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
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-U88c3v13nmGZx6sYl_oJXu9A5RktKqjqvyekWF-7ytDyRXyGF5cj0Kt",
    "y": "AdymlHz0iXxhEhayXQnNCvDX4h9htZaCJN34kfmC6Pv5ohQHdirVysSsU0qkAgDPgwRbmbnX9cw1Gfp=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_oJXu9A5RkTKqjqvjyeKF-7ytDyRXYgCF5cj0Kt",
  "y": "AdymlHvOiLxKKEhayXQmNcvDX4h9htZaCJN34kmCpV5ObHiraVVSsUdaQkAgDPrwQrJmbnX9cw1GfP-HqHZR1",
  "d": "AAhRON2r9cqXX1hg-RoI6R1tX5p2rUAYdmpHZoClXNM56KtscrX6zbKipQrCW9CGZH3T4ubpnoTKLDYJ_fF3_rJt"
}
```

Figure 2: Elliptic Curve P-521 Private Key

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

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

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

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

```
{
  "kty": "RSA",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "n": "n4EPTAOCC9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rHVTeT-0-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHz_AJmSCPmAJMRBSFKrKb2wqVwGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqkHHig3RBordaZ6Aj-oBHqFEHYpPe7Tpe-OfVFhd1E6cS6M1FZcD1NNLYD5lFhpPI9bTwJlsde3uhGqC0ZCuEHg81hwOHRtIQbSOFVbb9k3-tVTU4fg_3L_vniUFAKwuC LqKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUkrZ5ia1Qkqgc5gHdrNP5zw",
  "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.


```json
{
    "kty": "RSA",
    "kid": "bilbo.baggins@hobbiton.example",
    "use": "sig",
    "n": "n4EPtAOCc9AlkeQHPzHStgAbgxa7bT2LwUBZdrR8_KuKPEHld4rHTeT
-0-XV2jRoJndHhxJWTDvNd7nqQDVEi2QHz_AJmsCpMaJMRBSFKrKb2wqV
wGU_NaYOL-QtI2lzEc86XCOdAp5yqDlqHqKHHig3RQorda26Aj-oBQhFEHyP7e7Tpe-OFVHDh16C6S8M1FXzdC1NLYD51FhpP19zTJvIsd
3u9GgC0CuEHg8lhwzOHrtIQbS0FVbb9k3-vTU4f4g_3L_vn1UIFwKC
LqKns2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUKr5iaIQqg5g
HdRNP5zw",
    "e": "AQAB",
    "d": "bWUC9B-EFRo8kPgh02uyGPvMNKvYWntB_iikiH9k20eT-01q_I78e
iZkpXxQOUTEs2LsNRS-8uJbVQ-A1irkwMSMKI13XTgdrshCku9qRld
Y7sNA_AKZH-G6q142mINLRCe8W-n234ui_qofkLn9QWDQdpaIsA-b
MvWWSDFu2MUBYwkhMTMezLYGqOe04noqeq1hExBTHOBdkmXiuFhUq1BU
61-DqEiWxqg82xt2h-LMnT3046AOYJoRzoz75stSUQfGcshWTbnPS0Uj
D18Kkhyv071hfSjdRdM51y121hsFF4L_mHCuoFau7gdspFHPXjvOc
OpBrQzwQ",
    "p": "3S1xg_DwTXJc6b0695RoXygQCAZ5RnAvZ1n0ylhHtnUex_fp7AZ_9nR
a07Hx-_SFfGQeuta02TDjDAWU4Vupk8rw9JR0AzZ0NzfouIAmR_WCsm
gnQnqev1T7IyEsnh8UMt-n5CafhkikhzEsrmndH6LxOrvRJ1sPp62v8
JuQk",
    "q": "uKE2zh-cT6DERFk4k4e_jy78GfPUIaUyoSSJubBzp3Cubk3OQcs6grT
8bu_cuUDmIMZWmtmdqDyI515HRqu3MP15vMMQN81HTe2u21mkvqWn7an
5V5zhMIIiZ7z4yNhKuUFwOvvyY89EXxRd-hdq(RhX1sQAz192B3pF0J0
75Fc",
    "dp": "B8PvVXkvJr2l-G1Q7v3y9r6Kw5g9SahXBwsWUzp19T1vgl-YV85q
1N1brXyOgTs-IsXXR3-TanevuRPRtS0OdiMGQq8pbt26gLjYFKU_E9x
-RLUHz-0d9e9gXLKD4Vngp-pFQ_q29p5xWHOp0p09QF1hVcChRX
59ehik",
    "dq": "CLDMdGduhlc9o7r84rEUVn7pQzQ6PF83Y-1BzXz5NT-Tp0OZKF1pEr
AMVevKzFL14H6L50ML0W5s0FbwtXywZDm6sI6cg5ItbwWQIC3gnJk
bi_7k_vjGhWxXpA2XppvP-zyEkDERef-r4c_211Cq9AqC2yeL6dkK
T1cYF8",
    "qi": "3PiqXQNOzWee-sBv2gij289XP9XCF3VWqPzMKnIgQp7_Tugo6-N
ZBKCQsMf3HaEGBjTVJs_9cKB-TRXvaKe-72MaQj8VfBdysbubDNUkDh
jJ-GtiseaDwT7dcH0cfwxgFUHpQh7F0crJ6h62EpMF6xmujs4qMPP
z8aaI4"
}
```

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": "hJtXIZ2uSN5kbQfbtTNWbpmhkV8FJG-Onbc6mxCcYg"
}
```

Figure 5: HMAC SHA-256 Symmetric Key

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

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

This example illustrates a symmetric key used for encryption.

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

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

Figure 6: AES 256-Bit Symmetric Encryption Key

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

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

4. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

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

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

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

SXTigJ1lzIGEgZGFuZ2Vyb3VxIGJ1c21uZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZ2CwgYW5kIG1mIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldC4gdGhlcmXigJ1lzIG5vIGtub3dpbmcdZmlcm
UgeW91IG1pZ2h0IGJ1IHN0ZXBoIG9mZiBiB0by4

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:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
  .
  SXTigJ1lzIGEzZGFuZ2Vybj3VzIGJ1c2luZXNzLCBgc3RvcmNlIHN0cml0dGVBb2Nl
b24nCgkmaG9iYml0b24uZXhhbXBsZS4g
  .
  MRjdkly7-OyPTS3AXP41lIQGKa80A0ZmTuV5MEaHoxnW2e5CZ5N1KtaiFmKZopdHM1O2U4mwzJdQx996ivp83xug11I7PNDi84wnB-BDk0BwA78185hX-Es4JIwvDLJK3fWRa-XtL0RnlutYy746iyYTh_qHRD68BNt1uSNCrUCTJDT5aAE6x8W1Kt9eRo4QPocSadnHXFxtn8Is9zPzERV0ePPQdLuW3IS_de3xyIrdLaLgdjuPxAHh6L2axInc1U2podGUOKLUQSE_oI-ZnntKJ3F4uO2Dnd6Q2WJushZ41Ax_fCIE8u9ipH84ogoree7vjbU5y18kDquDg

Figure 13: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```json
{
  "payload": "SXTigJ1lzIGEzZGFuZ2Vybj3VzIGJ1c2luZXNzLCBgc3RvcmNlIHN0cml0dGVBb2Nl
b24nCgkmaG9iYml0b24uZXhhbXBsZS4g"
  ,
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9",
      "signature": "MRjdkly7-OyPTS3AXP41lIQGKa80A0ZmTuV5MEaHoxnW2e5CZ5N1KtaiFmKZopdHM1O2U4mwzJdQx996ivp83xug11I7PNDi84wnB-BDk0BwA78185hX-Es4JIwvDLJK3fWRa-XtL0RnlutYy746iyYTh_qHRD68BNt1uSNCrUCTJDT5aAE6x8W1Kt9eRo4QPocSadnHXFxtn8Is9zPzERV0ePPQdLuW3IS_de3xyIrdLaLgdjuPxAHh6L2axInc1U2podGUOKLUQSE_oI-ZnntKJ3F4uO2Dnd6Q2WJushZ41Ax_fCIE8u9ipH84ogoree7vjbU5y18kDquDg"
    }
  ]
}
```

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

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2tuZXNzLCBGcm9kbwZ2pmbcgb3VOIHlvdXIgZ29vci4gWW91IHNOZXAgb25obyB0aGUgc29mZGw5IGlmIGlvdXJkcywgcw5iIGlmIGlvdXJkcywgd2F0ZSBhbWUgZmVldCB0byB0aGUgcm9hZCBrZWVwIHZwaG9uaWxlLmF2ZSBub29zdGluZyBzdGprinting
```

```json
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",

  "signature": "MRjdkly7_-oTPTS3AXP4iIIGKa80A0ZmTuV5MEaHoxnW2e5CZS5N1KtainoFmKZopdhM1O2U4mwzJdQx996ivp83xuglI7PNDi84w
```

```json
  "n": "BDkoBwA78185hX-Es4J1wmDLJK3lfWRA-XtL0RnlHuYy746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8WlKt9eRo4QPocSadnHxXnt8is9UzpeERV0ePPqDLuW3IS_de3xyIrDaLgdjlUXUAb6L2aXic1UZpodGU0
```

```json
  "KLUQSE_oI-ZnmKJ3F4uOZDnd6QZnWJushZ41Ax_fciE8u9ipH84ogore e7vbU5y18kDquDg"
)
```

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

```
eyJhbGciOiJQUz4MCI6ImJpdG1jLmNhZ2dpbnNAd3o9
```

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).

```
eyJhbGciOiJQUz4MCI6ImJpdG1jLmNhZ2dpbnNAd3o9
```

Figure 18: JWS Signing Input

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

```
cu22eBqkYDKgI1TszDXGvaFFz6WGoz7fUDcfT0kk0y42miAh2zyBzk1xEsnk2I
```

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:

```
eyJhbGciOiJQUzIzSkM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9
.SXTigJlzIGEgSGFvZ2Vy63VzIGJlci21uZXNzLCBGb69kbywgZ29pbmcgb3V0IHBvdXIgZG9vi4g
WW91IHlvdXIgZ2VydGlctGF5IG1lIHNvbG9yZSB2YW5jZSB0byB0byBhcmRhaWQgYWN0
IHN0ZXAgZGF0YXJhIHNvZG9sZSB0byBkaXN0YW5jZSBmdXJvbGlvIG1lIHNvbGVzIHRlb
XNzLCBsaXZlZSB0byB0byBhcmRhaWQp
```

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

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2Vybc3VzIGJ1c2luZ2XNzlCBGcm9kbywgZ29pbmcgb3VOIHlvdXigZG9vci4gWW91IHN0ZXAgb25obyBOaGUgcm9hZCwgYW5kIGlmiHlvdSBkb24ndCBrZWVvIHlvdXigZ2mVldCwgGh1cmXi
  "signatures": [
    {
      "protected": "eyJhbGciOiJQU0M4NSIsImtpZCI6ImJpbGJvLmJhZ2dpbNAaG9iYmI0b24uZXhhbXBsXZS99",
      "signature": "cu22eBqyDKgI1TpzDXGvaFfz6WGoz7fUDcfT0kkoy42miAh2qyBzk1xEsnk21pN6-tPld6VrklHkqsGqDqHCDp608TTB5dDIt11Vo6_1LOLpccbUrhUIUSMbXUvdvXzg-U0D8biirEQflf2z8zGWSdiNAUf8ZnyPcGgVFn4422dNqiVJRmBqrYRxEx8P_i_jQ7p8VzdZ0TrxUeT31m8ds9hnr2lfJ8ItUjvAA2xez2Mlp8cBE5awDzT0qI0nuiP1aCN_2_jLAEQTlqRhtfa64QQSUmFAajVKBpBy17xho0uT0cbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw"
    }
  ]
}
```

Figure 21: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2Vybc3VzIGJ1c2luZ2XNzlCBGcm9kbywgZ29pbmcgb3VOIHlvdXigZG9vci4gWW91IHN0ZXAgb25obyBOaGUgcm9hZCwgYW5kIGlmiHlvdSBkb24ndCBrZWVvIHlvdXigZ2mVldCwgGh1cmXi
  "protected": "eyJhbGciOiJQU0M4NSIsImtpZCI6ImJpbGJvLmJhZ2dpbNAaG9iYmI0b24uZXhhbXBsXZS99",
  "signature": "cu22eBqyDKgI1TpzDXGvaFfz6WGoz7fUDcfT0kkoy42miAh2qyBzk1xEsnk21pN6-tPld6VrklHkqsGqDqHCDp608TTB5dDIt11Vo6_1LOLpccbUrhUIUSMbXUvdvXzg-U0D8biirEQflf2z8zGWSdiNAUf8ZnyPcGgVFn4422dNqiVJRmBqrYRxEx8P_i_jQ7p8VzdZ0TrxUeT31m8ds9hnr2lfJ8ItUjvAA2xez2Mlp8cBE5awDzT0qI0nuiP1aCN_2_jLAEQTlqRhtfa64QQSUmFAajVKBpBy17xho0uT0cbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-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
{
  "alg": "ES512",
  "kid": "bilbo.baggins@hobbiton.example"
}
```

Figure 23: JWS Protected Header JSON

eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9
hbbXBsZSJ9

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).

