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 ................................ 61
  5.4.5. Output Results ........................................ 63
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
5.9. Compressed Content ......................................... 84
  5.9.1. Input Factors ......................................... 84
  5.9.2. Generated Factors ..................................... 84
  5.9.3. Encrypting the Key .................................... 85
  5.9.4. Encrypting the Content ................................ 85
  5.9.5. Output Results ....................................... 86
5.10. Including Additional Authenticated Data .................. 88
  5.10.1. Input Factors ....................................... 88
  5.10.2. Generated Factors ................................... 89
  5.10.3. Encrypting the Key .................................. 90
  5.10.4. Encrypting the Content ............................. 90
  5.10.5. Output Results ..................................... 91
5.11. Protecting Specific Header Fields .......................... 93
  5.11.1. Input Factors ....................................... 93
  5.11.2. Generated Factors ................................... 94
  5.11.3. Encrypting the Key .................................. 94
  5.11.4. Encrypting the Content ................................ 94
  5.11.5. Output Results ..................................... 95
5.12. Protecting Content Only .................................... 97
  5.12.1. Input Factors ....................................... 97
  5.12.2. Generated Factors ................................... 98
  5.12.3. Encrypting the Key .................................. 98
  5.12.4. Encrypting the Content ................................ 98
  5.12.5. Output Results ..................................... 99
5.13. Encrypting to Multiple Recipients ....................... 101
  5.13.1. Input Factors ....................................... 101
  5.13.2. Generated Factors ................................... 101
  5.13.3. Encrypting the Key to the First Recipient ........ 102
  5.13.4. Encrypting the Key to the Second Recipient ....... 103
  5.13.5. Encrypting the Key to the Third Recipient ........ 105
  5.13.6. Encrypting the Content ................................ 106
  5.13.7. Output Results ..................................... 108
6. Nesting Signatures and Encryption ........................... 110
  6.1. Signing Input Factors ................................... 110
  6.2. Signing Operation ........................................ 112
  6.3. Signing Output .......................................... 112
  6.4. Encryption Input Factors ................................ 113
  6.5. Encryption Generated Factors ........................... 113
  6.6. Encrypting the Key ...................................... 114
  6.7. Encrypting the Content .................................. 114
  6.8. Encryption Output ....................................... 115
7. Security Considerations ...................................... 119
8. References .................................................. 119
  8.1. Normative References .................................... 119
  8.2. Informative References .................................. 120
Acknowledgements ............................................... 120
Author’s Address ................................................ 120
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": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sY1_oJXu9A5RkTKqjqvyjekWF-7ytDyRXygCF5cj0Kt",
  "y": "AdymlHvOilxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV50hOHiraVySsUdaQkAgDPwrQjmbnX9cw1Gfp-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.

```
{
    "kty": "EC",
    "kid": "bilbo.baggins@hobbiton.example",
    "use": "sig",
    "crv": "P-521",
    "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sY1_oJXu9A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
    "y": "AdymlHv0iLxXkEhavNYmNCvDX4h9htZaCJN34kfmC6pV5ohQHiraVySsUdaQkAgDPrwQrJmbnX9cw1Gf-P-HqHZR1",
    "d": "AAhRON2r9cqXX1hg-RoI6R1tX5p2rUAYdmpHZ0oClN4t56KtsrX6zbKipQrCW9CZH3T4ubpnoTKLDYJ_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.

```json
{
  "kty": "RSA",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "n": "n4EptAOcc9AlkeQHPzHStgAbga7bTZLwUBZdR8_KuKPEHLd4rHVTeT-O-XV2jRojdNhxJWTDvNd7nq00VEiZQHz_AJmSCpMaJMRBSFKrKb2wgVwGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5yQdQLrHqkHHig3RBordaZ6Aj-oBHqFEHYpPe7Tpe-OfVFhd1E6cS6M1FZcD1NNLYD51FhpPI9bTwJIsde3uhGgC02eEHg81hzwOHrTIqbs0FVbb9K3-tVTU4fg_3L_vniUFAKwucLgKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuauxwUkRz5iaiQkqc5gHdrNP5zw",
  "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.
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.

```json
{
  "kty": "oct",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037",
  "use": "sig",
  "alg": "HS256",
  "k": "hJtXIZ2uSN5kbQfbtTWNbpmhkV8FJG-0nbc6mxCcYg"
}
```

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-8nulDvOx8"
}
```

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]:

SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmlvdSBkZWRvdWVwIHlvdXIgZmVldCwgYW5kIGlmlvdSBkZWRvdWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJ1IHN3ZXBBIG9mZiB0by4

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.

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

Figure 9: JWS Protected Header JSON
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).

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

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:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsb2FkIiwieXNlcjoiS2FtbGl0aW9uIiwibG9rZWJuYW1lIjoiIn0.

-The resulting JWS object using the general JWS JSON Serialization:

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzL CBGcm9kbywgZ29pbmcgb3V0IHNsb3d IDi84nBhZ2dpbnNAaG9iYm10b24uZXhhbXBsb2Fk

   MRjdkly7_-oTPTS3AXP4ilOiQGKa80A0zmTuV5MEaHoxxW2e5CZ5N1Ktai oFmKZopdHM1O2U4wzJdBkzq996ivp83xuglI1PNDi84nBhZ2dpbnNAaG9iYm10b24uZXhhbXBsb2Fk

   ZCzwgY5WkIGlhdldld24ndCBZVWVdldWhlcmxi gJlzIG5vIGtub3dpbmcgd2h1cmaUgeW91ICp2Z2F0aG9uZmx0aDQw

   MIJ1Af_fCIE8u9ipH84ogoree7vjbU5y18kdqUdG

   Figure 13: JWS Compact Serialization

   Figure 14: General JWS JSON Serialization

   Miller                        Informational                    [Page 14]
The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBhcm9kbywgZ29pZmcgcb3V0IldXIGZvci4gWW91IHN0ZXAgb25obyB0aGUgcm9mZ2cgY29tbW9uIHRvbW8gdGhlcmxigJ1IG5vIGtub3dpbmcgd2hlcmUgeW91IGlwOjZjIHN3ZXB0IG9mZiB0by4",
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImlzaXRzX3M9RXh0b24uZXhhbXBsZSJ9",
  "signature": "MRjdkly7_-oIPTS3AXP4iIIGKa80AOzmuV5MabHoxnW2e5CZ5NIktainoFmKZopdHM02U4mx7JQx996ipv83xug1II7PND1i8w
nBBDkobwA78185hX-ES4JIwmdLJK3IfWRA-Xt0RnlruYv746iYTh_q
HRD68BNtluSNCruCUTJdt5aAE6x8W1Kkt9eRo4QPocSadmHXFxtnt8Is9UzpeERV0ePPQdLuw3IS_de3xyIrDaLgdjluPxAhb6L2aXic1U12podGU0
KLUQSE_oI-ZnmKJ3F4uOZDnd6Q2WJushZ41Axf_fcIe8u9ipH84ogore
e7vjbU5y18kDquDg"
}
```

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.

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

Figure 16: JWS Protected Header JSON

```
eyJhbGciOiJQUzIzQ4NCIsImtpZCI6ImltZDEiLCJhbGciOiJUZzMiLCJhZ2QiOiJhbmZzcC1hbGwiLCJpZCI6ImFjdGlvbiIsIm9wIjoiY29tcHpkIiwiZXhwIjoiY29tcHpkIiwiYXVkIjoiaHR0cHM6Ly91c2U6MjAwMC8iLCJ0b3Bvd3MiOiIyNDIifQ==
```

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).

```
eyJhbGciOiJQUzIzQ4NCIsImtpZCI6ImltZDEiLCJhbGciOiJUZzMiLCJhZ2QiOiJhbmZzcC1hbGwiLCJpZCI6ImFjdGlvbiIsIm9wIjoiY29tcHpkIiwiZXhwIjoiY29tcHpkIiwiYXVkIjoiaHR0cHM6Ly91c2U6MjAwMC8iLCJ0b3Bvd3MiOiIyNDIifQ==
```

Figure 18: JWS Signing Input

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

```
cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
```

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:

eyJhbGciOiJQUzMiM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGFgsZGFuZ2VybmVyIGZvcmQgYW5kIGluc3Rhbml6ZWQpIiwiaHR0cHM6Ly9wcmludHMub3Jn

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2VybyZvZIGJ1c2luZXNzLCBGcm9kbyygZ29pbmcgb3V0IldXIGZ9vci4gWW91IHN0ZXAgb25obyB0aGUgcm9hZCgwYW5kIGlmIldXlvdSBkb24ndCBzZW50IldXlvdXG1hcmxigJlZi5IGVtcmV2bmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0Ij8",
  "signatures": [
    {
      "protected": "eyJhbGciOiJQUzI6Inp3dmR5Mi4gWW91IHN0ZXAgb25obyB0aGUgcm9hZCgwYW5kIGlmIldXlvdSBkb24ndCBzZW50IldXlvdXG1hcmxigJlZi5IGVtcmV2bmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0Ij8",
      "signature": "cu22eBqkYDKgIltIzdXGvaFfz6WGoz7fUDcft0kko42miAh2qyBzk1xEsnk21pN6-tPd6Vrk1HkqsGdqHCdP608TTB5dDIIt1vN6_10LPcpcvUrhiUSMxbbXUvDvWXzg-UD8biiReQF1fz28zGWsvdINw8ZnyPegVFn442zDqiviVJRmBqyYRxe88_i_j7p8Vdz0TTrxUeT3lmd9s
hnr2lfJ78ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uPlaCN_2_jLAeQT1qRHTafa64QQSUmFAajVKPbByiy7xho0uUT0cbH51o6GYmJUAfmWjwZ6oD4iFKo8DYM-X72Eaw"
    }
  ]
}
```

Figure 21: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2VybyZvZIGJ1c2luZXNzLCBGcm9kbyygZ29pbmcgb3V0IldXIGZ9vci4gWW91IHN0ZXAgb25obyB0aGUgcm9hZCgwYW5kIGlmIldXlvdSBkb24ndCBzZW50IldXlvdXG1hcmxigJlZi5IGVtcmV2bmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0Ij8",
  "protected": "eyJhbGciOiJQUzI6Inp3dmR5Mi4gWW91IHN0ZXAgb25obyB0aGUgcm9hZCgwYW5kIGlmIldXlvdSBkb24ndCBzZW50IldXlvdXG1hcmxigJlZi5IGVtcmV2bmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0Ij8",
  "signature": "cu22eBqkYDKgIltIzdXGvaFfz6WGoz7fUDcft0kko42miAh2qyBzk1xEsnk21pN6-tPd6Vrk1HkqsGdqHCdP608TTB5dDIIt1vN6_10LPcpcvUrhiUSMxbbXUvDvWXzg-UD8biiReQF1fz28zGWsvdINw8ZnyPegVFn442zDqiviVJRmBqyYRxe88_i_j7p8Vdz0TTrxUeT3lmd9s
hnr2lfJ78ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uPlaCN_2_jLAeQT1qRHTafa64QQSUmFAajVKPbByiy7xho0uUT0cbH51o6GYmJUAfmWjwZ6oD4iFKo8DYM-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.

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

  Figure 23: JWS Protected Header JSON

  eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9
  hhbXBSZSJ9

  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).

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2Vybc3VzIGJlci21uZXNzdCBGcm9kb3VgZ29pZmcb3V0IHBvc3VwIHlvdXIgZmVjdW5kcm9sZSB0by4
```

Figure 25: JWS Signing Input

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

```
AE_R_YZCChjn4791jSQCrdP2CNYqHXCTZH0-JZGYNlaAjp2kgaluUIUnC9qvb
```

Figure 26: JWS Signature, base64url-encoded

### 4.3.3. Output Results

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:

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2Vybc3VzIGJlci21uZXNzdCBGcm9kb3VgZ29pZmcb3V0IHBvc3VwIHlvdXIgZmVjdW5kcm9sZSB0by4
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
  Z29pZmcgb3V0IHdvIGZ9vci1q4GW9l1HN0ZXAgb250by80aGUgcm9h
  ZCWgY5kIgImIHdvSBkb24ndCBrZWVvIHlvdXIg2mVldCwgdGhlcmXi
  gjIzIG5viGtbub3dpbmpg2hIcmUeW9l1GlpZ2h0IGJ1IHN3XzB0IG9m
  ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "AE_R_YZCChjn479jVSQrPZCNYqHXCTZH0-JZGYNl
      aAjP2kqalUIUIIUh9cvbuP1on7KRTzoNEuT4Va2cmL1eJAQy3mt
      PBU_u_sDDyYjnAMdxxXPn7XrT0lw-kvAD890j18e2puQens_IEKBp
      HAB1sbEPX6sfY80cGDqoRuB0mu9xQ2"
    }
  ]
}
```

Figure 28: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
  Z29pZmcgb3V0IHdvIGZ9vci1q4GW9l1HN0ZXAgb250by80aGUgcm9h
  ZCWgY5kIgImIHdvSBkb24ndCBrZWVvIHlvdXIg2mVldCwgdGhlcmXi
  gjIzIG5viGtbub3dpbmpg2hIcmUeW9l1GlpZ2h0IGJ1IHN3XzB0IG9m
  ZiB0by4",
  "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
  "signature": "AE_R_YZCChjn479jVSQrPZCNYqHXCTZH0-JZGYNl
  aAjP2kqalUIUIIUh9cvbuP1on7KRTzoNEuT4Va2cmL1eJAQy3mt
  PBU_u_sDDyYjnAMdxxXPn7XrT0lw-kvAD890j18e2puQens_IEKBp
  HAB1sbEPX6sfY80cGDqoRuB0mu9xQ2"
}
```

