Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)

Abstract

This document contains a set of examples using JSON Object Signing and Encryption (JOSE) technology to protect data. These examples present a representative sampling of JSON Web Key (JWK) objects as well as various JSON Web Signature (JWS) and JSON Web Encryption (JWE) results given similar inputs.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7520.

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

1. Introduction ....................................................5  
   1.1. Conventions Used in This Document ..........................5  
2. Terminology .....................................................6  
3. JSON Web Key Examples ...........................................6  
   3.1. EC Public Key ..............................................6  
   3.2. EC Private Key .............................................7  
   3.3. RSA Public Key .............................................8  
   3.4. RSA Private Key ............................................8  
   3.5. Symmetric Key (MAC Computation) ...........................10  
   3.6. Symmetric Key (Encryption) ................................11  
4. JSON Web Signature Examples ....................................11  
   4.1. RSA v1.5 Signature ........................................12  
      4.1.1. Input Factors ......................................12  
      4.1.2. Signing Operation ..................................12  
      4.1.3. Output Results .....................................13  
   4.2. RSA-PSS Signature .........................................15  
      4.2.1. Input Factors ......................................15  
      4.2.2. Signing Operation ..................................16  
      4.2.3. Output Results .....................................17  
   4.3. ECDSA Signature ...........................................19  
      4.3.1. Input Factors ......................................19  
      4.3.2. Signing Operation ..................................19  
      4.3.3. Output Results .....................................20  
   4.4. HMAC-SHA2 Integrity Protection ............................21  
      4.4.1. Input Factors ......................................22  
      4.4.2. Signing Operation ..................................22  
      4.4.3. Output Results .....................................23  
   4.5. Signature with Detached Content ...........................24  
      4.5.1. Input Factors ......................................25  
      4.5.2. Signing Operation ..................................25  
      4.5.3. Output Results .....................................26  
   4.6. Protecting Specific Header Fields .........................27  
      4.6.1. Input Factors ......................................27  
      4.6.2. Signing Operation ..................................27  
      4.6.3. Output Results .....................................28  
   4.7. Protecting Content Only ...................................29  
      4.7.1. Input Factors ......................................30  
      4.7.2. Signing Operation ..................................30  
      4.7.3. Output Results .....................................31  
   4.8. Multiple Signatures .......................................32  
      4.8.1. Input Factors ......................................32  
      4.8.2. First Signing Operation ............................33  
      4.8.3. Second Signing Operation ...........................34  
      4.8.4. Third Signing Operation ............................36  
      4.8.5. Output Results .....................................37  
5. JSON Web Encryption Examples ...................................39
5.1. Key Encryption Using RSA v1.5 and AES-HMAC-SHA2 ..........39
  5.1.1. Input Factors ...........................................39
  5.1.2. Generated Factors .......................................41
  5.1.3. Encrypting the Key ......................................41
  5.1.4. Encrypting the Content ..................................42
  5.1.5. Output Results ..........................................43
5.2. Key Encryption Using RSA-OAEP with AES-GCM ..................45
  5.2.1. Input Factors ...........................................46
  5.2.2. Generated Factors .......................................47
  5.2.3. Encrypting the Key ......................................48
  5.2.4. Encrypting the Content ..................................48
  5.2.5. Output Results ..........................................49
5.3. Key Wrap Using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2 ...52
  5.3.1. Input Factors ...........................................53
  5.3.2. Generated Factors .......................................54
  5.3.3. Encrypting the Key ......................................54
  5.3.4. Encrypting the Content ..................................55
  5.3.5. Output Results ..........................................56
5.4. Key Agreement with Key Wrapping Using ECDH-ES and AES-KeyWrap with AES-GCM ..............................59
  5.4.1. Input Factors ...........................................59
  5.4.2. Generated Factors .......................................60
  5.4.3. Encrypting the Key ......................................60
  5.4.4. Encrypting the Content ..................................60
  5.4.5. Output Results ..........................................61
5.5. Key Agreement Using ECDH-ES with AES-CBC-HMAC-SHA2 ........65
  5.5.1. Input Factors ...........................................66
  5.5.2. Generated Factors .......................................66
  5.5.3. Key Agreement ...........................................67
  5.5.4. Encrypting the Content ..................................67
  5.5.5. Output Results ..........................................68
5.6. Direct Encryption Using AES-GCM ..............................70
  5.6.1. Input Factors ...........................................70
  5.6.2. Generated Factors .......................................70
  5.6.3. Encrypting the Content ..................................71
  5.6.4. Output Results ..........................................72
5.7. Key Wrap Using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2 ...73
  5.7.1. Input Factors ...........................................73
  5.7.2. Generated Factors .......................................74
  5.7.3. Encrypting the Key ......................................74
  5.7.4. Encrypting the Content ..................................75
  5.7.5. Output Results ..........................................77
5.8. Key Wrap Using AES-KeyWrap with AES-GCM .....................79
  5.8.1. Input Factors ...........................................79
  5.8.2. Generated Factors .......................................80
  5.8.3. Encrypting the Key ......................................80
  5.8.4. Encrypting the Content ..................................80
  5.8.5. Output Results ..........................................82
1. Introduction

The JSON Object Signing and Encryption (JOSE) technologies -- JSON Web Signature [JWS], JSON Web Encryption [JWE], JSON Web Key [JWK], and JSON Web Algorithms [JWA] -- can be used collectively to encrypt and/or sign content using a variety of algorithms. While the full set of permutations is extremely large, and might be daunting to some, it is expected that most applications will only use a small set of algorithms to meet their needs.

This document provides a number of examples of signing or encrypting content using JOSE. While not exhaustive, it does compile a representative sampling of JOSE features. As much as possible, the same signature payload or encryption plaintext content is used to illustrate differences in various signing and encryption results.

This document also provides a number of example JWK objects. These examples illustrate the distinguishing properties of various key types and emphasize important characteristics. Most of the JWK examples are then used in the signature or encryption examples that follow.

All of the examples contained herein are available in a machine-readable format at <https://github.com/ietf-jose/cookbook>.

1.1. Conventions Used in This Document

This document separates data that are expected to be input to an implementation of JOSE from data that are expected to be generated by an implementation of JOSE. Each example, wherever possible, provides enough information both to replicate the results of this document and to validate the results by running its inverse operation (e.g., signature results can be validated by performing the JWS verify). However, some algorithms inherently use random data; therefore, computations employing them cannot be exactly replicated. Such cases are explicitly stated in the relevant sections.

All instances of binary octet strings are represented using base64url [RFC4648] encoding.

Wherever possible and unless otherwise noted, the examples include the JWS or JWE Compact Serialization, general JWS or JWE JSON Serialization, and flattened JWS or JWE JSON Serialization.

All of the examples in this document have whitespace added to improve formatting and readability. Except for JWE Plaintext or JWS Payload content, whitespace is not part of the cryptographic operations nor the exchange results.
Unless otherwise noted, the JWE Plaintext or JWS Payload content does include " " (U+0020 SPACE) characters. Line breaks (U+000A LINE FEED) replace some " " (U+0020 SPACE) characters to improve readability but are not present in the JWE Plaintext or JWS Payload.

2. Terminology

This document inherits terminology regarding JSON Web Signature (JWS) technology from [JWS], terminology regarding JSON Web Encryption (JWE) technology from [JWE], terminology regarding JSON Web Key (JWK) technology from [JWK], and terminology regarding algorithms from [JWA].

3. JSON Web Key Examples

The following sections demonstrate how to represent various JWK and JWK Set objects.

3.1. EC Public Key

This example illustrates an Elliptic Curve (EC) public key. This example is the public key corresponding to the private key in Figure 2.

Note that whitespace is added for readability as described in Section 1.1.

```
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLGsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sY1_oJXu9A5RkTkqjvqjyeKWF-7ytDyRXkggCF5cj0Kt",
  "y": "AdymlHvOiIxKehayXQhNCvDkX4h9htZaCJN34kfmC6pV5ohQHiraVysSudaQkAgDFrWqrJmbnX9cwIGfP-HqHZR1"
}
```

Figure 1: Elliptic Curve P-521 Public Key

The field "kty" value of "EC" identifies this as an Elliptic Curve key. The field "crv" identifies the curve, which is curve P-521 for this example. The values of the fields "x" and "y" are the base64url-encoded X and Y coordinates (respectively).
The values of the fields "x" and "y" decoded are the octets necessary to represent each full coordinate to the order of the curve. For a key over curve P-521, the values of the fields "x" and "y" are exactly 66 octets in length when decoded, padded with leading zero (0x00) octets to reach the expected length.

3.2. EC Private Key

This example illustrates an Elliptic Curve private key. This example is the private key corresponding to the public key in Figure 1.

Note that whitespace is added for readability as described in Section 1.1.

```json
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sY1_oJXu9A5RKTKqjqv+jekWF-7ytDyRXyGF5c+j0Kt",
  "y": "AdymlvOlxXkEhayXQnNCvDX4h9htZacJN34kfmC6pV5oHQhiraVysUdaQkAgDPDrwQrJmbnX9cwlGFP-HqHZRI",
  "d": "AAhRONZr9cqXX1hg-RoI6R1tX5p2rUAYdmpH2oC1XNM56KtsrX6zbKipQrCWW9CGZH3T4ubpn0TKLDYJ_ff3_rJt"
}
```

Figure 2: Elliptic Curve P-521 Private Key

The field "kty" value of "EC" identifies this as an Elliptic Curve key. The field "crv" identifies the curve, which is curve P-521 (also known as SECG curve secp521r1) for this example. The values of the fields "x" and "y" are the base64url-encoded X and Y coordinates (respectively). The field "d" value is the base64url-encoded private key.

The values of the fields "d", "x", and "y" decoded are the octets necessary to represent the private key or each full coordinate (respectively) to the order of the curve. For a key over curve P-521, the values of the "d", "x", and "y" fields are each exactly 66 octets in length when decoded, padded with leading zero (0x00) octets to reach the expected length.
3.3. RSA Public Key

This example illustrates an RSA public key. This example is the public key corresponding to the private key in Figure 4.

Note that whitespace is added for readability as described in Section 1.1.

```
{
  "kty": "RSA",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "n": "n4EPtAOCc9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rHVTdT-0-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHze_AJMScpMaJMRBSFKrKb2wqVwGU_NsYOYL-QtiWN21bzce6XC0dApr5ydQLRhQkHHiq3RbordaZ6AJo-oBhqFEHYpPe7Tpe-0fVfHd1E6cS6M1FzcD1NNLYD51FHpPI9bTwJlsde3uhGgC02CuEHg81hzwOHrtIQbSOFVb9k3-tVTU4fg_3L_vniUFAKwuCLqKnS2Ywdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUrKz5iaiQkqgc5gHdrNP5zw",
  "e": "AQAB"
}
```

Figure 3: RSA 2048-Bit Public Key

The field "kty" value of "RSA" identifies this as an RSA key. The fields "n" and "e" values are the modulus and (public) exponent (respectively) using the minimum octets necessary.

For a 2048-bit key, the field "n" value is 256 octets in length when decoded.

3.4. RSA Private Key

This example illustrates an RSA private key. This example is the private key corresponding to the public key in Figure 3.

Note that whitespace is added for readability as described in Section 1.1.
{ "kty": "RSA", "kid": "bilbo.baggins@hobbiton.example", "use": "sig", "n": "n4EPtAOC9AlkeQHPzHStgAbgs7bTzIwUBzDrR8_KuKPEHId4rHVTvT-O-XV2jRojdNhjxjWTDvNd7nqQ0Ve1ZQHsz_AJmScPmaJMRBSFKrb2wqV wGU_NsYOL-QtiWN21bzc6eX0CdApr5ydQLRqkHHig3RbordaZ6A-j-oBhQFEHYPpe7Tpe-0fVFDh16e6s6M1FzcDINNYL51FHP19bTwJ1sde 3uhGqC0ZCuHq81hzwOhrITqbs0FVvb9k3-tVTU4f3g_LnvnIUPAKwUC LqKmS2BhYwtd_mSnbLY7h_qixoR7jig3__kRhuaxwUKrZi5aiQkqg5c5 HdrNP5zw", "e": "AQAB", "d": "bWUC9B-EFRI08kg0yuGPvMNKvYNTBi_kiH9k20eT-O1q_I78e iZkpXXXQOUTEs2LsNRS-8uJbVQ-A1irkwMSMkK1j3XTGdhrhCku9qRld Y7sNA-AKZgH-Q661_42rINLRCe8W-nZ34ui_qOfkLnK9QWDcppiA-sb MrWW5SDFv2MUBYwkmHTEmLYcGqO0e4nogeqh1hExBTHB0BdKMiuPhUg18U 61-DqElWxqg82sXt2h-oLMnT3046AOYJoRi8975SUqFCGwEWTBnP5dJ dl18Kkhyv071hfSjdrPm5Pl1y21hsFf4L_mHCuoFau7gdsPfHpxxJVoC OpBrQzwQ", "p": "3S1xg_DwTXJcb6095RoXyqQCAZ5RnAv21no1yHtnUeFx7fpA7_9nR ao7HX_-SffQGqutao2TdDAWV4upk8rwj9R0Az2N2fvuIAMr_WcSmgspeNqNev71TlvEsnh8UMt-n5CafhkikzhEsrmdnH6Lx0rvRJlsPp6Zv8 buQ0k", "q": "uKE2dh-cf66ERF4k4e_jy78GFpYUIaUyoSSJuBzp3Cub3OCgs6grT 8bR_cu0DmlMZWmtdqDy195hrUeq3MP15vMMonN81Hezu2lmKwqW7an V5UzhM1i27z4YMuUFwObvY989kXxVRD-hdqRxhL5qAz192Z3pVFj0 7pFc", "dp": "BBFVvXxvJr2L-GQ7v3y9r6kW5g9sAhXbWSWuzp19TVgL-I-YV85q 11kbrxQc5-eSXXR3-TanevuRPRt5OBODiMGQp8pb26gljyfKU_E9xn -RULHz0-e9D9e9xLKXD4VGhnpz-PfQ_q29pk5xWhojP009qf1HvChrixR 59ehik", "dq": "CLDmdGdhuyc1c9o7r84rEUvn7pqQ6FP83Y-iB2x5NT-TpnOZKF1pEr AMVeKZFe141D1HqQ6LSM0WIsObWmxYWZdMs6I6q51tbWQGc3gnjJK bi_7k_vjQGHwHxgPaXZ2pVp-zYEkDERuf-ry4c_Zl1C9aqC2yeL6kdK T1cYF8", "q1": "3PiqvXQNU0zwMeE-sbVzgi289XP9CQF3VWqPzMKnIgQp7_Tugo6-N ZBkCQsMf3HaEGbjTVjs_4cK8-TRXvaKe-7ZMaQj8VFbdYkssb0NKDdHe j-GtisedAVV7t7dc0fXwxfUHPq7FoCrjFJ6h6ZEpMF6mujs4qMpsP z8a14" }

Figure 4: RSA 2048-Bit Private Key
The field "kty" value of "RSA" identifies this as an RSA key. The fields "n" and "e" values are the base64url-encoded modulus and (public) exponent (respectively) using the minimum number of octets necessary. The field "d" value is the base64url-encoded private exponent using the minimum number of octets necessary. The fields "p", "q", "dp", "dq", and "qi" are the base64url-encoded additional private information using the minimum number of octets necessary.

For a 2048-bit key, the field "n" is 256 octets in length when decoded, and the field "d" is not longer than 256 octets in length when decoded.

3.5. Symmetric Key (MAC Computation)

This example illustrates a symmetric key used for computing Message Authentication Codes (MACs).

Note that whitespace is added for readability as described in Section 1.1.

```
{
  "kty": "oct",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037",
  "use": "sig",
  "alg": "HS256",
  "k": "hJtXIZ2uSN5kbQfbtTNWbpmhkV8FJG-0nb6mxCcYg"
}
```

Figure 5: HMAC SHA-256 Symmetric Key

The field "kty" value of "oct" identifies this as a symmetric key. The field "k" value is the symmetric key.

When used for the signing algorithm "HS256" (HMAC-SHA256), the field "k" value is 32 octets (or more) in length when decoded, padded with leading zero (0x00) octets to reach the minimum expected length.
3.6. Symmetric Key (Encryption)

This example illustrates a symmetric key used for encryption.

Note that whitespace is added for readability as described in Section 1.1.

```
{
    "kty": "oct",
    "kid": "1e571774-2e08-40da-8308-e8d68773842d",
    "use": "enc",
    "alg": "A256GCM",
    "k": "AAPapAv4LbFbiVawEjagUBluYqN5rhna-8nuldDvOx8"
}
```

Figure 6: AES 256-Bit Symmetric Encryption Key

The field "kty" value of "oct" identifies this as a symmetric key. The field "k" value is the symmetric key.

For the content encryption algorithm "A256GCM", the field "k" value is exactly 32 octets in length when decoded, padded with leading zero (0x00) octets to reach the expected length.

4. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

All of the signature examples use the following payload content (an abridged quote from "The Fellowship of the Ring" [LOTR-FELLOWSHIP]), serialized as UTF-8. The payload is presented here as a series of quoted strings that are concatenated to produce the JWS Payload. The sequence "\xe2\x80\x99" is substituted for (U+2019 RIGHT SINGLE QUOTATION MARK), and quotation marks (U+0022 QUOTATION MARK) are added for readability but are not present in the JWS Payload.

"It\xe2\x80\x99s a dangerous business, Frodo, going out your "
"door. You step onto the road, and if you don’t keep your feet, "
"there\xe2\x80\x99s no knowing where you might be swept off "
"to."

Figure 7: Payload Content Plaintext
The payload -- with the sequence "\xe2\x80\x99" replaced with (U+2019 RIGHT SINGLE QUOTATION MARK) and quotations marks (U+0022 QUOTATION MARK) are removed -- is encoded as UTF-8 and then as base64url [RFC4648]:

SXTigJ1zIGEgZGFuZ2Vyb3VzIGJ1c2luZ2xvcm9kbywgZ29pbmcgZ2FtZ2UgZG9vcyBmb24tZG9vcyBkaXR0aDogUGFydmcgZ3Jvc3QgZG9vcyBrZnJhbWVzbyBzdXBwbGluZz4K

Figure 8: Payload Content, base64url-encoded

4.1. RSA v1.5 Signature

This example illustrates signing content using the "RS256" (RSASSA-PKCS1-v1_5 with SHA-256) algorithm.

Note that whitespace is added for readability as described in Section 1.1.

4.1.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- RSA private key; this example uses the key from Figure 4.
- "alg" parameter of "RS256".

4.1.2. Signing Operation

The following is generated to complete the signing operation:

- JWS Protected Header; this example uses the header from Figure 9, encoded using base64url [RFC4648] to produce Figure 10.

```
{
  "alg": "RS256",
  "kid": "bilbo.baggins@hobbiton.example"
}
```

Figure 9: JWS Protected Header JSON
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9

Figure 10: JWS Protected Header, base64url-encoded

The JWS Protected Header (Figure 10) and JWS Payload (Figure 8) are combined as described in Section 5.1 of [JWS] to produce the JWS Signing Input (Figure 11).

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9

Figure 11: JWS Signing Input

Performing the signature operation over the JWS Signing Input (Figure 11) produces the JWS Signature (Figure 12).

