Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 7, 2021

N. Madden
ForgeRock

May 6, 2021

# Public Key Authenticated Encryption for JOSE: ECDH-1PU
draft-madden-jose-ecdh-1pu-04

## Abstract

This document describes the ECDH-1PU public key authenticated encryption algorithm for JWE. The algorithm is similar to the existing ECDH-ES encryption algorithm, but adds an additional ECDH key agreement between static keys of the sender and recipient. This additional step allows the recipient to be assured of sender authenticity without requiring a nested signed-then-encrypted message structure.

## Status of This Memo

This Internet-Draft will expire on November 7, 2021.

## Copyright Notice

Table of Contents

## 1. Introduction

JSON Object Signing and Encryption (JOSE) defines a number of encryption (JWE) [RFC7516] and digital signature (JWS) [RFC7515] algorithms. When symmetric cryptography is used, JWE provides authenticated encryption that ensures both confidentiality and sender

authentication. However, for public key cryptography the existing JWE encryption algorithms provide only confidentiality and some level of ciphertext integrity. When sender authentication is required, users must resort to nested signed-then-encrypted structures, which increases the overhead and size of resulting messages. This document describes an alternative encryption algorithm called ECDH-1PU that provides public key authenticated encryption, allowing the benefits of authenticated encryption to be enjoyed for public key JWE as it currently is for symmetric cryptography.

ECDH-1PU is based on the One-Pass Unified Model for Elliptic Curve Diffie-Hellman key agreement described in [NIST.800-56A].

The advantages of public key authenticated encryption with ECDH-1PU compared to using nested signed-then-encrypted documents include the following:

- The resulting message size is more compact as an additional layer of headers and base64url-encoding is avoided. A 500-byte payload when encrypted and authenticated with ECDH-1PU (with P-256 keys and "A256GCM" Content Encryption Method) results in a 1087-byte JWE in Compact Encoding. An equivalent nested signed-then-encrypted JOSE message using the same keys and encryption method is 1489 bytes (37% larger).

- The same primitives are used for both confidentiality and authenticity, providing savings in code size for constrained environments.

- The generic composition of signatures and public key encryption involves a number of subtle details that are essential to security [PKAE]. Providing a dedicated algorithm for public key authenticated encryption reduces complexity for users of JOSE libraries.

- ECDH-1PU provides only authenticity and not the stronger security properties of non-repudiation or third-party verifiability. This can be an advantage in applications where privacy, anonymity, or plausible deniability are goals.

## 1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Key Agreement with Elliptic Curve Diffie-Hellman One-Pass Unified Model (ECDH-1PU)

This section defines the specifics of key agreement with Elliptic Curve Diffie-Hellman One-Pass Unified Model, in combination with the one-step KDF, as defined in Section 5.8.2.1 of [NIST.800-56A] using the Concatenation Format of Section 5.8.2.1.1. This is identical to the ConcatKDF function used by the existing JWE ECDH-ES algorithm defined in Section 4.6 of [RFC7518]. As for ECDH-ES, the key agreement result can be used in one of two ways:

1. directly as the Content Encryption Key (CEK) for the "enc" algorithm, in the Direct Key Agreement mode, or

2. as a symmetric key used to wrap the CEK with the "A128KW", "A192KW", or "A256KW" algorithms, in the Key Agreement with Key Wrapping mode.

A fresh ephemeral public key value MUST be generated for each message. When encrypting the message to multiple recipients using ECDH-1PU, the same ephemeral keys MAY be reused for multiple recipients [MRES].

In Direct Key Agreement mode, the output of the KDF MUST be a key of the same length as that used by the "enc" algorithm. In this case, the empty octet sequence is used as the JWE Encrypted Key value. The "alg" (algorithm) Header Parameter value "ECDH-1PU" is used in Direct Key Agreement mode.

In Key Agreement with Key Wrapping mode, the output of the KDF MUST be a key of the length needed for the specified key wrapping algorithm. In this case, the JWE Encrypted Key is the CEK wrapped with the agreed-upon key.

The following "alg" (algorithm) Header Parameter values are used to indicate the JWE Encrypted Key is the result of encrypting the CEK using the result of the key agreement algorithm as the key encryption key for the corresponding key wrapping algorithm:

| "alg" Param Value | Key Management Algorithm |
|---|---|
| ECDH-1PU+A128KW | ECDH-1PU using one-pass KDF and CEK wrapped with "A128KW" |
| ECDH-1PU+A192KW | ECDH-1PU using one-pass KDF and CEK wrapped with "A192KW" |
| ECDH-1PU+A256KW | ECDH-1PU using one-pass KDF and CEK wrapped with "A256KW" |

## 2.1. Special Considerations for Key Agreement with Key Wrapping mode

In Key Agreement with Key Wrapping mode, the JWE Authentication Tag is included in the input to the Key Derivation Function as described in section Section 2.3. This ensures that the content of the JWE was produced by the original sender and not by another recipient, as described in section Section 4.