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

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTrkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9
```

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).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTrkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZ3NzLCBGcm9kbywgZ29pbmcgd2hlcmUgeW91IHN0ZXAgb250byBk
b24ndCBzWVwIHlvdXIgcmxlc3NzIHN0ZXAgb250byBkZXJzaXRlIGJ1c2luZ3NzcyBwcm9kcyBwcm9kcyBw
```

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

```
s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

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:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWViZjMxNGJjNzAzNyJ9
```

```
SXTigJlzIGEgZGFuZ2Vvby5zc3dldGlvbi5iYXNlIGFuZCB2YXJ0aW9ucyB0byB0aGUgZGlh
```

```
s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyby3VvZIGJ1c2luZ2XNzLCBGcm9kbywg
Z29pbmcgbZVOIH1vdXIGZ9vc14gWW91IH02XAgb250byB0aGUgcm9h
ZCwgYW5kIGlmIH1vdSBkb24ndCBZZWV1IHvdXIg2mVldCwg0hlcmtX
iGJlzIG5vIGtub3dpbmcgd2hlcmtUgW91IG1pZ2h0IGJ1H3ZXB0IG9m
ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJUb250byB0aGUgcm9hZ2UgYXJnZXQgZm9ybXkg
RkOWITNDexYiU3M0JzMygNCg
      "signature": "s0h6KThzkfBBKLPspW1h84VsJZFTsPPqMDA7g1Md7p
0"
    }
  ]
}
```

Figure 35: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyby3VvZIGJ1c2luZ2XNzLCBGcm9kbywg
Z29pbmcgbZVOIH1vdXIGZ9vc14gWW91IH02XAgb250byB0aGUgcm9h
ZCwgYW5kIGlmIH1vdSBkb24ndCBZZWV1IHvdXIg2mVldCwg0hlcmtX
iGJlzIG5vIGtub3dpbmcgd2hlcmtUgW91IG1pZ2h0IGJ1H3ZXB0IG9m
ZiB0by4",
  "protected": "eyJhbGciOiJUb250byB0aGUgcm9hZ2UgYXJnZXQgZm9ybXkg
RkOWITNDexYiU3M0JzMygNCg
  "signature": "s0h6KThzkfBBKLPspW1h84VsJZFTsPPqMDA7g1Md7p
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-eef314bc7037"
}
```

Figure 37: JWS Protected Header JSON

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiliZmQ2LW
V1ZjMxNGJjNzAzNyJ9

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).

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiliZmQ2LW
V1ZjMxNGJjNzAzNyJ9
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZ3NzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXigZG9vcic4gWW91IHN0ZXAgb250byB0aGUgcm9hZ2FwYX5kIGlmIHlvdSBk
b24ndCBzZVwIHlvdXIgZmVjdXMgcHllcmXigJlzIG5vIgtub3dpbmcd2hlc
UgeW91IGlpZ2h0IGJlIHJnZXBlbGVkIg

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

s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0

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:

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9

s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0

Figure 41: General JWS JSON Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

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

```
{
    "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUtLTNDcxYi1iZmQ2LWVlZjMxNGJiNDAzNyJ9",
    "signature": "s0h6KThzkfBBBkLspW1h84VsJ2FTsPPqMDA7g1MD7p0"
}
```

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.

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

Figure 44: JWS Protected Header JSON
eyJhbGciOiJIUzI1NiJ9

Figure 45: JWS Protected Header, base64url-encoded

{
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
}

Figure 46: JWS Unprotected 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).

eyJhbGciOiJIUzI1NiJ9.
SXTigJlzIGEgZGFuZ2Vybyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHBvdXIgZG9vci4gWW91IHN0ZXAgb250cyB0aGUgcm9hZ2CuYW5kIGlmIHlvdSBk

Figure 47: JWS Signing Input

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

bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20

Figure 48: JWS Signature, base64url-encoded

4.6.3. Output Results

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:

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2VybyZvIGJ1c2luZXMzcGcm9kbywg
  Z29pbi5mcgu3VHIvdXHgZ9vc14gWW91IHJpZAcybGwvbnQwOAUgcm9h
  ZCgwYW5kIGlmIHlvdSBkb24ndCBrZGV2XHlvdXJgZmVldCwgZGhlcmXigJlzIG5vIGtub3dpbmcmd2h1cmUgeW91IG1pZiB0b24=
  ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "bWUSVaxorn7bEFldjytBd0kHv70Ly5pvbomzMWSOr20"
    }
  ]
}
```

Figure 49: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2VybyZvIGJ1c2luZXMzcGcm9kbywg
  Z29pbi5mcgu3VHIvdXHgZ9vc14gWW91IHJpZAcybGwvbnQwOAUgcm9h
  ZCgwYW5kIGlmIHlvdSBkb24ndCBrZGV2XHlvdXJgZmVldCwgZGhlcmXigJlzIG5vIGtub3dpbmcmd2h1cmUgeW91IG1pZiB0b24=
  ZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiJ9",
  "header": {
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "bWUSVaxorn7bEFldjytBd0kHv70Ly5pvbomzMWSOr20"
}
```

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).

```
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luc2ZvcmVzb3VyY2UgYW5kIHRoaXMgdGhpcyBhbmQgZmlsZS4gWW91IGlvdXIgZG9pbnQgYW5kIHRoaXMgZmlsZS4n
```

Figure 52: JWS Signing Input

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

```
xuLifqLGiblpv9zBpuZczWhNjlgARaLV3UxvxhJxZuk
```

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:

```
{
  "payload": "SXTigJ1zIGEgZGFuZ2VyZviZlGJ1c21uZXNhLCB0cm9kbywg
Z29pbiB0cm90I1lvdXIgZG9vaWRpZWQgYW5kIGlvc3QsIHRoZSB0aGUgcm90
ZU92ZXIgYW5kIGZyZXZzbyB0aGUgcm90ZSB0byBkaXZlciBhLg0=
",
  "signatures": [
    {
      "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eeff314bc7037"
      },
      "signature": "xuiLifqLGi1pv9zBpu2czWhNJl9ARnLV3UxvxhJx2u
k"
    }
  ]
}
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2VybzVzIGJ1c2luZXRzLCBGcm9kb3V0IG1lc3NhZ2UgZm9yZWN0IG1peC5hZ2UgY2F0aW9yIHN0cmluZw",
  "header": {
    "alg": "HS256",
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "xuLifqLGiblpv9zBpuZczWhNjlgARaLVVUxvvhJxZuk"
}
```

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.SXTigJlzIGEgZGFuZ2Vyb3VzIGJlclZvZ2lzLCAICmFzeXBlIGEgZGFuZ2Vyb3VzIHB0aGVyIGJylwIGR1cmxBc3ByaXkgUmVkdCB0byB0aGUgZmVjdCBsaXZlIGJ1c2l6ZSB0byB0aGUgZ2Fuc2VyIHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgZGF0YSB0byB0aGUgZmVjdCB0byB0aGUgZmVjdCBsaXZlIGJ1c2l6ZSB0byB0aGUgZ2Fuc2VyIHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgZGF0YSB0byB0aGUgZmVjdCB0byB0aGUgZmVjdCBsaXZlIGJ1c2l6ZSB0byB0aGUgZ2Fuc2VyIHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgZGF0YSB0byB0aGUgZmVjdCB0byB0aGUgZmVjdCBsaXZlIGJ1c2l6ZSB0byB0a

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

MIsjqtVlOpa71KE-Mss8_Nq2YH4Fghiocsqrqi5NvyG53u0imic1tcMdSg-qptrzZc7CG6Swv2Y13TD1qHzTUrL_1R22FcryNF1hKSw129EghGpwkpxaTn_THJTCglNbADk01MZBCdzwjXwqZc-1Rlp02HibUYyXSwo97Se0_evZKdjyvKSgsIqjytkSeAMbhMBdMma622_BG5t4sdbuCHTfp9ijmkio47AIwqkZV1aIZsv33uPuqBBCXbYoOJwt7mxFp7HmN1GoOSMxR_3thmXTCm4US-xiNOybhm8afKK64jU6_TPtQHIJeQJxz9G3Tx-083B745_AfYOnlC9w

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_Nq2YH4Fghiocsqrqi5NvyG53u0imic1tcMdSg-qptrzZc7CG6Swv2Y13TD1qHzTUrL_1R22FcryNF1hKSw129EghGpwkpxaTn_THJTCglNbADk01MZBCdzwjXwqZc-1Rlp02HibUYyXSwo97Se0_evZKdjyvKSgsIqjytkSeAMbhMBdMma622_BG5t4sdbuCHTfp9ijmkio47AIwqkZV1aIZsv33uPuqBBCXbYoOJwt7mxFp7HmN1GoOSMxR_3thmXTCm4US-xiNOybhm8afKK64jU6_TPtQHIJeQJxz9G3Tx-083B745_AfYOnlC9w"
}
```

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).

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

The following is the assembled second signature serialized as JSON:

```json
{
  "header": {
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "ARcVLnaJJaUG8fG-8t5BREVAuTY8n8YHjhWD01muhcdoFZFFjfISu0Cdkn9Ybdlmi54ho0x924Duz8sK7ZXkhc7AFM8ObLfTvNCrqcI3Jl2U5IX3uNhODH6v7xgy1Qahsn0fyb4zSAkje8bAWZ4vIfj5pCMYxmx4fgV3q7ZYhm5eD"
}
```

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

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LW
V1ZjMxNGJjNzAzNyJ9

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).

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LW
V1ZjMxNGJjNzAzNyJ9

Figure 68: JWS Signing Input

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

s0h6KThzkfBBBkBspW1h84VsJZFTsPPqMDA7g1Md7p0

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

```
{
"protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW1jNzAzNyJ9",
"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": "SXTigJ1zIGEgZGFuZ2Vybc3VjZS1uZXNjLCBpZC52aXNzIG9yZ2FuaWVkIGJ1c2luZw==",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiJ9",
      "header": {
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "MIsjqtVlOpa7lKE-Mss8_Nq2YH4Fhiocsqrgi5VyG55oumic1tcMdSg-qptrzZc7CG6Swv2Y13TiHzUUrL_1RZ2FcrvNyFiHkSwl29EghGpwkpxaTn_THJTc1NbADko1MZBCdwzJxqwZc-1RlpO2HibUYyXSw097BS0_ev2KdjvKSgsIqjytKSeAmbhM6Mma622_BG5t4sdbuCHTfjp9ijJmko47AIwqkZVlaIZsv33puUgBBCXbYoQJw7mxPfHmNiLg0OSxR_3thmXTCm4US-xiNoyhbm8afKx64jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOn1C9w",
    },
    {
      "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "ARcVLnaJJaUW8gfG-8t5BREVAuTY8n8YhwD01muhcYzOZ7FfjfoISu0CdMi9Ybd1mi54ho0ox24DUs2sK7XXkhc7AFM8ObL7TvNCrqlCI3jkl2U5IX3utNhODH6v7xy1Qahsn0fyb4zSAkje8bAZw4vIfj5pCMYxxm4fgV3q7ZYhm5eD",
    },
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGU5LTk0WIIHNDcxYi1iZmQzLWVI1jMxNGJjJNzAzNyJ9",
      "signature": "s0h6KThzkfBB8kLspWlh84VsJZFTspPqMDA7gLmd0z"
    }
  ]
}
```

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.
o "alg" parameter of "RSA1_5".

o "enc" parameter of "A128CBC-HS256".