MRjdkly7_-oTPTS3AXP4liQIGka80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmKZopdHM1O2U4mwzJdQx996ipv83xuglI7PNDi84wnB-BDkoBwa78185hX-Es4J1wmDLJK3lf0Wra-xtLOHnlutYv746IYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8wW1Kt9eRo4QFocSadnHGFxnt8Is9u2pERV0ePQPdLuW3IS_de3xyIrDaLDjluFxrUAhb6I2aXic1U12podGU0KLUQSE_oI-ZnmmKJ3F4uOZDnd6QZWJushZ41Ax_f c1e8u9ipH84ogoree7vjbU5y18kDquDg

Figure 12: JWS Signature, base64url-encoded

4.1.3. Output Results

The following compose the resulting JWS object:

- JWS Protected Header (Figure 9)
- Payload content (Figure 8)
- Signature (Figure 12)
The resulting JWS object using the JWS Compact Serialization:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAAGiYm10b24uZX
hhbXBsZSJ9

SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBgc9kbywgZ29pbmcgb3v0IHB
1vdXIGZG9vci4gW91IHOUZAgb25obyB0AGUgc9m9hZCgwYW5kIGJ1vdSBkb24
dCBrZWVwIHlvdXIGZmVlcmdGh1cmXiczIGI5IGItb3dpbmcgd2hlc 
UgeW91I1p2z010GJ1IHN3X0B0IG9mZiB0by4

MRjdkly7_-oTPTS3AXP41iQlGKa8O0A0zmTvuV5MEaHoxnW2e5C5Z51KtainoFmK
ZopdHM102U4mzwJdQx9961vp83xugl1I7PDNDI8wnB-BDkoBwa7818shX-Es4J
IwmdLJJK3lfWRA-XtLO0NtuYv7461YTh_qHRD66BNt1uSNCrUCTJd5aAE6x8w
W1Kt9eRo4QPoSadnHXFxt8Is9uzpERV0ePPQdLuW3IS_dezxyIrDaLgdj1uP
xUAhb6L2aXic1U12podG0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Ax_f
ce8u9ipH84ogoreee7vjbU5y18kDquDg

Figure 13: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{ "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBgc9kbywgZ29pbmcgb3v0IHB1vdXIGZG9vci4gW91IHOUZAgb25obyB0AGUgc9m9hZCgwYW5kIGJ1vdSBkb24ndCBrZWVwIHlvdXIGZmVlcmdGh1cmXiczIGI5IGItb3dpbmcgd2hlc 
UgeW91I1p2z010GJ1IHN3X0B0IG9mZiB0by4", "signatures": [ { "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAAGiYm10b24uZXhhbXBsZSJ9", "signature": "MRjdkly7_-oTPTS3AXP41iQlGKa8O0A0zmTvuV5MEaHoxnW2e5C5Z51KtainoFmKZopdHM102U4mzwJdQx9961vp83xugl1I7PDNDI8wnB-BDkoBwa7818shX-Es4JIwmdLJJK3lfWRA-XtLO0NtuYv7461YTh_qHRD66BNt1uSNCrUCTJd5aAE6x8wW1Kt9eRo4QPoSadnHXFxt8Is9uzpERV0ePPQdLuW3IS_dezxyIrDaLgdj1uPxAhb6L2aXic1U12podG0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Ax_fce8u9ipH84ogoreee7vjbU5y18kDquDg" } ] } 
```

Figure 14: General JWS JSON Serialization
The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vybj3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcg3V0IHldXIG9vcici4gWW91IHN0ZAXAgb250byB0aGUgcm9hZCwgYW5kIGlmlHldXIGb29nbmc2YXVsdGlucyBhIHN0ZXAgb24ndCBzZWNvIHldXIGZmVldCwgGh1cmtqJlZIG5vIGtub3dpbmcg2hlcmVgWW91IHN0ZAXAgb24ndCBzZWNvIHldXIGZmVldCgwNQ==",
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLiIiLCJ0eXBlIjoiRkA0d0VcX2Fpc2l6ZSIsInN1YiI6IjMyMjI1IiwidHlwZSI6W10sImlzMiI6ImJpdWVzdG9yIiwiY29kZSI6IjJoIiwibWFsZSI6IjJvc3RlZjoyci8iLCJzdWIiOiIiLCJpYXQiOjE2Mzg2NzY1MzUsImZ1bmN0aW9uIjp7InVzZXJfaWQiOiJidWNrZ9hIiwiY2hhbGl0dFJhbWVzIjoiMjAxMjIwMjMzNiIsImV4cCI6IjE2Mzg1ODU0MjUsImluaXRpYWwiOiJfX210cW9yZSIsImFjY291bnQiOjE2NjY4ODk1MjUsImlkIjoiMjAxIiwiaGF2b3J0IjoiaHR0cHM6Ly9zcGxvYWRzLmNvbS8iLCJ0eXBlIjoiRkA0d0VcX2Fpc2l6ZSIsInN1YiI6IjMyMjI1IiwidHlwZSI6W10sImlzMiI6ImJpdWVzdG9yIiwiY29kZSI6IjJoIiwibWFsZSI6IjJvc3RlZjoyci8iLCJzdWIiOiIiLCJpYXQiOjE2Mzg2NzY1MzUsImZ1bmN0aW9uIjp7InVzZXJfaWQiOiJidWNrZ9hIiwiY2hhbGl0dFJhbWVzIjoiMjAxMjIwMjMzNiIsImV4cCI6IjE2Mzg1ODU0MjUsImluaXRpYWwiOiJfX210cW9yZSIsImFjY291bnQiOjE2NjY4ODk1MjUsImlkIjoiMjAxIiwiaGF2b3J0IjoiMjAxIiwiZG9jZ3JvdW50cyI6IjIiLCJyZXNpemUiOiIiLCJzZXNzaW9uIjoiMjAxIiwiY29tcG9ydFZlcml0ZSI6IjE2Mzg2NzY1MzUsImZpcnN0IjoiMjAxIiwicGF5bGVzIjoiMjAxIiwiZGVzaWduZ3MiOiIiLCJyaWVwcyI6IjEyMzgwNjY5NDI1OTMxOTk3NTg1Iiwic2Vzc2l0ZSI6IjIwMjIifQ",
  "signature": "MRjdkly7_-oTPTS3AXP4iQIGKa80A02mTvuV5EAHoxnW2e5CZ51Ktai0oFmkZopdHM1O2U4mzwJdq996v9pr83xug1l7FND184wN-p-BDkoBwa78185h-8Es4JlwmdLJK3iFWRxBAxTL0RnlruYy746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8W1Kt9eRo4QpocSdnHFxnt8Is9UzpERV0eFPQdluW3IS_de3xyIrDaLDgjluFxUAhb6L2AxIc1U12podGU0KLQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_fciE8u9ipH84ogoree7vjBu5y18kDquDg"
}
```

Figure 15: Flattened JWS JSON Serialization

4.2. RSA-PSS Signature

This example illustrates signing content using the "PS384" (RSASSA-PSS with SHA-384) algorithm.

Note that RSASSA-PSS uses random data to generate the signature; it might not be possible to exactly replicate the results in this section.

Note that whitespace is added for readability as described in Section 1.1.

4.2.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- RSA private key; this example uses the key from Figure 4.
- "alg" parameter of "PS384".
4.2.2. Signing Operation

The following is generated to complete the signing operation:

- JWS Protected Header; this example uses the header from Figure 16, encoded using base64url [RFC4648] to produce Figure 17.

```json
{
  "alg": "PS384",
  "kid": "bilbo.baggins@hobbiton.example"
}
```

**Figure 16: JWS Protected Header JSON**

eyJhbGciOiJQUzI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9

**Figure 17: JWS Protected Header, base64url-encoded**

The JWS Protected Header (Figure 17) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 18).

eyJhbGciOiJQUzI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9

**Figure 18: JWS Signing Input**

Performing the signature operation over the JWS Signing Input (Figure 18) produces the JWS Signature (Figure 19).

cu22eBqkYDKgITlTpzDXGvaFFz6WGoz7fUDcfT0kk0y42miAh2qyBzk1xEsnk2IPN-tPd6VrkhKqsGqDqHCdP608TB5dDDIt11Vo6_1OLPpcbUrhiUSMxbbXUvdvWXzg-UD8biiriQF1fzZ8zGWVsdiNAUf82nyPEgVFm4422dNiVJRmBqrYRXe8P_ijQ7p8Vdz0TTtxUeT31m89shnr21fJT8ImUjvAA2XezZMlp8cBE5awDzT0qI0n6ulPlaCN_2_jLAeQTlqRhtfa64QQSUmFAajVKPbByi7xho0UTOcbH510a6GYmJUApfMwJwZ6oD4ifKo8DYM-X72Eaw

**Figure 19: JWS Signature, base64url-encoded**
4.2.3. Output Results

The following compose the resulting JWS object:

- JWS Protected Header (Figure 17)
- Payload content (Figure 8)
- Signature (Figure 19)

The resulting JWS object using the JWS Compact Serialization:

```
eyJhbGciOiJQUzMiM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2VybjVzIGJ1c2luZWNzLCBzb3V0IGJ1c2luZWNzLCAgd2hlcmUgeW91
IG1pZ2h0IGJlIHN3ZHJlZ3NzLCAuU01lZ2luZw==
```

Figure 20: JWS Compact Serialization
The resulting JWS object using the general JWS JSON Serialization:

```json
{
    "payload": "SXTigJlzIGEgZGFuZ2VybjZvIGJ1c2luZXNzLCBMcm9kbywgZ29pbmcgb3V0ImlvdXIgZGVmd2lvb24gY29tLmJhZ2dpbnNAaGB0b24uZWNobXBsbGV5",
    "signatures": [
        {
            "protected": "eyJhbGciOiJQUzIzMiIsImtpZCI6IjI2NjE2NzU1NzQ5Iiwia2V5IjoiNWZmM2Q5ZjctNjAxNy00MmU0LWJhOGEtMzJiM2UxNjU4MDYzIiwia3ViIjoyNCJ9",
            "signature": "cu22eBqkYDKgI1TzpDXGvaFfz6WGo7fUDcfT0kcyO4mJn55qBzK1xEnsk2Ip6vRk1HkqGqD9C6D8OTTB5dDDItUVo6_1LOPzcbUrhIUSMxBbXVdvWZg-UD8biReQFf2z8zGWVsidNAuf82nFEGFVFO422zDQiiVJRMkqRvX8P_iq72p8Vz07TrxUeT31m8d9shrr1jFT8ImUjvAAZ2M1p8cBE5awDz70qI0n6uiPlaCN_2_jLaEQTlqRhtfa64QQSMFAYJKPBByi7xho0uTOcbH510a66YMjUAFmWjw26D4ifKo8DYM-X72Eaw"
        }
    ]
}
```

Figure 21: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
    "payload": "SXTigJlzIGEgZGFuZ2VybjZvIGJ1c2luZXNzLCBMcm9kbywgZ29pbmcgb3V0ImlvdXIgZGVmd2lvb24gY29tLmJhZ2dpbnNAaGB0b24uZWNobXBsbGV5",
    "protected": "eyJhbGciOiJQUzIzMiIsImtpZCI6IjI2NjE2NzU1NzQ5Iiwia2V5IjoiNWZmM2Q5ZjctNjAxNy00MmU0LWJhOGEtMzJiM2UxNjU4MDYzIiwia3ViIjoyNCJ9",
    "signature": "cu22eBqkYDKgI1TzpDXGvaFfz6WGo7fUDcfT0kcyO4mJn55qBzK1xEnsk2Ip6vRk1HkqGqD9C6D8OTTB5dDDItUVo6_1LOPzcbUrhIUSMxBbXVdvWZg-UD8biReQFf2z8zGWVsidNAuf82nFEGFVFO422zDQiiVJRMkqRvX8P_iq72p8Vz07TrxUeT31m8d9shrr1jFT8ImUjvAAZ2M1p8cBE5awDz70qI0n6uiPlaCN_2_jLaEQTlqRhtfa64QQSMFAYJKPBByi7xho0uTOcbH510a66YMjUAFmWjw26D4ifKo8DYM-X72Eaw"
}
```

Figure 22: Flattened JWS JSON Serialization
4.3. ECDSA Signature

This example illustrates signing content using the "ES512" (Elliptic Curve Digital Signature Algorithm (ECDSA) with curve P-521 and SHA-512) algorithm.

Note that ECDSA uses random data to generate the signature; it might not be possible to exactly replicate the results in this section.

Note that whitespace is added for readability as described in Section 1.1.

4.3.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- EC private key on the curve P-521; this example uses the key from Figure 2.
- "alg" parameter of "ES512".

4.3.2. Signing Operation

The following is generated before beginning the signature process:

- JWS Protected Header; this example uses the header from Figure 23, encoded using base64url [RFC4648] to produce Figure 24.

```
{
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
}
```

Figure 23: JWS Protected Header JSON

Figure 24: JWS Protected Header, base64url-encoded
The JWS Protected Header (Figure 24) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 25).

Performing the signature operation over the JWS Signing Input (Figure 25) produces the JWS Signature (Figure 26).

The following compose the resulting JWS object:

- JWS Protected Header (Figure 24)
- Payload content (Figure 8)
- Signature (Figure 26)

The resulting JWS object using the JWS Compact Serialization:

eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBvcGVyZ2h5IGRva2V0eXMgVGhlIHByb2ZpbGUgYmVnaW5lZCBzdHJva2UgcmVzdCByb2NpdGVzIGFubm90IHRvIGl0IGRvY3MgY2VsbCIgYW5kIG1pZ2h0IGJlIHN3ZXB0IHRvIG1pZ2h0IGJlIHN3ZXB0

4.3.3. Output Results

Figure 25: JWS Signing Input

Figure 26: JWS Signature, base64url-encoded

Figure 27: JWS Compact Serialization
The resulting JWS object using the general JWS JSON Serialization:

```
{
    "payload": "SXTigJlzIGEgZGFuZ2VybdVzIGJlc2luZXNzLCBgm9kbywgZ29pbmcgb3V0I1dvXigZG9vci4gWW91IHNOZXAgb25obyBOaGUgcm9hZCwgYW5kIGlmIGlvdSBkdWlpbmgd2hlcUgwW91IGl0Z2h0IGN3ZXVB0IG9mZiB0by4",
    "signatures": [
        {
            "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
            "signature": "AE_R_YZCChjn4791jSQCrdPZCNYqHXCTZH0-JZGYNaAJP2kqalUIlUnC9qvbv9Plon7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDyYjnAMOxXps7XrT0lw-kvAD890j18e2puQens_IEKBpHAB1sbEPX6sFY80cGDqoRuBomu9xQ2"
        }
    ]
}
```

Figure 28: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
    "payload": "SXTigJlzIGEgZGFuZ2VybdVzIGJlc2luZXNzLCBgm9kbywgZ29pbmcgb3V0I1dvXigZG9vci4gWW91IHNOZXAgb25obyBOaGUgcm9hZCwgYW5kIGlmIGlvdSBkdWlpbmgd2hlcUgwW91IGl0Z2h0IGN3ZXVB0IG9mZiB0by4",
    "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
    "signature": "AE_R_YZCChjn4791jSQCrdPZCNYqHXCTZH0-JZGYNaAJP2kqalUIlUnC9qvbv9Plon7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDyYjnAMOxXps7XrT0lw-kvAD890j18e2puQens_IEKBpHAB1sbEPX6sFY80cGDqoRuBomu9xQ2"
}
```

Figure 29: Flattened JWS JSON Serialization

4.4. HMAC-SHA2 Integrity Protection

This example illustrates integrity protecting content using the "HS256" (HMAC-SHA-256) algorithm.

Note that whitespace is added for readability as described in Section 1.1.
4.4.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- HMAC symmetric key; this example uses the key from Figure 5.
- "alg" parameter of "HS256".

4.4.2. Signing Operation

The following is generated before completing the signing operation:

- JWS Protected Header; this example uses the header from Figure 30, encoded using base64url [RFC4648] to produce Figure 31.

```
{  
  "alg": "HS256",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
}
```

Figure 30: JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
```

Figure 31: JWS Protected Header, base64url-encoded

The JWS Protected Header (Figure 31) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 32).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
```

Figure 32: JWS Signing Input
Performing the signature operation over the JWS Signing Input (Figure 32) produces the JWS Signature (Figure 33).

Figure 33: JWS Signature, base64url-encoded

4.4.3. Output Results

The following compose the resulting JWS object:

- JWS Protected Header (Figure 31)
- Payload content (Figure 8)
- Signature (Figure 33)

The resulting JWS object using the JWS Compact Serialization:

Figure 34: JWS Compact Serialization
The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2VybjZvIzIGJlc2l0ZXJzLCBGCm9kbywg
Z29pmbmcgb3V0I1dvXG1gZ29vci4gWW91IHN0ZXAgb250byB0aGUgcmlnaHJpbmcgb2Yg
ZCw5YW5kIGlmaWxlIHdvBGRzZ2VvbWluZ3IgZ2V0IHN0ZXAgb2Yg
"signature": [
  {
    "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
RkOWItNDCxYi1iZmQ2LWVjMxNGJjNzAzNyJ9",
    "signature": "s0h6KThzkfBBKlSpWlh84VsJ2FTsPPqMDA7g1Md7p
0"
  }
]
```

**Figure 35: General JWS JSON Serialization**

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2VybjZvIzIGJlc2l0ZXJzLCBGCm9kbywg
Z29pmbmcgb3V0I1dvXG1gZ29vci4gWW91IHN0ZXAgb250byB0aGUgcmlnaHJpbmcgb2Yg
ZCw5YW5kIGlmaWxlIHdvBGRzZ2VvbWluZ3IgZ2V0IHN0ZXAgb2Yg
"protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOW
ItNDCxYi1iZmQ2LWVjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBKlSpWlh84VsJ2FTsPPqMDA7g1Md7p
0"
}
```

**Figure 36: Flattened JWS JSON Serialization**

4.5. Signature with Detached Content

This example illustrates a signature with detached content. This example is identical to other examples in Section 4, except the resulting JWS objects do not include the JWS Payload field. Instead, the application is expected to locate it elsewhere. For example, the signature might be in a metadata section, with the payload being the content.

Note that whitespace is added for readability as described in Section 1.1.
4.5.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- Signing key; this example uses the AES symmetric key from Figure 5.
- Signing algorithm; this example uses "HS256".

4.5.2. Signing Operation

The following is generated before completing the signing operation:

- JWS Protected Header; this example uses the header from Figure 37, encoded using base64url [RFC4648] to produce Figure 38.

```json
{
  "alg": "HS256",
  "kid": "018c0ae5-4d9b-471b-bfd6-eeff314bc7037"
}
```

Figure 37: JWS Protected Header JSON

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9

Figure 38: JWS Protected Header, base64url-encoded

The JWS Protected Header (Figure 38) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 39).

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9.

Figure 39: JWS Signing Input
Performing the signature operation over the JWS Signing Input (Figure 39) produces the JWS Signature (Figure 40).

s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0

Figure 40: JWS Signature, base64url-encoded

4.5.3. Output Results

The following compose the resulting JWS object:

- JWS Protected Header (Figure 38)
- Signature (Figure 40)

The resulting JWS object using the JWS Compact Serialization:

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9

s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0

Figure 41: General JWS JSON Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

Figure 42: General JWS JSON Serialization
The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOTkzIlNDcXl1IiIsZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspWl84VsJZFTspPqMDA7g1Md7p0"
}
```

**Figure 43: Flattened JWS JSON Serialization**

4.6. Protecting Specific Header Fields

This example illustrates a signature where only certain Header Parameters are protected. Since this example contains both unprotected and protected Header Parameters, only the general JWS JSON Serialization and flattened JWS JSON Serialization are possible.

Note that whitespace is added for readability as described in Section 1.1.

4.6.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- Signing key; this example uses the AES symmetric key from Figure 5.
- Signing algorithm; this example uses "HS256".

4.6.2. Signing Operation

The following are generated before completing the signing operation:

- JWS Protected Header; this example uses the header from Figure 44, encoded using base64url [RFC4648] to produce Figure 45.
- JWS Unprotected Header; this example uses the header from Figure 46.

```json
{
  "alg": "HS256"
}
```

**Figure 44: JWS Protected Header JSON**
The JWS Protected Header (Figure 45) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 47).

Performing the signature operation over the JWS Signing Input (Figure 47) produces the JWS Signature (Figure 48).

The following compose the resulting JWS object:

- JWS Protected Header (Figure 45)
- JWS Unprotected Header (Figure 46)
- Payload content (Figure 8)
- Signature (Figure 48)

The JWS Compact Serialization is not presented because it does not support this use case.
The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJ1lzIGEgZGFuZ2VybiZvZIGJ1c2luZXNzLCBzc3MvcmFuc3Bv
Z29pZmci4gWW91IHNoZSB0aGUgZ2V0aW50IHJlZmZlciB0byB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "bWUSVaxorn7bEF1djytBd0Kh70Ly5pvbomzMWSo20"
    }
  ]
}
```

Figure 49: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJ1lzIGEgZGFuZ2VybiZvZIGJ1c2luZXNzLCBzc3MvcmFuc3Bv
Z29pZmci4gWW91IHNoZSB0aGUgZ2V0aW50IHJlZmZlciB0byB0by4",
  "protected": "eyJhbGciOiJIUzI1NiJ9",
  "header": {
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "bWUSVaxorn7bEF1djytBd0Kh70Ly5pvbomzMWSo20"
}
```

Figure 50: Flattened JWS JSON Serialization

4.7. Protecting Content Only

This example illustrates a signature where none of the Header Parameters are protected. Since this example contains only unprotected Header Parameters, only the general JWS JSON Serialization and flattened JWS JSON Serialization are possible.

Note that whitespace is added for readability as described in Section 1.1.
4.7.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- Signing key; this example uses the AES symmetric key from Figure 5.
- Signing algorithm; this example uses "HS256".

4.7.2. Signing Operation

The following is generated before completing the signing operation:

- JWS Unprotected Header; this example uses the header from Figure 51.

```json
{
  "alg": "HS256",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
}
```

Figure 51: JWS Unprotected Header JSON

The empty string (as there is no JWS Protected Header) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 52).