```
eyJhbGciOiJFUzUxMjI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHVvdXlzcG9vcnJvb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
b24ndCBzZWNvIHlvdXlzcG9zImlcmdXb3V0IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
b24ndCBzZWNvIHlvdXlzcG9zImlcmdXb3V0IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
```

Figure 25: JWS Signing Input

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

```
AE_R_YZCChjn4791jSQCrdPZCNyqHXCTZH0-JZGYN1aAjp2kcaluUUIUnC9qvb
u9Plon7KRTzoNEuT4va2cmLLejAQy3mtPBu_u_sDDyYjnAMDrXPn7XrT0lw-kv
AD890j18e2puQens_IEKBpHAb1sbEPX6sFY8OcGdQorBomu9xQ2
```

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:

```
eyJhbGciOiJFUzUxMjI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHVvdXlzcG9vcnJvb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
b24ndCBzZWNvIHlvdXlzcG9zImlcmdXb3V0IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
b24ndCBzZWNvIHlvdXlzcG9zImlcmdXb3V0IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IHN0ZXAgb250byB0aGUgcm9hZ2wgYWF5IGlmIHIvdSBk
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJjIc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXJgZGVkci4gWW91IHNO2XAgb250byB0aGUgcm9hZCgwY25kIGlmIHlvdSBkb24ndCBrvZWVvIHzXlg2mVldCwgdGlcmXicgJlZIG5vIGluc3RhbmNlIGxlc3Qh",
  "signatures": [ {
    "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImlvdmFyNjUtMzEiLCJ0eXBlIjoxfQ",
    "signature": "AE_R_YzCChj4791jSCwdPZCNYqGXCTZH0-JZGYNl2m5kUw9hZlZ1cGx5IGluc3RhbmNlIGxlc3Qh",
  }]
}
```

Figure 28: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJjIc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXJgZGVkci4gWW91IHNO2XAgb250byB0aGUgcm9hZCgwY25kIGlmIHlvdSBkb24ndCBrvZWVvIHzXlg2mVldCwgdGlcmXicgJlZIG5vIGluc3RhbmNlIGxlc3Qh",
  "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImlvdmFyNjUtMzEiLCJ0eXBlIjoxfQ",
  "signature": "AE_R_YzCChj4791jSCwdPZCNYqGXCTZH0-JZGYNl2m5kUw9hZlZ1cGx5IGluc3RhbmNlIGxlc3Qh",
}
```

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.

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

Figure 30: JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiI2LWNlZXBlcl9pZDkiLCJhdF9zdWIiOiJPR0ZHQUEyNUEyIn0.
```

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).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiI2LWNlZXBlcl9pZDkiLCJhdF9zdWIiOiJPR0ZHQUEyNUEyIn0.
```

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

```
sO6KThzkfBBAkLspWIlh8V4VzZFTsPPqMDA7g1Md7p0
```

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:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOTItNDcxYjIiLCJ0eXBlIjoiR0JFQ0ZFT0QifQ
SxTigJlzIGEgZGFuZ2VtZW50b3VzIGFja2FnZSB0byB0aGUgYm9yZSBhbnZlcnNlcyB0aGUgY29tcGxleSB3
YXcgY29va2llcyB0byB0aGUgYm9ybWF0cyB0byB0aGUgYm9yZSBhcnRpZmljYXRlIHN0cyB0byB0aGUgY29t
ZSBJbmFib3IgY29tZSB0byB0aGUgYm9yZSBhcnRpZmljYXRlcyB0byB0aGUgY29tZSB0byB0aGUgYm9yZSBh
```

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

```json
{
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
Z29pbmcbg3V0IHIvdXigZG9vc14gWW91IHNoOZXAgb25obyB0aGUgcm9h
ZCwgYW5kIGlmlHIvdSBkb24ndCBZWVwIHldXIGzWldCwgGhlcmXigJlzIG5vIGtpub3dpbmcg2h1cmUgeW91IGlpZ2h0IGJ1N3XBO1G9m
ZiB0by4",
    "signatures": [
        {
            "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
RkOWIzNDcxYl1iZmQ2LWV1ZjMxNGJjNWYzNjY9",
            "signature": "s0h6KThzkfBBKlSpW1h84VsJZFTsPPqMDA7q1Md7p0"
        }
    ]
}
```

Figure 35: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
Z29pbmcbg3V0IHIvdXigZG9vc14gWW91IHNoOZXAgb25obyB0aGUgcm9h
ZCwgYW5kIGlmlHIvdSBkb24ndCBZWVwIHldXIGzWldCwgGhlcmXigJlzIG5vIGtpub3dpbmcg2h1cmUgeW91IGlpZ2h0IGJ1N3XBO1G9m
ZiB0by4",
    "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOW
ItNDcxYl1iZmQ2LWV1ZjMxNGJjNWYzNjY9",
    "signature": "s0h6KThzkfBBKlSpW1h84VsJZFTsPPqMDA7q1Md7p0"
}
```

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

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiIiLCJqYXZlciI6eyJ0eXBlIjoiRG9zaXRpb24iLCJzdWIiOiJhZG1pZCJ9
```

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).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlLTRkOWItNDcxYiIiLCJqYXZlciI6eyJ0eXBlIjoiRG9zaXRpb24iLCJzdWIiOiJhZG1pZCJ9
.SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZGVyb250cyBhbmQgZ3Jvc3QgdG8gZGV2ZXJzLg==
```

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:

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9

s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0

Figure 41: General JWS JSON Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

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

```
{
  "protected": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9"
  "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.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJlcl2luZXNzLCBGcm9kbywgZ29pbmcqbc3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZ2wgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXgZmVldCBrZwVlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IHN3ZXB0IG9mZiB0by4

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:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pcmcgb3V0IH1vdXIgZG9vci4gWW91IHN0ZXAgb250by90aGUgcm9hZCwgyW5kIGlmbH1vdSBkb24nCBrZWVvI1vdXIgZmVldCwgdGhlcmXgJ1lzIG5vIGtb3dpbmcgd2h1cmUgeW91IGlpZ2h0IGJ1HN3XBOIG9mZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-ee6314bc7037"
      },
      "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20"
    }
  ]
}
```

Figure 49: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pcmcgb3V0IH1vdXIgZG9vci4gWW91IHN0ZXAgb250by90aGUgcm9hZCwgyW5kIGlmbH1vdSBkb24nCBrZWVvI1vdXIgZmVldCwgdGhlcmXgJ1lzIG5vIGtb3dpbmcgd2h1cmUgeW91IGlpZ2h0IGJ1HN3XBOIG9mZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiJ9",
  "header": {
    "kid": "018c0ae5-4d9b-471b-bfd6-ee6314bc7037"
  },
  "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20"
}
```

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).

```
SXTigJ1zIGEgZGFuZ2Vyb3VzIGJ1c2luZXRlBGcm9kbywg229pbmcgb3V0IHBldvXigZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBzZWVwIHlvdXIgZ29pZ2h0IGJhcm5hbCBhbiB0aGUgcm9hZCBIQjJ1b25yIHRoZSB0byB0aGUgcm9hZCBIQjJ1b25y
```

**Figure 52: JWS Signing Input**

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

```
xuLifqLGiblpv9zBpuZccWhNjlgARaLV3UxvkhJxZuk
```

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

The following compose the resulting JWS object:

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

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

The resulting JWS object using the general JWS JSON Serialization:

```json
{
    "payload": "SXTigJ1zIGEgZGFuZ2Vybi3VzIGJ1c2luZWNzLCBGCm9kbywgZ29pbmcb3V0IHlvdXIgZG9vc2V5c2V5bmcgYWJsZS4gWW91IHN0ZXAgd2hpdGlvbmcgZ29vZ2xlIGlvdXIgZmVldC4g
    Zm9yIHN0ZXAgd2hpdGlvbiB0byB0aGUgc3lzdGVtLg==",
    "signatures": [
        {
            "header": {
                "alg": "HS256",
                "kid": "018c0ae5-4d9b-471b-bfd6-ee41307037"
            },
            "signature": "xuLifqLGibp9zBpuZczWhNj1gARaLV3UxvxhJxZuZiB0by4",
            "signature": "xuLifqLGibp9zBpuZczWhNj1gARaLV3UxvxhJxZuZiB0by4",
        }
    ]
}
```

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

```json
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZWNzLCBGcm9kbywgZ29pbmcgb3V0IExvci4gWW91IHByZXZpbGUgYW5kIHdlciBhbmQgdGhlcmVycm9yIGFyZSBIaXNoZWQgZWNjZXNzaW5lc3MgZmVcaW5hZ2UgZmVyaWVudCB0byB2YWx1ZSBhbmQgZGV2ZWwgaXMgd29ybGluZSB0byB0byB0aGUgcm9hZCgjIGZ2ZXJ0aW9uIHdpbGx5IG1pZ2h0IGJ1c3BlY3RvZ3MgYW5kaW5nIHRoZSB0byB0aGUgcm9iZXSwgS2Fyb2dlZSB0byB0aGUgc3RyaW5nIHRoZSB0byB0byB0byB5IG1vcmUgZmFzdC4",
  "header": {
    "alg": "HS256",
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvXhJxZuk"
}
```

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.

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

Figure 56: Signature #1 JWS Protected Header JSON

```
eyJhbGciOiJSUzI1NiJ9
```

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

```
{
  "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.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJlc2luZXNzLCBhbmltb24gZGVzdCB3aWR0aD9y
```

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

\[
\text{MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrqi5NvyG53uomicltcMdSg-qptrrzZc7CG6Swv2Y13TDiHzTUgL_1R22FcryNF1hkSw129EghGpwkpxaTn_THJTCglNbdOkolMZBIdwzJxwq2c-1Rlp02HiBbUYyXSw097BSe0_ev2KdjvvsKgsIqjytTKSeAMbhMBdMma622_BG5t4sdBuCHTfpj9iJmkio47A1wqkZV1aIZsv33uPUqB
BCXbyoQJw7mxFpFtHmN1GoOSMrXR_3thmXTCm4US-xiNOyhbhm8afKK64jU6_TpQHiJeQJxz9G3Tx-083B745_AfYOnlC9w}
\]

Figure 60: JWS Signature #1, base64url-encoded

The following is the assembled first signature serialized as JSON:

\[
\{
"protected": "eyJhbGciOiJSUzI1NiJ9",
"header": {
  "kid": "bilbo.baggins@hobbiton.example"
},
"signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrqi5NvyG53uomicltcMdSg-qptrrzZc7CG6Swv2Y13TDiHzTUgL_1R22FcryNF1hkSw129EghGpwkpxaTn_THJTCglNbdOkolMZBIdwzJxwq2c-1Rlp02HiBbUYyXSw097BSe0_ev2KdjvvsKgsIqjytTKSeAMbhMBdMma622_BG5t4sdBuCHTfpj9iJmkio47A1wqkZV1aIZsv33uPUqB
BCXbyoQJw7mxFpFtHmN1GoOSMrXR_3thmXTCm4US-xiNOyhbhm8afKK64jU6_TpQHiJeQJxz9G3Tx-083B745_AfYOnlC9w"
\}
\]

Figure 61: Signature #1 JSON

4.8.3. Second Signing Operation

The following is generated before completing the second signing operation:

o 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": "ARcVLnaJJaUGW8fG-8t5BREVAvTY8n8YHjawD01muhcdCoFZFFjfISu0Cdkn9Ybd1mi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrqcI3JKj2U5iX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxmx4fgV3q7ZYhm5eD"
}
```