Key Agreement with Key Wrapping mode MUST only be used with content encryption algorithms that are compactly committing AEADs as described in [ccAEAD]. The AES_CBC_HMAC_SHA2 algorithms described in section 5.2 of [RFC7518] are compactly committing and can be used with ECDH-1PU in Key Agreement with Key Wrapping mode. Other content encryption algorithms MUST be rejected. In Direct Key Agreement mode, any JWE content encryption algorithm MAY be used.

The requirement to include the JWE Authentication Tag in the input to the Key Derivation Function implies an adjustment to the order of operations performed during JWE Message Encryption described in section 5.1 of [RFC7516]. Steps 3-8 are deferred until after step 15, using the randomly generated CEK from step 2 for encryption of the message content.

## 2.2. Header Parameters used for ECDH Key Agreement

The "epk" (ephemeral public key), "apu" (Agreement PartyUInfo), and "apv" (Agreement PartyVInfo) header parameters are used in ECDH-1PU exactly as defined in Section 4.6.1 of [RFC7518].

When no other values are supplied, it is RECOMMENDED that the producer software initializes the "apu" header to the base64url-encoding of the SHA-256 hash of the concatenation of the sender's static public key and the ephemeral public key, and the "apv" header to the base64url-encoding of the SHA-256 hash of the recipient's static public key. This ensures that all keys involved in the key agreement are cryptographically bound to the derived keys.

### 2.2.1. "skid" Header Parameter

A new Header Parameter "skid" (Sender Key ID) is registered as a hint as to which of the sender's keys was used to authenticate the JWE. The structure of the "skid" value is unspecified. Its value MUST be a case-sensitive string. Use of this Header Parameter is OPTIONAL. When used with a JWK, the "skid" value is used to match a JWK "kid" parameter value [RFC7517].

## 2.3. Key Derivation for ECDH-1PU Key Agreement

The key derivation process derives the agreed-upon key from the shared secret Z established through the ECDH algorithm, per Section 6.2.1.2 of [NIST.800-56A]. For the NIST prime order curves "P-256", "P-384", and "P-521", the ECC CDH primitive for cofactor

Diffie-Hellman defined in Section 5.7.1.2 of [NIST.800-56A] is used (taking note that the cofactor for all these curves is 1). For curves "X25519" and "X448" the appropriate ECDH primitive from Section 5 of [RFC7748] is used.

Key derivation is performed using the one-step KDF, as defined in Section 5.8.1 and Section 5.8.2.1 of [NIST.800-56A] using the Concatenation Format of Section 5.8.2.1.1, where the Auxilary Function H is SHA-256. The KDF parameters are set as follows:

Z This is set to the representation of the shared secret Z as an octet sequence. As per Section 6.2.1.2 of [NIST.800-56A] Z is the concatenation of Ze and Zs, where Ze is the shared secret derived from applying the ECDH primitive to the sender's ephemeral private key and the recipient's static public key (when sending) or the recipient's static private key and the sender's ephemeral public key (when receiving). Zs is the shared secret derived from applying the ECDH primitive to the sender's static private key and the recipient's static public key (when sending) or the recipient's static private key and the sender's static public key (when receiving).

keydatalen This is set to the number of bits in the desired output key. For "ECDH-1PU", this is the length of the key used by the "enc" algorithm. For "ECDH-1PU+A128KW", "ECDH-1PU+A192KW", and "ECDH-1PU+A256KW", this is 128, 192, and 256, respectively.

cctag In Direct Key Agreement mode this is set to an empty octet string. In Key Agreement with Key Wrapping mode, this is set to a value of the form Datalen || Data, where Data is the raw octets of the JWE Authentication Tag, and Datalen is the big-endian 32-bit length of the authentication tag (in octets).

AlgorithmID The AlgorithmID value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data. In the Direct Key Agreement case, Data is set to the octets of the ASCII representation of the "enc" Header Parameter value. In the Key Agreement with Key Wrapping case, Data is set to the octets of the ASCII representation of the "alg" (algorithm) Header Parameter value.

PartyUInfo  The PartyUInfo value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data.  If an "apu" (agreement PartyUInfo) Header Parameter is present, Data is set to the result of base64url decoding the "apu" value and Datalen is set to the number of octets in Data.  Otherwise, Datalen is set to 0 and Data is set to the empty octet sequence.

PartyVInfo  The PartyVInfo value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data.  If an "apv" (agreement PartyVInfo) Header Parameter is present, Data is set to the result of base64url decoding the "apv" value and Datalen is set to the number of octets in Data.  Otherwise, Datalen is set to 0 and Data is set to the empty octet sequence.

SuppPubInfo  This is set to the keydatalen represented as a 32-bit big-endian integer followed by the octets of the cctag.

SuppPrivInfo  This is set to the empty octet sequence.

Applications need to specify how the "apu" and "apv" Header Parameters are used for that application.  The "apu" and "apv" values MUST be distinct, when used.  Applications wishing to conform to [NIST.800-56A] need to provide values that meet the requirements of that document, e.g., by using values that identify the producer and consumer.