```
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZ2XNzLCBGbcm9kbywgZ29pbmcgb3V0IHRlYXJlIGZ1bmN0aW9uIGFyZWhpc3RyaW5ncyBvZiB3aXRoIHRvb2tlbiB0byB0aGUgcm9hZ2UgYW5kIGFyZWhpc3RyaW5ncyBhbmQgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
```

Figure 52: JWS Signing Input

Performing the signature operation over the JWS Signing Input (Figure 52) produces the JWS Signature (Figure 53).

```
xuLifqLGiblpv9zBpuZczWhNjIgARaLV3UxvxhJxZuk
```

Figure 53: JWS Signature, base64url-encoded
4.7.3. Output Results

The following compose the resulting JWS object:

- JWS Unprotected Header (Figure 51)
- Payload content (Figure 8)
- Signature (Figure 53)

The JWS Compact Serialization is not presented because it does not support this use case.

The resulting JWS object using the general JWS JSON Serialization:

```json
{
  "payload": "SXTigJ1zIGEgZGFuZ2Vybj3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH1vdXIgZG9vc3cgWW91IHN0ZXAgb250byB0aGUgcm9kIHdlcmVldCBrZWVwIH1vdXIgZmVldCByZmVjdCgyJGJ1zIGvIGtub3dpbmcgd2hlcUg5IGZiB0b3xk",
  "signatures": [
    {
      "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "xuLifqLGiblpv9zBpuZczWhNjlgARalLV3UxvkhJxZuk"
    }
  ]
}
```

Figure 54: General JWS JSON Serialization
The resulting JWS object using the flattened JWS JSON Serialization:

```
{
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIIGJ1c2luZXMzLCBgcml9kbywg
Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
ZCwqYW5kIGlmIHlvdSBwb24ndCBrZWVwIHlvdXIgZmV1dCBrZGVhIcmXig
JlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IHN0ZXAgb250byB0
ZiB0byB4",
    "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eeff314bc7037"
    },
    "signature": "xuLifqLGiblpv9zBpu2czWhNj1gARaLV3UvxhJxZuk"
}
```

Figure 55: Flattened JWS JSON Serialization

4.8.  Multiple Signatures

This example illustrates multiple signatures applied to the same payload. Since this example contains more than one signature, only the JSON General Serialization is possible.

Note that whitespace is added for readability as described in Section 1.1.

4.8.1.  Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.
- Signing keys; this example uses the following:
  - RSA private key from Figure 4 for the first signature
  - EC private key from Figure 2 for the second signature
  - AES symmetric key from Figure 5 for the third signature
- Signing algorithms; this example uses the following:
  - "RS256" for the first signature
  - "ES512" for the second signature
  - "HS256" for the third signature
4.8.2. First Signing Operation

The following are generated before completing the first signing operation:

- JWS Protected Header; this example uses the header from Figure 56, encoded using base64url [RFC4648] to produce Figure 57.

- JWS Unprotected Header; this example uses the header from Figure 58.

```json
{
  "alg": "RS256"
}
```

**Figure 56: Signature #1 JWS Protected Header JSON**

eyJhbGciOiJSUzI1NiJ9

**Figure 57: Signature #1 JWS Protected Header, base64url-encoded**

```json
{
  "kid": "bilbo.baggins@hobbiton.example"
}
```

**Figure 58: Signature #1 JWS Unprotected Header JSON**

The JWS Protected Header (Figure 57) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 59).

eyJhbGciOiJSUzI1NiJ9.

**Figure 59: JWS Signing Input**
Performing the signature operation over the JWS Signing Input (Figure 59) produces the JWS Signature (Figure 60).

MIsjqtVlopa71KE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53uomic1tcMdSg-qptrrzZc7CG6Swv2Y13TDIgHzTUrL_1R2ZFcyrNF1Hksw129EghGpwkpxaTn_THJTCglNbAko1MZBcdwzJxwq2c-1RlpO2HiUYyXSwo97BSe0_evZKdjvvKSgsIqjytKSeAMbhMbdMma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZv1aI2sv33uPugB
BCXbYoQJwt7mxPfthMnN1GoOSMxR_3thmXTCm4US-xINOyhb8afK64ju6_TPtQHiJeQJxz9G3Tx-083B745_AfYOn1C9w

Figure 60: JWS Signature #1, base64url-encoded

The following is the assembled first signature serialized as JSON:

```
{
  "protected": "eyJhbGciOiJSUzI1NiJ9",
  "header": {
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "MIsjqtVlopa71KE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53uomic1tcMdSg-qptrrzZc7CG6Swv2Y13TDIgHzTUrL_1R2ZFcyrNF1Hksw129EghGpwkpxaTn_THJTCglNbAko1MZBcdwzJxwq2c-1RlpO2HiUYyXSwo97BSe0_evZKdjvvKSgsIqjytKSeAMbhMbdMma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZv1aI2sv33uPugB
BCXbYoQJwt7mxPfthMnN1GoOSMxR_3thmXTCm4US-xINOyhb8afK64ju6_TPtQHiJeQJxz9G3Tx-083B745_AfYOn1C9w"
}
```

Figure 61: Signature #1 JSON

4.8.3. Second Signing Operation

The following is generated before completing the second signing operation:

- JWS Unprotected Header; this example uses the header from Figure 62.

```
{
  "alg": "ES512",
  "kid": "bilbo.baggins@hobbiton.example"
}
```

Figure 62: Signature #2 JWS Unprotected Header JSON
The empty string (as there is no JWS Protected Header) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 63).

Figure 63: JWS Signing Input

Performing the signature operation over the JWS Signing Input (Figure 63) produces the JWS Signature (Figure 64).

Figure 64: JWS Signature #2, base64url-encoded

The following is the assembled second signature serialized as JSON:

```
{  
  "header": {  
    "alg": "ES512",  
    "kid": "bilbo.baggins@hobbiton.example"  
  },  
  "signature": "ARcVLnaJJaUGW8fG-8t5BREVAuTY8n8YHjwDO1muhcdCoFZFFjfISu0CdKn9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrqcI3JKl2Ut5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxxm4fgV3q7YYhm5eD"
}
```

Figure 65: Signature #2 JSON
4.8.4. Third Signing Operation

The following is generated before completing the third signing operation:

- JWS Protected Header; this example uses the header from Figure 66, encoded using base64url [RFC4648] to produce Figure 67.

```json
{
  "alg": "HS256",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
}
```

Figure 66: Signature #3 JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWV1ZjMxNGJjNzAzNyJ9
```

Figure 67: Signature #3 JWS Protected Header, base64url-encoded

The JWS Protected Header (Figure 67) and JWS Payload (Figure 8) are combined as described in [JWS] to produce the JWS Signing Input (Figure 68).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWV1ZjMxNGJjNzAzNyJ9.
```

Figure 68: JWS Signing Input

Performing the signature operation over the JWS Signing Input (Figure 68) produces the JWS Signature (Figure 69).

```
s0h6KThzKFBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 69: JWS Signature #3, base64url-encoded
The following is the assembled third signature serialized as JSON:

```json
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOTItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
}
```

Figure 70: Signature #3 JSON

4.8.5. Output Results

The following compose the resulting JWS object:

- Payload content (Figure 8)
- Signature #1 JSON (Figure 61)
- Signature #2 JSON (Figure 65)
- Signature #3 JSON (Figure 70)

The JWS Compact Serialization is not presented because it does not support this use case; the flattened JWS JSON Serialization is not presented because there is more than one signature.
The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJ1zIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBc
Z29pZmcbf3V0IHdvX1g9Vci4gWW91IHN0ZXAgb25obyB0aGUgcm9h
ZCwgYW5kIG1lIHBvdmU2dCBrZWVwIHBvdmVlCwgdGhlcmXigJlZiG5g
IItub3dpbmcg2hlcmlldGVyeW91IGlzZ2h0IGJ1IHN3ZXBlcmF0b
ZiB0by4",
  "signatures": [ { 
    "protected": "eyJhbGciOiJISUzI1NiJ9",
    "header": { 
      "kid": "bilbo.baggins@hobbiton.example"
    },
    "signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrgi5Nvy
G5Juoic1tcMdSg=qptrzzC7CG6Sw2Y13TIDigHzUrY1RZ5Fcy
NFhKsW129EgpwpkxaTN_THJTCg1NbADko1MZBCdwzJxw2z
-1RlpO2HibUYyxSw097Bse0-evZKdjjvVKSgsIjytKSeAhbIMyBM
ma622_BG5t4sdbuCHtFjp91Jmkio47AIwqkZiaZsv33uPqy5B
XbYqJw77xPftHmN1GoOSMxR3thmXTCm4US-xIN0yhbm8afKK6
4jU6_TpTQHiJeQJxz9G3Tx-083B745_AfYOn1C9w"
  },
  { "header": { 
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "ARcVLnaJJaUW8fG-8t5BREVAuTY8nYHjwDOlmuhc
dCoZ2FFjfISu0Cdhk5Ybdml54ho0x924DzUz8sK7Zkhc7AFM80b
UfTVnCrqC13Jk1Z5IX3utNhODH6V7xgy1Qahsn0fyb4zSAkje8b
AWz4vIfj5pCMYxMm4f4V5q77Yh5eEm"
  },
  { "protected": "eyJhbGciOiJISUzI1NiIsImtpZCI6IjA0OTk3NyI
5kW1NDcxY1liZmQ2LWV1ZjMxNGJjNzAzNyJ9",
    "signature": "s0h6KThzkfBBBKIlSpWh84VsjJFTsPPqMDA7g1Md7p
0"
  }
]
```

Figure 71: General JWS JSON Serialization
5. JSON Web Encryption Examples

The following sections demonstrate how to generate various JWE objects.

All of the encryption examples (unless otherwise noted) use the following Plaintext content (an abridged quote from "The Fellowship of the Ring" [LOTR-FELLOWSHIP]), serialized as UTF-8. The Plaintext is presented here as a series of quoted strings that are concatenated to produce the JWE Plaintext. The sequence \"\xe2\x80\x93\" is substituted for (U+2013 EN DASH), and quotation marks (U+0022 QUOTATION MARK) are added for readability but are not present in the JWE Plaintext.

"You can trust us to stick with you through thick and "
"thin\xe2\x80\x93to the bitter end. And you can trust us to "
"keep any secret of yours\xe2\x80\x93closer than you keep it "
"yourself. But you cannot trust us to let you face trouble "
"alone, and go off without a word. We are your friends, Frodo."

Figure 72: Plaintext Content

5.1. Key Encryption Using RSA v1.5 and AES-HMAC-SHA2

This example illustrates encrypting content using the "RSA1_5" (RSAES-PKCS1-v1_5) key encryption algorithm and the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

Note that RSAES-PKCS1-v1_5 uses random data to generate the ciphertext; it might not be possible to exactly replicate the results in this section.

Note that only the RSA public key is necessary to perform the encryption. However, the example includes the RSA private key to allow readers to validate the output.

Note that whitespace is added for readability as described in Section 1.1.

5.1.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- RSA public key; this example uses the key from Figure 73.
- "alg" parameter of "RSA1_5".
- "enc" parameter of "A128CBC-HS256".

```
{
"kty": "RSA",
"kid": "frodo.baggins@hobbiton.example",
"use": "enc",

"n": "maxhbsmBtdQ3CNrKvprUE6n9lYcregDMLYNeTAwCgLJ8SnPU9XlYeg7HVHqjxKDSHP21-F5j57sppG1wgAq2yznWxhYVhycm7RfgKxgNn_xAHzx6f3y7s-9M9PSNCwFC21h6eAKR4I00ehV9h1rmpM9p4datingZt5fS9W5UNwaAl1hrd-osGPQjJeI1deHtwx-ZTHu3C60Pu_LJI16Kh9wthewAmA4cR5Bd2ppgaY7A5gssJCUBtYJaNIHSoHKpruDjJKUMaZV0F0KpaA60P4i0ypBAdvjvM42Azj3BnAsySzE2haueTxvZB4eZOAjlyh2e_VOIKVMsDnryJYA
VotG1vMQ",
"e": "AQAB",
"dp": "Kn9tgoHfivT8v8pu5b9TnwyHwG5dK6RE0ufDlpCGjnJN7Z2e196377wybQ1PLAHmpIB5t7rheoAniRV1NClxuAqg8461xdTP4ntFEnqCksyO5jMA7i7-CL8vhpYy0wNF1esgMoVaPRMYMT9TW63hNMoAw7USZ_lHg6Oe1mYoVHT13FucjS8M68Nf4o1ENt43r2fPspEPGRd6e4pLc90ao-geP1GFUL1mRrdm-P8q8kvN3K1hNAtEgrQAgTTg7z80S-3VD0FgwFgdnbf1PNmUpUXO80pIF9KDIIfu_acc6fg14nsNAJqXe6RESvvhGPH2afjHqjSy_Fd2vpsj85bQQ",

"p": "2DwQmZ43FoTnQ81kUj3BMk5EH2m2izZAX5xEJ2MinJdysdTYkS1taEoekX9vbB2uxHdVhM6UnKCJ2iNk8Z0ayaLYHL0_G2iaXf9-uyynEpUsH7HUhkUlpFAzO0x1ZGyVjoxAdWNn3hiEFrrj1LZGSZ1OH-a3Qg1DDQj0OjQ2OYn",
"q": "te8LY4-W7IyagH1ExujmMgqTA1TeRbvoVLQnfLY2iXINnrDw2lQ93V5F099aP1ESelja2nw-6iKie-q77mcpfozfKvTUYf5hHr_jr_X2kfexj1Nh91hZHMv5p1sk2pe1S-GPHCC6g6R1Kol1q-idn_qxyusfW7WAX1sVFQfke8d6Et0",

"dp": "UyYKcL_or492vVocPzvLSp 청4L-3Z5wL48mwiwbpz0ylgd2x2HTQmJpFA1Z8q-zf9RmgJXkDrFS9KdxPtAsL1Wy8eCT5c125Fkdq3717VvRDo1tnX7x2Kdh8EERcWv8_4zXiTuT1_kIXZNU51vMQjWbIw2eTxlpsf10orYU",
"dq": "1Egco-QfpepH8FWd7mUFyZxn0kJBCogChY6YKu1HGc_p8Le9MbpFKEsZozEl1NehF3B0G2152-y12jzigQ7z2zucobrpa9FVYN0787A
CfzIG79dmV7R1PAdrZ103tkVXAdabu_9vs5rS-7HMtxkvRvxSUVY34J
TkX1HE",
"qi": "kC-1zZqQoFaZcr5lOtv8CoREKoVqayhIql6RGL-MzS4sCmRkxm5v2
1XKx6RTE1n_Aagajk1jeGlxTTThHD81g5f6CDMAr5u1hQGPsc7
G17CF1DZKBJMTQN6EshYxZfSxW08m10856Rzuhe0be1L6f9mkDcIyPBXx
2bQ_mM"
}
```

Figure 73: RSA 2048-Bit Key, in JWK Format
5.1.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 74.
- Initialization Vector; this example uses the Initialization Vector from Figure 75.

3qyTVhIWt5juqZUCpfRpqvwB956MEJL2Rt-8qXXSo

Figure 74: Content Encryption Key, base64url-encoded

bbd5sTkYwhAIqfHsx8DayA

Figure 75: Initialization Vector, base64url-encoded

5.1.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 74) with the RSA key (Figure 73) results in the following Encrypted Key:

laLxI0j-nLH-_BgLOXMoKkxmy9gffy2gTdvgzfTihJBouzxzg0V7yk1WClnQePFvG2K-pvSlWc9BR1azDrn50RcRaiz3TDON395H3c62tIouJJ4XaRvYHFjZTZZG Xfz8YAImcc91Tfki0XWC2F5Xbb71C1Q1DDH15t1pH77f2ff7xiSxh9oSeWYrcG TSLUeeCt36r1Kt3OSj7EyBQXo51N7IxbyhmAYFgIe7MvlrOTO1518NQeWXW8Vl zNmoxaGMny3YnGr5Wf6Qt2nBq4DaPdauuGUGEecelOi0x1BpyIfgvfjOh MBs9M8XL223Fg47x1GsMXdfuY-4jaqVw

Figure 76: Encrypted Key, base64url-encoded
5.1.4. Encrypting the Content

The following is generated before encrypting the Plaintext:

- JWE Protected Header; this example uses the header from Figure 77, encoded using base64url [RFC4648] to produce Figure 78.

```json
{
    "alg": "RSA1_5",
    "kid": "frodo.baggins@hobbiton.example",
    "enc": "A128CBC-HS256"
}
```

Figure 77: JWE Protected Header JSON

```
eyJhbGciOiJSU0ExXzUiLCJraWRQOiIjJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YW1wbGUlLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 78: JWE Protected Header, base64url-encoded

Performing the content encryption operation on the Plaintext (Figure 72) using the following:

- CEK (Figure 74);
- Initialization Vector (Figure 75); and
- JWE Protected Header (Figure 77) as authenticated data

produces the following:

- Ciphertext from Figure 79.
- Authentication Tag from Figure 80.

```
0fys_TY_na7f8dwSFxLliYdHa2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_raa0NhLQ6s1P2sv3Xz11p11_o5wR_RsSzrSBZ-wn13Jvo0mkpEEn1DmZvDu_k80WzJv7eZVEqiWKdyVzFhPpyQU28GLOpRc2VbVbK4dQKPdNTjPPEmRqcaGeTWZVyeSUv5k59y7ZxRu5vWFF6KrNtmRd28R4mD0jH5rM_s8uwIFcqt4r5Gx8TKaI0ztT5cb15lw3sRc7u_hg0yKV0iRytEAEs3v2kcfLkp6nbXdc_PKMdNS-ohP78T2O6_7uInMGhFex4ctHG7ve1HG1T93JfWDEQj5_V9UN1rhXNRyu-0fVmkZAKX3VW17lzA6BF430m
```

Figure 79: Ciphertext, base64url-encoded

```
kvKuFBXHe5mQr41qgobAUG
```

Figure 80: Authentication Tag, base64url-encoded
5.1.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 78)
- Encrypted Key (Figure 76)
- Initialization Vector (Figure 75)
- Ciphertext (Figure 79)
- Authentication Tag (Figure 80)

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUUICJ1bmMiOiJBMTI4Q0JDUlUtMjU2In0.
laLxI0j-nLH-_BgLOXKoXmy9gffy2gTdvqzftihJBUuLxq0V7yk1WClnQePF
vG2K-pvS1Wc9BRjaZDrn50RcRaI-_3TDON395H3c62tIouJ4XaRvYHFj2Z2G
Xfz8YAImcc91Tfk0WXC2F5Xbb71ClQ1DDH15lt1pH77f2ff7xiSh9oSeWYrCG
TSLueeC73rlKt3OSj7EyBQXoZ1N7IxbxyhMAfg1e7Mv1rOTI518NQxeXXW8V1
zNmoaxGMny3YnGir5Wf6Qt2nBq4gDaPdnaAuUGExec1Io1wx1BpyIfgfvjOh
MBs9M8XLZ33594731GmsMXdfuy-Y-4jaqVw
bbd5sTkYwhAIqfHsx8DayA
0fyts_TY_na7f8dwSfXLiydHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_r
aa0HnLQ6s1P2sv3Xzllp11_o5wR_RsSzrS82-wn13Jvo0mkpEEEn1Dm2vDu_k080
WzJv7e2VEqiWKdvy2FvPpXmYQ28GLopRc2VbVbK4dQKpdtNTjPPEmRqcaGeTWZV
yeSUvF5k59yJzxRuSvWf6KrNtmRdZ8R4mDOjH5rM_s8uwIFcqt4r5G8TKaI0
zT5CbL5Qlw3sRc7u_hg0yKVOiRytEAEs3vZkcfLkP6nbXdc_PkmNd-ohP78T2
06_juInMGrFe4ctHG7Ve1HG1t93jfWDEQ15_V9UN1rhXXrYu-0fVMKzAKX3vw
171zA6BP430m
.kvKuFBXHe5mQr4lqgobAUg
```

Figure 81: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "laLxI0j-nLH-_BgLOXmozKxmy9gffy2gTdvqzf
TihJBuuzxg0V7yk1WClnQePFvG2K-pvSlWc9BRJazDrn5OrcRai_
_3TDON395H3c62t1ouJJ4XaRvYHFjZTZ2GXfxz8YA1mcc91Tf
C2F5Xbb71CLQ1DHH151t1pH77f2ff7xiSxh9oSewYrcGTSLu
eCt36r1Kt3OSj7EyBQXo21N7IxbyhMAfgei7Mv1r0TOI518Nq
V1zNmoxaGr3y3yGir5Wf6Qt2nBq4qDaPdnaAuGUGcEeclI01w
1BpylfgfVfj0hMBs9M8XL223Fg47x1GsxMXdfuY-4jaqVw"
        }
    ],
    "protected": "eyJhbGciOiJSU0ExXzUicjJ7IiJ7Ijw\n5zGhvYmJpdGR9uLmV4YW1wbGUIClCJ1bmMiOjEBMTI4Q0JDLUhTMjU2In
0\n",
    "iv": "bbd5sTkXwhAIqfHsx8DayA",
    "ciphertext": "0fys_TV_na7f8dwSfXLIydHaA2DxUjD67ieF7fcVbIR62
JhJvG4z_FVNsiGc_\nra0HnLDQs1P2sv3Zxl1p11_o5wR_RszSrzS82-\nwmI3Jv0mkeEEnIDmZvDU_k80Wzjv7eZVEqiWkdyVzFhPpiyQU28Gl0pRc
2VbVbK4dqKPDnTjPEEmRqcaGerT2ZVyeSUvf5k59jZxRusVWFf6KrNn
tmRdZB4mD0jHiSm_s8uwIFCq4t4r5GX8TKaI0z5CBL5QLw3Sc7u_hg0\nKVOiRyTEAEs3V2kcflKp6nbXdc_PkMDNS-ohP78T206_7uInMGhFeX4c
thG7VeHg1T3JfWDEq15_V9UNlhxXNryu-0fVMkzAXK3W17JzA6BF4
30m",
    "tag": "kvKuFgBh5mQr41qgobAUG"
}
```

