings.

### [3.1.3.](#) Verification

Verification is performed by applying the verification algorithm defined in [[I-D.irtf-cfrg-eddsa](#)] to the public key (as public key), the JWS Signing Input (as message) and the JWS Signature value (as signature). All inputs are octet strings. If the algorithm accepts, the signature is valid, otherwise it is invalid.

## [3.2.](#) ECDH-ES

The following key subtypes defined here for purpose of ECDH-ES:

subtype:	ECDH Function:
X25519	X25519
X448	X448

The key type used with these keys is "OKP". These subtypes MUST NOT be used for signing.

### [3.2.1.](#) Performing the ECDH operation

The "x" parameter of "epk" field is set as follows:

Apply the appropriate ECDH function to the ephemeral private key (as scalar input) and the standard basepoint (as u-coordinate input). The output is the value for "x" parameter of "epk" field. All inputs and outputs are octet strings.

The Z value (raw key agreement output) for key agreement is determined as follows:

Apply the appropriate ECDH function to the ephemeral private key (as scalar input) and receiver public key (as u-coordinate input). The output is the Z value. All inputs and outputs are octet strings.

## [4.](#) Security considerations

Security considerations from [[I-D.irtf-cfrg-curves](#)] and [[I-D.irtf-cfrg-eddsa](#)] apply here.

Some algorithms interact in bad ways (e.g. "Ed25519" and "Ed25519ph"). For this reason, those algorithms have different subtypes, so keys for each are not mixed up.

Do not separate key material from information what key algorithm group it is for. When using keys, check that the algorithm is compatible with the key algorithm group for the key. To do otherwise opens system up to attacks via mixing up algorithms. It is particularly dangerous to mix up signature and MAC algorithms.

Do not assume that signature also binds the key used for signing, it does not (there are also other widespread signature algorithms where this binding fails, as such binding is not part of the definition of

secure signature primitive). As an example of such failure, the Ed25519ph signature of X under key (Ed25519ph,Y) is identical to Ed25519 signature of SHA512(X) under key (Ed25519,Y). And often it takes only setting a few bits of message (easy to do by brute force) to make the message valid enough to be processed in some very surprising way.

If key generation or batch signature verification is performed, a well-seed cryptographical random number generator is REQUIRED. Signing and non-batch signature verification are deterministic operations and do not need random numbers of any kind.

## [5.](#) Acknowledgements

Mike Jones for comments on initial pre-draft.

## [6.](#) IANA considerations

The following is added to JSON Web Key Types Registry:

- o "kty" Parameter Value: "OKP"
- o Key Type Description: Octet string key pairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 2](#) of [RFC-THIS]

The following is added to JSON Web Key Parameters Registry:

- o Parameter Name: "crv"
- o Parameter Description: The algorithm group of keypair
- o Parameter Information Class: Public
- o Used with "kty" Value(s): "OKP"
- o Change Controller: IESG
- o Specification Document(s): [Section 2](#) of [RFC-THIS]
  
- o Parameter Name: "d"
- o Parameter Description: The private key
- o Parameter Information Class: Private
- o Used with "kty" Value(s): "OKP"
- o Change Controller: IESG
- o Specification Document(s): [Section 2](#) of [RFC-THIS]

- o Parameter Name: "x"
- o Parameter Description: The public key
- o Parameter Information Class: Public
- o Used with "kty" Value(s): "OKP"
- o Change Controller: IESG
- o Specification Document(s): [Section 2](#) of [RFC-THIS]

The following is added to JSON Web Signature and Encryption Algorithms Registry:

- o Algorithm Name: "Ed25519"
- o Algorithm Description: Ed25519 signature algorithm
- o Algorithm Usage Location(s): "alg"
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
- o Algorithm Analysis Documents(s): [[I-D.irtf-cfrg-eddsa](#)]
  
- o Algorithm Name: "Ed25519ph"
- o Algorithm Description: Ed25519 signature algorithm with prehash
- o Algorithm Usage Location(s): "alg"
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
- o Algorithm Analysis Documents(s): [[I-D.irtf-cfrg-eddsa](#)]
  
- o Algorithm Name: "Ed448"
- o Algorithm Description: Ed448 signature algorithm
- o Algorithm Usage Location(s): "alg"
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG

- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
- o Algorithm Analysis Documents(s): [[I-D.irtf-cfrg-eddsa](#)]
  
- o Algorithm Name: "Ed448ph"
- o Algorithm Description: Ed448 signature algorithm with prehash
- o Algorithm Usage Location(s): "alg"
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG

- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
- o Algorithm Analysis Documents(s): [[I-D.irtf-cfrg-eddsa](#)]

The following is added to JSON Web Key Elliptic Curve Registry:

- o Curve Name: "Ed25519"
- o Curve Description: Ed25519 signature algorithm keypairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
  
- o Curve Name: "Ed25519ph"
- o Curve Description: Ed25519 signature algorithm with prehash keypairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
  
- o Curve Name: "Ed448"
- o Curve Description: Ed448 signature algorithm keypairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
  
- o Curve Name: "Ed448ph"
- o Curve Description: Ed448 signature algorithm with prehash keypairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.1](#) of [RFC-THIS]
  
- o Curve name: "X25519"
- o Curve Description: X25519 function keypairs
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): [Section 3.2](#) of [RFC-THIS]
- o Analysis Documents(s): [[I-D.irtf-cfrg-curves](#)]
  
- o Curve Name: "X448"
- o Curve Description: X448 function keypairs

- o JOSE Implementation Requirements: Optional



- o Change Controller: IESG
- o Specification Document(s): [Section 3.2](#) of [RFC-THIS]
- o Analysis Documents(s): [[I-D.irtf-cfrg-curves](#)]

## [7.](#) References

### [7.1.](#) Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", [RFC 4648](#), DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [I-D.irtf-cfrg-curves] Langley, A. and M. Hamburg, "Elliptic Curves for Security", [draft-irtf-cfrg-curves-11](#) (work in progress), October 2015.
- [I-D.irtf-cfrg-eddsa] Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", [draft-irtf-cfrg-eddsa-01](#) (work in progress), December 2015.

### [7.2.](#) Informative References

- [RFC7517] Jones, M., "JSON Web Key (JWK)", [RFC 7517](#), DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", [RFC 7518](#), DOI 10.17487/RFC7518, May 2015, <<http://www.rfc-editor.org/info/rfc7518>>.
- [RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", [RFC 7638](#), DOI 10.17487/RFC7638, September 2015, <<http://www.rfc-editor.org/info/rfc7638>>.

## [Appendix A.](#) Examples

To the extent possible, the examples use material lifted from test vectors of [[I-D.irtf-cfrg-curves](#)] and [[I-D.irtf-cfrg-eddsa](#)]

### [A.1.](#) Ed25519 private key

```
{"kty":"OKP","crv":"Ed25519",  
"d":"nWGxne_9WmC6hEr0kuwsxERJxWl7MmkZcDusAxyuf2A"  
"x":"11qYAYKxCrfVS_7TyWQH0g7hcvPapiMlrwIaaPcHURo"}
```

The hexadecimal dump of private key is:

```
9d 61 b1 9d ef fd 5a 60 ba 84 4a f4 92 ec 2c c4  
44 49 c5 69 7b 32 69 19 70 3b ac 03 1c ae 7f 60
```

And of the public key:

```
d7 5a 98 01 82 b1 0a b7 d5 4b fe d3 c9 64 07 3a  
0e e1 72 f3 da a6 23 25 af 02 1a 68 f7 07 51 1a
```

### [A.2.](#) Ed25519 public key

This is the public parts of the previous private key (just omits "d"):

```
{"kty":"OKP","crv":"Ed25519",  
"x":"11qYAYKxCrfVS_7TyWQH0g7hcvPapiMlrwIaaPcHURo"}
```

### [A.3.](#) JWK thumbprint canonicalization

The JWK thumbprint canonicalization of the two above examples is (linebreak inserted for formatting reasons)

```
{"crv":"Ed25519","kty":"OKP","x":"11qYAYKxCrfVS_7TyWQH0g7hcvPapiMlrwI  
aaPcHURo"}
```

Which has the SHA-256 hash of:

```
90facafea9b1556698540f70c0117a22ea37bd5cf3ed3c47093c1707282b4b89
```

### [A.4.](#) Ed25519 Signing

The JWS protected header is:

```
{"alg":"Ed25519"}
```

This has base64url encoding of:

```
eyJhbGciOiJFZDI1NTE5In0
```

The payload is (text):