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 .SXITigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZW NXzLCBGcm9kbywzZgpbmckb3V0IH lvdXigZG9vcj4gWW91IHJN0ZXAgb25obyB0aGUgcm9hZCwg5IG5kIGl0dCBvdSBk b24ndCBvZVwIHlvdXigZmVld3NldHMXigJlIHN3ZXB0IG9mZiB0by4 UgeW91I1p2Z2h0IGJlIHN3ZXB0IG9mZiB0by4

**Figure 68: JWS Signing Input**

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

s0h6KThzkfBBbkLspWIh84VsJZFTsPPqMDA7g1Md7p0

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

```json
{
    "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
    "signature": "s0h6KThzkfBBBkBpW1h84VzJZFTsPPqM7g1M7p0"
}
```

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": "SXTigJlzIGEgZGFuZ2Vybc3VzIGJ1c2luZWNzLCBGcm9kJ9
Z29pbnRmZ29vc2V5IHlvdXIgZGF0YXV0eW91IHlvdXIgZ29vZ2xlY3QgJG
JlZ1G5vIGTub3dpbmZlcmVjdG5ldGNoIGV4cG9ydG5ldGNoIGZvZXIt
"signature": [
  {
    "protected": "eyJhbGciOiJtZSJ9",
    "header": {
      "alg": "HS256",
      "kid": "bilbo.baggins@hobbiton.example"
    },
    "signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FghiocsqrqiNvy
G53uoimictcMdSg-qptrzZc7CG6Swv2Y13TDiHxTUrL_1R2ZFc
ryNFhKSw129EghGwpkpxaTn_THJTcNgADko1MZBCdwxJxwqZc
-I1lpO2HiUYyXSwO97BSe0_evZKdjvKSGsIqyjKSeAMBhMBdM
ma622_BG5t4sdbuCHtFjp9ijmkio47AIwqkVZ1v3s3puUgBBC
XbYoQJw7mxPftHmN1GoOSMxR_3thmXTcm4US-xiNOyhb8afKx6
4jU6_TPtQHJeQJxz9G3Tx-083B745_AfYOn1C9w"
  },
  {
    "protected": "eyJhbGciOiJtZSJ9",
    "header": {
      "alg": "ES256",
      "kid": "bilbo.baggins@hobbiton.example"
    },
    "signature": "ARcVLnaJJJaUWG8fG-8t5REgAvacY8n8HjwD01muhc
dCoFZFFjFSu0Cdkn9Ybd1mi54ho0x924DUz8sK7ZKhc7AFM80b
LfTvNCrqiC3Jk12U51X3utNhODH6v7xy1Qahsn0fyb4zSAkje8b
AWz4vIfj5pCMYxxm4fgV3q72Yh5m5E"
  },
  {
    "protected": "eyJhbGciOiJtZSJ9",
    "header": {
      "alg": "ES384",
      "kid": "bilbo.baggins@hobbiton.example"
    },
    "signature": "s0h6KThzkfBBkLspWh84VsJ2FTsPpM7g1Md7p
0"
  }
]
```

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

The following sections demonstrate how to generate various JWE objects.

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

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

Figure 72: Plaintext Content

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

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

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

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

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

5.1.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

{  
    "kty": "RSA",
    "kid": "frodo.baggins@hobbiton.example",
    "use": "enc",
    "n": "maxxbsmBtdQ3CNrKvpvrUE6n91YcregDMLYNeTAWclj8NnPUP9XIYegTHVQjxKDSHP21-F5j57sppG1wgQazyhnhvXhYNvcM7TfgqKxqN_xAHx6f3yq7s-M9P9SN2wF261l6ekqqR4I00Eh9v1rypM9P141B0up9t5fSWJ5U9hnaHAllrd-osQQPjjeIldeHtwx-ZTHu3c60Pu_LJi16hnK9wbaUaM4cR5Bdp2pgbay7ASgsjCUBtYJaNIHSoHxprUDjZKUMAZV0WOKPfa60PI4oypBadamente3J3BXaSYsEZhaueTXvZB4eZ0AJiyh2e_VOIKVMsnDrJYA1VoLtGlMq",
    "e": "AQAB",
    "d": "Kn9tq0HfiTIVi8uPu5bk97NtwyHwG5dK6RE0uFd1pCjGNJ7ZlE1963R7wvboIALHmpBnNTztrfehosAniV1NCIGxAXwQS461xiDT5p4ntEPncKc5yo5jMAj7i-CL8vhpYoYW9oFVfesgMoVaPRMYT9T63hNM0aAs7USZ_hLg6Oelmy0vHT31FucjSM8hNlf40IENt43r2fspgEPQRdE6fLc90aq-qeP1GFUlimRmdnm-P8gkvN3KH1NATeGrQAqTtgz8S-3VD0FgWfQbnb1PNniuPUx0OpI9KDIlu_acc6fQg0nsNaJqXe6RESvBGP2afjHqSy_FldvWj8bQQ",
    "p": "D2WqmZ43FoTnQ8IkUj3BmKrf5Eh2mizZA5xEJ2MinUE3sdTYKSLtaEoekX9vbBzuWxHDvM6uKcjz_2InK8Z0ayJLYHLO_G21axf9-unynEpUsH7HTkYlpYazOo12gYviojxAdWNN3h1EFrjLZG5701h-a3Q1DQJ0qOJ2VFMU",
    "q": "te8LY4-W7IyaqHlEuxjMqkTAqTeRb0VQLqfLY2xINnrDwqI93_VF099aP1EisLaj2zn-6iKIE-q7tmCPOzSIFvUyfz5HRj_XY2kfixJINb9ilrHm5v5liskZpeIS-GPCHC6gRIKolq1-ldn_qxyusfWv7WAX1SVQFQk6d6Et0",
    "dp": "UyFKcL_or492vVc0PzwLSplbg4L3-25wL48mwiwbpzoYg52zXHTQmjpFA1Z8q-zf9RmgJXKdFs9rkdPtaAsL1WYdeCT5cl25Fkdq317JVRDolInX7x2Kd8h8ERCreW8_4zXiTuTL_KiXZNU51vMqjWbIw2eXT1lpsfdOrYU",
    "dq": "IEgocO-QfepdH8FwD7mUyfXdn0kXBCogChY6KuIHCg_p8Le9MbpFKEsZeAL1N1EHf3oG6G15iz_ayUI2j2IoQ28zn0Urpa9fVYnot87ACfziG7Q9hM7rIPaderxIu30tkVXAdaBau_9vs5rS-7HMtxkVRxSUvJY14TkX1HE",
    "qi": "Kc-1zQoqFa2Cr510tOVCsREkoVQaAYhQigIRGL-MzS4cMkXRm5v21XYx6r6tElAaAgjgaj1kjisGlTTTHHD8Iga6fG0BMAr5uRl1hQpQSc7G17CF1D2kJBMqTN6EshYzZfxW08mIO8M6Rzu0beL6fG9mkDcIyPrBXX2bQ_mN"
}

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

The following are generated before encrypting:

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

```
3qyTVhIWt5juq2UCpfRgpvauwB956MEJL2Rt-8qKSo
```

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--_BgLOXMozKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WC1nQePF
vG2K-pvSIwC9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4XaRvYHFj2TZ2G
Xfz8YAImcc91Tfk0WXC2F5Xbb71C1Q1DDH151t1pH77f2ff7xi5xh9oSeYrcG
TSLUeeCt36r1Kt30Sj7EyBQXozIN7IxbyhMAfgIe7Mv1rOTO1518NQqeXXW8V1
zNmoxaGMny3YnGir5wf6Qt2nBq4qDaPdnaAuUGUEce1IO1wx1Bpyf1g1fjOh
MBs9M8XL223Fg47x1GxnMdxfuY-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**

eyJhbGciOiJSU0ExIiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJtd2lkdGlvIiwiZGlzdGluZ3MiOiJyYW5jaGFyZV9sb2dvIn0= eyJhbGciOiJSU0ExIiIsInR5cCI6IkpXVCJ9.eyJlbmMiOiJpdTJhIiwiZGlzdGluZ3MiOiJyYW5jaGFyZV9sb2dvIiwiZGV2aWNlIjoiR0FUS0ZFT0FUEkRGQUIiLCJ0b2tlbiI6IjI0YjBmLTc2MmUtMTZiZS05ODViLTQ5ZmUyMTM0ZjRkIiwiY2FtcGluZ3MiOiJyYW5jaGFyZV9sb2dvIiwicGFzc3dvcmUiOiJyYW5jaGFyZV9sb2dvIiwiY29tcGxleWVyIjoiR0FUS0ZFT0FUEkRGQUIiLCJ1c2VyX2FtZSI6IjIiLCJyZWdleS1pc3MiOlwiY2FtcGxleWVyIl0sImZvcmVudF9pZCI6InNLAiwiZXhwIjoiMzA5ZDIwZmMtMDM5MC00MjU4LWE4ZjMtMDM3NjI3NjY3OWUzLzIzIiwiZXhwIjoxODQ1NzIzMDAzLCJleHAiOjE1OTI4NjY2NzMsImF1ZXJ5IjoxODQ1MTIyMzI2LCJleHAiOjE1OTI4NjY2NzMsImV4cCI6MTkxNTMwMDIzMCwiaX预算IiowiI3NkODU1ODIzOTg0NzI2IiwiZG9tYWluX3JnYWt1bmF0aW9uIjoiT0FUS0ZFT0FUEkRGQUIiLCJrdW1ibGUiOiIyMDAxMjA5NjYwMjU5IiwiYXV0aF92IjoxNjM2MjE3MTA2fQ==
```

**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_na7f8dwSfxLxYdHa2AujD67ieF7fcVbIR62JhvG24_FNVSiGc_raa0HnLQs61P2sv3Xz11p11_o5wR_RsSrzs8Z-wnI3Jvo0mkpEElnDm2zvDu_k80WzJv7eZVEqiWKdyVzFphipyQU28GLopRc2VbVbK4dQKDPjTjPPEmRqcaGeTWZVyeSUrf5k59yJZxRUsvWf6KrNtmRdZ8R4mDOjHSrM_s8uwIFcq445G8XBTkAI01zT5Cbl5qLw3sc7u_hg0yKVOiRytEAes3v2kcfLkP6nbXdc_fMmdNS-ohP78T2O6_7uInMGhFeX4ctHG7Ve1HGIt933fWDEQiq5_V9UN1rhXNryu-0fVMkZAKX3VW17lzA6BP430m
```

**Figure 79: Ciphertext, base64url-encoded**

kVvKFBXHe5mQr4lggobAuG

**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--_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WC1nQePFvG2K-pvSlWC9BR1azDrn50RcRai__3TDON395H3c2tIoUJ4JaXaYHFjZTZzG
Xfz8YAImcc91Tfk0WXC2F5Xbb71ClQ1DDH15t1pH77f2ff7xiSxh9oSeWYrcG
TSLJeeCt36r1Kt3OSj7EyBQXoZ1N71xbyhMAfgIe7Mv1rOT01518NQqeXXW8V1
zNmoaGMny3YnGir5Wf6q62nBq4qDaPdnaAuGUeGece1I01wx1BpyfgvfjOh
MBs9M8XL223Fg47x1GsxMxdfuY=4jaqVw

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

```json
{
    "recipients": [
        {
            "encrypted_key": "laLxI0j-nLH-_BgLOXMoZkxmy9gffy2gTdvqzfTihJBuuxzg0V7yk1WC1nQePPvG2K-pvSIWc9BR1azDrn50RCRai__3TDON395H3c6tiouJJ4XaRvYHFjzTZ2Gxfz8YA1mcc91Tfk0WXC2F5Xbb71C1Q1DDH151t1pH77f2ff7xiSxh9oSewYrcGLSULueeCt36r1Kt3O5j7EybQXo21N7IxbByhMAfgeIe7Mv1rOTI518NqeqXW8V1zNmoxaGMny3YnGir5Wf6Qt2nBq4qDapnauuGUGEceci1OI1wx1BpyIfgfvfjOhMBs9M8XL223Fg47x1GsMXduY-4jaqVw"
        }
    ],
    "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdg9uLmV4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMju2In0",
    "iv": "bbd5sTkYwhAIqfHsx8DayA",
    "ciphertext": "0fys_TY_na7f8dwSfXLiYdHaA2DxUjd67ieF7fcVbIR62JhJvG24_FNVSiGC_raa0HnLQs1P2sv3Xz1lpl1_o5wR_RsSrZrSZ-wnI3Jvo0mkpEEnlDmZvDu_k8OWzJv7eZVEgiWKdyVzHFPpiyQU28LQpRc2VbVbK4dQXPdNTjPPErmRqaGetW2VyeSUvf5k59yJZxRusWVFf6KrNtmRdZ8R4mDOjHSrM_s8uw1Fcq4t4r5GX8TkaI0zT5CbL5l1w3sRc7u_hg0yKVOiriRyteEAe3v2kcfLkP6nbXdc_FkmDNS-ohP78T206_7uInMGhFeX4cTHG7ve1HGI93JFwDEq515V9UN1rNhXnrYu-0fVMkZARKX3VWi1zA6BP430m",
    "tag": "kvKuFBXHe5mQr4lqgobAug"
}
```

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