{
  "kty": "RSA",
  "kid": "frodo.baggins@hobbiton.example",
  "use": "enc",
  "n": "maxhbsmBtdQ3CNRkvprUE6n91YcregDMLYNeTAWclj8NnPU9XIYegTHVHqjyKDSHP21-F5j57sppG1wgdAqZyhNvXhYnvcM7FrgKqnx-zAHx6f3y7s-M9P3nCwFC21h6AqR4I00Ev99rpyM9Pi4lBUop95f5S9W5U
NwhAllhrd-osQgPjIeI1deHTwx-ZTHu3C60Ps_LJI16hKn9wbaUm4AcR5Bd2pgbaY7ASgsjCUBtYJaNIHSoHXprUdJZKUMaZvO0WOPKFa60PI4oy
pBadjvM2AzJ3BnxACYSsE2haueTXVzB4eZOAJiyh2e_VOIKVmsDrJYA
VotGlvmQ",
  "e": "AQAB",
  "d": "Kn9tqohHfiTVl8u5b9TnwyHwG5dK6RE0ufd1PCgJN7E1963R7wy
boQ1PLAHmpIBtztfrheoAniV1NCiQAxWq_hx4G6IxiSTp4nEp9cyKyo
5jMAji7-CL8vhypYoW9FpVescMoVpRMYT9W63hNM0aW5USZ_hLg6
OelmY0vHTI3ucjSM86Nff60IEN43r2fspqEPGRrdE6fpcLc90aq-geP
1GFUlimrRdnmd-P8q8kvN3K11N4TgqRQaoTTg8z0S-3VD0Fgwfnf1BPN
miuPUx08Opi9KDIfu_acc6fg14nsNaJqxe6RESvhGPH2afjHqSy_Fd2v
pzj85bQQ",
  "p": "2DwQmZ43FoTnQ8IkUj3BmKRf5Ek2mizZA5xEJ2minUE3sdYKSLtaE
oeKX9vbBzuWxHdVhM6umKACJ2iINk8ZoayLYHLO_G21aXF9-unynEpUsH
7HHTklplYaOo12zgVjoxAcWnn3h1EFrj2LZG5710H-a3Q1DDQQoJ0J
2VFmU",
  "q": "te8LY4-W7IyaqH1ExujjMqkTA1TeRbVOVLQnFLy2xINnrWdwiQ93_V
F099aP1ESuLja2n-6iKIE-q7mtCPozKVFyTUYfz5HRJ_XY2kfoexJN
91h2HmV5pikz2pezS-GLPC66gRlkOiq1d-qn_qyusfWv7WAX1SVQf8K
6dEt0",
  "dp": "UFyKcL_or492vVc0PzwL5p1bg4L3-25wL48miewspzYIdg2xHTH
QmjzpFA1Zq-zf9RmgJKxKrFs9rkdxtAsL1WydeCT5c125Fkdxg317JV
RD0linX7x2Kdh8ERCw8_4zXItuT1_KiXZNU5lMwJqWbIw2eTxlpsf
lo0rYU",
  "dq": "IeqQqQfnpbdH8Wd7m7UFyrX9on0KXBCogChY6Yku1HGc_p8Le9Mb
pFKEszEaLlN1Ehf3060G151z_auY1j2iOQ28znoUrpa9fVYNt087A
CfzIG7Q9mV7riFpAderZi03tkVXAdaBau_9vs5rS-7HMtxvRvSUyJ14
TkX1HE",
  "qi": "Kc-1zQoAfa2Zcr510tOVCERkQv9AhqIiRGLz-Mz4sCmRkxm5vZ
1XYx66EtEl-Aagjqja1kjieGlxTTThHD8Iga60GBMaAr5uR1hQqPsc7
G17CF12dKBMjqmN6EshYzZfxW08mIO8M8rzh0b0eL6fG9mkDcyPrBxx
2bQ_mM"
}

Figure 73: RSA 2048-Bit Key, in JWK Format
(NOTE: While the key includes the private parameters, only the public parameters "e" and "n" are necessary for the encryption operation.)

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.

3qyTVhIWt5juqZUCpfRgpvauwB956MEJL2Rt-8qXKSo

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--BgLOXMozKxmy9gffy2gTvqzfTihJbuuzxg0V7yk1WC1nQePFvG2K-pvSiWc9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4XaRvYHFj2TZ2G Xfz8YAlmcc91Tfkk0WXC2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcG TSLUeeCt36r1Kt3OSj7EyBQXoZ1N7IXbYhMAfGe7MVv1rOTO1518NQqeXXW8V1 zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnuAUuGUGEce1IO1wx1BpyIfqvfjOh 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.

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

Figure 77: JWE Protected Header JSON

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUiLCJ1bmMiOiJJBMTI4Q0JDLUhTMjU2In0
```

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_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvG24_FNVSiGc_r
aa0HnLQ6s1P2sv3Xz11p11_o5wr_RsSzrS8z-wn13Jvo0mkpEEn1DmZvDn_k80
WzJV7eZVeqiWKdyVZfPpiyQU28GLopRcZvBvB4dQKPdtNTjPPEmRquaGETWzV
yeSUv5kS5yJxZxRuSvWf6KrNtmRdZ8R4mdOjHsrsM_s8uwIFcqt4r5GX8TaIO
zT5CbiL5Qlw3rC7u_hg0yKVOirytEAEs3v2kcflkP6nbXdc_pkMDNS-ohP78T2
O6_7uInMGhFeX4ctHG7Ve1HGt93JfWDEQi5_V9UN1rhXNryu-0fVMkZAAX3VW
i7lzA6Bp430m
```

Figure 79: Ciphertext, base64url-encoded

```
kvKuFBXHe5mQr4lggobA Ug
```

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:

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.laLxI0j-nLH-_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePFvG2K-pvSlWc9BRIazDrn50RcRai__3TDON395H3c62tIouJ4XaRyYHFj2T22Gxfz8YAImc91Tfk0WXC2F5Xbb71Q1DDH151t1pH77f2ff7xi8xh9oSewYrcGSLJueeCt36rlKt30Sj7EyBQKoZlN7IxbyhMAfgIe7MvlrOT0I518NQqeXXW8V1zNmoxaGMny3YnGir5Wf6q72nBq4qDaPdnaAuUUGEeeelIO1wx1BpyIfgvfj0hMBS9M8XL223Fg47xlGswXXdfuY-4jaqVw
.lbbd5sTkYwhAiqfHsx8DayA.0fys_TY_na7f8dwSfXLiyDHaA2DxUjD67ieF7fcVbIR62JhvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xz1lpl1_05wr_RsSzrS8Z-wnI3Jvo0mkpEEn1DmZvDu_k8O
.0WzJv7eZVEq1WkDyVzHfPpiyQU28GLOpRc2VvBbK4dQKnPNTjPPEmRqcaGeTWZVyeSUvf5k59yJZxrSuSVF6kKrNtmRd28R4mdOjHSrM_s8uwIFcqt4r5G8TKaI0zT5cBlQ1ls3rc7u_hg0yKVOIrYtEAes3vzkcfLkP6nbXdc_PkMdNS-ohP78T2O6_7uInMGHeFex4tCH7ve1HG1T793JfWDEQI5_V9UN1rhXNryu-0fVMKzAKX3VW
i7izA6BP430m.kvKuFBXHe5mQr4lqgobAUg```

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

```json
{
  "recipients": [
  {
    "encrypted_key": "laLxI0j-nLH_-BgLOXMoZKxmy9gffy2gTdvgzfTihJhzuBxg0V7yk1WCnQePFvG2K-pvSlWc9BR1azDrn50RcRai__3TDON395H3c6t1ouJJ4XaRvYHFjzTZ22GXfz8YA1mcc91Tfk0WXC2F5Xbb71C1Q1DDH15t1pH77f2ff7xiSxh9oSevwYrcGTSUueeCt36r1Kt3OSj7EyBQXo21N7lbxbyhMAfgIe7Mv1rOTO1518NqexW8Vlzn5moxaGMny3ynGir5wf6Qt2nBq4qDaPdnaAuGUGEceelI01wx1BpyIfgvfohMBs9M8XL223Fg47x1GsMXdfuY-4jaqvVw"
  }
  ],
  "protected": "eyJhbGciOiJSU0ExUzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdg9uLmV4YW1wbGUiLCJlbnRmiOiJvZmMiLCJ1c2VySm9ic1wIW3UyIiwiZXhwIjoiOTUxNzE3ODI1OTFiZjY2OGQ4ZDAxM2QyN2Q4Y2Q1ZjU5NzIzZTBjZjQwZmNlZjI1ZDk2NWU4YmYiLCJ0eXBlIjoiSGFzeW5jZSA6d3BzfCJ9",
  "iv": "bbd5STkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7fBdwSfXLIydHaA2DxUjD67ieF7fcVbIR62JhJvGz4_FNVSiGC_raa0HnLQ6s1P2sv3Xz11p11_05wr_RsSrzS8Z-wnI3Jvo0mkpEEnlDmZvDu_k80WzJv7e2VEqiWKdyVzPhPpiyQU28LQopRc2VbVbK4dQKpNtfPPEmRqcaGetWZVyeSUvf5k59yJZxRuSvWFWF6KrNmRDZ8R4mD0jHSrM_s8uw1Fcqt4r5GX8TKaI0zT5Cbl5Qw3sRc7u_hgoyKVOiRytEAEs3v2kcfLkF6nbXdxC_PkmDNS-ohP78T206_7uInMGrFeX4ctH7G7e1HGlt93JfWDEQ15_V9UN1rhXnrYu-0fVMkZAKX3VWl71zA6BP430m",
  "tag": "kvKuFBXHe5mQr41qgobAug"
}
```

Figure 82: General JWE JSON Serialization
The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YW1wbGU1LJ1bmMiOiJJBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "laLxI0j-nLH-_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePFvG2K-pvS1Wc9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4xArvYHFj2TZ2GxFz8YAIWcc91Tfk0WXC2F5xb71ClQ1DDH151t1pH77f2ff7xiSxh9oSewYrcGTSLueeCt36r1Kt3OSj7EyBQx02N7IxbxhMAfgIe7MvlrOTOIS18NQeXXW8VlzNmoxaGMny3YnGir5Wf6Qt2nBq4QDaPdaAuugUGEEecllO1wx1BpyIfgvfj0hMBs9M8XL223Fg47x1GsmMXdfyV-4jaqVw",
  "iv": "bbdS5kTkywhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSFXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xz1lpl_05wR_RsSzr58Z-wnI3Jvo0mkpEE1Dm2vD_u_k80WzJv7eZVEqiWkdyVzPhPyiqUYQ28GLOpRc2VbVbK4dQKpdNTjpPEmRqcaGm7VzYeSUv5f59yJZxRsvWFF6KrNtmRdZ8R4mDOjHsrM_s8uwFctqt4r5G8XTKaI0zT5Cbl51w3sRc7u hg0yKVOiBytEAEesv2kcfLkP6nhXcd_PkmDNS-ohP78T206_7_uInMgHeX4ctHG7Ve1HG1T93JfWDEqI5_V9UN1rhXNry0-fVrMkZAKX3Vwi7lza6BP430m",
  "tag": "kvKuFBXHe5mQr41qgobA Ug"
}
```

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 uses 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",
  "use": "enc",
  "alg": "RSA-OAEP",
  "id": "samwise.gamgee@hobbiton.example",
  "k": "wbdxI55Vaan2XPY29Lg5hdmv2Xhvgoxkunznfzf2-5zVUxa6prHRrI4pPlAhqjRLZfWyndSmnHHRGp4AHl0h0ySJOw10Bi0ZB11XP2e-c-FyXJGcTy08hdkQW1rflhfTm42EW7Vv0Y4rgfa6uxylGwfpGr2Laroi1WCpnkNrg7182CuN3SQBiPGjXkmIy2t1_VWgGnL22Gpy1xjy51BldXpX3eXestspq51utNfoUTU8E4qdj3U1DiToVpG5swMLmnnnJiaw7sXRTtCivR4M5q52tdw-7v4wUR4779ubDuJ5analMV2S66-RPcnFazWSKxtBDnFJJDGIue7Tzizj1nmsOxq_q_yPb_U01WhO0ec85Fcft1hACpWG88chrOBenNYBDFkSyPyUc2L5CJAZAaTf32F2dA67dg1ITsc_FupfQ2kNgcE1LgprxXKhcVNYq86B-HorzhHZqtauzBFN5tvBuT-BpkcvJfNcFL1H3b8mb-H_ox35FjgbSAJKyoeqFKtpVxhd09tknwgf6Vfkg6UC18_TO1jMVfFTWXUxnlhhOOnzW6HSSzD1c9WrCuVzsvUMv54szlQw1f1yWfjg5qFDxQKls99gdcDaICAWM3yEBizuNee5a5dartHDx1xEB_HCHeSyeYbhbMjGfsavKn0aRzNSTyCuwhB1so12Xe_",
  "e": "AQAB",
  "n": "n7fzcJc3_WG59VEOBTKayzuSMM780OJQuzJ_nKb8H10ZG252oA7T4BxccOzQn5OH5uSChw91OcT0JvxpCpmqzaJ2glnirjcWZ-oBtV07gCAWq
  -B3ghf3iz1lk0srsrJHajicyC3H8hbsy4_WerrXg4MDNE4HYoijy68TcX
  2LYRQxUC5f5tJvxM86o1ex15G7VnQnDRtxUECwiwefmnrEveEogLx9E
  A-KmgAgtJiTSXqIXQWNUQGX17v_mV_Hr2YuImYcNhKxKv9P07868dk
  DkHo8v4U02LwA1O1UX98mkoqcw58A_Y2lBybVx1__SjGpSebhb-nqj
  h1lfQgOndfNhLnxclWTw7pcCztLnIMzAyeCWA7G71Fv-Rn9f1lIv9j26r7r-
  MSH9gsozuiHN2grGjD_jfRluuMA0184ffKl6bcqN1jXwPVh2zn0z01yD
  F-1LiqnqUYsepF6X3a2S0dkqBriguE6EvLuSYIDpJq3jDIsoLO1l
  oomgiJxUWt_GWEOGu28gp1yzm-9Q00nyhF1uhSR8aJAQWAIFImWH5W
  _1QT917-yrindr_2fWQo_iUgMsGzA7aOQzyZfPlyQy6z-t_yKuBG00-2B
  S_aWvjrjUc-ALp8AuYkjBZ-7CWH32fGWR48j1t-zomwrjn_mhnsPbGs0c
  9WsWqRzI-K8gE",
  "p": "7_2v30QzZ1LPcHyFylLABQ3XP85Es4hCdwcKkDeltaUXgV919etKghvM4hRko4bb0lkYVuLFmxIkCdpitz LYCAAdXKrAK3PbtSzid_XZ9nlsya_Q2wXb_yiTrFJYfdKUdmz94phuhNGFij7nr6NWXpihSHNWFEizD_AC3m
  Y46J961Y2LrnerVwAGNw53p07Db8yd_92pdA97vq2OZgtbYh9q6uma-
```
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.