Figure 82: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:
{
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YW1wbGUuLCJ1bmMiOiJJBMTI4Q0JDLUhTMjUIn0",
  "encrypted_key": "laLxI0j-nLH-_BgLOXMozKxmy9gfyy2gTdvqzfTihJBuuzxg0V7yk1WC1nQeFPGv2K-pvSLwc9BRIazDrn50RcRai__3TDON395H3c62tIouJ4xARvYHFj2T22GXfz8YAImcc91Tfk0WXCC2F5Xbb71CQ1DDH151t1pH77f2f7xiSxh90SewYrcGTSLUeeCt36r1Kt3OSj7EyBQXo21N7IxbyhMAfgie7Mv1rOTOI518NqexXXW8V1zNmoxaGMny3YnGir5WF6Qt2nBq4qDApndaAu6GUEecele1O1w1BpyIfgvfjOhMBs9M8XL223Fg47x1GsmXdfuY-4jaqVw",
  "iv": "bbd5sTkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSfXLIydHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xzil1p11_o5wr_RszrS82-wnI3Jv00mkpEEn1DmZvDu_k80Wzjv7e2VEqivKdyVzFhPpiyQU28GLOpRc2VbVbK4dQKpNdTj5PPEmRqcaGetWZVyeSUVf5k59yJzRsvWVF6KrNtmRdZ8R4mDOjHt5M_s8uwIFcqt4r5GX8TKa10zT5Cbl5Q1lw3Sc7u_hg0yKVOiRytEAEs3v2kcfLkP6nxXDC_PkmDNS-ohP78T206_7uInMGrFeX4ctHG7Ve1HGItT93JfWDEQ15_V9UN1rhXNryu-0fVMeZAKX3VWi71zA6BP430m",
  "tag": "kvKuFBXHe5mQr4lqgobAUG"
}

Figure 83: Flattened JWE JSON Serialization

5.2. Key Encryption Using RSA-OAEP with AES-GCM

This example illustrates encrypting content using the "RSA-OAEP" (RSAES-OAEP) key encryption algorithm and the "A256GCM" (AES-GCM) content encryption algorithm.

Note that RSAES-OAEP uses random data to generate the ciphertext; it might not be possible to exactly replicate the results in this section.

Note that only the RSA public key is necessary to perform the encryption. However, the example includes the RSA private key to allow readers to validate the output.

Note that whitespace is added for readability as described in Section 1.1.
5.2.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the Plaintext from Figure 72.
- RSA public key; this example uses the key from Figure 84.
- "alg" parameter of "RSA-OAEP".
- "enc" parameter of "A256GCM".

```json
{
    "kty": "RSA",
    "kid": "samwise.gamgee@hobbiton.example",
    "use": "enc",
    "n": "wbdxI55VaanZXPY29Lg5hdmv2XhvgAhoxUkanfzf2-5zVUxa6prHrR4P1AhogJRlZfytWwd5mmHHR2pAHIlh0ySJ9iOBioZB11XP2e-C-FyXJGcTy0HdKQWIfmTw42EW7Vv04r4gfao6uxjLGwpGrZLarchiWCPnkJNrg71S2CuNZ5QBP1GjXfkmy2t1-VEgGnL22GplYxj51BLdxXp3xSeStqo571utNf0U8E4qdzJ3UIDitoVDKpGwM1mmnJiWa7sxXRItBCiRV4M59gZtdw-7v4WuR4779ubDv55alMr2566-RPcnFazWSSxtBDnFJJDGIU7Tzizj1nmosXG_yPub_U01Wn0ec85FCft1AcPwG8schrObEnqHtBDNskYpUc2LC5J2TaTpf2dA67dg1TTsc_FupFQ2kNGcE1IlgprxKHCvNYq0868-HozjHjZcqttauBzFV95tbTu-5TpkcvJfNcFlhl3b8mb-H_o35FjgBShAlKycqefKTPvJvXhd09knwgf6Vkg6UC418_TOlMVFTXWUxnhFOOnzW6HSSsdLc99WvrCYSzr4V154sziG9wF1cYwF3g5qDxQKis99gcbaicAwM3yEBizuNeeCa5dartHDblxEB_HCHeYbhgbMjGfasvKn0aZRnsTyC0xHwB1olZEO",
    "e": "AQAB",
    "alg": "RSA-OAEP",
    "d": "n7fzjc3_WG59VEOBTkayzuSMNm780JQuZjNh_KbH8lOZ520aA7T4Bxc0cXqNsoE5uSClw91oCt0JvxPcpmqzaJ2glnjrjCW-oBtV7gCAwq083q4g3IhbokzzrJhayc3YHbh5y4_WerrXg4MDNE4Hojy68873cXuT2LYQRXv0UOF5TJXm80iex15StVnqNDRutxEUCwlewmnrveEogLx9EAA-KmgAjtIIsxxqIXQHWWWX1G7v_mV_HrZyUyMnCnKHyw9P70o0k0876Dhko8V4U0ZLWaAL0U1498mKoqwc58A_Y21BiyBvXs1_s5lPsEqbbh-nqfjhl0fNfNHlNcLwtW7pCztLznM2AyeCWA7G7IFv-Rn9f1iv9j2I67r-MSH9gqbu1HN2grGJd-jfrLmuHAlo84fFK16bcQj1JwXvPZhN20o1yDF-1LiQsqYyPeFpf6Xa2SodkgBriguE66ulvSYIDpJq3jDisgolML0iLOoomg1jXuUWlFWEOu28guqlyzm-9QO0u0yFh1huSrSAjaJQwAFITIwWH5W_ITQ9I7r-yrindr_2fwQ_i1UqMsGzA7aOGzZIfP1jry6z-tYKuBG00-28s_aWvjjuc-Alp8AuYkBLZ-7CWH32fGWK48j1t-zomrwJL_mhnsPbGsoC9WswGyRZI-K8gE",
    "p": "7_2v3Oq2z1FlFvhYflABQX3P85Es4hCdWckbDeltauxVgyV919etKgHvM4hRkOvbb1KkYvL3FXmIKCDtPl-zLCYAdXKrAK3PtSbtzIzD_ZX9n1sYa_QZwzPB_IrtFjVfKxd0dzs49uhUFqeGfp7jr6Nnxfip1HSHWEP1dz_2AC3mY46J961Y2RnreVwAGNw53p07Db8yD_92pDa979qcZOdgytB9Q6uma-
```
Figure 84: RSA 4096-Bit Key

(NOTE: While the key includes the private parameters, only the public parameters "e" and "n" are necessary for the encryption operation.)

5.2.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 85.
- Initialization Vector; this example uses the Initialization Vector from Figure 86.
mYMfsggkTAm0Tbvt1H2hyoXnbEzJQjMxmgLN3d8xXA

Figure 85: Content Encryption Key, base64url-encoded

-nBoKLH0YKLZPS19

Figure 86: Initialization Vector, base64url-encoded

5.2.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 85) with the RSA key (Figure 84) produces the following Encrypted Key:

rT99rwrBTbTT17JIM8fU3Elil7226HEB7IChCxNhuh71Ciud48LxeolRdtFF4nzQi
beYO19S_PJxAXwSXtDePz9hk-BbtsTBqC2UsP0dwjC9NhNupNNN9u9uHIrFtDyu
cv16hvAELe%G0hNV4v1zx2k701D89mAzfw-_kt3tuorpDU-CpBENfHIHXIq58
-Aad3FzMO3Fn9buEP2yXakLXXa15BUXQsUmpM4A1GD_H4Bd7V3u9h8Gkg8Bpx
KdUV5ScfJQTCyM6eJEBz3aSwiaK4T3-dwPwpuBohROQXBosJzSldnuHtVMt2pK
IIfux5BC6huIvmY7zkV77aIUrpYm_3H4zYyyMeq5pGqFMw2K8zpO878TR1zX7
p2fpyDxZ5y0ScFkkMozT_qiCwzTsz4LuYnt8hs429sGthXn9uqDq6wycMangaQ
f0Ts_yTwmY-agqVWDKhxjYNrf03N1wr6b5BE-t0dFwCASQj3uAGPGroO2AWBE3
8UjQb01VXnspY3WFc7WojyA7aA8DRn6MC6T-xDmMurxCOG7S2rscw51QQU
56MvZ1FOT0UvfuKBA03cxAX_nI8hL6jY2j0XQMmpDPT6Cbo8aKaO6E5
Jx9paBpnNm0OKH35j_QlrQhDWUN6A2Gg8iPaj69xDEdHAVCGRzN3woE12ozDR

Figure 87: Encrypted Key, base64url-encoded

5.2.4. Encrypting the Content

The following is generated before encrypting the Plaintext:

- JWE Protected Header; this example uses the header from Figure 88, encoded using base64url [RFC4648] to produce Figure 89.

```
{
    "alg": "RSA-OAEP",
    "kid": "samwise.gamgee@hobbiton.example",
    "enc": "A256GCM"
}
```

Figure 88: JWE Protected Header JSON

eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpcc2Utz2Fz2V1QGhvYmJpdG
9uLmV4YW1wbGUICJ1bmMiOiJBMjU2R0NNIn0

Figure 89: JWE Protected Header, base64url-encoded
Performing the content encryption operation over the Plaintext (Figure 72) with the following:

- CEK (Figure 85);
- Initialization Vector (Figure 86); and
- JWE Protected Header (Figure 89) as authenticated data produces the following:
  - Ciphertext from Figure 90.
  - Authentication Tag from Figure 91.

```
o4k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXklSgqBdpACm6UJuJowOH'C5ytjGyR
L-I-soPlwqMUF4UgRWMeaOGNw6vGW-xyM011TyxrXfVzIaRdhYTtLMRBvBWbEw
P7u1DRfvaOjgZv6ifa3brcAM64d8p51hhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sWwy7QmRkfhDjWBz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1qUlML
7Cc2GxgvI7qgWoOYIEc7aCf1LG1-8BovVWFd2KLK9vNoycinVHumwzK1uLWEdSV
maPpOs1Y2n525DxDfWaVFUFkQxMF56vn4B9QMPWAbnypNimbM8zVOw
```

Figure 90: Ciphertext, base64url-encoded

```
UCGiqJxhBI3IFVdPa1HHvA
```

Figure 91: Authentication Tag, base64url-encoded

5.2.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 89)
- Encrypted Key (Figure 87)
- Initialization Vector (Figure 86)
- Ciphertext (Figure 90)
- Authentication Tag (Figure 91)
The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpd2UzZ2VlQGhvYmJpDG9uLmV4YW1wbGUiLCJ1bmMiOiJBMIU2R0NNIn0.

Figure 92: JWE Compact Serialization

UCGiqJxhBI3IFVdPalHHvA
The resulting JWE object using the general JWE JSON Serialization:

```json
{
"recipients": [
{
"encrypted_key": "rT99rwrBTbII7IJM8fu3Eli7226HEB7IichCxNu
h7Iciud48LXeol1RdtTF4nzQibeYo155_PJsaAZXwSXTDePz9hk-Bb
tSTBqC2UsP0dwjC9NhNupNNu9uHIvftDyucvI6hvALe26OGnhNV4
vIzx2k70ID89mAzfw-_KT3tkuropDU-CpEnfIHXlQ58-Aad3FzMu
uo3Fn9bueEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8B
pxKdUV9SCjQTcYm6eJEBe3aSwIaK4T3-dwWpuBOhROQXBOsJzS1
asnuHtVMk2pKIIifux5BC6huIvmY7kzv77W7aIUrpym_3H4zYvyMeq
5pGqFmW2k8zpo8788R1zX7pZFPYDSXZyS0CFKkMozT_q1CWzTSz
4duYnt8hS4Z9sGthXn9dQd6wycMagQfOTs_1ycTWmY-agqWVDKx
jYNRf03NIwrTb5BE-tODFWCAsQj3uuAgPrO2AWBE38ujQb01vbXn
1SpvvY3WFc7WOJYTa7A8DNn6MC6T-xDMuxC0G7S2rcsw51QU
06Mv2TlF0t0UvfuKba03cxA_nIBIlLMJy2kOTxQmpmpDTrC6bo8a
KaOnx6ASE5Jx9paBpnNmoOKH35j_Q1rQhDWUN6A2Gg8iFayJ69xD
EdHAVCGRzN3woE1zozDRs"
}]
"protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpcoe2UzuZ2
FtZ2V1QGhvYmpdG9uLmV4YW1wbGUICJlbnMiOiJHBUdBjU2R0NNIn0",
"iv": "-nBoKLH0YKLZPSI9",
"ciphertext": "o4k2cnGN8rSSw3IDo1YuySkqes_t2m1Gxk1SgqBdpAcM6
UUjowOH6ytjyqYgRL-i-soPlwqMUf4ujRWWeaOGNW6vGW-xyMO11TYxr
XfVUlXairdYyEMR6vBBwBxWP7ualDRFva0JgZv61fa3brcAM6468p51
hNCizPersuwh5f-pPYzseva-TUaL8iWnctc-sSwy75QMqRkhDjwb20fz
6kFovEgj6X1i5s7EGLP5fbnyYGLa1QUiML7C2Gxgv17zw0YIEc7a
CfllL1-8BoVWFdZKLK9vNoycrYHunwzKluLWEbSvmaPpOs1y2n525Dx
DFwaVFUfKxwMF56vn4B9QMwAbnypNimbM8zVOW",
"tag": "UCGiqJxhBI13IFVdPa1HHvA"
}
```

Figure 93: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpd2VwZiIsInJvbGUiOiJBMjU2R0NNIn0",
  "encrypted_key": "rT99wrB8Tt7lJiM8fU3El7t226HEB7chCxNuh71C
  iu4d8Lxelo1rTtFF4nzQibeYo15S_JJsAXzwSxtDepz9hk-Blbl6w3nJCAu2U
  sp0dwj9CN6Nnu9uH1VtfByncvF6hvALez60GnhNV4v1xx2k70ID9
  mAsfEw_ _K73tkuorpuDU-CpBENfiHx1q58-Aad3fzMu03Fn9buEP2yXakL
  XYa15BUXQsupM4A1GD4_H4BD7V3u9h8Gkg8BpxKdUv9ScfJQTCyM6eJE
  Bz3aSwIAK3-0wBpuBohRQXBoJ2zL3asnuHTVmt2pK1Ifux5BC6wUI
  vmY7kzvV7wAUurpyYm_3H4zYvyMeq5pGqFmW2k8zpo878TR1z7pZfPYD
  SXZy50CfKZkkMo9z_T_qlCWZTSz4duYnt8s4Z9sGthXn9uQd6wycMqN
  fOTs_lycTWmY-aqWVDKhjYNRF03NiwrTab5BE-tOdfWCASQj3uuAgPGr0
  2AWBe38UqO01vXn1SpvY3WFC7WOJYaTa7A8Dx6M6CT-xDmMuCCOG
  7S2rscw51QQO6MvZ1F0t0UvfuKBA03cxA_nIBHLMjY2kOTMQmnpDP
  Tr6Cbo8aKan0x6ASE5jX9paBpnNmOOKH35j_Q1rQhDWUN6A2Gg8iFayJ
  69xDEdHACGxRzN3woElOzDRs",
  "iv": "-nBoKLH0YKL2PSI9",
  "ciphertext": "04k2cnGN8rSSw3IO1YuySkqes_t2m1GXk1sQgBdACm6
  UJuJowOC5ytjQyGRL-I-soPlwGMU4uGWWeaOGNw6VGW-xyM011TYx
  rXfVzIIaRdhYtEMRBBvWBEw7ua1DrfaOjgZv6Ia3brcAM64d8p51h
  hNcizPersuhw5f-5GYZeBuTUL8iWnctc-sSwy7SqmRkhDjwz0fz
  6Kf0vEgj064X115s5EG31l5fnhYGLa1QU1ML7c2Gxgy17zqWc0TIEC7a
  Cfl1GL1-8BboVWFdZKLK9vNoycrYHuwzKlUwEbsVmaFs0lyZ25DX
  DF1waVFUFQxMF56n4B9QmpWbnypNimbMBzVOW",
  "tag": "UGQigJxhBi13IFVdPaiHHVa"
}
```

Figure 94: Flattened JWE JSON Serialization

5.3. Key Wrap Using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2

The example illustrates encrypting content using the "PBES2-HS512-A256KW" (PBES2 Password-based Encryption using HMAC-SHA-512 and AES-256-KeyWrap) key encryption algorithm with the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

A common use of password-based encryption is the import/export of keys. Therefore, this example uses a JWKS Set for the Plaintext content instead of the Plaintext from Figure 72.
Note that if password-based encryption is used for multiple recipients, it is expected that each recipient use different values for the PBES2 parameters "p2s" and "p2c".

Note that whitespace is added for readability as described in Section 1.1.

5.3.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the Plaintext from Figure 95 (NOTE: All whitespace was added for readability).
- Password; this example uses the password from Figure 96 -- with the sequence "\xe2\x80\x93" replaced with (U+2013 EN DASH).
- "alg" parameter of "PBES2-HS512+A256KW".
- "enc" parameter of "A128CBC-HS256".

```
{
  "keys": [
    {
      "kty": "oct",
      "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
      "use": "enc",
      "alg": "A128GCM",
      "k": "XctOhJAkA-pD9Lh7ZqW_2A"
    },
    {
      "kty": "oct",
      "kid": "81b209e5-8332-43d9-a468-82160ad91ac8",
      "use": "enc",
      "alg": "A128KW",
      "k": "GZy6sIz6wl9NjOKB-jnmVQ"
    },
    {
      "kty": "oct",
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
      "use": "enc",
      "alg": "A256GCMKW",
      "k": "qC571_uu8cm7Nm3K-ct4GFjx8tM1U8CZ0NLDvQstiS8"
    }
  ]
}
```

Figure 95: Plaintext Content
5.3.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 97.
- Initialization Vector; this example uses the Initialization Vector from Figure 98.

5.3.3. Encrypting the Key

The following are generated before encrypting the CEK:

- Salt input; this example uses the salt input from Figure 99.
- Iteration count; this example uses the iteration count 8192.

Performing the key encryption operation over the CEK (Figure 97) with the following:

- Password (Figure 96);
- Salt input (Figure 99), encoded as an octet string; and
- Iteration count (8192)

produces the following Encrypted Key:

d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdtURtmeDVl
5.3.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 101, encoded using base64url [RFC4648] to produce Figure 102.