See Appendix A for an example key agreement computation using Direct Key Agreement mode, and Appendix B for an example sending to multiple recipients using Key Agreement with Key Wrapping mode.

## 3.  IANA considerations

This section registers identifiers under the IANA JSON Web Signature and Encryption Algorithms Registry established by [RFC7518] and the IANA JSON Web Signature and Encryption Header Parameters registry established by [RFC7515].

## 3.1. JSON Web Signature and Encryption Algorithms Registration

This section registers JWE algorithms as per the registry established in [RFC7518].

### 3.1.1. ECDH-1PU

Algorithm Name: "ECDH-1PU"
Algorithm Description: ECDH One-Pass Unified Model using one-pass KDF
Algorithm Usage Location(s): "alg"
JOSE Implementation Requirements: Optional
Change Controller: IESG
Specification Document(s): Section 2
Algorithm Analysis Document(s): [NIST.800-56A] (Section 7.3), [PKAE]

Algorithm Name: "ECDH-1PU+A128KW"
Algorithm Description: ECDH One-Pass Unified Model using one-pass KDF and "A128KW"
Algorithm Usage Location(s): "alg"
JOSE Implementation Requirements: Optional
Change Controller: IESG
Specification Document(s): Section 2
Algorithm Analysis Document(s): [NIST.800-56A] (Section 7.3), [PKAE]

Algorithm Name: "ECDH-1PU+A192KW"
Algorithm Description: ECDH One-Pass Unified Model using one-pass KDF and "A192KW"
Algorithm Usage Location(s): "alg"
JOSE Implementation Requirements: Optional
Change Controller: IESG
Specification Document(s): Section 2
Algorithm Analysis Document(s): [NIST.800-56A] (Section 7.3), [PKAE]

Algorithm Name: "ECDH-1PU+A256KW"
Algorithm Description: ECDH One-Pass Unified Model using one-pass KDF and "A256KW"
Algorithm Usage Location(s): "alg"

JOSE Implementation Requirements: Optional
Change Controller: IESG
Specification Document(s): Section 2
Algorithm Analysis Document(s): [NIST.800-56A] (Section 7.3), [PKAE]

### 3.2. JSON Web Signature and Encryption Header Parameters Registration

This section registers new Header Parameters as per the registry established in [RFC7515].

#### 3.2.1. skid

Header Parameter Name: "skid"
Header Parameter Description: Sender Key ID
Header Parameter Usage Location(s): JWE
Change Controller: IESG
Specification Document(s): Section 2.2.1

## 4. Security Considerations

The security considerations of [RFC7516] and [RFC7518] relevant to ECDH-ES also apply to this specification.

The security considerations of [NIST.800-56A] apply here.

When performing an ECDH key agreement between a static private key and any untrusted public key, care should be taken to ensure that the public key is a valid point on the same curve as the private key. Failure to do so may result in compromise of the static private key. For the NIST curves P-256, P-384, and P-521, appropriate validation routines are given in Section 5.6.2.3.3 of [NIST.800-56A]. For the curves used by X25519 and X448, consult the security considerations of [RFC7748].

The ECDH-1PU algorithm is vulnerable to Key Compromise Impersonation (KCI) attacks. If the long-term static private key of a party is compromised, then the attacker can not only impersonate that party to other parties, but also impersonate any other party when communicating with the compromised party. If resistance to KCI is desired in a single message, then the sender SHOULD use a nested JWS signature over the content.

When Key Agreement with Key Wrapping is used, the JWE Authentication Tag is included in the input to the Key Derivation Function, as described in section Section 2.3. Without this step, when the same

Content Encryption Key (CEK) is reused for multiple recipients, then any of those recipients can produce a new message that appears to

come from the original sender. If the MAC used by the content encryption algorithm is not compactly committing ([ccAEAD]) then it may be possible for a recipient to calculate an alternative message that produces the same authentication tag. An alternative is to encrypt the message separately to each recipient using Direct Key Agreement, or to sign the message using a nested signed-then-encrypted JOSE composition.

The security properties of the one-pass unified model are given in Section 7.3 of [NIST.800-56A].

# 5. References

## 5.1. Normative References

[NIST.800-56A]
Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key Establishment Using Discrete Logarithm Cryptography Revision 3.", NIST Special Publication 800-56A, April 2018.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[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/info/rfc7748>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

## 5.2. Informative References

[ccAEAD] Grubbs, P., Lu, J., and T. Ristenpart, "Message Franking via Committing Authenticated Encryption", IACR ePrint 2017/664, 2017.

[MRES] Bellare, M., Boldyreva, A., Kurosawa, K., and J. Staddon, "Multi-Recipient Encryption Schemes: Efficient Constructions and their Security", IEEE Transactions on Information Theory Vol. 53, Number 11, 2007.