5.2.2.3. Initialization Vector; this example uses the Initialization Vector from Figure 86.

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

rT99rwrBtBtIT1JM8fU3Eli7226HEB7IChCnxNuh71Ciud48LxeolRdtFF4nzQ1beY015s_PJsaAXZwSxTDePz9hrk-BBtsTBqc2Up0dwjc9NhNupNNu9uHIVftDyu
cvI6hvALeZ6OGhNv4V1xz2k701D9zmAzfw_kT3tkuorpDU-CpBENfIHX1Q58
-Aaad3Fzmu03F9n9eUP2yXakLXXa15BXQsumP4A1GD4-H4Bd7V3u9h8Gkg8Bpx
KdUv9ScfJQtYm6eJE8b3aSwIAK4T3-dwWpuBOhRQXBosJzS1asnuHtVMt2pK
IIffu5BC6huIVm7Y7kzv7W7aUrpYm_3H4zYyyMeq5pGqFmWk28zp0878TRLZ7
pZfPYDSXzyS0CfKkMozT_qicwZTSz4duYnt8hs429gthXn9uDq6wycMagnaQfOts_lycTWmY-agQWDKhhjYNRf03NiWt5Be-tODfWcasQj3uuAgPGOR02AWBe3
8uj061vXn1SppyyV23WFC7W0JvYaT7a8DRn6MC6T-xDMuxCG7S2rsuw5qQU06HVzT1F070UvfuKBA03caX_n1BIhLMjY2kOTxQmpDPT6Cbo8aKaOnx6ASe5
Jx9paBpnNmo0K35j_qlQhDWN6A2Gg8iFayJ69xDeEdHAVCRGzN3woE1ozDR

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.

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

Figure 88: JWE Protected Header JSON

eyJhbGciOiJSU0EiTiO0FUCiIImtpZC16InNhbxdp2UuZ2F2Zv1QGhvYmJpdG
9uLmV4YW1wbGUlCJlbnMiOiJBMjU2R0NNIn0

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_t2m1GXklSggBdpACm6UJuJowOHC5ytjqyR L-I-sop1wqMF4UgRWWeaOGNw6vGW-xyM011TYxrXfVzIIaRdhYtEMRBvBWBwEw P7ua1DRfvaOjg2v6Ifa3brcAM64d8p51hhNcizPersuhw5f-pGYzseva-TUtL8 iWmctc-sSwy7SqmRkfhdJwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML 7Cc2Gxgv7YqWo0YIEc7aCf1LG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV maPpOs1Y2n525DxDFWaVFUfKQxMF56vn4B9QmpWAbnypNimbM8zV0w
```

Figure 90: Ciphertext, base64url-encoded

```
UCGiqJxhBI3IFVdPalHHvA
```

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:

eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhXdpcc2UuZ2VtZ2V1QGhvYmJpdG9uLvYiVYIwGuICJlbmMiOiJBMjU2R0NNIn0.

Figure 92: JWE Compact Serialization
The resulting JWE object using the general JWE JSON Serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "rT99rwrBTbTI7IJM8fU3Eli7226HEB7IcCxCxNu
    h71Ciud48Lxeol1rDftFF4nznQ1beY015S_PJsAXzWtDzPz9hk-Bb
    tsTBqC2usPOdwjC9NhNupNN9hIIVfDyucvI6hvALEZ0GnhNV4
    v1zx2k7Q1D89mAzfw-_kT3tkuorpDU-CpBENfIHX1Q58-Aad3Fzm
    u03Fn9buEP2yXakLXYa15BUXQspUM4A1GD4_4H4bd7V3u9h8Gk8B
    pxKrUV9ScfJQTcYm6eJEBz3aSwIAK4T3-dwWpuB0hRQXBo5sz51
    asnuHtvM2tpIIJfux5BC6huIvmY7kzV7W7aUIrpmYm_3H4zYvyMeq
    5pGqFmW2k8zpO878TR1x7z72fP6DZSXZySC6fKkMozT_qiCwZTSz
    4duYnt8hsS4Z9sGthXn9uDqd6wycMaBnQfQTS_ylcTWmY-aqWVDKh
    jYNrf03niwRtb5BE-tCdfMlCASKQj3uuAqGro2AWBe38Uq0b1VxX
    1 SpyvY3WFe7WOJYhYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw51qQU
    06MvZlIot0UvfKBA03cxA_nIBIlMjY2bOxQnMn0DtPTr6Cbo8a
    KaOnx6ASE5Jx9paBpnNnOOKH35j_Qu1QhDWN6A2Gg8ifAeyJ69x
    DeHAVCGRzn3woE1zozDRs"
        }
    ],
    "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpd2UuZ2
    FtZ2VlQGhvYmJdpG9uLmV4YW1wbGUICJlbmMiOiJBMjU2R0NNIn0",
    "iv": "-nBoKL0YkLZPSI9",
    "ciphertext": "04k2cnGNrS5W3Do1yuyS1y5Qe965_t2m1GXklsQgBdpAcM6
    UJuJcwOHC5ytjQgRl-I-soFwqMUF4UgRWWeaG9Nw6vGW-xyM011TYx
    rXFvZIaRdh1tEMRbWbWbEw7ualDrfvaojzVv6Ifa3brcAM64d8p1h
    hNcizPsruhf5f-gyZsevTUAl8Iwnctc-SWw7SpmRkfhDw9x0fz
    6kFovEqj64XIIs7E6GpSfnbYGLa1QIuML7Cc2GxgvT7zqWo0YIEc7a
    Cf1L1G-8BboVnF6dZKlK9vNoycrrYHumwzKluWEbSVmaPpOslY2n525Dx
    DfWafVFUFQXzMF5v4nB9QMwPAbnyP9imbM8zV0w",
    "tag": "UCGiqJxhB1I3IFVdPa1HHVa"
}
```

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

```
{
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpd2U2U2FTZ2V1QGhvYmJpdG9uLmV4YW1wbGU1LCJlbmMiOiJjblBvZ242b2Ur20NNIn0",
  "encrypted_key": "rT99rBTbT171JM8FU3Ei17226HEB71cChCwxNuh71C
  iud48Lxoe1RdtFFnznq1beY0y15S_PjsAXzwXtDeFz9hk-BBtsTBqC2U
  aPoDwvj9NhNn9u9u9u9u9u9u1FvFlyucvI6hvALeZ6OghNV4v1xx2k70lD89
  mAzfw_kT3tkuorpDUCpBEFmF1Hx1Q58-Aaad3FzMuoc3F9buEP2yXakL
  XYa15BUXQsupM4A1GD4_h4Bd7V3u9h8Gkg8BpxKdUV9ScfJQcTym6eJE
  Bz3aSwIAK4T3-dwWpuBohROQXBosJzS1asnuHtVmt2pKIIfux5BC6huI
  vymY7kzV7w1IUrpy0m_3H4zYyyMeq5pGqFmW2k8zp0878TR1z7pZfFPYD
  SXZyS0cfKkKmozT_q1CwZTSz4duyNt8hS4z9SgthXn9uQdq6wycMmagnQ
  f0Ts_lycTWMy-aqWVDKhjYNRf03NiWrtb5BE-t0dfwCASQj3uuAgPGr0
  2AWBe38UjQb01vXn15pyV3Y3Wfc7WOJYaTa7A8DRn6MC67-xdMmugC0G
  S7zrscw51QQU06MvZT1FoTOu0vfuKBAo3ctxA_nIBIlMjY2kOTxOMmpDP
  Tr6Cbo8aKaOxixASE5Ex9paBnNmmO5HK35j_Q1rQhDWUN6A2G8iFayFJ
  69xEdEHACVGrZn3woEI2ozDRs",
  "iv": "-nBoKLH0YKlZFSI9",
  "ciphertext": "o4k2cnGN8rSSw31D01YuySkqs_t2m1GXk1SgQbAcM6U
  UJuJowOH5tjtqgYqRL-I-soPlwqMf4UeGrWneaO5nw6vGW-xyM01TlYx
  rXVfZxIArdhYtEMRBrWbBwEx7Ua1DRfvaOjg2v6Ia3brcAM64d8p51h
  hNcizPersuw5f-pgYzevea-TUaL8iWnctc-sSwy7QmRkfhDjwbrz0fz
  6kFovEgj64XI15sTE6l5p5fnbYGLa1QUiML7C2Gxgv7zqWo0YIEc7a
  CF1kGl-8Bb0wFEdZKlK9vNoyczRYHumwKlLuWEnSFvMPoO1y2n525Dx
  DfWaVFUFQxMF56vn4B9QMnPAbnypNimbM8zV0w",
  "tag": "UCGlqJxhBI31FvDPaHHVA"
}
```

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-AES-KW" (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".

```json
{
  "keys": [
    {
      "kty": "oct",
      "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
      "use": "enc",
      "alg": "A128GCM",
      "k": "XctOhJAkA-pD9Lh7ZgW_2A"
    },
    {
      "kty": "oct",
      "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
      "use": "enc",
      "alg": "A128KW",
      "k": "GZy6sIz6w19NJ0KB-jnmVQ"
    },
    {
      "kty": "oct",
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
      "use": "enc",
      "alg": "A256GCMKW",
      "k": "qC57l_uXCm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
    }
  ]
}
```

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:

```
d3qNhUWfqheyPp4H8sjOWsDYaioej4c5Je6r1UtFPWdgtURtmeDV1g
```

Figure 96: Password

Figure 97: Content Encryption Key, base64url-encoded

Figure 98: Initialization Vector, base64url-encoded

Figure 99: Salt Input, base64url-encoded

Figure 100: Encrypted Key, base64url-encoded
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.

```json
{
  "alg": "PBES2-HS512+A256KW",
  "p2s": "8Q1SzinasR3xchYz62ZcHA",
  "p2c": 8192,
  "cty": "jwk-set+json",
  "enc": "A128CBC-HS256"
}
```

Figure 101: JWE Protected Header JSON

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.
Figure 103: Ciphertext, base64url-encoded

0HlwodAhOCILG5SQ2LQ9dg

Figure 104: Authentication Tag, base64url-encoded

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:

eyJhbGciOiJQQkVTMi1IUzUzMitBMjU2SlciLCJwMnMiOiI4UTFTemluYXSM3hjaFl6N1paY0hBIiwicDJjIjo4MTkyLCJjdHkiOiIjfd2sc2V0K2pzb24iLCJ1bmMiOiIJBMTlQ0JDLUhTMjU2In0.
d3qNhUWfqheyPp4H8sjOWsDYajej4c5Je6rlUuFPWdgtURTmeDV1g.
VBICzVHNoLIR3F4V82uoTQ.
231-TblAV4n0WKVSSgCQrdg6GRqsUKxjruHTYxsTHAJLZ2nsnGIX86vMXqIi6IRsfywCFRzLxEcZBRnTvG3nhzPk0GDD7FMxUHPdjEYCNA_X0mzg8yZR9oyjo61TF6s14q9FZ2EhzgFCQL6h5Ev3vR75_hkBsnuoqoM3dwejXBTiOdN84PeqMb6asmas_dpSsz7H10fc5ni9xIz424givB1Yl1df6eexVml93R3foOJbmk2GBQZL SEG11v2cQsBgeprArSa7Q9f9t80coH8tBjgv08AtzXFFsx9qKvC92CLKdPQMT1VJKqtV4ru5LEVpBZXBnZrtVISOgyg6AiupaSrCrCDEPOGSuxvgtrokAKYPqmUXeRdjFJwafkYEkiuDC9vWGAi1DH2xtTafhJwcmwyIyzi4BqRpmsn_N-z15tuJYYuvKjv6ihbsV_k1hJGPAGxJ6wUpmwC4PTQ2izEm0TuSE8eMKdTw8V3kobXZ77u1Mwds4p.
0HlwodAhOCILG5Q2LQ9dg

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

```json
{
    "recipients": [
        {
            "encrypted_key": "d3qNhUWFqheyPp4H8sjOWsDYajoej4c5Je6r1UtFPWdgtURtmeDVlg"
        }
    ],
    "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFtemuYXNSM3hjaF6NlpaY0hBiwicDJjm4MTkyLCJjdHkiOiJqd2stc2V0K2zbo24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
    "iv": "VBiCzVHNQLiRV88uoTQ",
    "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjuh8YsTHAJLZ2nsnGIaX86vMXqi61RsfsywCRFzLxEcZBRnTv3nhzPkoGDD7FMyXHuHpDjYECA_NX0szg8yZR9oyjo61TF6si4g9FZ2EhzgFQCL0_6h5EVq3vR75_hkBsnuoq0M3wejXbIt odN84PeqMb6asm_s_dpsz7H10fC5ni9xIz424givB1YLdF6exVml93R3fO0Jbmk2GBQZLE_SGLlNv2cQsBgeprARsaQ7Bq9rT80coH8ItBjvgV08AatuXFFsx9KvC982KLDpFQMT1VJKkqtV4Ru5LEvpBZBnZrtViOqyg6AiuaSw-rCurcD_ePOGSuxvgtroAKYPqmuXueRdJFwafkYEkivDCV9wWGAiLDH2xTafhJwcmwyiyzi4Bqrpmdn_N-zt5tu6YyvVhjKv6ihbsV_k1hJGPGAxJ6wUmpwC4PTQ2izEm0TuSEBoMKdTw8V3kobXZ77ulMwDs4p",
    "tag": "0HlwodAhOCILG5SQ2LQ9dg"
}
```

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

```json
{
  "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU21ciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6NlpaYOhBiIwicDJjIjo4MTkyLCJjdHkiOiJq2stc2V0Kzpb24iLCJlbmMiOi1iJBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "d3qNhUWFqheyPp4H8sjOwsDYajej4c5Je6riUTFPDgtURtmeDVlg",
  "iv": "VBiCzVHNoLiR3F4V82uoTQ",
  "ciphertext": "Z3i-TbiAV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsgIX8v6MXqIi6IRsfywCRFz1xEcZBRnTvG3hzhzPkJG0GDD7FMyXhUHpDjvCyna-Xomzg8yZ9yjo61TF6si4q9F22EhzgFQCL0_6h5EVg3vR75_hkBnuuoqm3dwejXbtIodN84PeqMb6asmas_dpSsz7H10fc5ni9iZ424ivB1Yldf6exVmL93R3f0oOJbmK2GBQZL-SEG1lv2cQsBgeprARSaQ7Bq9t9T80coH8ItBjgV08AztXFFsx9qKvC982KLdPQMT1VJKkqtV4Ru5LEVpBZXbnZrtViSQyg6Aiuiwa-r-rCrcd_ePOGSuxvgtrokAKYPqmuXeRdjFjwafkyEKsiDCV9yWGAi1DH2XTaIhJwcmwlyzi4BqRmpdn_N-zi1tujYyuvKhjKv6ihbsV_k1hJGPGAXJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V3kobX77ulMwDs4p",
  "tag": "0HlwodAhOCILG55Q2LQ9dg"
}
```

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".


dict{
    "kty": "EC",
    "kid": "peregrin.took@tuckborough.example",
    "use": "enc",
    "crv": "P-384",
    "x": "YU4rRUzdmVqmRtWOs2OpDE_T5fsNIodcG8G5FWPrTPMyxpzsSOGaQLpe2FpxBmu2",
    "y": "A8-yxCHxfkFzb3hK2f1ljUYMyjUhsEveZ9ThuwFjH2sCNdtk5RU7D5-SkgfL1ETP",
    "d": "iTx2pk7wW-GqJKHCekFQb2EFyYc07RugmaW3mRrQA0UIpommT0IdnYK2xD1Zy-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:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 109.
- Initialization Vector; this example uses the Initialization Vector from Figure 110.