```json
{
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdg9uLmV4YW1wbGU1LCljbmMiOiJJBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "laLxI0j-nLH-_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePFvG2K-pvS1wc9BRIazDrn50RcRai__3TDON395H3c62tIouJ4XaRvYHFj2T22GXfz8YAImcc91Tfk0WXC2F5Xb71C1Q1DDH15tlpH77f2ff7x1Sxh9oSewYrcGTSLUeeC36r1Kt30Sj7EyBOxOz1N7IxbymMAfgIe7Mv1rOTOI5I8NQqeXXW8V1zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuGUeceleI0Iw1BpyIfgvfjOhMBs9M8XL223Fg47x1GsmXdfyU-4jaqVw",
  "iv": "bbd5TkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSFXLiYdHaA2DxUj671eF7fcVbR62JhJvG24_FNVSiGc_raa0HnLQ6s1P2sv3Xz1lipl_o5wR_RsSzrS8Z-wnI3Jvo0mkpEE1dm2vDu_k80WzJv7eZVEqiWKdyVzFPhpPiyqUZ28GLOpRc2VbVbK4dKQPDlntJPPEmRqcaTeT2vVeSUvff5k59yJZxRvSvWFf6KrNtmRdZ8R4mD0hSrrM_s8uwIFcqt4r5GX8TKa10zT5Cbl5Lq1w3sRc7u_hg0yKVOfyTEAEs3v2kcfLkP6nbXdc_PkMdns-ohP78T206_7uInMGhFeX4ctHG7Ve1HG1T93JfWDEQ15_V9UN1rhXNrYu-0fVMk2AX3VWi71zA6BP430m",
  "tag": "kvKuFBXHe5mQr4lgobAUg"
}
```

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 shows that the RSA private key is needed to allow readers to validate the output.

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

The following are supplied before beginning the encryption process:

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

```json
{
    "kty": "RSA",
    "kid": "samwise.gamgee@hobbiton.example",
    "use": "enc",
    "n": "wbdxI55Vaan2ZXPY29Lg5hdmv2XhvqAoxukanzfzf2-5zVUxa6prHRrI4pP1Ah0qJRIrZfYtwD5mmHRG2pAH1h0ySJ9wi0BioZBL1XP2e-c-FyXJGctyOhdKQW1rnhfTM42EZ7Vv04r4ga6uxjLGwpGr2Laroh1WCpnkNrg7Ll2CuNZQBPjgXfkmIy2t1_vWgGnL22Gp1yXj5Y1Bdx3XP3eStsqo571uNFoUTU8E4qdzJ3UDItvOvPlsMw1mnJiwAsXRTbCivR4M5q2tdw-7v4wuR4779bDuJ5aMV2S66-RPcnFazW5KxtBDmFJJDGUie7Tzizj1nms0Xq_yPub_U0iWn0ec85FCf1hACpWG8schrOBENqHBODFskYpUc2L25J2A2TaPF2dA67dg1TTSC_FupfQ2kNCe1LgprKHcWVYQb86B-Horzh2qtauBzFv5tbTUb-TpckvJfNcFL1H3b8mb-H_ox35FjgBAJLKyoeqFkTpyHxd09knwgfj6V1g6UC418_T01_jMVfFTWUXhnlfh0OnzW6HSSz4lc9w0CuVzsUMv54szidQ9wfc1yf3g5qfDxDQk1s99gcdaiCAwM3yEBizuNeeCa5dartHDblxEB_HCHeSyEybhbMjgfasvK0nA2nRTyCUxhW1bsol12E",
    "e": "AQAB",
    "alg": "RSA-OAEP",
    "d": "n7zfjc3_9g59V6EoBKtayzusMM7800JQFUz2JN_Kbh8102G25zoA7T4Bxcc0xQn5oZEl5uSiCw9910c0JvXcPcmqzaJZg1nirjwRz-oBtVY7gCAWq-B3qhF3ilz1bkosrzrJHajzcY33HBhsy4_werrXg4MDN4E4Hyojy68txCt2L3QrUXOc5fT3vXvMoixlsGtVqNqRtxUEwiewfmmrveEogLx9E-A-KmgAjtIlxSxQigXQWQX1g7v_mV_HrzYumIycHKhkRvp9E700ok876Dhkn084vU0zLwAl0IUx98mkogwcv58A_Y21BYvX1__s1lpPsEqbbh-nqj1hlf1og1dNfdflxncLw7PcztLnImZayeCWAG7z7IFv-R9nflfJy9j26r7r-MSH9sgbuiziHNzgG3d_jfRluuHMA0184ffKl1bocqN1JWvPHvNH20o1ydF-1lL0IqnyUSePF6X3a2SodkgBriquE6EvLusYIDpJq3jD1sogL8mnlLoomgjIxUwL_G2wEQG0u28glpIymz-90q0UNyheF1uhsR8aJaQWAIFImWHS5W_iQT917-yirr2FwOh0IUsGnA7aOQzZfFJry6-z_tYKubG00-28S_aWvjyUc-Alp8AuyKjBz-7CW3HF2Wgk448jI-t6rswjwLmnhsPbs0c9WsWqRzI-K8gE",
    "p": "7_2v3OQz2Z1PfcHyf1aLBQ3X85Es4hCdwCkhdDeltaUXGy9y19etKghvM4hrKp0b0k1kyULFmxIkCdtpi-zyLCAYdRXrAK3PtSbztLDXZ9n1sya_QzWkpXbIrfTFyJFvKxUdmz94pHUhPGFj7n6NWyfpiKSHWFE1zd_Ac3mY4J6J61Y2LrnnEvwAAGNw53p07Db8yD_92pa97vqc2Odgtbyh9q6uma-"
Figure 84: RSA 4096-Bit Key

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

5.2.2. Generated Factors

The following are generated before encrypting:

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

5.2.3. Initialization Vector;
this example uses the Initialization Vector from Figure 86.
Figure 85: Content Encryption Key, base64url-encoded

- nBoKLH0YkLZPSI9

Figure 86: Initialization Vector, base64url-encoded

5.2.3. Encrypting the Key

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

t99rwrBTbT17l13M8fuJELi226HEB7IchCnJnuh71Ciud48Lxe01RdtFF4nzQi beY015s_PjRsAXZwSxDePz9hk-BsTsTBqc2USPoWjC9NhNupNnu9uHIVftDyu cv16hvAeZ6OgNhNV4v1zx2k701D89mAzfw_-kT3tkuropDU-CpBENfIHX1Q58 -Aad3FzMo3Fn9uEsP2yXakLXYa15BUXQsupM4A1G04_148d7V3u9h8Gkg8Bpx KdUV9ScfJtQcYm6eJEBe3aSwIAk4T3-dwWpuB0hROQXBosJzS1asnuHtVMt2pK IIfux5BC6huIVy7kv7w7aiUrpYm_3H4zYyyMeq5pGQFM2k8zp087TR1Z7 pZfPyD5XZyS0CFKkMozT_qiwZTSz4duYnt8hS4Z9sGthXn9uDgd6wycMgnQ f0Ts_lcyCTWmx-aqWVDKhjYNrf03NiwRtb5BE-t0dFwCASQJ3uAgPGr02WBe3 8UjQb01vXn1spvyvY3ZF6C7W0VjaT7A8DRn6MC6t-xMmMxCO7G7S2rscw51QU 06MvZ1poUt0UvfKBa03cxA_n1B1hLMjY2k0QtmDmpDPr6Cbo8aK0nx6ASE5 Jx9paBpnNmoOOkH35j_q1rQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR s

Figure 87: Encrypted Key, base64url-encoded

5.2.4. Encrypting the Content

The following is generated before encrypting the Plaintext:

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

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

Figure 88: JWE Protected Header JSON

eyJhbGciOiJSU0EiT0FFUCIiImtpZCI6InNhbxzpc2UU2Z2VlQGhvYmJpdG 9uLmV4YW1wbGUcalculatorwJlbnMioiJBMjU2R0NNIn0

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_t2m1GXklSggBdpACm6UJuJowOHC5ytjqYgR L-I-soF1wqMUF4UgLd1EoGNw6vGW-xyM011LyrrXfVzIIaRdhYtEMRByBWbEw P7ua1DRfvaOjg2v6Iba3brcAM64d8p51hhNcizPersuhw5f-pGYZsevA-TUaL8 iWnctc-sSw7ySqmRkfhDjwbt0Fz6kFovEgj64X1l5s7E6GLp5FnbYGLa1QUIiML 7Cc2Gxgv17zgWo0YIEc7aCf1LG1-8BboVWFdZK1K9vNoycrYHumwrzKluLWElbSV 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:

eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpct2Uz2ZV1QGhvYmJpdG9uLmV4YW1wbGUICjJlbmMiOiJbMjU2R0NNIn0.

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

```json
{
  "recipients": [
    {
      "encrypted_key": "rT99rwrBTbTI71JIM8fU3El17226HEB7IchCxn
h71Ciud48Lxeol1rdtFF4nzQibeYo15S_PJsaAXZwSXtDePz9hk-Bb
tSTBqC2UsPOdwrjC9NhNupNN9uHIVFtDyucvI6hvALeZ60GnHV4
v1zx2k7qID89mAzfw_-_kT3tkuorpDU-CpBENfIHXI1Q58-Aad3Fz
uo3Fn9buEP2yXakLXYa15BUXQsupM4A1Gd4_4Hbd7V3u9h8Gk8B
px1KdU9VScfJQTcYm6eJEBz3aSwIAk4T3-dwWpuB0hRQXBoszS1
asnuHtVMt2pIIIfx5BC6huIvMy7kzV7W7aIUrEpYm_3H4zYvMeq
5pGqFmW2k8zp087TR1x7z2fPYDSXZySOcfKkKMoZT_qiCWzTSz
4duYnt8hS4z9sGthXn9uDqd6wycMgnQfoTS_lycTwM-yaqWVD Kh
jYNrf03NiwrRtb5BE-tOdfWcASQj3uuAgPrO2AWBe38UjQo1lvXn
1SppvYZ3WFc7wOjYAaTa7A8Rn6MC6T-xDmMux3C6G7S2rcsw51QQU
06MvZT1FoOt0UvfKBa03cxA_nIBh1MjY2KOTxQMmpDTPr6Cbo8a
KaOnx6ASE5Jx9paBpnNmo0KH35j_Qlq0hDWUN6A2Gg8iFayJ69xD
EdHAVCGRznN3woEI2ozDRs"
    }
  ],
  "protected": "eyJhbGciOiJSU0EtT0FFU1MxIjoi"।l
  "iv": "-nBoKLH0YkLZPS19",
  "ciphertext": "o4K2cnGN8rSSw3IDo1YuSkqesS_t2m1Gx1LsgGdpBcAm
UJuJcowOC5ytjQyRlx-J-soFlwqUMU4qGRWeaGNNw6vGW-xvX
rXfVzI1aRdh4tEMRBvWbEw7u1DRfavOjgZv6I3a3brcAM64d8p1h
hNcizPersuwh5f-pGYzzsev-TUA1L8wNCct-cSWy7SqmRkhfDjwbx0f
6kFovEgj64X1I5s7E6GlP5fnbYGLA1QjML7cc2GxgyV7zqWo0YIEc7a
Cf1LG1-8BboVWfdZKLR8vNoycryYMuvzKluRWebSVMaPp0s1Y2n525Bx
DfwAFvUFQxMF56vn4B9QMwWAbnypNimbM8zV0w",
  "tag": "$CGIgJxhBII3IFVdPa1HHAtA"
}
```