## Example of Ed25519 signing

This has base64url encoding of:

```
RXhbbXBsZSBvZiBFZDI1NTE5IHNPZ25pbmc
```

The JWS signing input is (concatenation of base64url encoding of the (protected) header, a dot and base64url encoding of the payload) is:

```
eyJhbGciOiJIJFZDI1NTE5In0.RXhbbXBsZSBvZiBFZDI1NTE5IHNPZ25pbmc
```

Applying Ed25519 signing algorithm to the private key, public key and the JWS signing input yields signature (hex):

```
53 18 48 60 b1 c6 83 7f 4d 54 22 e9 40 05 43 fd
47 1f 3a 69 c6 48 2c cb 15 9a 17 62 42 e2 21 b1
5c 72 63 9b fe a3 9b b2 08 f3 2c ab 1f 27 0f b8
36 57 1c 52 0b d8 ac 41 eb 45 b3 55 d0 77 19 01
```

Converting this to base64url yields:

```
UxhIYLHGg39NVCLpQAVD_Ucf0mnGSCzLFZoXYkLiIbFccm0b_q0bsgjzLKsfJw-4NlccU
gvYrEHrRbNV0HcZAQ
```

So the compact serialization of JWS is (concatenation of signing input, a dot and base64url encoding of the signature:

```
eyJhbGciOiJIJFZDI1NTE5In0.RXhbbXBsZSBvZiBFZDI1NTE5IHNPZ25pbmc.UxhIYLHGg
39NVCLpQAVD_Ucf0mnGSCzLFZoXYkLiIbFccm0b_q0bsgjzLKsfJw-4NlccUgvYrEHrRb
NV0HcZAQ
```

### [A.5.](#) Ed25519 Validation

The JWS from above example is:

```
eyJhbGciOiJIJFZDI1NTE5In0.RXhbbXBsZSBvZiBFZDI1NTE5IHNPZ25pbmc.UxhIYLHGg
39NVCLpQAVD_Ucf0mnGSCzLFZoXYkLiIbFccm0b_q0bsgjzLKsfJw-4NlccUgvYrEHrRb
NV0HcZAQ
```

This has 2 dots in it, so it might be valid JWS. Base64url decoding the protected header yields:

```
{"alg":"Ed25519"}
```

So this is Ed25519 signature. Now the key has: "kty":"OKP" and "crv":"Ed25519", so the key is valid for the algorithm (if it had other values, the validation would have failed).

The signing input is the part before second dot:

```
eyJhbGciOiJIJFZDI1NTE5In0.RXhhbXBsZSBvZiBFZDI1NTE5IHNPZ25pbmc
```

Applying Ed25519 verification algorithm to the public key, JWS signing input and the signature yields true. So the signature is valid. The message is base64 decoding of the part between the dots:

Example of Ed25519 signing

#### [A.6.](#) ECDH-ES with X25519

The public key to encrypt to is:

```
{"kty":"OKP","crv":"X25519","kid":"Bob"  
"x":"3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrFx-FG-IK08"}
```

The public key from target key is (hex):

```
de 9e db 7d 7b 7d c1 b4 d3 5b 61 c2 ec e4 35 37  
3f 83 43 c8 5b 78 67 4d ad fc 7e 14 6f 88 2b 4f
```

The ephemeral secret happens to be (hex):

```
77 07 6d 0a 73 18 a5 7d 3c 16 c1 72 51 b2 66 45  
df 4c 2f 87 eb c0 99 2a b1 77 fb a5 1d b9 2c 2a
```

So the ephemeral public key is X25519(ephkey,G) (hex):

```
85 20 f0 09 89 30 a7 54 74 8b 7d dc b4 3e f7 5a  
0d bf 3a 0d 26 38 1a f4 eb a4 a9 8e aa 9b 4e 6a
```

This is packed into ephemeral public key value:

```
{"kty":"OKP","crv":"X25519",  
"x":"hSDwCYkwp1R0i33ctD73Wg2_Og0m0Br066SpjqqbTmo"}
```

So the protected header could for example be:

```
{"alg":"ECDH-ES+A128KW","epk":{"kty":"OKP","crv":"X25519",  
"x":"hSDwCYkwp1R0i33ctD73Wg2_Og0m0Br066SpjqqbTmo"},  
"enc":"A128GCM","kid":"Bob"}
```