```
{
  "alg": "PBES2-HS512+A256KW",
  "p2s": "8Q1SzinasR3xchYz6ZZcHA",
  "p2c": 8192,
  "cty": "jwk-set+json",
  "enc": "A128CBC-HS256"
}
```

Figure 101: JWE Protected Header JSON

eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2SlciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6NiipaY0hBIIwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0

Figure 102: JWE Protected Header, base64url-encoded

Performing the content encryption operation over the Plaintext (Figure 95) with the following:

- CEK (Figure 97);
- Initialization Vector (Figure 98); and
- JWE Protected Header (Figure 102) as authenticated data produces the following:

- Ciphertext from Figure 103.
- Authentication Tag from Figure 104.
5.3.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 102)
- Encrypted Key (Figure 100)
- Initialization Vector (Figure 98)
- Ciphertext (Figure 103)
- Authentication Tag (Figure 104)
The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJQQkVTMi1IUzUxMitBMjI2S1ciLCJwMnMiOiI4UTFTemluYXNSM3hjaF16NlpaYOhBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pb24iLCJlbmMiOiJBMjI2Sm13UyIiLCJpYiI6IkZkaXZkZg==
d3qNhUWeqheyPp4H8sjOWsDYaiej4c5Je6r1UtFPWdgtURtmeDV1g

VBiCzVHNoLiR3F4V82uoTQ

Figure 105: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "d3qNhUWfqheyPp4H8sj0Wsyjaoe4c5Je6I1UtFPWdgtURtmeDV1g"
        }
    ],
    "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2SlciLCJwMnMiOiI4UTFTemluYXNSM3hjaF16NlpaY0hBiwiicDJjIjo4MTkyLCJjdHkiOiJq2stc2V0K2pzb24iLCJbmiMiOiJBMTI4Q0JDLUhTMjU2In0",
    "iv": "VBicZVHNoLiR3F4V82uoTQ",
    "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIXS6vMXqIi61RsfywCRFzLxExZBRnTvG3nhzPkOGDD7FMyXhUHpDjeYCPW_XOmzg8yZR9oyj61TF6siq9FZ2EhzgFQCL6_6h5EVg3vR75_hkBsnuoqoM3dwejXBiOdN84PegMb6asmas_dpSsz7H10fC5ni9xIz424giYBh1YLld6exVmL93r3foOJbmk2GBQZL_SEG11v2cQsBgeprARsaQ7Bq99tT80ccH8ltBjgV08AtzXFFsx9qKvC982KLdPQMT1VJKkgtV4Ru5LEVpBZXBnZrtViSOgy6Aiws-rCrC_ePOGSuxvgtrokAKYPqmXE8RdjijwafKYEkiuDCV9vWGAi1DH2xTafhJwcmywIyzi4BqRpmkn_N-z15tuJYyuvKkJv6ihbsV_k1hJGPAXj6wUwmwC4PTQ2izEm0TuSE8oMKdTw8V3kobXZ77ulMwDs4p",
    "tag": "0HlwodAhOCILG58G2LQ9dg"
}
```

Figure 106: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```
{
    "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6NiipaY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzbi241LCJlbmMiOiJBMjIwMTI4Q0JDLUhTMjU2In0",
    "encrypted_key": "d3qNhUWFqheypp4H8sjoWSyajoej4c5Je6riUTFPWdgtURteDV1g",
    "iv": "VBiCzVHN0LiR3F4V82uoTQ",
    "ciphertext": "23i-Tb1AV4n0WKSStgQrdg6GRqsUKxjruHXYsTHAjLZ2nsnG1X6vMXqi1i6IRsfywCRFzlxEcZBRnTvG3nhzPko0GDD7FMyxUhpDdYEwCA6_mz8yZR9oyjoyo61TF6i4g9FZ2EhzgFQColo_6h5Ev3vR75_hkBsnuoqoM3dwejXeTiodN84PegM6asmas_dszz7H10fC5ni9xIz424givB1YldF6exVmL93R3fo0OJbmk2GBQZL SEG11v2cQsBgeprAR5aq7Bq9tT80coH8tBjgV08AtzXFFsx9qKVC982KLkdPQMT1VJKktv4R5LEVPBZXBn2rtVIoS0gg6AiuaS-rCrc_ePOGSuxvtroAKYPgmUXeRdjFJwafkYEkiuDCV9vWGAi1iDH2xTafhJwcmwywIyzi14BqRpmdn_n-zl5tuJYyvKkijKv6ihbsV_k1hJGPGAxJ6wUmpwC4PTQ2izEm0TuSE8oMKdT w8V3kobXZ77ulMwDs4p",
    "tag": "OHlwodAh0OCILG5SQ2LQ9dg"
}
```

Figure 107: Flattened JWE JSON Serialization

5.4. Key Agreement with Key Wrapping Using ECDH-ES and AES-KeyWrap with AES-GCM

This example illustrates encrypting content using the "ECDH-ES+A128KW" (Elliptic Curve Diffie-Hellman Ephemeral-Static with AES-128-KeyWrap) key encryption algorithm and the "A128GCM" (AES-GCM) content encryption algorithm.

Note that only the EC public key is necessary to perform the key agreement. However, the example includes the EC private key to allow readers to validate the output.

Note that whitespace is added for readability as described in Section 1.1.

5.4.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- EC public key; this example uses the public key from Figure 108.
"alg" parameter of "ECDH-ES+A128KW".

"enc" parameter of "A128GCM".

{
    "kty": "EC",
    "kid": "peregrin.took@tuckborough.example",
    "use": "enc",
    "crv": "P-384",
    "x": "YU4rRUzdmVqmRtWOs2OpDE_T5fsNIoXcG8G5FPrTPMyxpzsSOGaQLpe2FpxBmu2",
    "y": "A8-yxCHxkfB3hKZf1jUYMjUhSeveZ9ThuwFjH2sCNdtk8RJU7D5-SkgAF11ETP",
    "d": "iTx2pk7W-GqJkHoEkFQb2EFyYc07RugmaW3mRrQVAOUIpommT0IdnYK2xD1zh-j"
}

Figure 108: Elliptic Curve P-384 Key, in JWK Format

(Note: While the key includes the private parameters, only the public parameters "crv", "x", and "y" are necessary for the encryption operation.)

5.4.2. Generated Factors

The following are generated before encrypting:

o AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 109.

o Initialization Vector; this example uses the Initialization Vector from Figure 110.

Figure 109: Content Encryption Key, base64url-encoded

mH-G2zVqgzUtbnW_

Figure 110: Initialization Vector, base64url-encoded

5.4.3. Encrypting the Key

To encrypt the Content Encryption Key, the following is generated:

o Ephemeral EC private key on the same curve as the EC public key; this example uses the private key from Figure 111.
Performing the key encryption operation over the CEK (Figure 109) with the following:

- The static Elliptic Curve public key (Figure 108); and
- The ephemeral Elliptic Curve private key (Figure 111)

produces the following JWE Encrypted Key:

0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2

Figure 112: Encrypted Key, base64url-encoded

5.4.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 113, encoded to base64url [RFC4648] as Figure 114.

```json
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx510xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iuEDsQ6wNdNg3",
    "y": "sp3p5SghZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEwZ0yQTA-JdaY8tb7E0",
    "d": "D5H4Y_5PSKZvhfVFbcCYJ0tcGZygRgfZkpsBr591cmnhe9sw6nkZ8WfwhinUfWJg"
  },
  "enc": "A128GCM"
}
```

Figure 113: JWE Protected Header JSON
Performing the content encryption operation on the Plaintext (Figure 72) using the following:

- CEK (Figure 109);
- Initialization Vector (Figure 110); and
- JWE Protected Header (Figure 114) as authenticated data produces the following:
  - Ciphertext from Figure 115.
  - Authentication Tag from Figure 116.

WuGzxmcreYjpHGJoal7EBg

Figure 116: Authentication Tag, base64url-encoded
5.4.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 114)
- Encrypted Key (Figure 112)
- Initialization Vector (Figure 110)
- Ciphertext (Figure 115)
- Authentication Tag (Figure 116)

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJFQ0RILUVTK0EXMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2luXhhbXBSZISIsImVwayI6eyJrdHkiOiJFQyIsImNydii611At
Mzg0IiwieCI6InVCbzcRrSFB3Nmtiang1bDB4b3dyZF9vWXpC8WF6LUDlpl1NH
hBRkY11pV2dlDEVLNnl1RUrzUT3TmRO2zMiLCJ5Ijoic3AzcDTR2haVkMy
ZmFydW1JLWU5L0UyTW84S3BvWXJGRHI1eVBOVnRXNFBrXdaT3lRVEEtSmRhWT
h0YjdPMCFJ9LCJlbnMiOiJBMTE4R0NNIn0.
```

Figure 117: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
        }
    ],
    "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6ICJCI6InVCbzRrSFB3NmtniB0bD
B4b3dy2F9vWXpCbWF6LudRIp1NHBKzkY1lpV21dEVLNml1RURzUT
Z3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkMyZmFydW1JLUU5Um5vTWE4S3
BvWXJGRH1eVBOVnRXNFBnRXdaT3lRVEEtSmRhWT0YjdFMCMJ9LCJlbm
Mi0iJBMJ4R0NNIn0",
    "iv": "mH-G2zVqgztUtNW_",
    "ciphertext": "tkZuOO9h950gHJmkkrfLBisku8rGf6nzVxhRM3sVohXg
5NJ76oID71pA1-cpWJRCySpaAzu5dOR3Sp7QuEkmKx8-3RCMhSMzS
XaEwDdXta9Mn5B7cCBOJKB0IgEnj_qfoi1Ii-uEUKup0Z8aLT2GHfp105
jMwbbKKe2yK3mjF6SBAsgicQDVKccY9BLuux1RmC3ORXaM0JaHPB93Y
csdSGgipBWMvNRU1ErkjcMgMoT_wtCex3wO3xDLkjXiUEr2hWgeP-nkU
ZTPU9EOGSPj6fAS-b9z87RCPnxZdj_IVyC6QWcqAuO7WbHzJEPc4jVn
tRJ6K53NqPQSp991324080Uqj4ioYezbS6vTPzQ",
    "tag": "WuGzxmcreYjpxOGJoa17EBg"
}
```

Figure 118: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdHVjaJvcml9IiIsImNydiI6IlAtMzg0IiwieCI6InVCbzRrSFB3Nmtiang1bD
B4b3dy2F9vWXpCbfWF6Lulrip1NHHbrKzrzYl1pV2d1dEV1Nnm1111RURzUT
Z3TrO2zzMiLCJtIjoic3AzcDVR2haVkJMy2mFydlWlJLWU5Si1yTW84S3
BvWXJGRH1leVBOvRXNFBnRxdaT31RVEEEtSmRWhTH0YjdFMCJ9LCJ1bm
MiOijBMI4R0NNIn0",
  "encrypted_key": "ODjJBxRi_kBcC46IkUS_Jk9BqaQeHdv2",
  "iv": "mH-G2zVqgzUtbnW_
  "ciphertext": "tk2U009h95OqHJmkkrfLBisku8rgf6nzVxhRM3sVohXg
5NJ76oID7lpnAI_cPWJRCjSpAaUZ5dOR3Spy7QuEkMkx8-3RCMhSYMz
XaEwdXta9Mn5B7cCBoJKB0IgEn/_qBo1hIi-ueKUp0Z8aLT2GHfp105
jMwbKkKez2yK3mjf6SBgsigcDVcKcY9BLuzx1RmC3ORxAM0JaHPB93Y
cdSDGgpgBWMvrNU1ErkjcMgMoT_wtCex3w03XdkjXiUer2hWgeP-nkU
ZTPU9EgGSpj6fAS-bSz87RCPpxZdj_iVyC6QWcqAu07WNhzJEpc4jVn
rj6K53NgPQ5p911324080Uqi41oyezbS6vTP1Q",
  "tag": "WuGzxmcreYjpHGoa17EBg"
}
```

Figure 119: Flattened JWE JSON Serialization

5.5. Key Agreement Using ECDH-ES with AES-CBC-HMAC-SHA2

This example illustrates encrypting content using the "ECDH-ES" (Elliptic Curve Diffie-Hellman Ephemeral-Static) key agreement algorithm and the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

Note that only the EC public key is necessary to perform the key agreement. However, the example includes the EC private key to allow readers to validate the output.

Note that whitespace is added for readability as described in Section 1.1.
5.5.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- EC public key; this example uses the public key from Figure 120.
- "alg" parameter of "ECDH-ES".
- "enc" parameter of "A128CBC-HS256".

```json
{
  "kty": "EC",
  "kid": "meriadoc.brandybuck@buckland.example",
  "use": "enc",
  "crv": "P-256",
  "x": "Ze2loSV3wrroKUN_4zhwGhCqo3Xhu1td4QjeQ5wIVR0",
  "y": "HlLtdXARY_f55A3fnzQbPcm6hgr34Mp8p-nuzQCE02w",
  "d": "r_kHyZ-a06rmxM3yESK84r1otSg-aQcVStkRhA-iCM8"
}
```

Figure 120: Elliptic Curve P-256 Key

(NOTE: While the key includes the private parameters, only the public parameters "crv", "x", and "y" are necessary for the encryption operation.)

5.5.2. Generated Factors

The following is generated before encrypting:

- Initialization Vector; this example uses the Initialization Vector from Figure 121.

```
yc9N8v5sYyv3iGQT926IUg
```

Figure 121: Initialization Vector, base64url-encoded

(NOTE: The Content Encryption Key (CEK) is not randomly generated; instead, it is determined using ECDH-ES key agreement.)
5.5.3. Key Agreement

The following is generated to agree on a CEK:

- Ephemeral private key; this example uses the private key from Figure 122.

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "mPUKT_bAWGHIhg0TpjjqVsP1rXWQu_vwVOHHtNkdYoA",
  "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs",
  "d": "AtH35vJsQ9SGjYfOsjUxXYQKrPH3FjZHmEtSKoS8cM"
}
```

Figure 122: Ephemeral Private Key, in JWK Format

Performing the ECDH operation using the static EC public key (Figure 120) over the ephemeral private key (Figure 122) produces the following CEK:

```
hzHdlfQIAEehb8Hrd_mFRhKsKLEzPfshfXs916areCc
```

Figure 123: Agreed-to Content Encryption Key, base64url-encoded

5.5.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 124, encoded to base64url [RFC4648] as Figure 125.

```
{
  "alg": "ECDH-ES",
  "kid": "meriadoc.brandybuck@buckland.example",
  "epk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "mPUKT_bAWGHIhg0TpjjqVsP1rXWQu_vwVOHHtNkdYoA",
    "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs"
  },
  "enc": "A128CBC-HS256"
}
```

Figure 124: JWE Protected Header JSON
Performing the content encryption operation on the Plaintext (Figure 72) using the following:

- CEK (Figure 123);
- Initialization Vector (Figure 121); and
- JWE Protected Header (Figure 125) as authenticated data produces the following:

- Ciphertext from Figure 126.
- Authentication Tag from Figure 127.

**Figure 125: JWE Protected Header, base64url-encoded**

**Figure 126: Ciphertext, base64url-encoded**

**Figure 127: Authentication Tag, base64url-encoded**

### 5.5.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 114)
- Initialization Vector (Figure 110)
- Ciphertext (Figure 115)
- Authentication Tag (Figure 116)
Only the general JWE JSON Serialization is presented because the flattened JWE JSON Serialization is identical.

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOlJFQ0RILUVTTiw2IjoibWVyaWFkb2MuYnJhbmr5YnVja0B1dWNrbGFuZC5leGFtcGxlIiwizXBrIjw7Imt0eSI6IjI1VrIj7IjoiUC0yNTYiLCJ4IjoiVHlwZyIiLCJpZCI6IjE5OTgifQ.
```

```
yc9N8v5sYyv3iGQT926I Ug.
```

```
BoDlwPnTypYq-ijvmQvAYJLB5Q61-F3L1gQomlz87yw40PKbWE1zSTEFjDFhu9
IPrO8A9mL4m7iDFW4A-1ZxyHtElDw4lRKGMeSDiqAYtswTTmzmsNa-_4F_e
vAPUmw10-ZG45Mngq4uhMlfm_D9rBtWolqZSF3xGNNkPOQKF1C1818wjrli7-
IXgyirlKQsbbhqHzxv8icYa6L124j03C-AR21e1r7URUHAr798ySo0zU0lw1-
sD5P2134NDCe19ko1AfsXWmySPOeRb2N15ZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K7679C9Azp73gKZD0DyUn1mn0WW5LMyx_yJ-3A0Q8pwZBFg-ZyJ61
95_JGG2m9Csg
```

Figure 128: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
    "protected": "eyJhbGciOlJFQ0RILUVTTiw2IjoibWVyaWFkb2MuYn
Jhbmr5YnVja0B1dWNrbGFuZC5leGFtcGxlIiwizXBrIjw7Imt0eSI6IjI1VrIj7IjoiUC0yNTYiLCJ4IjoiVHlwZyIiLCJpZCI6IjE5OTgifQ",
    "iv": "yc9N8v5sYyv3iGQT926I Ug",
    "ciphertext": "BoDlwPnTypYq-ijvmQvAYJLB5Q61-F3L1gQomlz87yw40PKbWE1zSTEFjDFhu9
IPrO8A9mL4m7iDFW4A-1ZxyHtElDw4lRKGMeSDiqAYtswTTmzmsNa-_4F_e
vAPUmw10-ZG45Mngq4uhMlfm_D9rBtWolqZSF3xGNNkPOQKF1C1818wjrli7-
IXgyirlKQsbbhqHzxv8icYa6L124j03C-AR21e1r7URUHAr798ySo0zU0lw1-
sD5P2134NDCe19ko1AfsXWmySPOeRb2N15ZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K7679C9Azp73gKZD0DyUn1mn0WW5LMyx_yJ-3A0Q8pwZBFg-ZyJ61
95_JGG2m9Csg",
    "tag": "WCCKNa-x4BeB9hIDIFuuhg"
}
```

Figure 129: General JWE JSON Serialization
5.6. Direct Encryption Using AES-GCM

This example illustrates encrypting content using a previously exchanged key directly and the "A128GCM" (AES-GCM) content encryption algorithm.

Note that whitespace is added for readability as described in Section 1.1.

5.6.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 130.
- "alg" parameter of "dir".
- "enc" parameter of "A128GCM".

```
{
  "kty": "oct",
  "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
  "use": "enc",
  "alg": "A128GCM",
  "k": "XctOhJAkA-pD9Lh7ZgW_2A"
}
```

Figure 130: AES 128-Bit Key, in JWK Format

5.6.2. Generated Factors

The following is generated before encrypting:

- Initialization Vector; this example uses the Initialization Vector from Figure 131.

refa467QzzKx6QAB

Figure 131: Initialization Vector, base64url-encoded
5.6.3. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 132, encoded as base64url [RFC4648] to produce Figure 133.

```json
{
    "alg": "dir",
    "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
    "enc": "A128GCM"
}
```

Figure 132: JWE Protected Header JSON

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MTdiWmI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0

Figure 133: JWE Protected Header, base64url-encoded

Performing the encryption operation on the Plaintext (Figure 72) using the following:

- CEK (Figure 130);
- Initialization Vector (Figure 131); and
- JWE Protected Header (Figure 133) as authenticated data produces the following:

- Ciphertext from Figure 134.
- Authentication Tag from Figure 135.

```
JW_i_f52hww_ELQFGaYyeAB6HYGcR55919TYnSovc23XJoBcW29rHP8yZOZG7YhLpTlbjFuvZPQS-m0IFtVcXKZXdH_lr_FrdYt9HRUYkshtrMmIUAYGmUnd9zMDB2n0cRDIAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_BIwCPTjbb9o0sBdcdREEMJMy2BH8ySWMV1lgPD9xy1-aQpGbSv_F9N4IAXscj5g-NJsUPbjk29-s7LGAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZRSInZI-wjsY0yu3cT4_aQ3i0o-tie-F8Ios61EkgyIQ4CWao8PFMj8TTnp
```

Figure 134: Ciphertext, base64url-encoded

```
vbb32Xv1lea20tmHAdccRQ
```

Figure 135: Authentication Tag, base64url-encoded

Miller                        Informational                    [Page 71]
5.6.4. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 133)
- Initialization Vector (Figure 131)
- Ciphertext (Figure 134)
- Authentication Tag (Figure 135)

Only the general JWE JSON Serialization is presented because the flattened JWE JSON Serialization is identical.

The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQyY3Mi02MTdiNWl0NTI0M2EicmVxdWVzdC1pZG4iLCJlbmMiOiJBMTI4R0NN
..refa467QzzX6QAB.JW_i_f52hwv_ELQPGaYyeAB6HYGcR55919TYnSovc23XJoBcW29rHP8yZ0ZG7YhLpTlbjfFuvZPjQS-m0IFtVcXkXZdH_lR_FrdYt9HRUYksht
mIAyGmUnd9zMDB2n0cRD1HzFVeJUDxkUwVAE7_YGRPdcqMy1BoCO-FBdE-Nceb4h3-FtBP-c_BwCPTjb9o0SbdcdREEMJMy2BH8ySWMV1lgPD9yi-aQpGbSv_F9N41ZAxscj5g-NJsnUPb9k29-s7LAAGb15wEBtXphVCgyy53CoIKLHHeJHxex45Uz9aKZRSInZI-wjsY0yu3CT4_aQ31lo-tIE-F8ios61EKgyIQ4Cwao8FFMj8TTnp
vbb32Xv1lea20tMAdccRQ

Figure 136: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2tODY3Mi02MTdiNW1ONTI0M2EiLCJ1bnMiOiJ1bmti4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hwv_ELQP6ayAB6HYGcR55919TYnSovc23XJoBcW29rHP8yZ0Z7Yh LPCtibzFuVZPjQS-mOIfTvckXkZxh_1r_FrdYt9HRUYkshtrMmIUAyGmUnd9zMBB2n0cRIHHzFVeJUDxkUwVVAE7_YGRPdcqMyiBoCO-FCBe-NceB4h3-FtBP-c_BtwCPTj9b9o0SbdcdREEMJMyZBH8ySMWVi1gPD9x1i-aQqGbSv_F9N41ZAxscj5g-NJswjUbjK29-s7LJAGb15wEBtXphVCggy53CoIKLHHeJHxex45Uz9aKZSRSIn2I-wjsY0yu3cT4_aQ310-tIE-F8Ios61EKgyIQ4CWao8PFMj8TTn",
  "tag": "vbb32Xv1le20tmHAdccRQ"
}
```

Figure 137: General JWE JSON Serialization

5.7. Key Wrap Using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2

This example illustrates encrypting content using the "A256GCMKW" (AES-256-GCM-KeyWrap) key encryption algorithm with the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

Note that whitespace is added for readability as described in Section 1.1.