[PKAE] An, J., "Authenticated Encryption in the Public-Key Setting: Security Notions and Analyses", IACR ePrint 2001/079, 2001.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

## Appendix A. Example ECDH-1PU Key Agreement Computation with A256GCM

This example uses ECDH-1PU in Direct Key Agreement mode ("alg" value "ECDH-1PU") to produce an agreed-upon key for AES GCM with a 256-bit key ("enc" value "A256GCM"). The example re-uses the keys and parameters of the example computation in Appendix C of [RFC7518], with the addition of an extra static key-pair for Alice.

In this example, a producer Alice is encrypting content to a consumer Bob. Alice's static key-pair (in JWK format) used for the key agreement in this example (including the private part) is:

```
{"kty":"EC",
 "crv":"P-256",
 "x":"WKn-ZIGevcwGIyyrzFoZNBdaq9_TsqzGl96oc0CWuis",
 "y":"y77t-RvAHRKTsSGdIYUfweuOvwrvDD-Q3Hv5J0fSKbE",
 "d":"Hndv7ZZjs_ke8o9zXYo3iq-Yr8SewI5vrqd0pAvEPqg"}
```

Bob's static key-pair (in JWK format) is:

```
{"kty":"EC",
 "crv":"P-256",
 "x":"weNJy2HscCSM6AEDTDg04biOvhFhyyWvOHQfeF_PxMQ",
 "y":"e8lnCO-AlStT-NJVX-crhB7QRYhiix03illJOVAOyck",
 "d":"VEmDZpDXXK8p8N0Cndsxs924q6nS1RXFASRl6BfUqdw"}
```

The producer (Alice) generates an ephemeral key for the key agreement computation. Alice's ephemeral key (in JWK format) is:

```
{"kty":"EC",
 "crv":"P-256",
 "x":"gI0GAILBdu7T53akrFmMyGcsF3n5dO7MmwNBHKW5SV0",
 "y":"SLW_xSffzlPWrHEVI30DHM_4egVwt3NQqeUD7nMFpps",
 "d":"0_NxaRPUMQoAJt50Gz8YiTr8gRTwyEaCumd-MToTmIo"}
```

Header Parameter values used in this example are as follows. The "apu" (agreement PartyUInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Alice" and the "apv" (agreement PartyVInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Bob". The "epk" (ephemeral public key) Header Parameter is used to communicate the producer's (Alice's) ephemeral public key value to the consumer (Bob).

```
{"alg":"ECDH-1PU",
 "enc":"A256GCM",
 "apu":"QWxpY2U",
 "apv":"Qm9i",
 "epk":
  {"kty":"EC",
   "crv":"P-256",
   "x":"gI0GAILBdu7T53akrFmMyGcsF3n5dO7MmwNBHKW5SV0",
   "y":"SLW_xSffzlPWrHEVI30DHM_4egVwt3NQqeUD7nMFpps"
  }
}
```

The resulting one-pass KDF [NIST.800-56A] parameter values are:

Ze This is set to the output of the ECDH key agreement between Alice's ephemeral private key and Bob's static public key. In this example, Ze is the following octet sequence (in hexadecimal notation):

```
9e 56 d9 1d 81 71 35 d3 72 83 42 83 bf 84 26 9c
fb 31 6e a3 da 80 6a 48 f6 da a7 79 8c fe 90 c4
```

Zs This is set to the output of the ECDH key agreement between Alice's static private key and Bob's static public key. In this example, Zs is the following octet sequence (in hexadecimal notation):

```
e3 ca 34 74 38 4c 9f 62 b3 0b fd 4c 68 8b 3e 7d
41 10 a1 b4 ba dc 3c c5 4e f7 b8 12 41 ef d5 0d
```

Z This is set to the concatenation of Ze followed by Zs. In this example, Z is the following octet sequence (in hexadecimal notation):

```
9e 56 d9 1d 81 71 35 d3 72 83 42 83 bf 84 26 9c
fb 31 6e a3 da 80 6a 48 f6 da a7 79 8c fe 90 c4
e3 ca 34 74 38 4c 9f 62 b3 0b fd 4c 68 8b 3e 7d
41 10 a1 b4 ba dc 3c c5 4e f7 b8 12 41 ef d5 0d
```

keydatalen This value is 256 - the number of bits in the desired output key (because "A256GCM" uses a 256-bit key).

cctag This value is the empty octet string.

AlgorithmID This is set to the octets representing the 32-bit big-endian value 7 - 00 00 00 07 in hexadecimal notation - the number of octets in the AlgorithmID content "A256GCM", followed by the octets representing the ASCII string "A256GCM" - 41 32 35 36 47 43 4d (in hex). The complete value is therefore: 00 00 00 07 41 32 35 36 47 43 4d

PartyUInfo This is set to the octets representing the 32-bit big-endian value 5, followed by the octets representing the UTF-8 string "Alice". In hexadecimal notation: 00 00 00 05 41 6c 69 63 65

PartyVInfo This is set to the octets representing the 32-bit big-endian value 3, followed by the octets representing the UTF-8 string "Bob". In hexadecimal notation: 00 00 00 03 42 6f 62

SuppPubInfo This is set to the octets representing the 32-bit big-endian value 256 - the keydatalen value. In hexadecimal notation: 00 00 01 00

SuppPrivInfo This is set to the empty octet sequence.