Nou2ueKlP70ZXDbq9UrRwg

Figure 109: Content Encryption Key, base64url-encoded

mH-G2zVqgzUttnW_

Figure 110: Initialization Vector, base64url-encoded

5.4.3. Encrypting the Key

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

- Ephemeral EC private key on the same curve as the EC public key; this example uses the private key from Figure 111.
```json
{
  "kty": "EC",
  "crv": "P-384",
  "x": "uBo4kBHp6kbjx510xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iuEDsQ6wNddNg3",
  "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEw2OyQTA-JdaY8tb7E0",
  "d": "D5H4Y_5PSKZvhfVFbcCYJ0tcGZygRgfZkpsBr591cmmhe9sW6nkZ8WfwhinUfWJg"
}
```

Figure 111: Ephemeral Elliptic Curve P-384 Key, in JWK Format

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": "uBo4kBHp6kbjx510xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iueDsQ6wNddNg3",
    "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEw2OyQTA-JdaY8tb7E0"
  },
  "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.

Figure 114: JWE Protected Header, base64url-encoded

Figure 115: Ciphertext, base64url-encoded

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:

```
eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtp2CI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guXhhbXBsZSIsmWwayI6eyJrdHkiOiJFQyIsImNydiI6I1At
Mzg0IiwieCI6InVCbzRsSFBJNmtiang1bDB4b3dy2F9vWxpCbWF6LdLRlp1NH
hBRkZrYlplpV2ldEVLMnl1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcDVTR2haV
MyZmFYdW1jLiU5SUytW84S3BvWXJGRHI1eVBOVnRXXdTa73lRVEEtSmRhWT
h0YjdfMCJ9LbmMioiJBMTI4R0NNIn0
.0DJjBXri_kBeC46IkU5_Jk9BqaQeHdv2.
mH-G2zVggztUtnW_
.tkZu099h50gHjmkkrfLBisku8rGf6nzVxhRM3sVOhxRgz5NJ76oID71pnAi_cP
WJRCjSpAaUZ5dOR3Spy7QuEkmx8-3RCmHYMzSXaEwDdXta9Mn5B7cCBOJKBO
IgEnj_qfohII-euEkUpO8aALTZGHfpi05JmwbKkTe2yK3mjF6SBAsgicQDVckc
Y9BLluzxlRmc3ORxaMOJaHP93YcdSDGgqBWMrNU1ErkjcMqMr7 wtCex3w0
3XdlkjXluEr2hWgP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyc6WcqAu
07WNhjzJEPc4jVntRJ6k3NpPq5p991324080Uqj4icYeze6vTP1Q
.WuGzmc+reYjpHGola17EBg
```

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

```json
{
  "recipients": [
    {
      "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
    }
  ],
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2AdHVja2Jvcm91Z2guZWhhXBS2ZISsImVwayI6eyJrJrdHkiOlJFQyIsImNydiI6I1AtMzg0IiwieCI6InVCbzzRrSFBB3Nmtianq1bD
    "B4b3dyZF9vWXpCbfW6LUdLRip1NHhBRkZrY11pV2dlDVLNm11RURzUTZ3mROZzMiLCJ5Ijoic3AzcDVTR2haVkJMy2mFYdW1JLWU5S1UYTW84S3
    BvWXJGRHIIeVBOVnRXNFbRXd31tRVEEtSmRhWTh0YjdFMCoJ9LClJbm
    MiOlJBMI4R0NNIn0",
  "iv": "mH-G2zVqgzUtW_",
  "ciphertext": "tkZuO09h950qHJmkkrlBisku8rGf6nVxhRM3sV0hXg5NJ76oID7lpnAI_cPWJRCjSpaAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMz
    mXaEwDdxtra9Mn5B7cB0JKB01gEnj_qfo1hIi-uEkUpOZ8aLTZGfp105
    jMwbKkTe2yK3mjF6SBAsgicDVCvckY9BLuux1RmC3ORXaM0JahPB93Y
    cdSDgpgBWMVrNU1ErkjcMgMoT_wtCex3wO3XdxkxJxiErz2hWgeP-nkU
    ZTPU9EoGSpj6fAS-bsz87RCPrxZdj_iVyC6QWcqAu07WNhjzJEPc4yVntR6K3NgPQs91324080Uqj4icYezbs6vTP1Q",
  "tag": "WuGxmc4cYjphGJoal7EBg"
}
```

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

```
{
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncllnRvbtAdHVja2Jvcmm912gu2XhhBXBsZSI1mVwayI6eyYrdHkiOiJFQyIsImNydiI6I1AtMzg0IiwieCI6I1hKbGtJbHlHbDByb3dyZm9vWkxCbWFLUDlRIp1NHhBKzYrYl1pV2d1EVLm1lRURzUZT3TmROZzMiLC5iIjoic3AzcvDVT2haVMy2mYdW1JLU5S1uyTW84S3BvWXJRHI1eVBOVnRXNFbRXdaT31RVFtStSmRhWTh0YjdFMCJ9LCJ1bmMiOiJ0MTI4R0NNIn0",
  "encrypted_key": "0DJjBXri_kBcC46IkJU5_Jk9BqaQeHdv2",
  "iv": "mH-G2zVqgztUtntn_",
  "ciphertext": "tkZuOO9h95OGHJmkkrfLBisku8rGf6nzXvnhRM3sV0hXg5NJ76oID71npAI_CPWJRCjSpqAaUZ5dOR3Spy7QuEkmx8-3RCMhSYMz5XaWdDxta9Mn5bCboJKB01gEnj_qfo1hIi_uEkUpOZ8aLTZHgpf05jMwbKkTe2yKSmjF6SBAsgicQDVcKcyYBLuzz1RmC3ORXaM0JaHPB93ycdsDgppgBWMrNU1ErkJcqMoT_wtcex3w03XdlkXrXr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrrZddj_iYvC6QWcqAu07WnhjzJEPc4jVntrJ6K3lNpPQ5p9913Z408OUqj4icYezeS6yVT1Q",
  "tag": "WuGzxmcreYjspHGoa17EBg"
}
```

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".

```
{
    "kty": "EC",
    "kid": "meriadoc.brandybuck@buckland.example",
    "use": "enc",
    "crv": "P-256",
    "x": "Ze21oSV3wrrcKUN_4zhwGhCgo3Xhultd4QjeQ5wIVR0",
    "y": "HlLtdXARY_f55A3fnzQbPcm6hgr34Mp8p-nuzQCE0Zw",
    "d": "r_kHyZ-a06rmxM3yESK84rlotSg-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_bAWGH1hg0TpjjqVsP1rXWQu_vwV0HHTNkdYoA",
  "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs",
  "d": "AtH35vJsQ9SGjYfOsjUxYXQKrPH3FjZHmEtSKoSN8cM"
}
```

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_bAWGH1hg0TpjjqVsP1rXWQu_vwV0HHTNkdYoA",
    "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.

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)

Figure 125: JWE Protected Header, base64url-encoded

Figure 126: Ciphertext, base64url-encoded

Figure 127: Authentication Tag, base64url-encoded
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:

```
ejJhbGciOiJFQ0RILUVTTiwia2lkIjoibWVyaWFkb2MuYnJhbmrR5YnVja0BidWNrbGFuZC5leGFtGtcGxlIiwiZXBrIjp7Imt0eSI6IiIiLCJ4IjoiVFVUc0lTbWFhXkZJcmFrIiwiaWlkIjoiaGcwVHBqanFwciIiLCJve3VybCI6Imh0dHBzOi8vd3d3Lmd1c2Vybi5uZXQifQ
```

```
yc9N8v5sYyv3iGQT926IUg
```

Figure 128: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
    "protected": "eyJhbGciOiJFQ0RILUVTTiwia2lkIjoibWVyaWFkb2MuYnJhbmrR5YnVja0BidWNrbGFuZC5leGFtGtcGxlIiwiZXBrIjp7Imt0eSI6IiIiLCJ4IjoiVFVUc0lTbWFhXkZJcmFrIiwiaWlkIjoiaGcwVHBqanFwciIiLCJve3VybCI6Imh0dHBzOi8vd3d3Lmd1c2Vybi5uZXQifQ",
    "iv": "yc9N8v5sYyv3iGQT926IUg",
    "ciphertext": "BoDlwPnTypYq-iwjmQvAYJLb5Q6l-F3LiqQomlz87yW4OPKbWE1zSTEFjDfhU9IPIOSA9bm4m7iDFwa-1ZXvHteLdtw4R1XRGMEsDItaytskTTzmzNa-q4F_evAPum1o-2ZG45Mnq4uhM1fm_D9rbWolqZSF3xGNNkpoMQKFC1818wzjRli7-IXgyirLQsbhhqRzkw81cYs6aH124j03C-AR2le1r7URuHArM79BY8soZU01lzW1-sD5PZ314NDCceiXKkoIAfsxJWmySPOeRb2Nial5UL4mYpvKDIwmyzGd65KqVw7MsFfiK767G9C9Az7p5ZK0DyUn1mn0WWS5LmyX_yJ-3AROq8p1W2BfG-ZyJ6195_JGG2m9Csg",
    "tag": "WCcKNa-x4BeB9hIDIFuWng"
}
```

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".

```json
{
  "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

eyJhbGciOiJkakZjIaWQ0i3N2M3ZTJiOOC0Z2zLTQ1Y2Yt0DY3Mi02M

di5W1ONTI0M2EiLCJ1bnMiOiJBTMT4RONNIn0

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.

```
W_i_f52hww_ELQPgaYyeAB6HYGcR58591TYosVoc23XJoBw29rHP8yZOZ7Y
hLpTbFpvzPjQS-m0IFvCkEZdH_lq_FrdYt9HRUYksrtrMmIAeyGmUnd9zM
DB2n0CRIHAzFVeJUdUwVAV7_YGRPdcqMylBoCO-F8dE-Nce4h3-FtBP-c_
BIwCPTjb9o0SbdcR7EMJMyZH8yWMVl1gPD9yxi-aQpGbSw_F9N4IZAxscj5
g-NJsUPbjk29-s7LJAgb15wEbtXphVCggyy53CoIKLHeJHXex45Ur9aKZRSIn
ZI-wjsY0yu3cT4_aQ3ilo-tIE-F8Ios61EKgyIQ4CWao8PBMj8TTnp
```

Figure 134: Ciphertext, base64url-encoded

```
vbb32Xv1lea20tmHAdccRQ
```

Figure 135: Authentication Tag, base64url-encoded
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:

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1YzEtODY3Mi02MTdiNWlONTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0...

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

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTEyY2Y2ODY3Mi02MTdiNWl0NTIOM2EiLCJjbmMiOiJjBMTI4R0NNIi0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPagaYyeAB6HYGcR55919TYnSovc23XJ
oBcW29rHP8yZOG7YhLPti1bJFuvZpJQS-moIFtVcXkZXdh_lrl_FrdYt9
HRUYkshtrMmIUYgmUn9zMDb2n0cRIHAFveJuDxkuWVae7_YGrpdc
qMyiBoCo-FbdE-Nceh4h3-FtBP-cBIwCPTjb9o08bdcdREEMJMyMBH8
ySwMVilgPD9yxi-aQpgBsv_F9N4IZAxscj5g-NJsuPbjk29-s7LJAgb1
5wEBtxXphVCgyy53CoIKLHHeJHXenx45Uz9aKZRSInZI-wjsY0yu3cT4-
  aQ3iio-tiE-F8ios61EKgyIQ4CWao8PFMj8TTnp",
  "tag": "vbb32Xvllea20tmHAdccRQ"
}
```

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-ct4GFjx8tM1U8CZ0NLBvdQsTiS8"}

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.

UWxARpat23nL9ReIj4WG3D1ee9l4r-Mv5QLuFXdy_rE

Figure 139: Content Encryption Key, base64url-encoded
gz6NjyEFNnm_vm8Gj6Fw0FQ

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.

KkYT0GX_2jhIfqN_

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.

1Jf3HbOApxMEBkCMOoTnnABxs_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.