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

```json
{
  "protected": "eyJhbGciOiJSU0EtT0FFU01JUkVIRU9MRSIsImtpZCI6IjBOb3ZlZ29vZ2ljLmJhY2Nob3IgMjIiLCJoZWFyZ3JpcHQiOjF9",
  "encrypted_key": "rT99rwrBTbTI7IJM8fU3Ei17226HEB7IqCwxNuh71C
  iud4Flxoe0fDrdTF4nzQ1beYo155_PJsAAXwzSxtDePz9hk-BBtsTBqC2U
aPoDwJc9NhNpuNu9uH1VfDyucV76hVAleZ6QnhNV4v1zx2k70I89
mAzfW-kt3tkuorpDU-CpBENfIHX1Q58-Aaad3FzMu03F9buEP2yXakL
XYa15BUXQsupM4A1GD4_H4BD7V3u9h8Gkg8BpxKdUV9ScfJQTCy6m6E
BZ3aSwIAK4T3-dwWpuBOhROQXBosJ8sIasnuHtVmt2pKlIfux5BC6huI
vmY7kw7V7aUIrpYm_3H4zYyyMeq5pGqFmW2k8zoP0878TR1z7pZfYPD
SXZyS0CfKkKMoZt_qiCwZTSz4duYnt8hs4Z9gsGthXn9uDq6wycMqnQ
fOTs_lycTWMvY-aqWVDkhjYNYf03NiWTb5BE-tOdFwCASQ3juuAgPGr0
ZAWBe38UjQbo1LvXn1spyvY23WFC7WOJYaTa7A0DRn6MC6T-XdmXucG0
S2rscw51QQU6MvZT1FoT0UvfKBAo3cxA_nIBIHLMjY2kOTxQMmpDP
Tr6Cbo8aKaOnnxASE5EJx9paBpN00KH35j_Q1rQhDWUN6A2G8iFayJ
69xDeEdHAVCGRzN3woEI2ozDRe",
  "iv": "-nBoKLHYoKLZPS9I",
  "ciphertext": "o4k2cngN8rSSw31DolYuySkqes_t2m1GXk1SggBdpACm6
UJuJowOHC5ytjygYqRI-I-soPlwqMf4UgRWWeaOGNw6vGW-xyM011TXY
xrXfVzIaRadhtEMRbWbBwEw7ua1DRfvaOjgjV6Ia3brcAM64d8p51h
hNcizPersuw5-fpGyVseva-TUA8L8iWncctc-sSwy7SqmRkhfDjwb0rFZ
6kFovEgj64XI15sTEGlp5fnbYGa1QUiML7C2Gxgv7qzW00YECaCf1LGl-8BfoWFDdKLX9vNocyrYHumwzKlUAWEvSvMaP0s1Y2n525Dx
DfwaVFUFxQxM56vn4B9QMnPAbnypNimbM8zV0w",
  "tag": "UCG1qJXhBI3IFVdPaIHhVA"
}
```

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-AESKW" (PBES2 Password-based Encryption using HMAC-SHA-512 and AES-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 JWK Set for the plaintext instead of the Plaintext from Figure 72.
Note that if password-based encryption is used for multiple recipients, it is expected that each recipient use different values for the PBES2 parameters "p2s" and "p2c".

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

5.3.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the Plaintext from Figure 95 (NOTE: All whitespace was added for readability).

- Password; this example uses the password from Figure 96 -- with the sequence "\xe2\x80\x93" replaced with (U+2013 EN DASH).

- "alg" parameter of "PBES2-HS512+A256KW".

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

```
{  
  "keys": [  
    {  
      "kty": "oct",  
      "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",  
      "use": "enc",  
      "alg": "A128GCM",  
      "k": "Xct0hJAKa-pD9Lh7ZgW_2A"
    },  
    {  
      "kty": "oct",  
      "kid": "81b20965-8332-43d9-a468-82160ad91ac8",  
      "use": "enc",  
      "alg": "A128KW",  
      "k": "GZy6sIz6w19NJ0KB-jn_mvQ"
    },  
    {  
      "kty": "oct",  
      "kid": "18ec08e1-bfa9-4d95-b205-2b4d1d4321d",  
      "use": "enc",  
      "alg": "A256GCMKW",  
      "k": "qC57I_uXcm7Nm3K-ct4GFjx8tMlU8CZ0NLvbdQst1S8"
    }
  ]
}
```

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.

uwsjJXaBK407Qaf0_zpcpmr1Cs0CC50hIUEyGNEt3m0

Figure 97: Content Encryption Key, base64url-encoded

VBiczVHNoLiR3F4V82uoTQ

Figure 98: Initialization Vector, base64url-encoded

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.

8Q1SzinasR3xchYz6ZZcHA

Figure 99: Salt Input, base64url-encoded

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:

d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6r1UtFPWdgURtmeDV1g

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.

```
{
  "alg": "PBES2-HS512+A256KW",
  "p2s": "SQ1SzimasR3xchYz62ZcHA",
  "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:

eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6N1paY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbnMiOiJBTMI4Q0JDLUhTMjU2In0.

d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdgtURtmeDV1g.

VB1c2zVHNoLiR3F4V82uoTQ.

231-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYSvTHAJLZ2nsnGIX86vMXqIl6IRsfywCFzLxEcZBRnTvG3nhzPk0GDD7FMyXuHpDjEYCNA_XOmgg8yZR9oyj061
TF6s14q9FF22EhzFQzLCOO_6h5EVg3vR75_hkBsnuoqoM3dwejXBlIodN84PeqMb
6asmas_dpSsz7H10f5ni9xIz4244iVb1YL1ldF6exVmL93R3fOoOJbmk2GBQZL
SEG11v2cQsBgeprArsaQ7Bq99tT80oH8ItBjvO8AtzXXFsxx9kKvC982KLKd
PQMT1VJKqtvV4RvuLeVpBZXbnZrtVISOgyg6AiuaS-rCrcD_ePOGSuxvgtrok
AKYPgmxUeRdJFJwafkYEkiiDCV9vWGAl1DH2xTafhJwcmwywyzi4BqRpmdn_N-zl5tuJYyuvKk5ivhbsV_k1hJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdT8V
3kobXZ77ulMwDs4p.

0HlwodAhOCILG5SQ2LQ9dg

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

```json
{
    "recipients": [ {
        "encrypted_key": "d3qNhUWFqheyPPp4H8sjOWsDYajoej4c5Je6r1UtFPWdgtURtmeDV1g"
    }],
    "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2Si1cILCJwMnMiOiI4UTTFemluYXNSM3hjaF6N1paY0hBIiwicDJjIo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbmMiOiJBMTHMjU2In0",
    "iv": "vBiCzVHNoLi3F4V82uoTQ",
    "ciphertext": "23i-Tb1AV4n0WKVSgcQrdg6GRqsUKxjrhuHXYsTHAJLZ2nsnGIX86vMXqi61RsfywCRFzLxEcZBRnTvG3nhzPki0GDD7FMyXhUhpDjEYCNAXOmzg8yZR9oyjo61TF6si4q9FZ2EhzgFCuLO_6h5EVq3vR75_hkBsnuoqoM3wejXbtIoD84PeqMb6asmas_dpSsz7H10fC5ni9x1z424givB1YLldF6exVmL93R3foOJbmk2GBQZL_SEG1lv2cQsBeprARsaQ7Bq9rt80coH8ItBjgv08A7zXFFsx9qKvC982KLKdFQMT1VJKkqtV4Ru5LEVpBZXbnZrtViSOyyg6AiwaS-rCrcD_ePOGSuxvgtrokAKYPqmXUeRdjFJwafkYEkiuDCV9vWGAi1DH2xTaTfJwcmwIyzi4BqRpmdn_N-z15tuJYyuvKhjKv6ihbsv_k1hJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V3kobXZ77ulMwDs4p",
    "tag": "0HlwodAhOCILG5SSQ2LQ9d9"
}
```

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

```json
{
  "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU21ciLCJwMiOiI4UTFTemluYXNSM3hjaFl6NIpaYOhB1wicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2zpb24iLCJlbmMiOi1JBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "d3qNhUWfqheyPp4H8sj0WsDYaacej45Je61uTFPWdgtURtm8DV1g",
  "iv": "VBiCzVHNoLiR3F4V82uoTQ",
  "ciphertext": "23i-Tb1AV4n0WKVSScQrdg6GRqsUtxrhuXYSyTHAJLZ2nsgIIXV8zvMxQxI6IRsRfzwCRZ5iXEcZ780nTvg3nHzvp0GD7FMyxhUHPDjEYCNAXQmz8yZy9oym61TF6si4qFU2EhzgFQCL0_6h5Evg3v9R75_hkBSnuqoM3dwejXBlTiodN84PegMb6asmas_dpSsz7H10fc5ni9Iz424ivB1YLldF6exVml93R3fo0Qbmk2GBQZLESG11vz2QBsBgeprAESAQ7Bq9tT80coH8ItBjgV08AxXF85x9Kvc982KLdPQTM1VJKkqtV4Ru5LEVPBZXBnZrtViSOeyg6AiwaSr-rCrCdePOGUSxvgtrokAKYPmqXUERdfJwafKYEkiuDCV9vWGAI1DHk2TaWfJyeywiyi4BqRpmnN-z15tuJYYuvKhhXv6ihbV_k1hJGPGAXJ6wUpmw4CPTQ2izEm07TuSE0oMKdTw8V3kobXZ7u1MwDs4",
  "tag": "0HlwodAhOCILG5SQ2LQ9dg"
}
```

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.
o "alg" parameter of "ECDH-ES+A128KW".

o "enc" parameter of "A128GCM".

```
{
  "kty": "EC",
  "kid": "peregrin.took@tuckborough.example",
  "use": "enc",
  "crv": "P-384",
  "x": "YU4rRUzdMvqmRtWOs2OpDE_T5fsNIOdcG8G5FWPrTPMypxzsSQGxFL1v",
  "y": "A8-yxCHxk5Fz3hKZfI1jUYMjUhsEvez9THuwFjH2sCNdtksRU77D5-SkgFL1ETP",
  "d": "iRx2pk7wW-GqJKhEcEKFQb2EFyYc07RugmaW3mRRQVAOUpiommTOIdn
  YK2xIDh-j"
}
```

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

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

5.4.2. Generated Factors

The following are generated before encrypting:

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

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

Nou2ueK1P70ZXD5b9UrRwg

Figure 109: Content Encryption Key, base64url-encoded

mH-G2zVqgztUtNw_

Figure 110: Initialization Vector, base64url-encoded

5.4.3. Encrypting the Key

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

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

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

produces the following JWE Encrypted Key:

0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2

Figure 112: Encrypted Key, base64url-encoded

5.4.4. Encrypting the Content

The following is generated before encrypting the content:

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

```
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx510xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iuEDsQ6wNdNg3",
    "y": "sp3p5SGrZhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEwZ0yQT-A-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:

eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guZXhhbXBsZSIsmVwayI6eyJrdHkiOiJFQyIsImNydiI6I1At
Mzg0IiwieCI6InVcbzRsFBJNntianger1bDB4b3dy2F9vWXpCbf6fLl1sNH
hBRkZrY1lpV21dEVNml1URUzUTZ3TmR0ZzMiLCJ5Ijoic3AzcDVTR2haVkm
YmFydW1JLW5U5UyTw84S3BvWXJGRHi1eVBOVnRXdRXdaT3LRVEEtSmRhWT
h0YjdFMCJ9LCJlbnMiOiJBMTI4R0NNIn0.