Concatenating the parameters AlgorithmID through SuppPrivInfo results in a FixedInfo value in Concatenation Format (as per Section 5.8.2.1.1 of [NIST.800-56A]) of (in hexidecimal notation):

```
00 00 00 07 41 32 35 36 47 43 4d 00 00 00 05 41
6c 69 63 65 00 00 00 03 42 6f 62 00 00 01 00
```

Concatenating the round number 1 (00 00 00 01), Z, and the FixedInfo value results in a one-pass KDF round 1 hash input of (hexadecimal):

```
00 00 00 01 9e 56 d9 1d 81 71 35 d3 72 83 42 83
bf 84 26 9c fb 31 6e a3 da 80 6a 48 f6 da a7 79
8c fe 90 c4 e3 ca 34 74 38 4c 9f 62 b3 0b fd 4c
68 8b 3e 7d 41 10 a1 b4 ba dc 3c c5 4e f7 b8 12
41 ef d5 0d 00 00 00 07 41 32 35 36 47 43 4d 00
00 00 05 41 6c 69 63 65 00 00 00 03 42 6f 62 00
00 01 00
```

The resulting derived key, which is the full 256 bits of the round 1 hash output is:

```
6c af 13 72 3d 14 85 0a d4 b4 2c d6 dd e9 35 bf
fd 2f ff 00 a9 ba 70 de 05 c2 03 a5 e1 72 2c a7
```

The base64url-encoded representation of this derived key is:

```
bK8Tcj0UhQrUtCzW3ek1v_0v_wCpunDeBcIDpeFyLKc
```

## Appendix B. Example ECDH-1PU+A128KW Key Agreement computation with A256CBC-HS256

This example uses ECDH-1PU in Key Agreement with Key Wrapping mode ("alg" value "ECDH-1PU+A128KW") to encrypt a JWE for multiple recipients using the JWE JSON Serialization. The example uses X25519 key pairs, as described in [RFC8037]. Alice is sending an identical message to Bob and Charlie. Because Bob and Charlie are using the same curve (X25519), Alice reuses the same ephemeral key-pair for both recipients and includes it in the JWE Protected Header. If this was not the case, Alice should generate a separate ephemeral key-pair for each recipient and include it in each per-recipient header instead.

Alice's static key pair, represented as an OKP JWK (including the private component) is:

```
{"kty": "OKP",
 "crv": "X25519",
 "x": "Knbm_BcdQr7WIoz-uqit9M0wbcfEr6y-9UfIZ8QnBD4",
 "d": "i9KuFhSzEBsiv3PKVL5115OCdsqQai5nj_Flzfkw5jU"}
```

Bob's static key-pair (in JWK format) is:

```
{"kty": "OKP",
 "crv": "X25519",
 "x": "BT7aR0ItXfeDAldeeOlXL_wXqp-j5FltT0vRSG16kRw",
 "d": "1gDirl_r_Y3-qUa3WXHgEXrrEHngWThU3c9zj9A2uBg"}
```

Charlie's static key-pair (in JWK format) is:

```
{"kty": "OKP",
 "crv": "X25519",
 "x": "q-LsvU772uV_2sPJhfAIq-3vnKNVefNoIlvyvg1hrnE",
 "d": "Jcv8gklhMjC0b-lsk5onBbppWAx5ncNtbM63Jr9xBQE"}
```

Alice generates an ephemeral key-pair on the same curve. Alice's ephemeral key-pair (in JWK format) is:

```
{"kty": "OKP",
 "crv": "X25519",
 "x": "k9of_cpAajy0poW5gaixXGs9nHkwg1AFqUAFa39dyBc",
 "d": "x8EVZH4Fwk673_mUujnliJoSrLz0zYzzCWp5GUX2fc8"}
```

## B.1. JWE Protected Header

The JWE Protected Header is as follows. The "apu" (agreement PartyUInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Alice" and the "apv" (agreement PartyVInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Bob and Charlie". The "epk" (ephemeral public key) Header Parameter is used to communicate the producer's (Alice's) ephemeral public key to the consumers (Bob and Charlie).

```
{"alg":"ECDH-1PU+A128KW",
 "enc":"A256CBC-HS512",
 "apu":"QWxpY2U",
 "apv":"Qm9iIGFuZCBDaGFybGll",
 "epk":
  {"kty":"OKP",
   "crv":"X25519",
   "x":"k9of_cpAajy0poW5gaixXGs9nHkwg1AFqUAFa39dyBc"}}
```

## B.2. JWE Per-Recipient Unprotected Headers

The following JWE Per-Recipient Unprotected Header values are used for Bob and Charlie respectively:

```
{"kid":"bob-key-2"}
{"kid":"2021-05-06"}
```