```
{
  "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:

eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlM1ZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMzIiLCJraWQidGhlMjUyMCIsIjIwMjQ3MTQ4IiwicmFjZ3JvdW5kIjoiQTEyOENCQy1IUzI1NiJ9

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

```json
{
  "recipients": [
    {
      "encrypted_key": "lJf3HbOApxmEBkCMoTnnABxs_CvTWUmZQ2ElL
vYNok"
    }
  ],
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS
liZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YyoiJrZlBkdV
ZRM1QzSDZ2bmV3c0ta3N3IiwiaX5iOiJlallUEMedYXzJqSGxmcU5fIi
wiZ5jiioiQTEyOENCQy1IUzI1NiJ9",
  "iv": "g6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFm10Ro7w7G1QiaZpI80aiVgD8E
qoDZHyFKFBupS8iaEeVlgMqWmsujKuoVgzR3YfzoMd3GxEm3VxNhzWyW
tZKX0gxKdy6HgLvqGNb2zJ1qcpDiF8q2_62EVAbr2uSc2oaxFmFuIQ
HLcqAHxy51449xkz7ewzZaGV3eFqhpco8o4DijxG5_7kp3h2cajRfd
ygunuxUbgWgLqaeNQajtJmSMFuEOSAw9Hdeb6yhdTynCMru-kqtO5Dec
41T2OMZKpnxc_F1_4yDJFcgqb5CiDSmA-psB2k0Jt_jxA44UPI61oONK7z
zFIu4gBfjJnCndsZfdvG7h8wGjV98QhrKEnR7xKZ3Kr0_qR1B-gxpNk3
xWU",
  "tag": "DKW7jr4WaRSNfbXVP1T5g"
}
```

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

```json
{
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJpdiI6Ii16Ii
  pIbGztT8iLClJraWQiOiIzOGVjMDh1M1ZwM5LTrkOjEuYjIwNz0yYj
  RkZDFkNDMyMWQiLCJ0YmciOiIxOjIzZi1BMjU2IiwiZmNhIiwiNzIi
  "encrypted_key": "lJf3Hb0AxpMEBkCXM0nABx6_CyTWUmZQ2ElvYN
  k",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-2hjlUm_y_0_bKfM0Ro7w7GIq5a2pI8OaiVgD8E
  qoD2HyFKBFupS8iaEeVlgMqWmsuJKuoVgz3YfzoMd3GxEm3VxNhzWyW
  tZKX0xKdy6H6vqoGNbZCzljqcpDf8q2_62EVAbr2uSc2oaxFmFIQ
  HLcqaHxy51449xkjZ7ewZaGV3eFqhpco8o4DijXaG5_7kp3h2cajRfD
  gmunxUbWgLqaeNQaJtvJMSmFUeO5azw9Hdeb6yhdTyCrmu-kqtO5Dec
  4I12OZKpnxc_F1_4yDFcqb5ClDSmA-psB2k0TjtxA4j4UPI61oONK7z
  ZFrU4gBfjJCnds2fdvG7h8wGjV98QhrKEnR7xKZ3KCr0_qR1B-gxP
  xWU",
  "tag": "NvBveHr_vonkvflfnUrmBQ"
}
```

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".
5.8.2. Generated Factors

The following are generated before encrypting:

o AES symmetric key as the Content Encryption Key; this example uses the key from Figure 152.

o Initialization Vector; this example uses the Initialization Vector from Figure 153.

aY5_Ghmk9KxWPBlu_glx1w

Figure 152: Content Encryption Key, base64url-encoded

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:

CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx

Figure 154: Encrypted Key, base64url-encoded

5.8.4. Encrypting the Content

The following is generated before encrypting the content:

o JWE Protected Header; this example uses the header from Figure 155, encoded to base64url [RFC4648] as Figure 156.
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

ER7MWJZ1FBI_NKvn7Zb1Lw

Figure 158: Authentication Tag, base64url-encoded
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:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0.CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx.Qx0pmsDa8KnJc9Jo

AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvrRkH0tP1bTdhtFJgJxeVmjkJLD61AhnhWGetdg1l9ADsnWgL56NywxSYjU1ZEHcGkd3EKU0vjHi9gTlb90qSYFfeF0LwkcTtjbYKcsiNJQkcIplyeM03OmuLYSoYJVSpf7ej6zaYcMv3WwdxDF18REwOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHlQeg7gJ6xW0JqHDFn_H-puQsmthc9Zg0ojmJfqqFvePUTxLAF-KjcBTS5dNy6egwkYT0t8EIHK-oEsKytZRa

a827MOZ7UGxGIMvEmxrGCFeJal4s1v2-gaqK0kETHkaSqdYw0FkQZF

ER7MWJZ1FBI_NKvn7Zb1Lw

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

```json
{
  "recipients": [
    {
      "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1cLiLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MJzMGFkOTFhYzgiLCJ1bmMiOiJBMTI4R0NNIn0",
  "iv": "X70pmsDa8KKnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkH0tP1bTdhTFJgJxeVmJkLD61AhnWGetgd1lc9ADsnWgL56NywxSYjU12EHcGkd3EkU0vji9gTlb90qSYFeF01WkcTjtjBYKcsiNJQkIp1yeM03OmuiYS0YJVSfpf7e76zaYcMv3WwdxDF18REwOhNImk2Xld2JXq6BR53TSFkyTY7FwVLug-1GwtGH1Qeq7gDT6xW0JqHDPn_H-prQsmthc9z0qojMfqqFVeTUXiAF-KjcBTS5dNy6egkwYt0t8EIHK-oESkYt2Raa82Z7OZ7UGxGIMvEmxrGCpeJa14s1v2-gaqK0kEThkaSqdYw0FsQF",
  "tag": "ER7MWJZ1FBI_NKvn72b1Lw"
}
```

Figure 160: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJBMTI4S1cLiLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MJzMGFkOTFhYzgiLCJ1bmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx",
  "iv": "X70pmsDa8KKnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkH0tP1bTdhTFJgJxeVmJkLD61AhnWGetgd1lc9ADsnWgL56NywxSYjU12EHcGkd3EkU0vji9gTlb90qSYFeF01WkcTjtjBYKcsiNJQkIp1yeM03OmuiYS0YJVSfpf7e76zaYcMv3WwdxDF18REwOhNImk2Xld2JXq6BR53TSFkyTY7FwVLug-1GwtGH1Qeq7gDT6xW0JqHDPn_H-prQsmthc9z0qojMfqqFVeTUXiAF-KjcBTS5dNy6egkwYt0t8EIHK-oESkYt2Raa82Z7OZ7UGxGIMvEmxrGCpeJa14s1v2-gaqK0kEThkaSqdYw0FsQF",
  "tag": "ER7MWJZ1FBI_NKvn72b1Lw"
}
```

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.

by_BDcIwDEVX-QNU3QEOrIA4pq1DokYxchxVvbEDGzIjbioOSJwc-f___HPj8u
8KVFPvTvAp1VE1-wzo0YjNzo3C7R5v72pV5f5X382VwjYQpqZKAyjziZor2B7kQ
PSy6oZIXUnDYbVKN4jNXi2u0yB7t1qSHTjmMODf9QgvrDzfTIQXnyQRuUya4zI
WG3vTOdir0v7BRHFYWq3k1k1A_gSDJqtcBF-GZxw8

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_IzVQwkGgzlFDwPi

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

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwicmVfbm90IjoiREVGIn0

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.

```text
HbDtOsdaIloYziSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyez
SPt0fdLVfe6Jf5y5-JaCap_JQ8cb5opbmT60uWGl8blyiMQmOn9J---XhhL1Yg0
m-BHaqfD05iTOWxPxFMUedx7WCy8mxdDGhj0aBMG6152PwM-wSE_o2B3jDbrYBK
hpYA7qi3AyijnCJ7BP9rr3U8kkExCpG3mK420TjOw
```

Figure 168: Ciphertext, base64url-encoded

```text
VILuUwuIxaLVmh5X-T7kmA
```

Figure 169: Authentication Tag, base64url-encoded

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:

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04NjE2MGFkOTFhYzgiLCJibmMiOiIJBMTI4R0NNIiwieWiOiREVGiI0
.5vUT2WOTQxKWeckM_IzVQwkGgzz1FDwPi
.p9pUq6XHY0jfEZIl
HbDtOsda1IoYziSx25KEeTxmwnh8L8jKMFNC1k3zmMI6V88hry57tD26ljXyez
SPt0fdLVfe6Jf5y5-JaCap_JQb5b5opbmT60uWGml8lyiMQmOn9J--XhhLYy0m-BHaqFD05iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2Bj3jDbvYBK
hpYA7qi3AyijnCJ7BF9rr3U8xkExCpG3mK420TjOw
.VILuUwuIxaLVmh5X-T7kmA
```

Figure 170: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
    "recipients": [
        
            "encrypted_key": "5vUT2WOTQxKWeckM_IzVQwkGgzz1FDwPi"
    
        
    ],

    "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04NjE2MGFkOTFhYzgiLCJibmMiOiIJBMTI4R0NNIiwieWiOiREVGiI0",

    "iv": "p9pUq6XHY0jfEZIl",

    "ciphertext": "HbDtOsda1IoYziSx25KEeTxmwnh8L8jKMFNC1k3zmMI6V88hry57tD26ljXyez
SPt0fdLVfe6Jf5y5-JaCap_JQb5b5opbmT60uWGml8lyiMQmOn9J--XhhLYy0m-BHaqFD05iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2Bj3jDbvYBK
hpYA7qi3AyijnCJ7BF9rr3U8xkExCpG3mK420TjOw",

    "tag": "VILuUwuIxaLVmh5X-T7kmA"
}
```

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

```
{
    "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJJBMTI4R0NNIiwiemlWjoiREVGi0",
    "encrypted_key": "5vUT2WoTQxKWcekJM totalmente_1",
    "iv": "p9pUq6XY0j7EZl1",
    "ciphertext": "HbDtOsdailoYzIsrX25KEeTmwnh8L8jKMFNc1k3zmM16V B8hry57tDZ61jXyezSPt0fdLVfe6Jf5y5-JaCap_JQBbcb5obpmT60uWG ml8bly1MQmOn9J--XhnlYg0m-BHaqfD05iTOWxPxFMUedx7WCy8mxdGh j0aBMG6152PsM-w5E_o2B3jDbbrYKhpYA7qi3AyijncJ7BP9rr3U8kxEx Cpg3mK420Tjow",
    "tag": "VILuUwuiXxLVmh5X-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.
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.

75m1ALsYv10pZTKPWrsqdgl

Figure 174: Content Encryption Key, base64url-encoded

veCx9ece2orS7c_N

Figure 175: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc21vbilse30sInRlQnl0OLjAiXxbImZuIix7fS
widsGV4dCIsIk1lcmhZG9jIEJyYW5keWJ1Y2siXxbIm4iLHt9LCJ0ZXh0Iix7fS
IkJyYW5keWJ1Y2siLCJ2Y2FyZCIsW1sidmVyc21vbilse30sInRlQnl0OLjAiX
x8mIix7fS

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_ZzH76Ta1kJyRFgOV9MIpnx4X

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.

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

Figure 178: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

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.
5.10.5. Output Results

The following compose the resulting JWE object:

- 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": "4YiiQ_ZzH76TaIkJmYfRFgoV9MIpnx4X"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWRQOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgIClJbmMiOiJBMTI4R0NNIn0",
  "iv": "veCx9ece2orS7c_N",
  "aad": "WyJ2Y2FyZCI6W1siY2Vzc2lkIjIwMjMwMjMwMyIiLCJ2Y2FyZCI6IiJ9",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Llyf3iPppGf3Pj17wNZqteJ0Ui8p74SchQP8xygMl0FRwCNzeIa6s6BcEtP8qEFlqTUEyiNKOWDNoFl4T_4NFqF-p2Mx8zkbKXi7oPK8KNarFbyxIDvICNqBLba-v3uzXbdB89fzOJI-Lv4PjoFAQGHrgv1rjXAmKbgkft9cB4WejZw8MldBhc-V_KWZsIrsLNygon_gJWd_ek6LNq5NRehVaqf9t7r4aq3FBXw0xCys1s5PhCdaggyZ3kUf12OkwKnWUb9XVD1C6XhLlqHhCwXDG59weHrRDQeHyMROb1joV3X_bUTJDnKBFOod7nLz-cj48Jmx35ncZTPbQAkFKV",
  "tag": "voAh_Rajnpy_3hOtqvZHRA"
}
```

Figure 182: General JWE JSON Serialization

Miller                        Informational                    [Page 92]
The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFg0V9Mipx4X",
  "aad": "WyJ2Y2FyZCIwLsidmVyc2lvb1ise30sInRleHQiLCJ0LiIsIiJd",
  "iv": "veCx9ece2orS7c_N",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3pJj7wNzqteJ0U1p74SChQP8xygMloFRCwNCze1a6s6Cetp8qEFiqTUEyiNkOWDNoF14T_4NFQf-p2Mx8zkbKk7oPK8KNarFbxyIDvICNqBLba-v3uzXbdB99fzoI-Lv4PjOFQqGHrv1rjXAmKbgkft9cB4Wey2w8M1dBhcc-V_KWzs1rsLNygong_JJWd_e6l6On5NRehVApqf9zrxB4ag3FBxB0xCys35PhCdaggy2kfUfi20kwKnWbogXVD1c6HxLliqHhCwXD59weHRDQeHyMRoB1joV3X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV",
  "tag": "vOah_Rajnp_y_3h0tqvZHRA"
}
```

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".
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.