0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2
.mH-G2zVqgztUtnW_
..tkZuO09h950gHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID71pnAi_cP
WJRCjSpAaUZ5dOR3Sp7QuEkMkx8-3RCmhsMyMsXaEwDdXta9Mn5B7cCB0JKB0
IgEnj_qfo1hi1-i-uEkUpOZ8aLTZGhfpl05jMwbbKkTe2yK3mjF6SBAsgicQDVCkc
Y9BLluux1RmC3ORXaM0jaHPB93yccDSGg9BWMrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXiuErhWgeP-nkUZTPU9EoGSPj6fas-bSz87RCFzxZdj_iVyc6Qwcau
07WNhjzJEPc4jVntRJ6K5NgPq5p9913Z4080Uqj4icYeTbS6vTP1Q
..WuGzxcncreYjpHGjala7EBg

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

```
{
  "recipients": [
    {
      "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
    }
  ],
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6IiIsImVwayI6Ijg5Z29uZyIiLCJ0bFlmciI6IiJ9",
  "iv": "mH-G2zVqgzUttnW_",
  "ciphertext": "tkZuO09h950qHJmkkrlBisku8rGf6nzVxhRM3sVOhXgZ5NJ76oID7lpnAI-cPWJRCJsSpAaUZ5dOR3Spy7QuEkMx8-3RCMhSYMzszXaEwDdxta9Mn5b7cB0JK801gEnj_qf01hIi-uEkJpU28aL TZGhp105jMwbKkTe2yK3mjF6SBAsxicQDVcKcY9Blluzx1RmC30RXaM0JahPB93YcdS4gjgPBMVnU1ErkycMcMoT-wtCex3w03XdLkJXlUerz2hWgeP-nkUZTPU9EoGSpj6fAS-bsz87RCPxZdj_iVyc6QWcqAn07WnhjzJEPc4jVntRJ6K53NgPQP9132Q08OUnj4iOYeZbS6vTP1Q",
  "tag": "WuGzxmcreYjpHGJoal7EBg"
}
```

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

```
{
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVVyIsImtpZCI6InBlcm
  VncmUlunRvb2tAdHVja2Jvcm91Z2gu2XhhbXBsZISIsImVwayI6eyJrdH
  kiOiJFQyIsImNydiI6IiAtMzg0Iiwic16InVCbzRrSFB3Nmtiang1bD
  B4b3dYZ9vWXpCbWF6LUdLR1p1NHzBRkZrY1lpV2d1EVLNm11RURzUT
  Z3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkJMy2mFYdW1JLUU5SlUyTW84S3
  BvWXJGRHiIeVBOVnRXNFBnRXdaT31RVEEeSmRhWTh0YjdFMCJ9LCJ1bm
  MiOiJBMTI4R0NNIn0",
  "encrypted_key": "0DJjBXri_kBc46IkU5_Jk9BqaQeHdv2",
  "iv": "mH-G2zVqgztUtnW_",
  "ciphertext": "tkZuOO9h95OgHJmkkrlBisku8rGf6nzVhxRM3sVOhXgz
  5NJ76oID71pnAi_CPWJRCjSpAaUZ5dOR3Spy7QuEkmmKx8-3RCMhSYMzs
  XaEwDdxta9Mn5b7CBoJKB01gEnj_qfo1hIi-uEkUpO28aLTZGFp105
  jMwbKkTe2yK3mjF6SBAsgicQDVcky9BLuzx1RmC3ORXaM0JaHPB93Y
  cdSDGgpgBWMrNU1ErkjcMqMoT_wtCex3w03XdlkjXIuErzHqgP-nkU
  ZTPU9EoGSP+j6fAS-bSz87RCPrxZdj_iVyc6QWcqaU07WNhjzJEPc4jVn
  tRj6K5NgPQ5sp991324080Uqj4icYezbS6vTP1Q",
  "tag": "WuGzxmcreYjpHGJoal7EBq"
}
```

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": "Ze2loSV3wrrocKUN_4zhwGhCgo3Xhu1td4QjeQ5wIVR0",  
  "y": "HlLtcXARY_f55A3fnzQbPcm6hgr34Mzp8p-nuzQCE0Zw",  
  "d": "r_kHyZ-a06rnxM3yESK84r1otSg-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.

yc9N8v5sYv3iGQT926IUg

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_bAWGH1hg0TpjqVsPlrXWQu_vwVOHHtNkdYoA",
    "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_bAWGH1hg0TpjqVsPlrXWQu_vwVOHHtNkdYoA",
        "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs"
    },
    "enc": "A128CBC-HS256"
}
```

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

- CEK (Figure 123);
- Initialization Vector (Figure 121); and
- JWE Protected Header (Figure 125) as authenticated data produces the following:
  - Ciphertext from Figure 126.
  - Authentication Tag from Figure 127.

5.5.5.  Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJFQ0RILUVTIlwia2lkIjoibWVyaWFkb2MuYnJhbmrR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6I0I1YiIucGc6c2h5bWV5
Ino0ODBcIiwiZXBrIjppcCwiXzV1bGQ6MDAiLCJ1c2VyX2Z1bmN0b3J5
IjppZCwiY2xvc2QiO<n>uIz</n>bWVyaWFkb25jcy50YWJsZSoi
.
yc9N8v5sYyv3iGQT9261Ug
.
Figure 128: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

{  "protected": "eyJhbGciOiJFQ0RILUVTIlwia2lkIjoibWVyaWFkb2MuYn
   JhbmrR5YnVja0BidWNrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6I0I1YiIucGc6c2h5bWV5
   Ino0ODBcIiwiZXBrIjppcCwiXzV1bGQ6MDAiLCJ1c2VyX2Z1bmN0b3J5
   IjppZCwiY2xvc2QiO<n>uIz</n>bWVyaWFkb25jcy50YWJsZSoi
   "iv": "yc9N8v5sYyv3iGQT9261Ug",
   "ciphertext": "eyJhbGciOiJFQ0RILUVTIlwia2lkIjoibWVyaWFkb2MuYn
   JhbmrR5YnVja0BidWNrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6I0I1YiIucGc6c2h5bWV5
   Ino0ODBcIiwiZXBrIjppcCwiXzV1bGQ6MDAiLCJ1c2VyX2Z1bmN0b3J5
   IjppZCwiY2xvc2QiO<n>uIz</n>bWVyaWFkb25jcy50YWJsZSoi
   "tag": "WCCKNa-x4BeB9hIDIFuWug"
}

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:

o Plaintext content; this example uses the content from Figure 72.

o AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 130.

o "alg" parameter of "dir".

o "enc" parameter of "A128GCM".

```json
{
    "kty": "oct",
    "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
    "use": "enc",
    "alg": "A128GCM",
    "k": "XctOhJAkA-pD9Lh72gW_2A"
}
```

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

5.6.2. Generated Factors

The following is generated before encrypting:

o 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.

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

Figure 132: JWE Protected Header JSON

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

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

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

```
JW_i_f52hww_ELQPgaYyeAB6HYGcR55919TYnSovc23XJoBcw29rHP8yZOZG7Y
hLpTlbjFuvZPjQs-m0IFTvCkZXdH_lr_Frdyt9HRUYkshtrMmIUAYmUnd9zM
DB2n0cRDlHAFvFELUxkUwVAE7_YGRPdcqMyiBoCO-FbdE-Nceb4h3-FtBP-c_
BIwCPTj9o0SbdcdREEMJMyZH8ySwMVilgPD9yxi-aQpGbSv_F9N4IZAxcj5
g-NJsuPbjk29-s7LJAGb15wEBtxphVCgyy53CoIKLHeJHXex45Uz9aKZRSIn
ZI-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp
```

Figure 134: Ciphertext, base64url-encoded

```
vbb32Xv1lea20tmHA dccRQ
```

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:

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTEzLTQ1Y2YtODY3Mi02MThiNDI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0.

refa467QzzKx6QAB

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

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTEzLTQ1Y2tODY3M0ZMTdiWN1ONT1OM2EiLCJjbmMiOiJBMlI4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcR55919TYnSovc23XJ
  oBcW29rHP8yZOZG7YhLpTlbjFuvZPjQS-m0IFtvCxcKXzdH_Lr_FrdYt9
  HRUYkshtrMmIUAyGmUnd9zMDB2n0cRDIHAFVeJUDxkUwVAE7_YGRPdc
  qMyiBoCO-FbdE-Nceb4h3-FtBP-c_BlwCPTjb9o0SbdcdREEMJMyZBH8
  ySWMVilgPD9yxi-aQpgBsv_F9N4IZAxscc5g-NJsUPbjsk29-s7LJAGbl
  5wEBtxXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRISInZI-wjsY0yu3cT4_
  aQ3i1o-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp",
  "tag": "vbb32Xv1lea20tmHAdccRQ"
}
```

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".
```json
{
  "kty": "oct",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "use": "enc",
  "alg": "A256GCMKW",
  "k": "qC57l_uxcm7Nm3K-ct4GFjx8tM1U8CZOuNLbvdQstiS8"
}
```

Figure 138: AES 256-Bit Key

5.7.2. Generated Factors

The following are generated before encrypting:

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

Figure 139: Content Encryption Key, base64url-encoded

```
gz6NjyEFNm_vm8Gj6FWoFQ
```

Figure 140: Initialization Vector, base64url-encoded

5.7.3. Encrypting the Key

The following is generated before encrypting the CEK:

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

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

- AES symmetric key (Figure 138);
- Initialization Vector (Figure 141); and
- The empty string as authenticated data produces the following:
  - Encrypted Key from Figure 142.
  - Authentication Tag from Figure 143.

```
1Jf3HbOApxMEBkCM0oTnnABxs_CvTWUmZQ2E1LlvYNok
```

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.
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:

```plaintext
eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYy1ZDFkNDMwMTlcIiwicm9vdFwvcm9vdElkIiwibG9jZDEiOiJrZlBkdVZRM1QzSDZ2bmV3IiwiaWQiOiJLa1lUMEdYXzJqSGxmcU5fIiwiZW50aWZmZCI6IkJ9.
LjF3HbOApxMEBkCMOoTnnABxs_CvTWUmQ2ElLvYNok.
gz6NjyEFNm_vm8Gj6FwFCQ
.Jf5p9-ZhJ1Jy_IQ_byKFm10Ro7w7G1QiaZpI80aiVgD8EeqoDZHtFKFBupS8iae eVi0gMqWmsuJKuoVgzR3YfzoMd3GxE3VxNhzWyWtZKX0qxRdy6HglvqoGNb2Cz LjqcpDif8q2_62EVAbr2uSc2oaxFmU9QHLCqAHxy51449xkj7ewzZaGV3eFq hpco8oDiJxA57tkp3h2cajRfFgymuxUbwLqaeNQaJtvJmSMFupEOSAzw9Hde b6yhdtnCMru-kqt05Dec41T2OMZKpnxc_F1_4yDjFzqbi5C1DsmA-psB2k0Jt jxAj4UPI61oONK7zzFIu4gBfjJCndsZfdvG7h8wGj9v98QhrKEnR7xKZ3KC0_qR 1B-gxPNk3xWU.
.DKW7jrb4WaRSNfbXVPfT5g
```

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

```json
{
  "recipients": [
    {
      "encrypted_key": "lJf3HbOApxmEBkCMoOtTnnABxs_CvTWUUmZQ2E1L
vYNok"
    }
  ],
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMDM1
iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWMzLCJ0YyI6IjI2MzQ2MDk0OTA
NzI3In0iLCJ1c2VyX25ldXNlciI6IkJhY2JjZjM2NzRiYzgwZjg0ZmU2
NjQ1NzIiLCJyb2xlIjoiQTEyOENCQy1IUzI1NiJ9",
  "iv": "gz6NjyEFN_mvm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFm10Ro7w7GIQiaZp18oaiVgD8E
goDZhHyFKFBupS8iaEeVlGqWmsuJKuovGzr3YfzoMd3GxEm3VxNhzWyW
tZKx0qxKdy6HhVqGbNzCzlJqcpDf8q2_62EVAbr2uSc2oaxFmFuIQ
HLcqAhxy51449xkjz7ewzZaGV3eFghpc08o4DijXaG5_7kp3h2cajRfd
gymuxUbwGdlaeNQaJtvJmSMFuEOSAzw9Hdeb6yhdtynCRmu-kgtO5Dec
41T2OMZKpnxc_F1_4yDJFcb5CidxMa-psB2kOjtjxAj4UPI61oONK7z
zFIu4gBfjJCndsZfdvG7h8wGjV98QhrKeR7xKZ3Kr0_qR1B-gxpnK3
xWU",
  "tag": "DKW7jr4WaRSMfbXVP1T5g"
}
```