## B.3. JWE Shared Unprotected Header

This JWE uses the "jku" Header Parameter to reference a JWK Set. This is represented in the following JWE Shared Unprotected Header value as:

```
{"jku":"https://alice.example.com/keys.jwks"}
```

## B.4. Additional Authenticated Data

Let the Additional Authenticated Data encryption parameter be ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:

```
[123, 34, 97, 108, 103, 34, 58, 34, 69, 67, 68, 72, 45, 49, 80, 85,
43, 65, 49, 50, 56, 75, 87, 34, 44, 34, 101, 110, 99, 34, 58, 34,
65, 50, 53, 54, 67, 66, 67, 45, 72, 83, 53, 49, 50, 34, 44, 34, 97,
112, 117, 34, 58, 34, 81, 87, 120, 112, 89, 50, 85, 34, 44, 34, 97,
112, 118, 34, 58, 34, 81, 109, 57, 105, 73, 71, 70, 117, 90, 67, 66,
68, 97, 71, 70, 121, 98, 71, 108, 108, 34, 44, 34, 101, 112, 107,
34, 58, 123, 34, 107, 116, 121, 34, 58, 34, 79, 75, 80, 34, 44, 34,
99, 114, 118, 34, 58, 34, 88, 50, 53, 53, 49, 57, 34, 44, 34, 120,
34, 58, 34, 107, 57, 111, 102, 95, 99, 112, 65, 97, 106, 121, 48,
112, 111, 87, 53, 103, 97, 105, 120, 88, 71, 115, 57, 110, 72, 107,
119, 103, 49, 65, 70, 113, 85, 65, 70, 97, 51, 57, 100, 121, 66, 99,
34, 125, 125]
```

### B.5. Content Encryption Key

Alice generates the following 512-bit Content Encryption Key (CEK) for A256CBC-HS512 (shown in hexadecimal):

```
ff fe fd fc fb fa f9 f8 f7 f6 f5 f4 f3 f2 f1 f0
ef ee ed ec eb ea e9 e8 e7 e6 e5 e4 e3 e2 e1 e0
df de dd dc db da d9 d8 d7 d6 d5 d4 d3 d2 d1 d0
cf ce cd cc cb ca c9 c8 c7 c6 c5 c4 c3 c2 c1 c0
```

### B.6. Initialization Vector

She then generates the following random JWE Initialization Vector (IV):

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

### B.7. JWE Plaintext

The plaintext of the message Alice sends to Bob and Charlie is the UTF-8 bytes of the string "Three is a magic number." (without the quotes). The octets of the plaintext are:

```
[84, 104, 114, 101, 101, 32, 105, 115, 32, 97, 32, 109, 97, 103, 105, 99,
32, 110, 117, 109, 98, 101, 114, 46]
```

### B.8. Content Encryption

Alice performs authenticated encryption on the plaintext with the AES_256_CBC_HMAC_SHA_512 algorithm using the CEK as the encryption key, the JWE Initialization Vector, and the Additional Authenticated Data value above. This algorithm is described in [RFC7518]. The resulting ciphertext (in base64url encoding) is:

```
Az2IWsISEMDJvyc5XRL-3-d-RgNBOGolCsxFFoUXFYw
```

The resulting JWE Authentication Tag is (in base64url encoding):

```
HLb4fTlm8spGmij3RyOs2gJ4DpHM4hhVRwdF_hGb3WQ
```

### B.9. Key Agreement for Bob

The KDF input parameters for Bob are as follows:

Ze This is set to the ECDH key agreement output between Alice's ephemeral private key and Bob's static public key. In this example, Ze is the following octet sequence (in hexadecimal):

```
32 81 08 96 e0 fe 4d 57 0e d1 ac fc ed f6 71 17
dc 19 4e d5 da ac 21 d8 ff 7a f3 24 46 94 89 7f
```

Zs This is set to the ECDH key agreement output between Alice's static private key and Bob's static public key. In this example, Zs is the following octet sequence (in hexadecimal):

```
21 57 61 2c 90 48 ed fa e7 7c b2 e4 23 71 40 60
59 67 c0 5c 7f 77 a4 8e ea f2 cf 29 a5 73 7c 4a
```

Z Z is the concatenation of Ze followed by Zs. In this example, the value of Z is:

```
32 81 08 96 e0 fe 4d 57 0e d1 ac fc ed f6 71 17
dc 19 4e d5 da ac 21 d8 ff 7a f3 24 46 94 89 7f
21 57 61 2c 90 48 ed fa e7 7c b2 e4 23 71 40 60
59 67 c0 5c 7f 77 a4 8e ea f2 cf 29 a5 73 7c 4a
```

keydatalen This value is 128 - the number of bits in the desired output key (because "ECDH-1PU+A128KW" uses a 128-bit key-wrapping key).