5.7.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- AES symmetric key; this example uses the key from Figure 138.
- "alg" parameter of "A256GCMKW".
- "enc" parameter of "A128CBC-HS256".
{  
  "kty": "oct",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "use": "enc",
  "alg": "A256GCMKW",
  "k": "qC571_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQs7sI8"
}

Figure 138: AES 256-Bit Key

5.7.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 139.
- Initialization Vector for content encryption; this example uses the Initialization Vector from Figure 140.

Figure 139: Content Encryption Key, base64url-encoded

Figure 140: Initialization Vector, base64url-encoded

5.7.3. Encrypting the Key

The following is generated before encrypting the CEK:

- Initialization Vector for key wrapping; this example uses the Initialization Vector from Figure 141.

Figure 141: Initialization Vector for Key Wrapping, base64url-encoded
Performing the key encryption operation over the CEK (Figure 139) with the following:

- AES symmetric key (Figure 138);
- Initialization Vector (Figure 141); and
- The empty string as authenticated data

produces the following:

- Encrypted Key from Figure 142.
- Authentication Tag from Figure 143.

`lJf3HbOApxMEBkCMoOTnnABxs_CvTWUmZQ2E1LvYNok`

Figure 142: Encrypted Key, base64url-encoded

`kfPduVQ3T3H6vnewt--ksw`

Figure 143: Authentication Tag from Key Wrapping, base64url-encoded

5.7.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 144, encoded to base64url [RFC4648] as Figure 145.

```json
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "kfPduVQ3T3H6vnewt--ksw",
  "iv": "KkYT0GX_2jHlfqN-",
  "enc": "A128CBC-HS256"
}
```

Figure 144: JWE Protected Header JSON
Performing the content encryption operation over the Plaintext (Figure 72) with the following:

- CEK (Figure 139);
- Initialization Vector (Figure 140); and
- JWE Protected Header (Figure 145) as authenticated data produces the following:
  - Ciphertext from Figure 146.
  - Authentication Tag from Figure 147.

Figure 145: JWE Protected Header, base64url-encoded

Figure 146: Ciphertext, base64url-encoded

Figure 147: Authentication Tag, base64url-encoded
5.7.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 145)
- Encrypted Key (Figure 142)
- Initialization Vector (Figure 140)
- Ciphertext (Figure 146)
- Authentication Tag (Figure 147)

The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDBmMjQwMDk5MTg3IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIiwiaXNzIjoiQTEyOENCQy1IUzI1NiJ9

.1Jf3HbOApxEyEBkCMMtnnAbXS_CvTWUmQ2EiLVYNok

.gz6NjyEFNnm_vm8Gj6FwoFq

.Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8EqoDZHylFKFBupS8iaeEvIGMqWmsuJKuoVqzRJYfzoMd3GxEm3VxNhzWyWtZX0gxKdy6HgLvqoGNbZCzLjqcDiF8qq2_6EVAbr2uSc2oaxFmFuIQHLCqAHxys0I444kxkJ7ewzZaGV3eFqhpco8o4DijxaG5_7kp3h2caJrFdgymuxUbgWgqaeNQaJtvJmSMFwUOEAzw9Hdeb6ydYnnCRMu-kqtO5Dec41T20MZKpnxc_F1_4yDJFcbq5CiuSmA-psB2k0JtjxAj4UPI61oONK7zzFlu4gBfjCnds2fdvG7h8wgjV98QhrKEr7xKZ3KCr0_qR1B-gxpNk3xWU

.DKW7jrb4WaRSNfbXVP1T5g

Figure 148: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "lJf3HbOApxMEBkCMOoTnnABxs_CvTWUmZQ2E1LvYNok"
    }
  ],
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRM1QzSDZ2bmV3dC0ta3N3IiwiaXYoiJLal1UMEEdYxzJqSGxmcU5fIiwizW5jIjoiQTEyOENCQy1IuI1NiJ9",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJ1Jy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVqD8E
qoDZHyFKFBupS8iaEeV1gMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhZWyW
XZK0gxKdy6HgLvqoGNbZCgLqcpDiF8q2_62EVAbr2uSc2oaxFmFuIq
HLCqAHxy51449kxj7ewz2aGV3eFghpco8o4DiXaG5_7kp3h2cajRfd
gymuxUbWgLqaeNQaJtvJmSMFuEOSAzw9Hdeb6yhdtynCRmu-kqtO5Dec4I1T02M2Knxc_F1_4yDJFcqB5CdSmA-psB2k0JtjxAj4UPI6IoONK7z
zFIu4gBfjJCndsZfdVG7h8wGjV98QhrKEr7xKZ3KCr0_qRLB-gxpNk3
xNU",
  "tag": "DKW7jrb4WaRSNfbXVP1T5g"
}
```

Figure 149: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
    "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJpdiI6IktrWVQwR1hfMm
piBjZt18iLCJraWQiOiJOGVjMDhjM1iZmE5LTRkOTUtYjIwNS0yYj
RkZDFkNDMyMWQ1LCJ0YWciOiJrZlBkdVZRM1QzSDZ2bmV3dC0ta3N3Ii
wiZw5jIjoiqTEyOENCQy1IuzIINiJ9",
    "encrypted_key": "lJf3HbOApxMEBkCMOoTnnABxs_CvTWUumZQ2ElLvYNo
k",
    "iv": "gznjyEFNm_vrn#g6j6Fw0FQ",
    "ciphertext": "Jf5p9-2hJjLy_qyKfim0Ro7w7G1QiaZp180aiVqD8E
qoDZHyFKFBupS8iaEeVfgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhzwWyW
tZKX0gxKdy6HglVqoGnbZCzLjQcpDiF8q2_62EVAbr2uSc2oaxFmFuIQ
HlcqAHxy514494kJz7ewzZaGV3eQghpc08o4DjXaG5_7Kp3h2ca1RfD
gymuxUbWqLqaeNqAjtvJmSMFuEOAzw9Hdeb6yhdTynCRmu-kqto5Dec
41T2OMZKpunxc_Fl_4yDFCqb5C1DSmA-psB2k0JtjxAj4UI61oONK7z
zFlu4gBfjCnzs2fdvG7h8wGy98QhrKEtR7xKz3KCr0_qR1B-gxpNk3
xWU",
    "tag": "NvBveHr_vonkvlfnUrmbQ"
}
```

Figure 150: Flattened JWE JSON Serialization

5.8. Key Wrap Using AES-KeyWrap with AES-GCM

The following example illustrates content encryption using the "A128KW" (AES-128-KeyWrap) key encryption algorithm and the "A128GCM" (AES-128-GCM) content encryption algorithm.

Note that whitespace is added for readability as described in Section 1.1.

5.8.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- AES symmetric key; this example uses the key from Figure 151.
- "alg" parameter of "A128KW".
- "enc" parameter of "A128GCM".
Figure 151: AES 128-Bit Key

5.8.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key; this example uses the key from Figure 152.
- Initialization Vector; this example uses the Initialization Vector from Figure 153.

\[ \text{aY5_Ghmk9KxWPBLu_glx1w} \]

Figure 152: Content Encryption Key, base64url-encoded

\[ \text{Qx0pmsDa8KnJc9Jo} \]

Figure 153: Initialization Vector, base64url-encoded

5.8.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 152) with the AES symmetric key (Figure 151) produces the following Encrypted Key:

\[ \text{CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx} \]

Figure 154: Encrypted Key, base64url-encoded

5.8.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 155, encoded to base64url [RFC4648] as Figure 156.
Figure 155: JWE Protected Header JSON
{
  "alg": "A128KW",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
  "enc": "A128GCM"
}

Figure 156: JWE Protected Header, base64url-encoded
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

Performing the content encryption over the Plaintext (Figure 72) with the following:
- CEK (Figure 152);
- Initialization Vector (Figure 153); and
- JWE Protected Header (Figure 156) as authenticated data

produces the following:
- Ciphertext from Figure 157.
- Authentication Tag from Figure 158.

Figure 157: Ciphertext, base64url-encoded
AwliP-KmWgsZ37BvzCefNen6VTbrK3QMA4TkvRkH0tPlbTdhtFJgJxeVmAkJLD61AhnWGetdg11c9ADsmWgL56NyxwSYjU1ZEHcGkd3EkU0vjHi9gT1b90qSYFfeF0Lwc1T7jybYKcsiNQkcp1yeM03Omu1YS0YJVSsP7e76zaYcMv3WwrdxF18REwoHNMk2X1d2JXq6BR53TSFkyT7PwVLuc-1GwGHIQeg7gDT6xW0JqHDPn_H-puQsmthc9ZgOojmJfqqFvETUxLAF-Kjc bel'S5dNy6egwkYtOt8EIHK-oEsKYtZRa a8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kETHsSaQdYw0F0kQZF

Figure 158: Authentication Tag, base64url-encoded
ER7MWJZ1lFBI_NKvn7ZblLw
5.8.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 156)
- Encrypted Key (Figure 154)
- Initialization Vector (Figure 153)
- Ciphertext (Figure 157)
- Authentication Tag (Figure 158)

The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDkZ2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTPhYzgiLCJlbmNlMi0iJBMJ49R0NNIn0

Figure 159: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```
{
   "recipients": [
      {
         "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIqx"
      }
   ],
   "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MjgKOTEyZygiLCJlbmMiOiJBMTI4R0NNIn0",
   "iv": "Qx0pmsDa8KnJc9Jo",
   "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkHtPlbTdhTFjgJxeVmjL61AhnWGetdq11c9AdsnWgL56NywxSYyjU1ZEHcGkd3EkU0vjhi9g1b90qSYFf0LwkcTtjbYKCs1NJQkcpIpm4eM03Onm1YSoYJVspf7e6zaYcMv3Ww6xDF18REWOhNImk2Xl2Jxg6BR53TSFkyT7PwVLq-1gwtGH1Qeg7gDT6xW0JqHDPnP_h-puQsmthc92g0jmxjFqFvETUXLAF-KjcBSTS5dNy6egwkYrOt8EIHk-oEsKytZRaa876Z07UGxGIMvEmxrGCpeJa14slv2-gaq0kETHkaSqdYw0FkQ2F",
   "tag": "ER7MWJZ1FBI_NKvn7zblLw"
}
```

Figure 160: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
   "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MjgKOTEyZygiLCJlbmMiOiJBMTI4R0NNIn0",
   "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIqx",
   "iv": "Qx0pmsDa8KnJc9Jo",
   "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkHtPlbTdhTFjgJxeVmjL61AhnWGetdq11c9AdsnWgL56NywxSYyjU1ZEHcGkd3EkU0vjhi9g1b90qSYFf0LwkcTtjbYKCs1NJQkcpIpm4eM03Onm1YSoYJVspf7e6zaYcMv3Ww6xDF18REWOhNImk2Xl2Jxg6BR53TSFkyT7PwVLq-1gwtGH1Qeg7gDT6xW0JqHDPnP_h-puQsmthc92g0jmxjFqFvETUXLAF-KjcBSTS5dNy6egwkYrOt8EIHk-oEsKytZRaa876Z07UGxGIMvEmxrGCpeJa14slv2-gaq0kETHkaSqdYw0FkQ2F",
   "tag": "ER7MWJZ1FBI_NKvn7zblLw"
}
```

Figure 161: Flattened JWE JSON Serialization
5.9. Compressed Content

This example illustrates encrypting content that is first compressed. It reuses the AES symmetric key, key encryption algorithm, and content encryption algorithm from Section 5.8.

Note that whitespace is added for readability as described in Section 1.1.

5.9.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- Recipient encryption key; this example uses the key from Figure 151.
- Key encryption algorithm; this example uses "A128KW".
- Content encryption algorithm; this example uses "A128GCM".
- "zip" parameter of "DEF".

5.9.2. Generated Factors

The following are generated before encrypting:

- Compressed Plaintext from the original Plaintext content; compressing Figure 72 using the DEFLATE [RFC1951] algorithm produces the compressed Plaintext from Figure 162.
- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 163.
- Initialization Vector; this example uses the Initialization Vector from Figure 164.

Figure 162: Compressed Plaintext, base64url-encoded
5.9.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 163) with the AES symmetric key (Figure 151) produces the following Encrypted Key:

5vUT2WOtQxKWcekm_IzVQwkGgz1FDwPi

Figure 165: Encrypted Key, base64url-encoded

5.9.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 166, encoded to base64url [RFC4648] as Figure 167.

```json
{
  "alg": "A128KW",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
  "enc": "A128GCM",
  "zip": "DEF"
}
```

Figure 166: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiemlwIjoiREVGIn0

Figure 167: JWE Protected Header, base64url-encoded
Performing the content encryption operation over the compressed Plaintext (Figure 162, encoded as an octet string) with the following:

- CEK (Figure 163);
- Initialization Vector (Figure 164); and
- JWE Protected Header (Figure 167) as authenticated data produces the following:
  - Ciphertext from Figure 168.
  - Authentication Tag from Figure 169.

```
HbDtOsdailoYziSx25KEeTwmwnh8L8jKMFNC1k3zmMI6VB8hry57tDZ61jXyez
SPt0fdLVfe6Jf5y5-JaCap_jQBcb5opbmtT60uWGM1blyiMQmOn9J--Xhh1Yg0
m-BHaqfD05iTOWxPxFmUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbYBK
hpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420Tj0w
```

Figure 168: Ciphertext, base64url-encrypted

```
VILuUwuIxaLVmh5X-T7kmA
```

Figure 169: Authentication Tag, base64url-encrypted

5.9.5. Output Results

The following compose the resulting JWE object:

- JWE Protected Header (Figure 167)
- Encrypted Key (Figure 165)
- Initialization Vector (Figure 164)
- Ciphertext (Figure 168)
- Authentication Tag (Figure 169)
The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJBMTI4SlciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTPhYzgiLCJlbmMiOlBMTI4R0NNIiwiemlwIjoiREVGIn0
.5vUT2WOtQxKWeckM_IzVQwkGgz1FDwPi
. p9pUq6XHY0jfEZi1
. HbDtOsdai0YziSx25KEeTwmwnh8L8jKMFNc1k3zmMI6VBhry57tDZ61jXyezSPT0fdLVfe6Jf5y5-JaCap_JQcb5opbmT60uWGm18blyiMQmOn9J--Xhh1Yg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2BjDbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
. VILuUwuIXaLVmh5X-T7kmA

Figure 170: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

{  "recipients": [  {    "encrypted_key": "5vUT2WOtQxKWeckM_IzVQwkGgz1FDwPi"  }  ],  "protected": "eyJhbGciOiJBMTI4SlciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTPhYzgiLCJlbmMiOlBMTI4R0NNIiwiemlwIjoiREVGIn0",  "iv": "p9pUq6XHY0jfEZi1",  "ciphertext": "HbDtOsdai0YziSx25KEeTwmwnh8L8jKMFNc1k3zmMI6VBhry57tDZ61jXyezSPT0fdLVfe6Jf5y5-JaCap_JQcb5opbmT60uWGm18blyiMQmOn9J--Xhh1Yg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2BjDbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw",  "tag": "VILuUwuIXaLVmh5X-T7kmA"  }

Figure 171: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzDktYTQ2OC04MjE2MGFkOTPhYzgiLCJlbmMiOiJBMTI4R0NNIndlJmI6
  wiemlwIjoiREVGIn0",
  "encrypted_key": "5vUT2WOTQxKWcekm_IzVQwkGg21FDwPi",
  "iv": "p9pUq6XHY0jfeEZI1",
  "ciphertext": "HbDtOsdai1oYziSx25KEeTxmwh8L8jKMFNc1k3zmMI6V
  B8hry57tDZ61jXyezSPt0fdLVfe6Jf5y5-JaCap_JQbc5opbmT60uWG
  m18blyiMqmOn9J--Xhh1Yg0m--BHaqfDO5iTOWxPxFMUedx7WCy8mxgDH
  j0aBMG6152PsM-w5E_o2B3jDbbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxE
  xCpG3mK420TjOw",
  "tag": "VILuUwuIxalVmh5X-T7kma"
}
```

Figure 172: Flattened JWE JSON Serialization

5.10. Including Additional Authenticated Data

This example illustrates encrypting content that includes additional authenticated data. As this example includes an additional top-level property not present in the JWE Compact Serialization, only the flattened JWE JSON Serialization and general JWE JSON Serialization are possible.

Note that whitespace is added for readability as described in Section 1.1.

5.10.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- Recipient encryption key; this example uses the key from Figure 151.
- Key encryption algorithm; this example uses "A128KW".
- Content encryption algorithm; this example uses "A128GCM".
- Additional Authenticated Data; this example uses a vCard [RFC7095] from Figure 173, serialized to UTF-8.
["vcard",
  [
    "version", {}, "text", "4.0" ],
  ["fn", {}, "text", "Meriadoc Brandybuck" ],
  ["n", {},
    "text", [
      "Brandybuck", "Meriadoc", "Mr.", ""
    ]
  ],
  ["bday", {}, "text", "TA 2982" ],
  ["gender", {}, "text", "M"]
]

Figure 173: Additional Authenticated Data, in JSON Format

NOTE: Whitespace between JSON values was added for readability.

5.10.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 174.
- Initialization Vector; this example uses the Initialization Vector from Figure 175.
- Encoded Additional Authenticated Data (AAD); this example uses the Additional Authenticated Data from Figure 173, encoded to base64url [RFC4648] as Figure 176.

75m1ALsYv10pZTKPWrsqdg

Figure 174: Content Encryption Key, base64url-encoded

veCx9ece2orS7c_N

Figure 175: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc21vbiiS30sInRleHQiLCI0LjAiXSiXbIm2uIix7fS
widGV4dCIsIkJyYW5keWJ1Y2siXSiXbIm4iLHt9LCJ0ZXh0Iixb
IkJyYW5keWJ1Y2siLCJNZXJpYWRvYysIKkiyLiIsIiJdXSxBImJkYXxiLHt9LC
J0ZXh0IiwiVEEgMjk4MiJdLPSiZ2VuZGVyIix7fSwidGV4dCIsIkJ0iXVi

Figure 176: Additional Authenticated Data, base64url-encoded
5.10.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 174) with the AES symmetric key (Figure 151) produces the following Encrypted Key:

4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X

Figure 177: Encrypted Key, base64url-encoded

5.10.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 178, encoded to base64url [RFC4648] as Figure 179.