And sender computes as the DH Z value as X25519(ephkey,recv\_pub)  
(hex):

```
4a 5d 9d 5b a4 ce 2d e1 72 8e 3b f4 80 35 0f 25  
e0 7e 21 c9 47 d1 9e 33 76 f0 9b 3c 1e 16 17 42
```

The receiver computes as the DH Z value as X25519(seckey,ephkey\_pub)  
(hex):

```
4a 5d 9d 5b a4 ce 2d e1 72 8e 3b f4 80 35 0f 25  
e0 7e 21 c9 47 d1 9e 33 76 f0 9b 3c 1e 16 17 42
```

Which is the same as sender's value (the both sides run this through  
KDF before using as AES128-KW key).

#### [A.7.](#) ECDH-ES with X448

The public key to encrypt to is (linebreak inserted for formatting  
reasons):

```
{"kty":"OKP","crv":"X448","kid":"Dave"  
"x":"PreoKbDNIPW8_AtZm2_sz22kYnEHvbDU80W0MCfYuXL8PjT7QjKhPKcG3LV67D2  
uB73BxnvzNgk"}
```

The public key from target key is (hex):

```
3e b7 a8 29 b0 cd 20 f5 bc fc 0b 59 9b 6f ec cf  
6d a4 62 71 07 bd b0 d4 f3 45 b4 30 27 d8 b9 72  
fc 3e 34 fb 42 32 a1 3c a7 06 dc b5 7a ec 3d ae  
07 bd c1 c6 7b f3 36 09
```

The ephemeral secret happens to be (hex):

```
9a 8f 49 25 d1 51 9f 57 75 cf 46 b0 4b 58 00 d4
ee 9e e8 ba e8 bc 55 65 d4 98 c2 8d d9 c9 ba f5
74 a9 41 97 44 89 73 91 00 63 82 a6 f1 27 ab 1d
9a c2 d8 c0 a5 98 72 6b
```

So the ephemeral public key is  $X_{448}(\text{ephkey}, G)$  (hex):

```
9b 08 f7 cc 31 b7 e3 e6 7d 22 d5 ae a1 21 07 4a
27 3b d2 b8 3d e0 9c 63 fa a7 3d 2c 22 c5 d9 bb
c8 36 64 72 41 d9 53 d4 0c 5b 12 da 88 12 0d 53
17 7f 80 e5 32 c4 1f a0
```

This is packed into ephemeral public key value (linebreak inserted for formatting purposes):

```
{"kty":"OKP","crv":"X448",
"x":"mwj3zDG34-Z9ItWuoSEHSic70rg94Jxj-qc9LCLF2bvINmRyQdlT1AxbEtqIEg1
TF3-A5TLEH6A"}
```

So the protected header could for example be (linebreak inserted for formatting purposes):

```
{"alg":"ECDH-ES+A256KW","epk":{"kty":"OKP","crv":"X448",
"x":"mwj3zDG34-Z9ItWuoSEHSic70rg94Jxj-qc9LCLF2bvINmRyQdlT1AxbEtqIEg1
TF3-A5TLEH6A"},"enc":"A256GCM","kid":"Dave"}
```

And sender computes as the DH Z value as  $X_{448}(\text{ephkey}, \text{recv\_pub})$  (hex):

```
07 ff f4 18 1a c6 cc 95 ec 1c 16 a9 4a 0f 74 d1
2d a2 32 ce 40 a7 75 52 28 1d 28 2b b6 0c 0b 56
fd 24 64 c3 35 54 39 36 52 1c 24 40 30 85 d5 9a
44 9a 50 37 51 4a 87 9d
```

The receiver computes as the DH Z value as  $X_{448}(\text{seckey}, \text{ephkey\_pub})$  (hex):

```
07 ff f4 18 1a c6 cc 95 ec 1c 16 a9 4a 0f 74 d1
2d a2 32 ce 40 a7 75 52 28 1d 28 2b b6 0c 0b 56
fd 24 64 c3 35 54 39 36 52 1c 24 40 30 85 d5 9a
44 9a 50 37 51 4a 87 9d
```

Which is the same as sender's value (the both sides run this through KDF before using as AES256-KW key).

Author's Address

Ilari Liusvaara  
Independent

Email: ilariliusvaara@welho.com