cctag This is set to the octets of the JWE Authentication Tag, prefixed by the length of the authentication tag (number of octets) as a big-endian 32-bit unsigned integer. For the "A256CBC-HS512" algorithm used in this example, the tag is 32 octets in size (00 00 00 20 in hex). The complete value of the cctag parameter for this example (in hex) is:

```
00 00 00 20 1c b6 f8 7d 39 66 f2 ca 46 9a 28 f7
47 23 ac da 02 78 0e 91 cc e2 18 55 47 07 45 fe
11 9b dd 64
```

AlgorithmID  This is set to the octets representing the big-endian value 15 - 00 00 00 0F in hexadecimal notation - the number of octets in the ASCII encoding of "ECDH-1PU+A128KW", followed by the octets representing that string - 45 43 44 48 2d 31 50 55 2b 41 31 32 38 4b 57 (in hex).  The complete value is therefore 00 00 00 0f 45 43 44 48 2d 31 50 55 2b 41 31 32 38 4b 57

PartyUInfo  This is set to the octets representing the big-endian value 5 followed by the octets of the UTF-8 encoding of "Alice": 00 00 00 05 41 6c 69 63 65 (in hex).

PartyVInfo  This is set to the octets representing the big-endian value 15 followed by the octets of the UTF-8 encoding of "Bob and Charlie": 00 00 00 0f 42 6f 62 20 61 6e 64 20 43 68 61 72 6c 69 65 (in hex).

SuppPubInfo  This is set to the octets representing the 32-bit big-endian encoding of the keydatalen followed by the octets of the cctag.  The complete value is as follows (in hex):

```
00 00 00 80 00 00 00 20 1c b6 f8 7d 39 66 f2 ca
46 9a 28 f7 47 23 ac da 02 78 0e 91 cc e2 18 55
47 07 45 fe 11 9b dd 64
```

SuppPrivInfo  This is set to the empty octet sequence.

Concatenating the parameters AlgorithmID through SuppPrivInfo results in a FixedInfo value in Concatenation Format (as per Section 5.8.2.1.1 of [NIST.800-56A] of (in hexadecimal notation):

```
00 00 00 0f 45 43 44 48 2d 31 50 55 2b 41 31 32
38 4b 57 00 00 00 05 41 6c 69 63 65 00 00 00 0f
42 6f 62 20 61 6e 64 20 43 68 61 72 6c 69 65 00
00 00 80 00 00 00 20 1c b6 f8 7d 39 66 f2 ca 46
9a 28 f7 47 23 ac da 02 78 0e 91 cc e2 18 55 47
07 45 fe 11 9b dd 64
```

Concatenating the round number 1 (00 00 00 01), Z, and the FixedInfo value results in a one-pass KDF round 1 hash input of (hexadecimal):

```
00 00 00 01 32 81 08 96 e0 fe 4d 57 0e d1 ac fc
ed f6 71 17 dc 19 4e d5 da ac 21 d8 ff 7a f3 24
46 94 89 7f 21 57 61 2c 90 48 ed fa e7 7c b2 e4
23 71 40 60 59 67 c0 5c 7f 77 a4 8e ea f2 cf 29
a5 73 7c 4a 00 00 00 0f 45 43 44 48 2d 31 50 55
2b 41 31 32 38 4b 57 00 00 00 05 41 6c 69 63 65
00 00 00 0f 42 6f 62 20 61 6e 64 20 43 68 61 72
6c 69 65 00 00 00 80 00 00 00 20 1c b6 f8 7d 39
66 f2 ca 46 9a 28 f7 47 23 ac da 02 78 0e 91 cc
e2 18 55 47 07 45 fe 11 9b dd 64
```

The resulting derived key, which is the first 16 octets of the round 1 hash output is:

```
df 4c 37 a0 66 83 06 a1 1e 3d 6b 00 74 b5 d8 df
```

The derived key is then used with the "A128KW" key-wrapping algorithm described in [RFC7518] to encrypt the CEK, resulting the following JWE Encrypted Key (in base64url encoding with line breaks for display purposes only):

```
pOMVA9_PtoRe7xXW1139NzzN1UhiFoio8lGto9cf0t8PyU-sjNXH8-LIRLycq8CHJQ
bDwvQeU1cSl55cQ0hGezJu2N9IY0QN
```

## B.10.  Key Agreement for Charlie

The KDF input parameters for Charlie are as follows:

Ze This is set to the ECDH key agreement output between Alice's ephemeral private key and Charlie's static public key.  In this example, Ze is the following octet sequence (in hexadecimal):

```
89 dc fe 4c 37 c1 dc 02 71 f3 46 b5 b3 b1 9c 3b
70 5c a2 a7 2f 9a 23 77 85 c3 44 06 fc b7 5f 10
```

Zs This is set to the ECDH key agreement output between Alice's static private key and Charlie's static public key.  In this example, Zs is the following octet sequence (in hexadecimal):

```
78 fe 63 fc 66 1c f8 d1 8f 92 a8 42 2a 64 18 e4
ed 5e 20 a9 16 81 85 fd ee dc a1 c3 d8 e6 a6 1c
```