```json
{
  "alg": "A128KW",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
  "enc": "A128GCM"
}
```

Figure 178: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

Figure 179: JWE Protected Header, base64url-encoded

Performing the content encryption operation over the Plaintext with the following:

- CEK (Figure 174);
- Initialization Vector (Figure 175); and
- Concatenation of the JWE Protected Header (Figure 179), ".", and the base64url [RFC4648] encoding of Figure 173 as authenticated data

produces the following:

- Ciphertext from Figure 180.
- Authentication Tag from Figure 181.
The following compose the resulting JWE object:

- JWE Protected Header (Figure 179)
- Encrypted Key (Figure 177)
- Initialization Vector (Figure 175)
- Additional Authenticated Data (Figure 176)
- Ciphertext (Figure 180)
- Authentication Tag (Figure 181)

The JWE Compact Serialization is not presented because it does not support this use case.
The resulting JWE object using the general JWE JSON Serialization:

```
{"recipients": [
  {"encrypted_key": "4Yi1Q_ZzH76TaIkJyfRPGovg9MiDnx4X"}
],
"protected": "eyJhbkGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTPhYzgiLCJlbmMiOiJJBMTI4R0NNIn0",
"iv": "veCx9ece2orS7c_N",
"aad": "WyJ2Y2FyZCIsW1sidmVyc21vbIIs30sInRleHQiLCI0LjAiXxbImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4iLHt9LCJ0Zxh0IixbIkJyYW5keWJ1Y2siLCJhZG9jIEJyYW5keWJ1Y2siY3RvYisIk1lyLiIiIiJdXSxbImjXxKiLHt9LCJ0Zxh0IiwiveEgMj44Mj45LFsiZ2VuZGVyIix7fSwidGV4dCIsIk0iXV1d",
"ciphertext": "Z_3cbr0k3bVM6N3oSNmH7Lyf3iPppGf3Pj17wN2qteJ0U1p9S7cP8xygM1oPFWCNzeA6s6BcEtp8qEFiqTEyJiNkOWDNoF14T_4NFqF-p2Mx8zbKxi7oPK8KHarFbyxDVICNqBLba-v3uzXbdB89fzOI-Lv4Pj0FAQGHRgv1rJXAmKbkgkft9cB44WeyZw8M1dbBhc-V_KWZslrsLNyg0n_JJWd_ek6LQn5NRcHvApqf9ZRxR4aq3FXBx0CxCys3P+Cdaggy2kufU120kWnWUbQXVD1C6xLIIqHhCwXDG59wRHRDqehyMROb1joV3X_bUTJDnKFO0d7nLz-cj48JMr3SnC2TpbQAkPv",
"tag": "v0A_H_Rajnpy_3hOtqvZHRa"
}
```

Figure 182: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04M
MyLTQzZDktYTQ2OC04MjE2MGFkOTPhYzg1LCJlbnMiI0IJBMTI4R0NNIn
0",
  "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFgOV9Mlpx4X",
  "aad": "WyJ2Y2FyZCIwI1sidmVyc21vbisi3osInRleHB1LCI0jAiXSe
Im2uiix7fswidGV4dCiIsIk1lcmlhZG9jIxEyYW5keWJ1Y2siXSeiMi4i
LHt9LCJ0XH0iixbiKJyYW5keWJ1Y2siLCJlNCJzJpYWRvYyIsIlkyLiiS
IiJrXSeiXbiMjkYXxiLiHht9LCJ0XH0iiviVEEgMjk4MiJdLFlisi2VuzG
VyiIix7fswidGV4dCiIsIk0iXViD",
  "iv": "veCx9ecejO2orS7c_N",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmH7Lyf3iPPgf3jPj1wNZqteJ0
U18p74SchQP8ygM1oFRWCNzeIa6s6BcEtpe8qEFlqTUEyI1kOWDNsOf14
T_4NFqF-p2Mx8kRKX7oPK8KNarFbyxiDvICNqBLba-v3uzXbDB89fz
O1-Lv4PjOFAQGHrgv1rjXAmBkgkft9cB4WeyZw8M1dbBhc-V_KWzslr
LNYgon_JJWd_ek6LQn5NHrevApqf92rXa4q3FXB0xCys35FhCdaggy
2kfu120kwKnWUBgXV1C5I6XLI1qHhCwXDG59weHRDQeHyMRoBljoV3
X_bUTJDnKBF0o7nLz-cj48JMX3SncZC7pBQAkFV",
  "tag": "vOaH_Rajnpy_3hOtqVZHRA"
}
```

Figure 183: Flattened JWE JSON Serialization

5.11. Protecting Specific Header Fields

This example illustrates encrypting content where only certain JOSE Header Parameters are protected. As this example includes parameters in the JWE Shared Unprotected Header, only the general JWE JSON Serialization and flattened JWE JSON Serialization are possible.

Note that whitespace is added for readability as described in Section 1.1.

5.11.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- Recipient encryption key; this example uses the key from Figure 151.
- Key encryption algorithm; this example uses "A128KW".
- Content encryption algorithm; this example uses "A128GCM".

Miller Informational
5.11.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 184.
- Initialization Vector; this example uses the Initialization Vector from Figure 185.

WDgEptBmQs9ouUvArz6x6g

Figure 184: Content Encryption Key, base64url-encoded

WgEJsDS9bkoXQ3nR

Figure 185: Initialization Vector, base64url-encoded

5.11.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 184) with the AES symmetric key (Figure 151) produces the following Encrypted Key:

jJIcM9J-hbx3wnqhf5FlkEYos0sHsF0H

Figure 186: Encrypted Key, base64url-encoded

5.11.4. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 187, encoded to base64url [RFC4648] as Figure 188.

  {  
    "enc": "A128GCM"
  }

Figure 187: JWE Protected Header JSON

eyJlbmMiOiJBMTI4R0NNIn0

Figure 188: JWE Protected Header, base64url-encoded
Performing the content encryption operation over the Plaintext with the following:

- CEK (Figure 184);
- Initialization Vector (Figure 185); and
- JWE Protected Header (Figure 188) as authenticated data

produces the following:

- Ciphertext from Figure 189.
- Authentication Tag from Figure 190.

Figure 189: Ciphertext, base64url-encoded

fNYLqpUe84KD45lvDiaBAQ

Figure 190: Authentication Tag, base64url-encoded

5.11.5. Output Results

The following compose the resulting JWE object:

- JWE Shared Unprotected Header (Figure 191)
- JWE Protected Header (Figure 188)
- Encrypted Key (Figure 186)
- Initialization Vector (Figure 185)
- Ciphertext (Figure 189)
- Authentication Tag (Figure 190)

The JWE Compact Serialization is not presented because it does not support this use case.
The following JWE Shared Unprotected Header is generated before assembling the output results:

```json
{
  "alg": "A128KW",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
}
```

Figure 191: JWE Shared Unprotected Header JSON

The resulting JWE object using the general JWE JSON Serialization:

```json
{
  "recipients": [
    {
      "encrypted_key": "jJIcM9J-hbx3wnghf5FlkEYos0sHsF0H"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D
               M3swKkjOWqYztWswFLYMj5YeiLt_StAn21tHmQjuNt64T8d4t6C7kc9O
               CCJ1IHaolUV4MyOt80MoPb8fZYbNKnqplzYJgL58g8N2v460gyG637d6
               uuKpwhAnTGm_zWhqc_srOvgILkzyFXPq1hBAURbc3-8BqeRb481R1-5
               g5UjWVD31gLcLCN_P7AW8mIlFvUNXBPJ3nOWL4teUP8yHlbWcL83olU
               4UAgL48x-8dDkH23cJykibVSQju-f7e-1xreHwxzWLh1Ngbbre0dEwK3
               HX_xM01jUz77Krppgoutp5qKg31-_xMINmf",
  "tag": "fNYLqUe84KD451vDiaBAQ"
}
```

Figure 192: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
"protected": "eyJlbmMiOiJBMTI4R0NNIn0",
"unprotected": {
"alg": "A128KW",
"kid": "81b20965-8332-43d9-a468-82160ad91ac8"
},
"encrypted_key": "jJIcM9J-hbx3wnqh5f1kEYos0sHsF0H",
"iv": "WgeEJsDS9bkQxQ3nR",
"ciphertext": "jJIBcyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D
M3swKkjowQyZtW6LYm5Yelht_STAn2itHmQJuNt64TBD4t6C7kC90
CCJJIHao1Uv4MyOt80MoPb8f2YbNkqplzYJgl5h58g8N2v460qyG637d6
uuKpwhAnTgm_zWhqc_srovq1LkzxyFXPqihBAURbc3-8BqeRb48iR1|-_5
g5UjWVD31giLCN_P7AW8mIiFvUNXBPJK3nOWL4teUPS8yHLbWeL8301U
4UAgL48x-8dDKh23JykibS0Jhu-f7e-1xreHWxzl1NqBbre0dEwK3
HX_xM0ljUz77KrppegoutpF5qaKg31-_xMIMrf",
"tag": "fNYLqqUe84KD45lvDiaBAQ"
}
```

Figure 193: Flattened JWE JSON Serialization

5.12. Protecting Content Only

This example illustrates encrypting content where none of the JOSE header parameters are protected. As this example includes parameters only in the JWE Shared Unprotected Header, only the flattened JWE JSON Serialization and general JWE JSON Serialization are possible.

Note that whitespace is added for readability as described in Section 1.1.

5.12.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 72.
- Recipient encryption key; this example uses the key from Figure 151.
- Key encryption algorithm; this example uses "A128KW".
- Content encryption algorithm; this example uses "A128GCM".
5.12.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key; this example the key from Figure 194.
- Initialization Vector; this example uses the Initialization Vector from Figure 195.

KBocAP130QPV3vkcZlXnzQ

Figure 194: Content Encryption Key, base64url-encoded

YihBoVOGsR1l7jCD

Figure 195: Initialization Vector, base64url-encoded

5.12.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 194) with the AES symmetric key (Figure 151) produces the following Encrypted Key:

244YHfO_W7RMpQW81UjQrZcq5LSyqiPv

Figure 196: Encrypted Key, base64url-encoded

5.12.4. Encrypting the Content

Performing the content encryption operation over the Plaintext (Figure 72) using the following:

- CEK (Figure 194);
- Initialization Vector (Figure 195); and
- Empty string as authenticated data

produces the following:

- Ciphertext from Figure 197.
- Authentication Tag from Figure 198.
5.12.5. Output Results

The JWE Compact Serialization is not presented because it does not support this use case.

The following JWE Shared Unprotected Header is generated before assembling the output results:

```json
{
  "alg": "A128KW",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
  "enc": "A128GCM"
}
```

Figure 199: JWE Shared Unprotected Header JSON

The following compose the resulting JWE object:

- JWE Shared Unprotected Header (Figure 199)
- Encrypted Key (Figure 196)
- Initialization Vector (Figure 195)
- Ciphertext (Figure 197)
- Authentication Tag (Figure 198)
The resulting JWE object using the general JWE JSON Serialization:

```json
{
    "recipients": [
    {
        "encrypted_key": "244YHiO_W7RMpQW81UjQrZcq5LSyqiPv"
    }
    ],
    "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
    },
    "iv": "YihBoVOGsR117jCD",
    "ciphertext": "qtPIMMaOBrgASL10dNQhoa7Gqrk7Eallvwht7R4TT1uq-arsVCPaIeFwQfzrS6oEUWbBtxEasE0vc6r7spHyVzimCIVJEuRJYoAHFSP3eqQp4Ic15DSqyXjw_L3svybhHYUGyQuTMUQEDjgjJfBoifwHIsDsRPeBz1NomgeifVPq5GTCWFo5k_MNIQRR2Wj0AHC2k7JZfu2iWjUHLF8ExFZL4n1msvJu_mvfMYiikfNfs2AudISOa6073yPZtL04k_1FI7WDf
rb2w7OqKLWDXz1pcxchPVOLQwpA3mFRNKdY-bQz4Z4KX91fzc1ne31N4
-8BKnmojpw-0QjKdLQGkC445FB_Klt1DQXw2sBF",
    "tag": "e2m0V7m7Jvjk2VpCKKS-kyg"
}
```

Figure 200: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
    "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
    },
    "encrypted_key": "244YHiO_W7RMpQW81UjQrZcq5LSyqiPv",
    "iv": "YihBoVOGsR117jCD",
    "ciphertext": "qtPIMMaOBrgASL10dNQhoa7Gqrk7Eallvwht7R4TT1uq-arsVCPaIeFwQfzrS6oEUWbBtxEasE0vc6r7spHyVzimCIVJEuRJYoAHFSP3eqQp4Ic15DSqyXjw_L3svybhHYUGyQuTMUQEDjgjJfBoifwHIsDsRPeBz1NomgeifVPq5GTCWFo5k_MNIQRR2Wj0AHC2k7JZfu2iWjUHLF8ExFZL4n1msvJu_mvfMYiikfNfs2AudISOa6073yPZtL04k_1FI7WDf
rb2w7OqKLWDXz1pcxchPVOLQwpA3mFRNKdY-bQz4Z4KX91fzc1ne31N4
-8BKnmojpw-0QjKdLQGkC445FB_Klt1DQXw2sBF",
    "tag": "e2m0V7m7Jvjk2VpCKKS-kyg"
}
```

Figure 201: Flattened JWE JSON Serialization
5.13. Encrypting to Multiple Recipients

This example illustrates encryption content for multiple recipients. As this example has multiple recipients, only the general JWE JSON Serialization is possible.

Note that RSAES-PKCS1-v1_5 uses random data to generate the ciphertext; it might not be possible to exactly replicate the results in this section.

Note that whitespace is added for readability as described in Section 1.1.

5.13.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the Plaintext from Figure 72.
- Recipient keys; this example uses the following:
  - The RSA public key from Figure 73 for the first recipient.
  - The EC public key from Figure 108 for the second recipient.
  - The AES symmetric key from Figure 138 for the third recipient.
- Key encryption algorithms; this example uses the following:
  - "RSA1_5" for the first recipient.
  - "ECDH-ES+A256KW" for the second recipient.
  - "A256GCMKW" for the third recipient.
- Content encryption algorithm; this example uses "A128CBC-HS256".

5.13.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 202.
- Initialization Vector; this example uses the Initialization Vector from Figure 203.
5.13.3. Encrypting the Key to the First Recipient

Performing the "RSA1_5" key encryption operation over the CEK (Figure 202) with the first recipient’s RSA key (Figure 73) produces the following Encrypted Key:

dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w6Y5G4XJQsNNIBqyvUUA OcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXihidAYWVajJIA KMXMvFRMV6id1Rr076DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-w qjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxtj_oxtr9c1N1n2MbeYS rRicJK5xodvWgkpIdkMHo4LvdrRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-d bHcGlhhqc_wGgho9fLMK8J0ArLcMDNQ

Figure 204: Recipient #1 Encrypted Key, base64url-encoded

The following is generated after encrypting the CEK for the first recipient:

- Recipient JWE Unprotected Header from Figure 205.

```json
{
  "alg": "RSA1_5",
  "kid": "frodo.baggins@hobbiton.example"
}
```

Figure 205: Recipient #1 JWE Per-Recipient Unprotected Header JSON
The following is the assembled first recipient JSON:

```json
{
    "encrypted_key": "dYOD28kab0Vvf4ODgxAJgHCZIcOp8M51zjwj4w6Y5G4XJQsNNIB1yvVUAo0pL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJJdXidAFYWMvjJiaKMXMvFRMV6iD1Rr076DFthg2_AV0_tSiv6xSEIFq1txnYpmp91tc5WjQOgb-wqjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvym2wxj-oFnr9c1Lnl2M6beYsrRicJK5xodvWgpIkdMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-dbHcGlihq_cWGgho9fLMK8JOArYLCMDNQ",
    "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
    }
}
```

Figure 206: Recipient #1 JSON

5.13.4. Encrypting the Key to the Second Recipient

The following is generated before encrypting the CEK for the second recipient:

- Ephemeral EC private key on the same curve as the EC public key; this example uses the private key from Figure 207.

```json
{
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRcl1z5p5_r0OjeqT-I68I8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
    "y": "VDrryFJh-Kwd1ElJgmnj5EO-CTHAZ53MC7PjjpLioy3ylEjI1pOMbw91fZ84pbm",
    "d": "1DKHfTv-PiifVw2VBMH_ZiVcwOMxkOyANS_lQHJcrDxVY3jhVCvZPwMxJKIE793C"
}
```

Figure 207: Ephemeral Private Key for Recipient #2, in JWK Format
Performing the "ECDH-ES+A256KW" key encryption operation over the CEK (Figure 202) with the following:

- Static Elliptic Curve public key (Figure 108).
- Ephemeral Elliptic Curve private key (Figure 207).

produces the following Encrypted Key:

ExInT0io9BqBMYF6-maw5tZlgoZXThDlzWKsHixJuw_e1Y4gSSId_w

Figure 208: Recipient #2 Encrypted Key, base64url-encoded

The following is generated after encrypting the CEK for the second recipient:

- Recipient JWE Unprotected Header from Figure 209.

```json
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjypLioy3ylEjIlpOMb91fzZ84pbfm"
  }
}
```

Figure 209: Recipient #2 JWE Per-Recipient Unprotected Header JSON
The following is the assembled second recipient JSON:

```json
{
    "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw_e1Y4gSS1d_w",
    "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
            "kty": "EC",
            "crv": "P-384",
            "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xA
 n2DtMRb25Ma2CX",
            "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1Ej1p0
 Mbw9fZ84pbfm"
        }
    }
}
```

Figure 210: Recipient #2 JSON

5.13.5. Encrypting the Key to the Third Recipient

The following is generated before encrypting the CEK for the third recipient:

- Initialization Vector for key wrapping; this example uses the Initialization Vector from Figure 211.

AvpeoPZ9Ncn9mkBn

Figure 211: Recipient #2 Initialization Vector for Key Wrapping, base64url-encoded

Performing the "A256GCMKW" key encryption operation over the CEK (Figure 202) with the following:

- AES symmetric key (Figure 138); and
- Initialization Vector (Figure 211)

produces the following:

- Encrypted Key from Figure 212.
- Authentication Tag from Figure 213.
The following is generated after encrypting the CEK for the third recipient:

- Recipient JWE Unprotected Header; this example uses the header from Figure 214.

```json
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "59Nqh1L1YtVIhfD3pgRGvw",
  "iv": "AvpeoP29Ncn9mkBn"
}
```

Figure 214: Recipient #3 JWE Per-Recipient Unprotected Header JSON

The following is the assembled third recipient JSON:

```json
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1L1YtVIhfD3pgRGvw",
    "iv": "AvpeoP29Ncn9mkBn"
  }
}
```

Figure 215: Recipient #3 JSON

### 5.13.6. Encrypting the Content

The following is generated before encrypting the content:

- JWE Protected Header; this example uses the header from Figure 216, encoded to base64url [RFC4648] as Figure 217.
Performing the content encryption operation over the Plaintext (Figure 72) with the following:

- CEK (Figure 202),
- Initialization Vector (Figure 203), and
- JWE Protected Header (Figure 217) as the authenticated data produces the following:
  - Ciphertext from Figure 218.
  - Authentication Tag from Figure 219.
The following is generated after encrypting the Plaintext:

- JWE Shared Unprotected Header parameters; this example uses the header from Figure 220.

```json
{
  "cty": "text/plain"
}
```

Figure 220: JWE Shared Unprotected Header JSON

5.13.7. Output Results

The following compose the resulting JWE object:

- Recipient #1 JSON (Figure 206)
- Recipient #2 JSON (Figure 210)
- Recipient #3 JSON (Figure 215)
- Initialization Vector (Figure 203)
- Ciphertext (Figure 218)
- Authentication Tag (Figure 219)

The JWE Compact Serialization is not presented because it does not support this use case; the flattened JWE JSON Serialization is not presented because there is more than one recipient.
The resulting JWE object using the general JWE JSON Serialization:

```json
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOq8M51zj
                      wj4w6Y5G4XJQosNNIBgyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-
                      vhW6me4bWrBf7cHwEQdXihidAYWvajJIAKXMxvFRMV6iD1r076
                      DFlhg2_AV0_tSi6xSEIFqt1xnYPpmf91tc5WJDQGb-wqjw0-b-S
                      1laS1lQVbu78dQ7Fa0zAVzzjHX-xxyM2wxj_otxr9c1N1n2MBe
                      YSrRicJK5xodWgkp1dkMHo4LvdhRRvzoKzlic89jFwPlbBq_V4n
                      5trGuExtp_-dbHcGlihqc_wGgho9fLMK8JOArYLCMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBM6-faw5t2goZXThD1zWKsHi
                      xJuw_elY4gSSId_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3pi5wKCRe1lizp5_0OjeqT-I681zg2b8mv8diRhs
                      E2xAn2DtmRb2s5Ma2C XD",
          "y": "VDrFyFhj-Kwd1EjAgm05o-CTHAZ53MC7PjppLioy3y1Ej
                      1lpOMbw91f9xZ84pbfm"
        }
      }
    },
    {
      "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-Wy
                      TpS1E",
      "header": {
        "alg": "A256GCMKW",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59Nh11yTYVhfD3pgRGvw",
        "iv": "AvpeoPZ9Ncn9mkBn"
      }
    }
  ],
  "unprotected": {
    "cty": "text/plain"
  },
  "protected": "eyJlbmMiOiIBMTI4Q0JDLUhTMjU2In0"
}
```
"iv": "VgEIHY20EnzUtZF12RpB1g",
"ciphertext": "ajm2Q-OpPXCr7-MHXicknb1lsxLdXXK_yLds0KuhJzfwK
04SjdxQeSw2L9mu3a_k1cS55kCQ_3xlkcVKC5yr__Is48V0oK0k63_QRM
9tBURMFqLBByJ8vOYQX0oJW4VUHJlmGhF-tVQWB7Kz8mr8zeE7txF0MSa
P6ga7-siXxStr7_0G07Th1djh-zGT0wxM5g-VRoRtq0K6AXpLlwEqRp7p
kt2zRm0ZAXqSpel06FJP7FHLdyEFnD-zDIzukLpCzhzMDLlw2-8I14FQ
rgi-iEuzHgIJFIjn2wh9Tj0cg_kOZy9BqMRZbmXMY9YQjor2_p_JYG3
ARAIF3ojDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVh2ofDQdLcht
azE",
"tag": "BESYyFN7T09KY7i8zKs5_g"
}

Figure 221: General JWE JSON Serialization

6. Nesting Signatures and Encryption

This example illustrates nesting a JSON Web Signature (JWS) structure within a JSON Web Encryption (JWE) structure. The signature uses the "PS256" (RSASSA-PSS) algorithm; the encryption uses the "RSA-OAEP" (RSAES-OAEP) key encryption algorithm and the "A128GCM" (AES-GCM) content encryption algorithm.