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

```
{
    "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJpdjI6IktwVWQwR1hfMm
pIbGztX18iLCJraWQiOiJrZ1BkdVZRM1QzSDZ2bmV3dCoa3N3iwiZW5jIjoiQTEyOENCQy1IUzI1NiI",
    "encrypted_key": "lJf3HbOApxMEBkCMo0nABxs_CvTWUmZQ2E1LvYN0k",
    "iv": "gz6NjyEFN_vm8Gj6FwoFQ",
    "ciphertext": "Jf5p9-ZhJy_Jy_JQ_byKFmI0Ro7w7GIQ2pZ18OaiVgD8E
qoD2HyFKFupS8iaEeVIGmQwmsuJKuoVgzR3YfzoMd3GxEm3VxNhzWYw
7ZX0gkJy6H9lvqGNgBZcljWqcdPiF8q2_62EVbbr2uSc2oaxFmF0IQ
HLcqAHxy51449xkjZ7ewzaGV3eFqhpco8o4DijXaG5_7kp3h2cajRfD
gymuxUBwQlgaeNQaJtymSMFuEOSAzw9Hdeb6yhdTynCRmu-kqtO5Dec
41T2OM2Knxc_Fl_4yDFJFpq5CidSmA-psB2kOJtjxAj4UPI61oONK7Z
zFIu4gFfjJCnadsZfdvG7h8wGjV98QhrKEnR7xKZ3KCr0_qR1B-gxpNk3
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".
{  "kty": "oct",  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",  "use": "enc",  "alg": "A128KW",  "k": "GZy6sI26wl9NJOKB-jnmVQ"
}

Figure 151: AES 128-Bit Key

5.8.2. Generated Factors

The following are generated before encrypting:

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

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

aY5_Ghmk9KxWPBLu_glxlw

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:

- 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
.Qx0pmsDa8Knc9Jo
AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvrRkH0tPlbTdhtFJgJxeVmjkJLD6
1AlhnWGetdg11c9ADsnWqL56NywxSYjU1ZEHcGkd3EkU0vjHi9gTlb90qSYFfe
F0JwkcTtjbYKcSiNJKQcIplyeM030muiYSoYJYVSpf7ej6zaYcMvJWdxDF18RE
wOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGH1Qeg7gDT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJffqfVETUxLAF-KjcBTS5dNy6egwkYtOt8EIHK-oEsKYZfRa
a8Z7MOZ7UGxGIvEmxrGCFeJa14slv2-gaqK0kETHkaSqdYw0FkQZ
.ER7MWZ1FBI_NKnv7Zb1Lw
```

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

```json
{
  "recipients": [
    {
      "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIqX"
    }
  ],
  "protected": "eyJhbGciOiJBMTEI4SlciLCJraWQiOiI4MjI4MyIjMCkzNjQ0MzMyLTQzZDktYTQ2OC04MjE2MjFkOTFhYzgiLCJ1bmlQOiIJBMTI4R0NNIn0",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkH0tP1bTdhTFJgJxeVmJkLD61AhnWdgdg11c9AdsnWgL56NyxwSYjU1ZEHCgKd3EkU0vJhIl9gTl9b90qSYFfF0LwkcTjbyKCIiNJQkIplYeM03Omu1YSoYJVSgf6j6zaYcV3WwdxDF18REoHnINmk2X1d2JKxg6BR53TSFkY7T
  PWVLuq-1GwTH1Qeg7gDT6xW0JqHDPn_H-puQsmthc9290o0jMfqqFVe
  TUXLAF-KjcBTS5dNv6gegwktYo08EIHK-oESkYtZRaaz87M0Z7UGxGIMvEmxrGCPeJa14slv2-gaqK0EThkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI__NKvn72b1Lw"
}
```

Figure 160: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJBMTEI4SlciLCJraWQiOiI4MjI4MyIjMCkzNjQ0MzMyLTQzZDktYTQ2OC04MjE2MjFkOTFhYzgiLCJ1bmlQOiIJBMTI4R0NNIn0",
  "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIqX",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QA4TkvRkH0tP1bTdhTFJgJxeVmJkLD61AhnWdgdg11c9AdsnWgL56NyxwSYjU1ZEHCgKd3EkU0vJhIl9gTl9b90qSYFfF0LwkcTjbyKCIiNJQkIplYeM03Omu1YSoYJVSgf6j6zaYcV3WwdxDF18REoHnINmk2X1d2JKxg6BR53TSFkY7T
  PWVLuq-1GwTH1Qeg7gDT6xW0JqHDPn_H-puQsmthc9290o0jMfqqFVe
  TUXLAF-KjcBTS5dNv6gegwktYo08EIHK-oESkYtZRaaz87M0Z7UGxGIMvEmxrGCPeJa14slv2-gaqK0EThkaSqdYw0FkQZF",
  "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.

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:

5vUT2WOtQxKcekJzVQwkGgzlFDwPi

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

eyJhbGciOiJBMTE4SiJcLCJraWQiOiI4MW1iI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFk0TFhYzgiLCJlbmMiOiJBMTE4R0NNIiwieiwimlwIjoiREVGIN0

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.

Figure 168: Ciphertext, base64url-encoded

%HbDtOsdailOYziSx25KEeTxmwnh8L8jKMFNC1k3zmMI6VB8hry57tDZ6ljXyezSPt0fdLVf6Jf5y5-JaCap_JQ8cb5opbmtT60uWGl8blyiMqmOn9j--XhhLYg0m-BHaqfD05iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbryBKhpYA7qi3AyijncJ7BP9rr3U8kxExCpG3mK420Tj0w%

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:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04NjE2MGFkOTFhYzgiLCJibmMiOiJiJBMbMTI4R0NNIiwimiwiioIREVGIn0
 .
5vUT2W0tQxKwcekM_IzVQwkGgz1FDwPi
 .
p9pUq6XHY0jfEZI1
 .
HbDtOsdai0oYziSx25KEeTwxmwhh8L8jKMFNc1k3zmMI6VB8hra57tD261jXyez
SPt0fdLVfe6Jf5y5-JaCap_QoBc5opbmT60uWgml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMjedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7q13AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
 .
VILuUwuIxaLVmh5X-T7kmA

Figure 170: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
    "recipients": [
        {
            "encrypted_key": "5vUT2W0tQxKwcekM_IzVQwkGgz1FDwPi"
        }
    ],
    "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04NjE2MGFkOTFhYzgiLCJibmMiOiJiJBMbMTI4R0NNIiwimiwiioIREVGIn0",
    "iv": "p9pUq6XHY0jfEZI1",
    "ciphertext": "HbDtOsdai0oYziSx25KEeTwxmwhh8L8jKMFNc1k3zmMI6VB8hra57tD261jXyez
SPt0fdLVfe6Jf5y5-JaCap_QoBc5opbmT60uWgml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMjedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7q13AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw",
    "tag": "VILuUwuIxaLVmh5X-T7kmA"
}
```

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

```
{
   "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkJTPhYzgiLCJ1bmMiOiIJBMTI4RONNIiwmlwjoireGVi0",
   "encrypted_key": "5vUTZWoTQxKWceKpM_IzVQwGgz1FDwP",
   "iv": "p9pUq6XY0jfEZi1",
   "ciphertext": "HbDtOsdai1oYziSx25KEeTxmwnh8L8jHmFNe1c3zmM6V9
B8hr57tDZ61jXyEzSPt0fdLVfe6JF5y5-jAcP_JQcb5obpmT60uWG
ml8blyMqM0n9j--XhhlY90m-BHaqFD05iTOWxPxFMUedx7WCy8mxgDH
j0aBMG6152PsM-w5E_o2B3j0BrYBHhpYA7qi3AyijnCJ7BP9rr3U8kxe
xCpG3mK420TjOw",
   "tag": "VILuUwuIxaLVMhX7-7kmA"
}
```

Figure 172: Flattened JWE JSON Serialization

5.10. Including Additional Authenticated Data

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

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

5.10.1. Input Factors

The following are supplied before beginning the encryption process:

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

Figure 173: Additional Authenticated Data, in JSON Format

NOTE: Whitespace between JSON values was added for readability.

5.10.2. Generated Factors

The following are generated before encrypting:

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

75m1ALsYv10pZTKPWrsqd

Figure 174: Content Encryption Key, base64url-encoded

veCx9ece2orS7c_N

Figure 175: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc2lvbiIse30sInRleHQiLCIljAiXzsbImZuIix7fS
widGV4dCIsIk1lcmlhZG9jIEJyYW5kew1iY2siXSxbIm4iLHt9LCJ0ZXh0Iix7fS
IkJyYW5kew1iY2siLCJNZXJpYyIsIk1yLiIsIiJdXSxbImJkYXkiLHt9LC
J0ZXh0IiwicVEgMjk4MiJdLFSiZ2VvZGVyIix7fSwidGV4dCIsIk0iXV1d

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

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

4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X

Figure 177: Encrypted Key, base64url-encoded

5.10.4. Encrypting the Content

The following is generated before encrypting the content:

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

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

Figure 178: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTfHzgILCJlbmN0IjEiLCJlbmMiOiJBMTI4R0NNIn0

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_ZzH76TaIkJmYfRFg0v9MiPnx4X"
    }
  ],
  "protected": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXaWQiLCJhdWQiOiJSYXNzdWNoIiwiZXhwIjoxNjcyMDYxODA4LCJpYXQiOjE2MjMxMDI0NTF9.577vX52aJQyF95jYQ6JrHw",
  "iv": "veCx9ece2orS7c_N",
  "aad": "WyJyYW5keWJ1Y2siXSxxbXHt9LCJOZxboIixbIKJyYW5keWJ1Y2siLXJNViYyYyYsIKlyLiIiJdXsbImJkYXkiLHt9LCJOZxboIiwiveEegMj4MiJdLFSiZ2VvZGVyIixfswidu94dCISIK0iXV1d",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmH7Lyf3iPppGf3Pj7wNZgte0Ui8p74SchOP8xqyM1oFRWCNeiA6s6BcEtp8gEFigTUEyiNKOWDNoF14T_4NFqF-p2Msx8zkKxI7oPKK7arFByxIDvICNqBLBa-v3uzXbdB89fzO1-Lv4PjoFAQGhRqv1rXAmKbgkf79cB4WeyZw8M1dbBhcvV_KWZs1rsLNyg0n-JJWd_e+k6LQn5nRehApqf9zrB4aq3FXBw0xCys35PhCdaggy2kfUf12OkwKnWUbogXVD1c6HxL1qHhCwXDG59weHRDQeHyMRO8JljoV3X_bUTJDNKBFo0d7nLz-cj48Jdx35nC2TpbQAkJV",
  "tag": "vOah_Rajnpy_3hoTqVZVRA"
}
```

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

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04Mz
  MyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTE1R0NNIn
  0",
  "encrypted_key": "4YiiQ_Zzh76TaIkJmYfRFg0V9MiPx4X",
  "aad": "WyiJY2FyZCIsWlsidmVyc2lvbiIse30sInRleHQiLCIOJjAiXsxb
  ImZuIix7fSwB4dCIsIkI1cmNhd2FjY2hhY3QiLHt9LCJ0ZXh0IixbIkJyYW
  5keWJ1Y2siSXsxBIm4iLHt9LCJ0ZXh0IixbIkJyYW5keWJ1Y2siLCJN
  LZjPyYW5keYIsiklYLiIiJdXsxbImJkJXkLHiLT9LCJ0ZXh0IiwVEEgMjk4
  Mi3dLFsiz2VuZGVyIix7fSwidGV4cISIk0iXV1d",
  "iv": "veCx9eece2orS7c_N",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPpGf3p1j17wNqteJ0
  Ui8p74SchQP8xygM1oFRCNzeiA6s6BcEtp8qEFigTUEyiNk0OWDN0j14
  T_4NFqF-p2Mx8zkKxI70PK8KNaFBbyxIDvICNqBLba-v3uzXbdB89fz
  Oi-LvPjOFAPQGHqV1rjXAmKbgkft9cB4WeyZw8MldBhc-v_KWZlrs
  LNygJogm_JWd_eK6Lq5NRehhPqf9Zrbx4aq3FXBoxCys35Pchdaggy
  2kfUf120kwKnWbgXVD1C6HxLIlgHhCwXDG59weHRDQeHyMRoB1joV3
  X_bUTJdKBF0od7nLz-cj48JMX3SnCZTpbQAkFV",
  "tag": "v0A_l-Rajnpy_3hOtqv2HRA"
}
```

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.

Figure 189: Ciphertext, base64url-encoded

fNYLqpUe84KD45lvDiaBAQ

Figure 190: Authentication Tag, base64url-encoded

5.11.5. Output Results

The following compose the resulting JWE object:

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

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

```
{
  "alg": "A128KW",
  "kid": "81b20965-832-43d9-a468-82160ad9ac8"
}
```

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": "81b20965-832-43d9-a468-82160ad9ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoX3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D M3swKkJW/QyZtWsFLYmJ5Yelht/StAn21tHmQJuunT64T8D4t6C7kC9Q CCJ1iHAo1Uv4MyOt8OmoPb8fZyybNKqplzYjgIL58q8N2vz040gyG637d6 uuKphwANrTGM_Wh9c_sru2g1kzyFXPqiHbAURbc3-8BqeRb41R1-5 q5UjWVD31giLCN_P7AW8miFvUNXBPJK3nOWL4teUPSB8yHLbWeL83o1U 4UAgL48x-8dDkH23JykiVSQju-f7e-1xreHWXzWLHs1NqBrre0dEwK3 HX_xM0Ljz77Krppegoutpf5qaKg3l-_xMINm",
  "tag": "fNYLqpUe84KD451vDiaBAQ"
}
```