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:

```
{
  "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:

```
{
  "recipients": [
    {
      "encrypted_key": "jJIcM9J-hbx3wnqhf5F1kEYos0sHsF0H"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D M3swKkJOwQyZtWsFLYmj5YeLht_StAn21tHmQJuunT64T8D4t6C7kC9Q CCJ1IHAo1Uv4MyoT80MoPb8fZYbNKqplzYjgIL58g8N2v460gyG637d6 uuKPhwANtGm_zWhqc_srOvgiLkzyFXPq1hBAURbc3-8BqeRb481R1-__5 q5UjWVD3lgiLCN_P7AW8mIlfVvUNXBPK3nOWL4teUPS8yHLbWeL83o1U 4UAgl48x-8dDkH23JykiV5Sjfu-f7e-1xreHWXzWLHs1NqBbre0dEwK3 HX_xM0LjUz77Krppgezoutpf5qaKg3l-_xMINmf",
  "tag": "fNYLqpu84KD45lvDiaBAQ"
}
```

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

```
{
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "encrypted_key": "jJIcM9J-hbx3wnqhfh5F1kEYos0sHsF0H",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossGw3uD9DPsqLL2D
  M3swKkJwQy2tWsFLYMj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC90
  CCJ1IHa0lUv4MyOt80MoPb8fZyBnkqplzYJgIL58g8N2v46OgyG637d6
  uuKPwhAnTGM_zWhqc_srOvgiLkzyFXPq1hBAURbc3-8BqeRb48iR1-_5
  g5UjwVD31giLCN_P7AW8mIvfVuxNPBJK3nOWL4teUPS8yHLbWEl83o1U
  4UAgL48x-8dDkH23JykibV8qju-f7e-1xreHWXzWLHs1NgBbre0dEwK3
  HX_xMOLJuz77Krppegoutpf5qaKg31-_xMINnf",
  "tag": "fNYLqpUe84KD45lVdiaBAQ"
}
```

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.

KBooAF130QPv3vkcZlXnzQ

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:

244YfO_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.
Figure 197: Ciphertext, base64url-encoded

e2m0Vm7JvjK2VpCKXS-kyg

Figure 198: Authentication Tag, base64url-encoded

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:

```
{
  "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": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoVOGsR117jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQh0a7Gqrk7Eallvwt7R4TT1uq-
  arsVCPaEfwQfzrSS6oEUNbBtxEasE0vc6r7spvyVziMCVJEuRJyoAHF
  SP3eqQPb4c1SDSyxjw_L3svybhHYUyGuTmQEDjggjFBoifwHIsDs
  RPeBz1NmqgeivFq5GTCWFO5K_MNJQRR2Wj0AHCG27J2fu21WjUHLF8
  ExF2ZL41nsyv Ju_mviFMyikFnsZAudISOa6073yPz04k_1FI7Wdf
  rb2w70gKLDx1pcoxhPVOLqwpA3mFNRKdy-bQz424KX91fz1cne31N4
  -8Bkmjow-QdQkJdLOGkC445Fb_K1t1DQXw2sBF",
  "tag": "e2m0Vm7JvK2VpCKXS-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": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv",
  "iv": "YihBoVOGsR117jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQh0a7Gqrk7Eallvwt7R4TT1uq-
  arsVCPaEfwQfzrSS6oEUNbBtxEasE0vc6r7spvyVziMCVJEuRJyoAHF
  SP3eqQPb4c1SDSyxjw_L3svybhHYUyGuTmQEDjggjFBoifwHIsDs
  RPeBz1NmqgeivFq5GTCWFO5K_MNJQRR2Wj0AHCG27J2fu21WjUHLF8
  ExF2ZL41nsyv Ju_mviFMyikFnsZAudISOa6073yPz04k_1FI7Wdf
  rb2w70gKLDx1pcoxhPVOLqwpA3mFNRKdy-bQz424KX91fz1cne31N4
  -8Bkmjow-QdQkJdLOGkC445Fb_K1t1DQXw2sBF",
  "tag": "e2m0Vm7JvK2VpCKXS-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:

dYOD28kab0Vvf40DgxVAJxgHcSZICSOp8M51zjwj4w6Y5G4XJQsaNNIBiqvUUAGcpeL7s7-cFe7pio7gV_Q06WmCSa-vhW6me4bWrf7cHwEQJdXihidAYWVajJIAKMXvFRMv6i1Rr076DFthg2-AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-wqjw0-b-S1laS11Qvbup78dQ7Fa0zAVzzjHX-xyvM2wxj_0txr9c1N1Lz2MbeYSrRicJKxodvWgkpIdkMHo4LvdhRRvzozKzlic89jFwP1nBq_V4n5trGuExtp_-dBHcGlihqc_wGgho9fLMK8JOArYLCmdNQ

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": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zwj4w6Y5G4XJQsNNBiqyvUUAOcpL7S7-cFe7Piog7VQ06WmCSa-vhW6me4bWrbF7cHwEQJdXihiDAYWVajJ1aKMXmVRMV6i61Rr076DFthg2_AV0_tSiV6xSEIFqt1xnYPmp91tc5WJDOGb-wqwj0-b-S1laS1lQVbuP78dQ7Fa0xAVzzjHX-xvyM2wxj_otxr9c1N1L3MbeYSrRicJK5xodWgkpIdkMHo4LvdhRRvzoKzlic889jFWPlnBq_V4n5trGuExtp-_dbHcGlihqC_wGgho9fLMK8JOArYLCMDNQ",
  "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": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8dRhsE2xAn2DtMRb5Ma2CX",
  "y": "VDrRyFJh-Kw4dEjAqmgj5Eo-CTHAZ53MC7PjppLio3y1EjI1pOMbw9Ifz88pbfm",
  "d": "1DKHFtV-PiiFVv2VBHM_ZlVcwOmxoeyANS_1QHJcrDxVY3jhVCvZPwMxJKE793C"
}
```

**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:

```
ExInT0io9Bq8MYF6-maw5tZlgo2XThD1zWKshHixJuw_elY4gSSId_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.

```
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1Ej1IpOMbW9lfzZ84pbfM"
  }
}
```

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

```
{
  "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuwe1Y4gSSID_w",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "Uzdvk3pi5wKCRc1izp5_r00jeqT-I68i8g2b8mva8diRhsE2xA
           n2DtMRb25Ma2CX",
      "y": "VDrRyFJh-Kwd1EjAgnj5Eo-CTHAZ53MC7PjjpLioy3y1EjI1p0
           Mbw91fz84pbfm"
    }
  }
}
```

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": "59Nqh1LlYtVIhfD3pgRGvw",
    "iv": "AvpeoPZ9Ncn9mkBn"
}
```

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

The following is the assembled third recipient JSON:

```json
{
    "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmskt9-WyTpS1E",
    "header": {
        "alg": "A256GCMKW",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59Nqh1LlYtVIhfD3pgRGvw",
        "iv": "AvpeoPZ9Ncn9mkBn"
    }
}
```

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.

Figure 218: Ciphertext, base64url-encoded

BESYyFN7O9KY7i8zKs5_g

Figure 219: Authentication Tag, base64url-encoded
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": "dYOD28kab0Vvf4ODgxVAJxgHcSZICSOp8M51zjwj4w6y5g4XJQsNNibgyvuUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-vhv6wme4bwr-Bf7cHwEqJdXihidAYWVajJIAKMXMvFRMV6i1Dr076DFthg2_AV0_VI6xSEIFqt1xnYppmP91tc5WDQGb-wqjw0-b-S11aSl1QvbU784Q7Fa0zAVzzjHX-xvyM2wxj_otxr9c1N11nZMbeYSrIcJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWP1nBq_V4n5trGuExtp_dbHcGlihq_c_wGgho9fLMK8JOArYLCMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5tZ1go2XThD1zWKsHiXJuwe1Y4gSSIDw",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3p5wKCrc1izp5_r00jeqT-I6818g2b8mvaBdiRhsE2xAx2D5MRb25Ma2CX",
          "y": "VDrRyFJh-Kwd1EjAgnj5Eo-CTHAZ53MC7pjjPloy3y1EjIlpOMbw91fzZ84pbfm"
        }
      }
    },
    {
      "encrypted_key": "a7CclAejo_7JSuPB8zeagXRam8dwCfmkt9-WyTps1E",
      "header": {
        "alg": "A256GCMKW",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59NqhilLytVlhF3pgRGvw",
        "iv": "AvpeoPZ9Ncn9mkBn"
      }
    }
  ],
  "unprotected": {
    "cty": "text/plain"
  },
  "protected": "eyJlbnMlOiJBMTI4Q0JDLCThMTuZ2In0"
}
```
"iv": "VgEIHY20EnzUtZFl2RpB1g",
"ciphertext": "ajm2Q-OpPXCr7-MHXicknbl1sxLdXxk_yLds0KuhJzfWK04SjdxQeSw2L9mu3a_kIc55kCQ_3xlkcVKC5yr__Is48V0ocK0k63_QRM9tBURNFqLByj38väYQX0oJW4VUHJLmGhF-tVQBWB7Kz8mr8zeE7txF0MSaP6ga7-s1yXstR7_g07Thd1jh-zGTOwxM5g-VRORtg0K6AXpLwEqR7pkt2zRM02AXqSpe106FJ7FHLDrEFnD-zDIZukLpCbzhzMDLLw2-8I14FQrg1-iEuzHgIjFIJn2wh9Tj0cg_kOZy9BqMRZbmYXMY9YqjorZ_P_JYG3ARAIF3ojDNqpdYe-K_5Q5crGJSNyij_ygEiItR5jssQVH2ofDQdLchtazeE",
"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
eyJpc3MiOiJob2JiaXRvb15leGFlcGxlLiwiZXhwIjoxMzAwODE5MzgwLCJodHlwZU9zaG9wIl0.

Figure 223: Payload Content, base64url-encoded

{
  "kty": "RSA",
  "kid": "hobbiton.example",
  "use": "sig",
  "n": "kNpPBfDM6fycv5i-QCHA-Q-K8gsC3HJb7FYhYaw8xhBnJa-t8q01DkWgLq4XYV-ffWXjJv5fWR1ZEGU4Gm2iMfEMedOtg7TRQ3tepgKfMJGg6iy6f出了Zx2gEnnisnhsfaA9GjwRT5tmKPhbls-hwx11USAT-AIe1nqbGcF2eE5Z55SGBBoAROVdUJyqETDmgMrizkKV4ZjD8-1h4oVB07bka6cLQdHjUyUYSh_Er20DxX30Ky197piciKTs-QKXmm8ivyRCmuxx22ZoPUinded2Bk50i4GnWALhLz222k8CsRdyf-7dg7z41rP0d2EvtuAp4bX4AKrla4rfTfw",
  "e": "AQAB",
  "p": "yKYwON1KqMrQ1gB0OdT1Ni7cbDNUUs2Rh-pBAxD_mIkweMt4Mg0-B2iyVvMr8shorhonV7vxcCqgcbAAATGW-haAfUehWjxWJS-3KccRM8toL4e0q7M-idRDBXSoe722-CV2x_zCY3RP8642r13WgQGDI44M6BkUsjcy9-c",
  "q": "uND4o15Z0KdZ8fVfJw589plv1QVQ3Nei1rlnRUPHkxkaAzDzccGqrWMwPcGfFsn35cQELxeU76-51VYQg0HwYV10hVXH7Q7gaGu-483AD3ENC123FrOnF45m7_2ooAstJDe49MeLTTQKrSI1S_KvqpyrfPTczPczkh9k",
  "dp": "jmTbQqqa8ouaymjhJSCnsveUXmQMc2AneQJRQkFqGz-2V2PKNBPKvVyi5fb2-L3tM30W2d2iNDyRUXW1T7V510KwPTABSTOnTqAmYChG18kXxdlhcrtVx1dBakC6saxwi_TzGGY2MVXzc2ZnCvCXH4qj5SoXOrfP3FHFU",
  "dq": "R9FUU8B0VzEkTKX13-5-WusE4DjHmnde21lul3rifBdfLpg_P-iWpBBag9wzQ1c-J7SZcJcQkEJdv5yd2C7rn26kpzWh_rml8zscAk1gsunLt9CGJAYz-s-GWY1JGShFazF52Pb41rLC0JYuEaQMIRzypY77-0LahpMDAOhLk",
  "qi": "S8c7Zknw6PHi7kjcwtttQOPlVmRfwrIRlFAViuDb8NW9CrV_7F20qUZCqmmHTYumwGFHII1WkRq7anleWajJx7C_1b3fq_al4gH3-Be-EKIHi6gIMazuRtLUOcThrExDbF5dybsciDnFwRULwERZ4N1BeOnxYuPqxwKd9QzwMr0"
}
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.