Note that RSASSA-PSS uses random data to generate the signature, and RSAES-OAEP uses random data to generate the ciphertext; it might not be possible to exactly replicate the results in this section.

Note that whitespace is added for readability as described in Section 1.1.

6.1. Signing Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the JSON Web Token [JWT] content from Figure 222, encoded as base64url [RFC4648] to produce Figure 223.

- RSA private key; this example uses the key from Figure 224.

- "alg" parameter of "PS256".

```json
{
  "iss": "hobbiton.example",
  "exp": 1300819380,
  "http://example.com/is_root": true
}

Figure 222: Payload Content, in JSON Format

Figure 223: JWS Content, in JSON Format

Miller                        Informational                   [Page 110]
eyJpc3MiOiJ1b2JiaXRvbVsiEGFcGxldiIwZ2xhIjoxMzAwODExMzgwLCA6HlZ0bW1hIjwiaXNfcm9vdCI6bCI6NzA6

Figure 223: Payload Content, base64url-encoded

{
  "kty": "RSA",
  "kid": "hobbiton.example",
  "use": "sig",
  "n": "kNrPB5DXML6fcyv5i-OHQA-QK8gsC3HJb7FYhYyaw8hXbNJa-t8q01DkWlZgQXYV-ffWxJv5GGr1ZE4G5U2fMEegTzYTrQR3tepgKFjMGg6Ifz6k12Nsz2Gonsn1Shfz-A9GJwRTmtKBk1s-hwxi1U5AT-AIe1NqBgcFvE5W25-SGGBoaROVdUYxqETDqgM1zK4VzjD2-1h4OV07bkapcLQdHpJUuUyR_Er20DXXs30Ky197PciXKTS-QKXnnm81ivyRCmuz22z0PUIn2dBC50iG4MAHLh2Z2k8Csrdfy-7dg7z41Rp6D0zeEvtaUp4bX4AKral4rTfw",
  "e": "AQBQ",
  "d": "ZLTXEpE9-W-n2Vba-HW5uVPjtjxvWXC1JFOpsJdsea89gRMx34gEOEtNoyC2un3C3ZLTji-mju5RA78Sc76YJd3zVw0U0I8MBeG6-i0nvogobNo7K57-xjTJU272Ejor9k9B7z62KwDDq7HFCdDeUcYeCfHvc7iL_6TibVbAhOFOWNt1QlJgEgwVYd0zybNGKlfdfnEpEwyHoMwY6HMGvqNErfgP7iZ0YzHUT535x6j74VKcA7ZduFkhUauiysySEW7mxZMbfj1vdJjiIY9LD1fiz30Xv4ckqohKF5GONU6tNMmMNgAD6giViyEleIPrlxl1tBbCHL4dRW-zrpHgAQ",
  "p": "yKYyC5IaqWMrQLgIBO6T1NlbDiscDNUUe2X-RhpBaxD_mIkweM74Mg-0-B2iSyVMrsiVhOron7VvxCqagcBAATGw-hAafUehWxjWbSw-3KccRM8toL4e0q7-M-irdDBXsoe722-CV2xZC3jR3Pq642R13Wg5QM4D4bUK5ZyJcY-c",
  "q": "uND4ol5V30KDzf8vFw589p1v1QVQ3NoilrinRUPHkkxaAzDzccGrWMWpGxGFbNL3w5CqELeU76-5IYYQoOHwYV10hVXQR7sgaGu-483Ad3ENcl23FrOnF45m7_2ooastJde4M9ElTTQKsR1B1سكرVPspCzPczkh9Kk",
  "dp": "jmTnEoqZqa8ouaynjhJSCnseUXnMQC2gAnerQJRQkFQYzu-zV2PKPKNbhPvKvY1F5b2-L3t3OW2d2lNdYRUW1T7V510KrPTABStOngAmYChG18XKd1hcrt5Vf1dbakc6saw1_TZG7Y2MVXz2ZnCvCmX4VjRjXof3PHFbU",
  "dq": "R9FuVvU88V0zEzTkrX13-5-WusE4DJhmdeZ1u3rifBdfLpq-P-1WPBbGaqg9wz1c-G7SsdCqjkeJD5vd5yd27rn6kzpmBh_nbMz3scAk1qsnnt9C7JGAY7-sGWyljGSHFzafjFp52ThB4lCJOYuEam5RIzpyY77_oLAhpmDA0OhkLk",
  "qi": "SbtC7zknW6hPITkjwttQQPLVmfwirfrLEVAtu8BN9CrV_7F2OgUZCqmzH7yAumGFH1lWVf7p7a1eWaaJjxK_Lb3aqal4QH3e-P5E1KjH6IMazuRtZLUROCThrExDf5ydbsciDnfrUWLErZ4N1Be0bnxYuPqxxKd9QzW0mO"
}

Figure 224: RSA 2048-Bit Private Key, in JWK Format
6.2. Signing Operation

The following is generated to complete the signing operation:

- JWS Protected Header; this example uses the header from Figure 225, encoded using base64url [RFC4648] to produce Figure 226.

```
{
  "alg": "PS256",
  "typ": "JWT"
}
```

Figure 225: JWS Protected Header JSON

eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9

Figure 226: JWS Protected Header, base64url-encoded

Performing the signature operation over the combined JWS Protected Header (Figure 226) and payload content (Figure 222) produces the following signature:

```
dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIA
kIvnlqskChirjKv9ESZNUCF4jvfyzPS-nqjJxYoA5ztWOyFk2cZNIPXjcJXSQ
wXPo9tEe-v4VSqgD0aKHzPyog4N6Cz1lkph1UlsYDI67_bLL7elg_vkjfMp5
_W515LuUYGMeh6hxQIanUXf9EwV2JmvTMu2-vBOWy0Sniy1EFo72CRTvmtr1f5
ARo05MNiiY3KtUxeF-SOMd-LEYwW9S1kohYzMVAZDDOrVbvTKVRHpeYNaK75KE
QqdCEEkS_rskZS-Qtt_nlegTWh1mEYaa
```

Figure 227: JWS Signature, base64url-encoded

6.3. Signing Output

The following compose the resulting JWS object:

- JWS Protected Header (Figure 226)
- Payload content (Figure 223)
- Signature (Figure 227)
The resulting JWS object using the JWS Compact Serialization (which is the plaintext input to the following encryption operation):

eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlLiwiZXhwIjoxMzAwODE5MzgwLCJodHRwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0.

Figure 228: JWS Compact Serialization

6.4. Encryption Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 228.
- RSA public key; this example uses the key from Figure 84.
- "alg" parameter of "RSA-OAEP".
- "enc" parameter of "A128GCM".

6.5. Encryption Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 229.
- Initialization Vector; this example uses the Initialization Vector from Figure 230.

Figure 229: Content Encryption Key, base64url-encoded

GbX1i9kXz0sxXPmA

Figure 230: Initialization Vector, base64url-encoded
6.6. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 229) with the RSA key (Figure 84) produces the following Encrypted Key:

```
a0JHRoITfpX4qRewImjlStn8m3CPxBVlueY1VhjurCyrBg3I7YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FxnJVOGrG-bi13mwy7RoYhjTkBEC6P7sMYMXXx4g
zMedpiJHQveyI-zkZV7A9matpgevAJWrXzOysYGTtowoSN6gtUvt1Laivjvb21
O0u14YxSHV-ByK1kyeetRp_fuYJxHoKLQL9P4244sKx2WGYb4zsB1PF4ssl_e5I
R7nany-25_Umc2urosNkoFz9cQ82MypZP8ggbQjyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dMVIY7qLeKpxcV1lWQ_2FefEBqRxIxV1jLeZivjNKzogCq3-I
apSjVFmJbxjpyLT8muawolyy1XXMuinIpNc0Y3n4KkrXLRcCteX85m4I1HDMZ
a38s1Hpr56fFpsEa-Jltmt-a9iEDtOzhtxz8AXy9tsCA2V2XBWNG8c3kJusAa
mBK0Ywfk7JhLRdOnJj1JLhn7TI4UxDp9dCmUXEN6z0v23W15qJlEXN5tqnb1p
ymooeWAHCT4e_Owbimlg0AEpTHUdA2iiLN9s9WTX_H_TXuPC8yDDhi1smxS_X_x
pkIHk1IHWDOlx03BpqDTivpKkBYwqP2UZkcxqX2F0_GnVrNw1K7Lgw6FSQvDO0
```

Figure 231: Encrypted Key, base64url-encoded

6.7. Encrypting the Content

The following is generated before encrypting the Plaintext:

- JWE Protected Header; this example uses the header from Figure 232, encoded using base64url [RFC4648] to produce Figure 233.

```json
{
  "alg": "RSA-OAEP",
  "cty": "JWT",
  "enc": "A128GCM"
}
```

Figure 232: JWE Protected Header JSON

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYW1iI6IkExMjhHQ00ifQ

Figure 233: JWE Protected Header, base64url-encoded
Performing the content encryption operation over the Plaintext (Figure 228) with the following:

- CEK (Figure 229);
- Initialization Vector (Figure 230); and
- JWE Protected Header (Figure 233) as authenticated data produces the following:
  - Ciphertext from Figure 234.
  - Authentication Tag from Figure 235.

Figure 234: Ciphertext, base64url-encoded

KnIKEhN8U-3C9s4gtSpjSw

Figure 235: Authentication Tag, base64url-encoded

6.8. Encryption Output

The following compose the resulting JWE object:

- JWE Protected Header (Figure 233)
- Encrypted Key (Figure 231)
- Initialization Vector (Figure 230)
- Ciphertext (Figure 234)
- Authentication Tag (Figure 235)
The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ.
a0JHRoITfpX4rQwemJ1stn8m3CPxBV1ueY1VhjxurCyrB9g3t17YuCRyjphDOC4S
E7zXbrZn6NyQq-A-gqT0FXnqJVOGr-gbi13mwy7RoYhjTkPAC6P7sMNN6xXg
zMedipiHJQveyl-I-zkZ7V9matpgevAAWrxZ0UsYsYTtowoSN6gtUVt1Laivjvb21
O0ui4YxSHV-ByKj1kjeypPf_fuYjxHoKLQL9P424sXk2WGYb4zsBIIPF4ssl_eI
R7nanye-25-UmC2uroskNkoFz9cQ82MyqZP8ggbQjyPN-FpP4z-So6yV64x6yzDU
F_J5CIdl-Qv6H5dMV1Y7q1eKpCv11WQ_2PefEBqxXvIjLeZ1vJNkzogCq3-I
apSjVFnmjBxjypYLT8muaw01yy1XXMuinIpNCoy3n4KnZKXUCCt3esX85m4IIHMZ
a38s1hpz56fPpseMA-Jltmt-a91EDtOzhtxZ8Axy9tsCAZ2V2XBNW8c3kKusAA
mBKOYwfk7hLRDOnJj1JLhn7TI4Xdp9dcMUXEN6z0v23W15qJIXNtqnbIp
ymoOEWAHCT4e_Owbimlg0AepTHUDA2itILNs9WTX_H_TXuPC8yDDDh1spxs_S_x
pKIHkIIHWDOLx03BpqDTipvKkBYwqP2UZkcqyX2Fo_GnDrNw1K7Lgxw6FSQvDO
0.
GbX1i9kXz0sxXPma.
S2I4IVKkmwpa2l_pQXX3mHv1ANnOU4Wf9-utWYUcKBNgCe2OMfM66cSJ8k2Q
kxaj3D_860MG79ofomwtky3OFYMeGRjtpMt90AVVLSAXB0_UTCBGy8gb3C2bWLX
q2fJAJaORUpPkm-BimZY81zVBU1hc7HzQEPcpu33szMsFhjn41P_idrZj2g12
TNqKdt8znUPauKTKNOH1DD4fuzvDYFDIaqGPY1L5sVR0b1XpxDgkEszM-9C
hMPqW1QNhuzUX_zul3bwrjw7ruGZs4cUScY3n8yE3AHCLurqls-A9mz1K38xEl
uL1814fg9tLejdkAuQZjPbqfHQBjE4lWGD5mE0dQ-Mtzf4NhkkIwx-YKBbxo2
zi3Q_isyJKuils7YWW-HTR_vqptFt0bj7WjFzvB0TZ3dvs0GaTVpH2dyWwunUr
lx4gMNuzBdwG6ubtSDEEUZ5py0d_OtWeUSyCybKd-aM7tXg26qJc21gYjlf
hn9zy-W19sOZGUZgFjPaw2Kpvnj-t-0_ES96kogjJLxS1M9Y5xnwZMyNc
9E1wngosC-hVwzlyF0S1rukt914_SLLXeM17o03phcTMxt1MrzR88NKUIWkB
siXMCjy1Noe1MD-SHdp5dmM
.KnIKEhN8U-3C9s4gtSpjSw

Figure 236: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```
{
"recipients": [
{
"encrypted_key": "a0JHRoITfpX4qRewImj1Stn8m3CPxBV1ueY1Vh
juCyBrg3I7yYCFhjhpDOOS4E7rXbr2FN6NyQq-A-gqT0FXqNjVO
GrG-bi3mwy7RcYhjTkBEC6P7sMYMXx4gzMedp1JHQVeyI-zkZV
7A9matqgeVAjWzx3yFy2YTtwSN6gtUVT1laivjvb2l0ou1wY
SV-ByKkicyetP_fuYJKoQL9P424sKx2WGYb4zSkIPF4sdl_e
5IR7nany-25_UmC2urosNkoFz9cQ82Myp2P8gbqQyPN-Fpp4Z-
06yV64x6y6zDUF5_JCld1-Qv6H5dMV1Y7qleKpXcV11wo_2FeEBq
XxXvJiJeZivjNkgqC3-IapSjVFmJbXjyYLT8muaw0lyy1XXM
uinIpc0NcY3n4KkRlLrCcxe85m411HMMZa38shPr56fPPseMA-JL
	mt-a13EdtOzhtxz8AXy9tsCAV2YBGN9c3KusAAmBOYyfK7J
hLDRGoJnJlJLHn7tI4UXdp9dCmUXEN6z0v23w15qJIEJXJtg
bNp'moowWAHCT4e_Owbin1qOeThiRd2iiNsw9WTX_H_TXUCy8Dh
il3mS_x_xpxiHkiIHWDOLx03BpqDTlvpKkBYwpP2UZcqxqX2Fo_
GnVrNw1K7Lgxw6FSqVD00"
}
],
"protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIImVuYy
I6IKeMjhHQ0IfQ",
"iv": "GbXl19kXz0sxXpMa",
"ciphertext": "SZI4IfKHmwpazl_pJOXX3mHv1AAnO4wF9-uTYWYkRb
ngCe2OMF66cS38k2QXaaQD3_R60MG9foIacmtky3GFpMxMGRbptMT9oA
VLSx9B0-UTCGBYgB3c2BWLxZ1IfJAAoJRUPRk-BImYZt8zvBul7Hs
QePCpu33zSzmFhj4lP_idjz_gIZvN9KfO8dzdUNAuBkTMDNoH1lD4fu
zVdYDfa1fQyP55sVRwbXpDgOuEs2M-9ChMPFqW1QNhzuX_2ul3brvJ
wr7nu2Gz4cUScY3n8yE3AHCLIurqsl-A9mziX3XeaulV1814F9tLejd
kAuQzJpbq6HQBJe4tWgD5EEdq-Mtz4努hpm1Wx-YKoB_xo2zi3Q_isYj
KUois7wWw-HTr_qvxFtOibj7WwJf2vzB0TZ3dvs6aTvPH2dyWwumUr1x
4gmPUZ6w06ubfYSDPZeWEbEm-eQ-Mt7XfX2g6qJo2lgY
jlIfn9g7-W19sO6ZrGuPqFjuwXHpvtn_j-0_E596kogjJLxS11UM0Y5
Xmn2WMyNc9E1wongsCp-hVuypyP0sIruktM194_SLIqM170o3phcTMX
tlMizR8skUW1bSxjXMcjy1Noue7MD-ShDp5dmM",
"tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

Figure 237: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "encrypted_key": "a0JHRoITfpX4gRewImjlStn8m3CPx8dXVlueY1VhjurC
  yrB3I7YhCRYjhpDOOS4E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bi13
  mwy7RoYhTkJkBE6C6P7mSMYMMx4gzmMedpiJHqVeyI-zkZV7A9matpgevA
  JrWrXzOyUsYGTtowSN6gtUVt1alvjbvB2100u14YxSHV-ByK1kyeetRp_f
  uYJxHoKQLqL942sKx2WGYy4zsBIFP4ssl_e5lR7nany-25_UmC2uros
  NKofz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDUF_5JCl1-d1-Qv6
  H5dMVlY7qlKeKpXcVIIW0_2FeEBqXXvIjLeZivjNkzogCq3-IapSJyF
  nMjBxjvpYLT8muawolVy1XXMunInPncOY3n4KKrXLrCtexeS85m4lIHMZ
  a38lsHpr56FPsEaM-Jltmt-a9IEdOztzxtx8Axy9tsCAZV2XBNGw8c3
  kJusAamBKOYWfK7JhLk0gOnJj1Lln7T14Ux7D9cmUXEN8z0v23W15q
  JIEXNjtnbipiymoeeWAHC4T4e_Owimb1g0AEpTHUdA2ii1lns9WTX_H_TX
  uPC8yDh1lsmxs_X_xpkiIHkiiHWDOLox3BpqDTivpKkBWyqP2UZkcxqX
  2Fo_GnVrNw1K7Lgxxw6FSQvD0O",
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYy
  I6IKeXmhjHQ001ifQ",
  "iv": "GbX1i9kXz0sxXpMa",
  "ciphertext": "SZI4lVKhmpwazl_pJQX83mHv1ANnOU4wf9-utWYUcKrBN
  gC2OFMf66cSj8k2QXqaQD3_R60MG9ofofomtzy3GFxMeGRjtpM79QAv
  VLSaAXB0_UTC6ByBg3C2bxWx2lfJAJoJRUPRk-BimYY5lVBu1hC7Hs
  QePcPu33S2MsFhn4IP_idrJz_g12TNkKd8ztznPauKTDNOH1DD4fu
  zvDYfDAf0qyPl5sVwbiXpXGokEsZm-9ChMnPq1QNhzuX_2ul3bcvJr
  wz7nuGZs4cUCy3n8yE3AHCLurgls-A9mz1X8xeAuv1814Fg9tLejd
  kAuQ2jJpbeQHQBje41wGD5e0dQ-MtZ4Nhkk1Wx-YKBb_Xo2zi3Q_1sYj
  KUuis7yWw-HTr_vqvtF0b7j7WjF2vB0T23dvsoGaTvPH2dyWwumUr1x4
  qmPUzBw7D0ubfYSDUEEzSpv0d_OtWeUSyCYYBKd-aM7xg26qJo21gY
  jlfhn9zy-W19sOCZGuSgFJPhawXhpvvnj_t-0_ES96kogjJLxS1M9U9Y5
  XmmZ2MyNc9ElnwogsCg-hVuvyzFP0s1ruktm194_SL1xgM7003phcTMx
  t1MizR88KUy1KsliMCjy1Noure7MD-ShDp5dmM",
  "tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

Figure 238: Flattened JWE JSON Serialization
7. Security Considerations

This document is designed to provide examples for developers to use in checking their implementations. As such, it does not follow some of the security considerations and recommendations in the core documents (i.e., [JWS], [JWE], [JWK], and [JWA]). For instance:

- it does not always generate a new CEK value for every encrypted example;
- it does not always generate a new Initialization Vector (IV) value for every encrypted example; and
- it does not always generate a new ephemeral key for every ephemeral key example.

For each example, data that is expected to be generated for each signing or encryption operation is isolated to sections titled "Generated Factors".

8. References

8.1. Normative References


8.2. Informative References


Acknowledgements

Most of the examples herein use quotes and character names found in the novel "The Fellowship of the Ring" [LOTR-FELLOWSHIP], written by J. R. R. Tolkien.

Thanks to Richard Barnes, Brian Campbell, Mike Jones, and Jim Schaad for their input and review of the text. Thanks to Brian Campbell for verifying the Compact Serialization examples.

Author’s Address

Matthew Miller
Cisco Systems, Inc.

EMail: mamille2@cisco.com