Z  Z is the concatenation of Ze followed by Zs. In this example, the value of Z is:

```
89 dc fe 4c 37 c1 dc 02 71 f3 46 b5 b3 b1 9c 3b
70 5c a2 a7 2f 9a 23 77 85 c3 44 06 fc b7 5f 10
78 fe 63 fc 66 1c f8 d1 8f 92 a8 42 2a 64 18 e4
ed 5e 20 a9 16 81 85 fd ee dc a1 c3 d8 e6 a6 1c
```

The FixedInfo value is identical to that computed for Bob. Concatenating the round number 1 (00 00 00 01), Z, and the FixedInfo value results in a one-pass KDF round 1 hash input of (hexadecimal):

```
00 00 00 01 89 dc fe 4c 37 c1 dc 02 71 f3 46 b5
b3 b1 9c 3b 70 5c a2 a7 2f 9a 23 77 85 c3 44 06
fc b7 5f 10 78 fe 63 fc 66 1c f8 d1 8f 92 a8 42
2a 64 18 e4 ed 5e 20 a9 16 81 85 fd ee dc a1 c3
d8 e6 a6 1c 00 00 00 0f 45 43 44 48 2d 31 50 55
2b 41 31 32 38 4b 57 00 00 00 05 41 6c 69 63 65
00 00 00 0f 42 6f 62 20 61 6e 64 20 43 68 61 72
6c 69 65 00 00 00 80 00 00 00 20 1c b6 f8 7d 39
66 f2 ca 46 9a 28 f7 47 23 ac da 02 78 0e 91 cc
e2 18 55 47 07 45 fe 11 9b dd 64
```

The resulting derived key, which is the first 16 octets of the round 1 hash output is:

```
57 d8 12 6f 1b 7e c4 cc b0 58 4d ac 03 cb 27 cc
```

The derived key is then used with the "A128KW" key-wrapping algorithm described in [RFC7518] to encrypt the CEK, resulting the following JWE Encrypted Key (in base64url encoding with line breaks for display purposes only):

56GVudgRLIMEElQ7DpXsijJVRSWUSDNdbWkdV3g0GUNq6hcT_GkxwnxlPIWrTXCqRp
VKQC8fe4z3PQ2YH2afvjQ28aiCTWFE

## B.11. Complete JWE JSON Serialization Representation

The complete JWE JSON Serialization for these values is as follows (with line breaks within values for display purposes only):

```
{
```

```
     "protected":
      "eyJhbGciOiJFQ0RILTFQVStBMTI4S1ciLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwiYXB1Ijoi
       UVd4cFkyVSIsImFwdiI6IlFtOWlJR0Z1WkNCRGFHRnliR2xsIiwiZXBrIjp7Imt0eSI6Ik9L
       UCIsImNydiI6IlgyNTUxOSIsIngiOiJrOW9mX2NwQWFqeTBwb1c1Z2FpeFhHczluSGt3ZzFB
       RnFVQUZhMzlkeUJjIn19",
     "unprotected":
      {"jku":"https://alice.example.com/keys.jwks"},
     "recipients":[
      {"header":
        {"kid":"bob-key-2"},
       "encrypted_key":
        "pOMVA9_PtoRe7xXW1139NzzN1UhiFoio8lGto9cf0t8PyU-sjNXH8-LIRLycq8CHJQbDwv
        eU1cSl55cQ0hGezJu2N9IY0QN"},
      {"header":
        {"kid":"2021-05-06"},
       "encrypted_key":
        "56GVudgRLIMEElQ7DpXsijJVRSWUSDNdbWkdV3g0GUNq6hcT_GkxwnxlPIWrTXCqRpVKQC
         fe4z3PQ2YH2afvjQ28aiCTWFE"}],
     "iv":
      "AAECAwQFBgcICQoLDA0ODw",
     "ciphertext":
      "Az2IWsISEMDJvyc5XRL-3-d-RgNBOGolCsxFFoUXFYw",
     "tag":
      "HLb4fTlm8spGmij3RyOs2gJ4DpHM4hhVRwdF_hGb3WQ"
    }
```

## Appendix C. Document History

-04 Added requirement to include the JWE Authentication Tag in the KDF input when using Key Agreement with Key Wrapping mode to ensure security against insider threats when sending to multiple recipients. Added worked example for a multi-recipient JWE in Appendix B.

-03 Corrected typos and clarified wording. Removed unnecessary references.

-02 Removed two-way interactive handshake protocol section and example after discussion with Hannes Tschofenig.

-01 Added examples in Appendix A and a two-way handshake example.

Added "skid" Header Parameter and registration. Fleshed out Security Considerations.

## Author's Address

Neil Madden
ForgeRock
Broad Quay House
Prince Street
Bristol  BS1 4DJ
United Kingdom

Email: neil.madden@forgerock.com