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

Figure 225: JWS Protected Header JSON

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

dPpMqwRZxFyi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprzOOWaIAkIvn1qskChirjKvY9ESZNUCP4JjvfyPS-nqjJxYoA5ztWOyFk2cZNIPXjcJXsQwXPO9tEe-v4VSqgD0aKHzFxYog4N6Cz11Kphi1UsYDSI67_bLL7elg_vkjfMmp5_WS15LuUYGMeh6hxQ1aUXf9EwV2JmvTMuZ-vBOWy0Sniy1EFo72CRTvotrIf5AR0oSMNl1Y3KtUxeP-SOmD-LEYwW9SlkohYzMVAZDDOrVbv7KVRHpeYNaK75KEQqdCEEKs_rskZ5-Qtt_nleqTWh1mEYaA

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.
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiZXhwIjoxMzAwODE5MzgwLCJodHRwOi8vZXhhbXBsb2Fkcy8yMDQ1MjM5MDA4MDc0NzIwLXNjaWVzdC92YWxjIiwiZXhwIjoxMzAwODE5MzgwLCJuYmYiOiJ0b2N1bWVudF9yZXNpemUifQ.

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:

a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueY1VhjurCyrBg3I7YhCRYjphDOOS4E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bi13mwy7RoYhjTkbEBC6P?7sMYMXXx4gZMedpiJHb7YEuY-zkZV7A9matpgevAkJWrXzOUysYGTtwoSN6gtUVtt1Laiyjbv21O0u14YxSHV-ByK1kyeetRp_fuYJxHoKQQL9F424sKx2WGYb4zsBIPF4ssl_e51R7nany-25_UmC2urosNkoFz9cQ82MypZP8qgbQjyPN-FppZ-5o6yV64x6yzDUF_5JCIdl-Qv6H5dMVIY7qleKpXcvV11WO_2FefEBqXXvIjLe2ivjNkzogCq3-IapSjVFnmjBxjpYLT8muawolyy1XMMuinIpNcOY3n4KKrXLrCceX85m4IIHMZa38s1Hpr56fPseMA-Jltmt-a9iEDtOzhtxxz8AXy9tscAZV2XBWNG8c3kJusAamBK0Ywfk7JhLRDgOnJjllJln7T14UxDP9dCmUXEN6z0v23W15qJIEXXJtqnblpymoowAHCT4e_Owhbmlg0AEpTHUdA2iINs9WTX_H_TXuPC8yDDhilsmxS_X_xpkIHkiIHDOLx03BpqDTivpKkBYwqP2UZkcxqX2Fo_GnVrNw1K7Lgxxw6FSQvDO

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

eyJhbGciOiJSU0EtT0FFUCISImN0eSI6IkpXVCJ9

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.

```
SZI4IvKhmwpazl_pQXX3mHv1ANnOu4Wf9-utWYUcKrBNgCe2OFMf66cSJ8k2QkxaQ03_R6OMGE9ofomwtky3GFxMeGRjtpMt9OAvVLsAXB0_UTCBGbq3C2bWLXq2lfJAAoJRUPRk-BimYZY81zVBUiHc7HsQePcpu33SzMsfHjn41P_idrJz_g1ZTNgKdt8zdnuPauKTKDN011D4fuzvDYfDIAfgGPyL5sVRwbIXpXdGokEszM-9ChMPqW1QNHzuX_zUl3bvrJwr7nuGZs4cUScy3n8yE3AHCLurgls-A9mz1X38xEauiV1814Fg9tLejkdAuQzjPbqehQBJe4IWGD5Ee0dQ-Mtz4NnhkINw-YKBB_Xo2zI3Q_1sYjKUuis7yYW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUr1x4gmPUzBdwT06ubfYSDUEEz5py0d_OtWeUSycYBKDaM7txXg26qJo21gYjLfh9ny-W19sOCZGUugFjPHawXHpnvnt-0_ES96kogjJLx1IMU9Y5Xmnw2MyNc9ElnwogCs-hVuvzyF0sIrulkm194_SLlxgMl7o03phcTMxtlMizR88NkUIWkBsiXMCIjy1Noue7MD-ShDp5dmM
```

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:

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6I1EmXmhjHQ0ifQ.
a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueYVhjrCyrgJi7YhCrYjphDOOS4E7rXbr2Fn6NyQ-A-gq10FXqNvJVOGr-g-b113mwy7RoYjhTkBFEC67sMYMXx4g
zMedpiJHQVey-I-zkZV7a9matpegveAWrXzOUyeYGTtwoSN6gtUTl1aiyjvb21O0u14YxSHV-ByK1kyeHoKIQ7L9P424sKx2WGy4bzsBIFPF4ssl_e51
R7nany-26_Umc2urosNkoFz9cQ82My2P8gbQJyPN-Fpp4Z-5o6yV64x6yzUF
F_5JCIdl-QvV6H5dMVIY7qleKpxCV11WQ_2FeEFbqXxXvIiLeZivjNkzogC3q-I
apSjVFqMjBxypYL78muawolfylxXMuinIhOC0Y3n4KKrXLRceteX85m4IIMHz
a38s1hpr56fPPseMA-Jltmt-a9iEDo2htxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBKOYwfk7JhLRDgOnJjJiJLhh7TI4UxpD9dCmUXEN6z0v3W15qJIEQJtJqnbIp
ymooreWAHCT4e_Owbmlg0AEpTHUdA2liLNsW9TX_H_TXUC8yDh11smKs_X_x
pk1HkiiIHWDOLx03BpqDTipvKBYwqP2UZkcxqX2Fo_GnVrNw1K7Lqwx6FSQvDO
0.
GbX1i9kXz0sxXpMA
.
SZI4IVKHmwpazl_pJQXX3mHv1AANnOU4Wf9-utWYUCrKrBGNeCe2OFMf66cSJ8k2Q
kxaQD3_R60MGE9ofomwtky3CGxMeGRjtpMt90aVVLsoAXB0_UTCBGyBq3C2bWLX
qZIfJAAoJRUprk-BimY2Y81zVBulhch7HseQPcpu333ZMsFHjn41p_idrJz_glZ
TNgKD8zdnUPauKTKDNH01D4fuzvDfDIAfGPyL5sVRwbiXpdGokEsZM-9C
hMfQw1QNhzuX_2ul3bvr7Wv7nG2s4uScy3n8e3AHCLurgis-A9mz1l38xEa
ulV184F9tLejdpkuZj2fBeHQBjE4iwG5E0dQ-Mt24NhckIwx-YKBb-Xo2
Zi3Q_l5yJUnls7ywW-HrVqVf0bj7Wf2vZo9TZ3dvsCoTvPH2dyWum6Ur
lx4gmPzBd8O6ubfYSDUEEz3pYo0d_OtWeUSycCBKd-aM7xQg26qO2lgYyjlf
hn9zy-W19sOClGuzfGjPhawXHpvnj_t-0_ES96kogjJLxSL1IMU9YSnmwZMYnc
9E1wnogscG-hvVvzyP0iairuktm194_SLXgM17o03phcTMxt1MizR88NKi1WkB
siXMCjy1No1u7MD-ShDp5dmM
.
KnIKEhN8U-3C9s4gtSpjSw

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

```
{
  "recipients": [
    {
      "encrypted_key": "a0JHRoITfpX4qRewImj1Stn8m3CPxBV1ueY1VhjurCyrBg3l7YhCRYjpD0OS4E7rXbr2F6N6NyQq-A-qtqFXqNjVOGrG-bi13myw7r0Yyjtk3k6aBE6F7sMyMXXx4gzMexdjiIyHQYe-vzKZ
7A9matpegveA7Wrx0UysYGTtwoSN6gtUVt1Laivyvb2100u4YhXS
Hv-YxKlyeetRp_fuYjxHoKQl9I4q24sKx2WGYy42zsBIF4s6l_e
5Ir7nany-25_UMC2urozNkoFz9cQ82Zmp2Fp8q8qJyPN-FnP44z-5
06yV6x6yDUF_5JCd1-qV6H5dMV9Y7q1eKpXcvl1lWO_2FeE8q
XxXvIjLeZivvNkzogCq3-IapSjVFnMjBxjpYL8amw901yyXXM
uinIpnOY3n4KkrXLrcCtceX85m4IbHMAz38s1Hpr56FPPsEaMA-J1
tmt-a9iEDtohtxx-8AXy9tscaZ2V2X8BNG8c8ckJusAamBK0yYwkJ7
hLRDgOnJjILhnn7TI4UxPd96cmUXEN6z0v23W15qIJEXNkJqnb1p
ymoeoWAHCT4e_Owbin1g0AEpTHUdAziiLna96W8X_HlXuPC8yYDh
i1sMxS_X_xpkIhiIHD01Lx03BpqDTivkkBYYrP2UZkcesxQX2FO_
GnVrNw1K7Lgxxw6FSQVdO0"
  }
],
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6Ij6kXVCIsImVuYy
I6IkExMjhhQ00ifQ",
  "iv": "GbXl19kXz0sxXPmA",
  "ciphertext": "SZI4IvKHmpwpa1l_pQXX3mHvlANnOU4Wf9-uN7Y6ucKzBN
qCe2OOFm66c3Jy2kZQxxjQxD360M9G9fomwtky3GFXwGrjtpYh7OAv
VLsAXBO_UTC7BYgBc3c26WLxQz1fJAAoJRUPRk-BimYzY1zVBuLh7c5h
QeCPnpu33szMsFHiin4p_1drdz_gJ2ZNgKDt8zdhN4ufaKTDDNH1DD4f
zvDYDIAfGPyG5zVrWbiXpXdGokEszM-9ChMPqWlQNhzuX_2i3bvrJ
wr7nuGz5cUSy3n8yE3AHCLrugs-A9mz1X38eaud1V1814Fg9teLj3
kAuOZzpjPqehQ8JbE4IwGD5EoDq-Mt44NnhkIWx-YKbb_Xo2z13Q_1sYj
KUuis7yWW-HTr_vqveFtob7jW7WFf2ZsO2T3svogaTvPH2dyWwumUr1c
gnMrPUzBdTO6ubfY5DUEE3zy0_0d OtweUSyCyYB0d-aM7Xq26g6jOzg1pY
jLfhm23zy-Wl9oOCZGuzgFJPhawXhpnvj-i-0_ES96kogjJLxS11MU9Y5
XmmwZMyNc9E1EwngsCg-hVuvzyP0sIruktMl94_SL1yknM17o03phecTmX
1mizr88UNKU1KbBsiMCcj1Noue?MD-ShDp5dmM",
  "tag": "KnIKEhN8U-3C9s4gSpjSw"
}
```

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

```
{
  "encrypted_key": "a0JHRoITfpX4gRewImjlStn8m3CPxBV1ueYlVhjurC
yrB317YhCRYjphDOOS4E7rXbr2Fn6NyQq-A-gqT0FxqNjVOGr-bil13
mwy7RoYhjTbEC6P7sMYMYXnx4gzMedplJHQVeyI-zkZV7A9matpgewvAJ
WrXzOUysYGtowoSN6gtUv11aiyvb2100u14YxSHV-ByKlkyeetRp_f
uYxHoKLQL9P424sKxZ0GEyb4zsBIPF4sssl_e5IR?nany-25_UmC2uros
NkoFz9Q82Myp28qgbQJyPN-Fpp4Z-5o6yV64x6yzUF_5JCId1-Qv6
H5dMVl7Y7leRpxcV11lW0_2FeFbEqXXvXtjLeZivjNzkogCq3-IapSjVF
nMjBxjpyYL78maaowoly1XXMuinIpNc0Y3n4KrXLrCctexX85mdIIHmZ
a38s1Hpr56fPPseMA-Jltmt-a9iEDtRzhtxZ8AXY9tscAZV2XBNWg8c3
kJusAamBKOYwfk7JhLrDqOnjJjJLhn7TI4UEdp9dCmUXEN6ozv23W15q
JIEXNjtnqlpmoocWAmHCT4e_Owbimlg0AEpTHUda2iilns9WHT_X
uPC8yDDhilsmsxS_X_xpkIRhiIHWDOMLx03BpqDTipvKkBYwqP2U2kcqX
2FO_GnVrWn1K7Lgwx6FSQvD00",
  "protected": "eyJhbGciOiJSU0Ei

"iv": "GbX1l9kXz0oxxPmA",
  "ciphertext": "SZI4IvKhnwpszl_pJQQX3mHv1ANnOU4W9f-utWYUCrBN
gCe2OFMf66cSJ8k2QkxqAQD3_R60MG9efomwtky3GFxMeGRjtpMt90Av
VLsAXB0-UTCbGyBgq3C2bWLQxZ1fJAAoJRUPRK-BimYZY81zVBu7c7Hs
QePcpu33SzMsFhjn41P_idrJz_g1ZTNqKd879nUPAUkTKDNOH11DD4f
uzDYRdIAfqFyld5sVRLwbikxPdGokEszM-9ChMPw1QnzuU_x413bvrJ
r7nuGZs4G5Cf3n82EBaClergsl-A9mz1X38xEauV1814F9t7ejd
kAuQzjPbqeqBQ9e4iWGD5e0dQ-Mt4nNhh11Wx-YKBb_0z21z3QIsyJ
KUi9s7yWW-HTr_vqvtObj7WJF2vzbOTz3dvs0GaTvPH2dyWwwUmUr1x4
qMf0U2BdToU6ubYSDUDEzzSpy0d_0tWeUSycCyYBDK-aM7tXq26qJo2lgY
JlFhnn9zy-W19sOC2GuzgFJPhawXHpvn_t-0_ES96kogjLJL91U95Y
Xmnw2My2cNcEFlwngosCg-hVuyzyP0sIruktmt94cSLkxgM17o3phcTMx
t1Mi2z88NKUIWkBsiXMCjy1Noue7MD-ShDp5dmM",
  "tag": "KnIEhN8U-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