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-hbx3wnqhf5FlkEYos0sHsF0H",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRjxM2yLQOTqjCDKV3H30ossGw3u9DPsQLL2D
M3swKkjoWQy2tWsFLYMj5YeLht_StAn21tHmQJuNt64T8D4t6C7kC90
CCJ1IHAo1Uv4MyOt80MoPb8fZyNkqPlzYJgIL58g8N2v460qyG637d6
uuKPwhAnTGM_zWhqc_srOvglKzyFxpqlhBAURbc3-8BqerB48iR1-_5
g5UjWd31giLCN_P7Aw8mIifVvUNXBPK3nOWL4teUPS8yHLbWebL80JU
4UAgL48x-8dDkJH23JykibVSju-f7e-1xreHWxWlHs1NqBbre0dEwK3
HX_xM0LJu77Krppgeoutpf5qaKg31-_xMINmf",
  "tag": "fNYLqPue84KD451vDiaBAQ"
}
```

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

YihBoVGsR117jCD

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.
5.12.5. Output Results

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

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

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

Figure 199: JWE Shared Unprotected Header JSON

The following compose the resulting JWE object:

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

```json
{
  "recipients": [
    {
      "encrypted_key": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoV0GsR117jCD",
  "ciphertext": "qtPIMMaOBrgASL10dNQh0a7Gqkrk7EalIvwht7R4T1uq-ArsVCPaIeFwQFzrSS6oEUWbBtxEasEOvC6r7sphyVziMCVJEuRJyoAHAFC3pEqQPb4IClSDSgyYxjwL3sylvbhHYUYjQuTuMqEDjgjJfbOifwHIsDsRPeBziANmqeifVpq5GTcWFo5K_MN1QURR2Wj0AHC2k7J2fu21wJUHLF8ExF2LZ4n1msvju_mviMYiikFnsZAudISOa6073yP2tL04k_1FI7WDFrb2w70qKLWXz1pcxohPVOLWqpA3mFNRKdy-bQz424XX91fc1cne31N4-8Bkmjojpw-0dOjQkJdLRGkC445Fb_K1tlDQxw2sBF",
  "tag": "e2m0V7m7VjK2VpCKXS-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": "YihBoV0GsR117jCD",
  "ciphertext": "qtPIMMaOBrgASL10dNQh0a7Gqkrk7EalIvwht7R4T1uq-ArsVCPaIeFwQFzrSS6oEUWbBtxEasEOvC6r7sphyVziMCVJEuRJyoAHAFC3pEqQPb4IClSDSgyYxjwL3sylvbhHYUYjQuTuMqEDjgjJfbOifwHIsDsRPeBziANmqeifVpq5GTcWFo5K_MN1QURR2Wj0AHC2k7J2fu21wJUHLF8ExF2LZ4n1msvju_mviMYiikFnsZAudISOa6073yP2tL04k_1FI7WDFrb2w70qKLWXz1pcxohPVOLWqpA3mFNRKdy-bQz424XX91fc1cne31N4-8Bkmjojpw-0dOjQkJdLRGkC445Fb_K1tlDQxw2sBF",
  "tag": "e2m0V7m7VjK2VpCKXS-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:

dYOD28kab0Vvf4ODgxVAJxGhScZIS0p8M51zjw4w6Y5G4XJQaNNIBqyvUUA
OcpL7S7-ccFe7Pi07qV_Q06WmCSa-vhW6me4bWBrF7cHwEQJdXihiidAYVvajJIA
KMXMvFRMv6I1Rr076DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-w
qjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9c1N1Lz2MbeYS
rRicJK5xovdWgkpIdkMCHo4LvdhRRvzoKzlic89jFWPlnBq_V4nStrGuExtp_-d
bHcGlihqC_wGgho9fLMK8JOArYLCMDNQ

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

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

- Recipient JWE Unprotected Header from Figure 205.

```
{
  "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:

```
{
  "encrypted_key": "dYOD28kab0Vvf40DgxVAJxGhSzZICSOp8M51zjwj4w6Y5G4XQsNN1BiqvYUUAocepL757-cFe7Pio7gV_Q06WmcSa-vhW6me4b
  WrBF7cHwEQJdxIihdAYYWavjJ1aKMXvFRMV6iDr076DFthg2_AV0_tSiV6x5IFq1xnYFpmP91tc5WJDOG6t-wqjw0-b-S1laS11QvbP78dQ7
  Fa0zAVzzjHX-xvyM2wxj_oxtr9c1N1Lz2MbeYSrRicJK5xodWgkpIdkJHo4LvdhRRvzoKzlic89jFWP1nBq_V4n5trGuExtp-_dBbGlhlgc_wG
  gho9fLMK8JOArYLcMDNQ",
  "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.

```
{
  "kty": "EC",
  "crv": "P-384",
  "x": "Uzdvk3p15wKCRc1izp5_r00jqt-Ti6818g2b8d1RhsE2xAn2DtsMrB5M2a2CX",
  "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAz53MC7PjppLioy3ylEj1pOMbw91fzZ84pbfm",
  "d": "1DKHfTv-PiifVv2VBHM_Z1Vcw0Mxk0yANS_1qHzcrDxVY3jVcVZPwMxJKIE793C"
}
```

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

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

produces the following Encrypted Key:

ExInT0io9BqBMYF6-maw5tZlgoZXThDlzWKsHixJuw_e1YgSSIId_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-168i8g2b8mva8diRhsE2xAn2DtMrb25Ma2CX",
        "y": "VDrRyFJh-Kwd1EjAqmg5Eo-CTHAZ53MC7PjpLioy3y1EjIlpOMb
    w91fzZ84pbfm"
    }
}
```

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

```
{
  "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw_e1Y4gSSID_w",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
      "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEjI1pOMbw91fz824pbfm"
    }
  }
}
```

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.
a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E

Figure 212: Recipient #3 Encrypted Key, base64url-encoded

59Nqh1LlYtVIhfD3pgRGvw

Figure 213: Recipient #3 Authentication Tag from Key Wrapping, base64url-encoded

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_7JSuPB8zeagxXRam8dwCfmkt9-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.

ajm2Q-OpPXCr7-MHXicknb1lxLdXxK_yLds0KuhJzfWK04SjdxEeSw2L9mu3a
_k1C55kCQ_3xlkcVKC5yr___Is48V0oK0k63_QRM9tBURMFqLBvJ8vOYQX0oJW4
VUHJLmGhF-tVQWB7Kz8mr8zeE7txFOMSaP6ga7-siYxStr7_G07Thd1jh-zGT0
wxM5g-VROSmq0K6AxPllwEqRp?pkt2zRM0ZAXqspeio6FJ7FHLjEfNnD-zDIZu
kLpCbczhzMDLlw2-8I14FQorgi-iEuzHgIIF1Jn2wh9Tj0cG_kOZy9BqMRZbmXYM
Y9YQjorZ_P_JYG3ARAIIF30jDNpdYe-K_5Q5crGJSDNyij_ygEItR5jssQVH2
ofDQdLChtaZE

BESYyFN7T09KY7i8zKs5_g

Figure 218: Ciphertext, base64url-encoded

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.

```
{
  "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": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zj
wj4w6y5g4XJQsNNIBqgyyUUA0cpl7S7-cFe7Pio7gV_Q06WmCSa-
vhW6me4bWw-Bf7cHWQJdXihiDAYWVajJ1aRMXWfFRMv61D1Rr076
DFthg2_AVo_s1V6xSEIFqt1xnYPpmP91tc5WJDQGb-wqjw0-b-S
1laS11QVbuF78Q7Fa0zAVzzjHx-xvyM2wxjOttxr9c1N1IzZMbe
YSrRicJK5xodWgkpIzdkMHo4LvdhRRvzcKzlic89jFwpInBq_V4n
5tvGuExtp_dBhcGl1hqC_wGgho9fLMK8JOArYLcM1DqN",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5t2loZXThD1zWKSHi
xJuw_elY4gS51d_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3p15wKCRclizp5_r00jeqT-16818g28mva8diRhs
E2xAn2dtMRebZSi2CW",
          "y": "VDrRyFJh-Kwd1EjAgm5Eo-CTHAZ53MC7pjpLioy3y1Ej
IlpOMbw91fz84pbm"
        }
      }
    },
    {
      "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfMkt9-Wy
TpsI1E",
      "header": {
        "alg": "A256GCMKW",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59Nqhl1YtVIfD3pgRGwv",
        "iv": "AvpeoP29Ncn9mkBn"
      }
    }
  ],
  "unprotected": {
    "cty": "text/plain"
  },
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0"
}
```
"iv": "VgEIH2Y0EnzUt2F12RpBlg",
"ciphertext": "ajm2Q-OpWXcr7-MHlcknbl1sxLdXxK_ylds0KuhJzfWk 04SjdxQeSw2L9mu3a_kiCS5kCQ_3xl1kcVKC5yrl_Is48VOocK0k63_QRM 9tBURNMFqLBj38v0YQXw0JW4VUHJLMghF-tVQWB7Kz8mr8zeE7txF0M5a P6ga7-sIxXtr7-G0TThd1jh-zGT0wXM5g-VRORe0K6AxpLlwEqR07pk t2zRM02AXgpe1O6Pf7FHLDyEFnD-zDIZukLpC0zhMDLLw2-8I14FQ rgi-iEuzHgIjFIJn2w9Ht0cg_k0Zy9BqmRZbmYXY9YqorZ_P_JYG3 ARAIF30jDNqpdYe-K_5Q5crGJSNYiy_ygEItR5jssQVH2ofDqDCh t aze",
"tag": "BESYyFN7T09KY7i8zKsS_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".

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

Figure 222: Payload Content, in JSON Format
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwibXhwIjoxMzAwODE5MzgwLCJodHRwOi8vZ2hhbXBsZXN5b21vbnQxNzI5LmNhc3QuY29tIiB9

Figure 223: Payload Content, base64url-encoded

{
  "kty": "RSA",
  "kid": "hobbiton.example",
  "use": "sig",
  "n": "KnPzBDM6fcyv5i-Q8QAQ-K8gsC3HJb7FYhYaw8xhBNj-a-t8q0LDKwL2qXYV-ffWxJv5GGr1Z4G5U21fMEegTDyTRq3tepgKFjMGg6iy6fklNNx2gEonsnHsf2A9GjwRlmtKpbkl-hxwi1USAT-A1e11q86cF2v5Ewz5_SGGBoaROVdUYxqETDqgM1z5cKV42jZD8-nfr40VB07rkac6LQdHjJUySH_Er20Dxx30Ky197ciXKT5-QXKmm8ivyRCmu22ZOPI
nd2BKC5oG4MwALhaL2Z2k8CsRdfy-7dg7z41rP602eEeVtaUp4bX4AKra14tFtw",
  "e": "AQAB",
  "d": "ZLE_TlxpE9-W-n2Vba-HWvuYPtjxvwVXCLjFopJsedea8g9RMx34qEOEtncYc2un3C23Lti-jmu5RAT8YSc76YJds3ZVw0Uio8MBeG6-iOnvgobbbNk757-xjxTJU72EjOr9kb7z6ZKwDDq7HfycDhUCeCyHFvc71L_6TibVhhAhFOWN1qLjEgWVY0rbyNGKi6fnpEbwyHoMwY6HM1qvnEFgP7iZ0YzHUT535x6j4VKcdA7zuFkhuauyseySEW7mz6Mfj1vdjJiy9L0DifIz30Xv4ckogqHF5GONU6tnMmmNgAd6jViyE1elPrX11tBc1I4bRWr-zrpHgAQ",
  "p": "yKWyO1AwqMRQ1gBoOa1cBNuUs2Rh-pBaxD-nIkweMtMg-0-B2iSVhMr8shhorhonV7vxcQacgBAATGW-hAaFuehWjxWsH-3KccRM8toL4e0q7M-idRD0BXSoe722-CV2x_zCY3RP8q642R13WqGDIM4MbUK2syc9-c",
  "q": "uD4o15V30KDzf8vFJw589plv1vQV3NellrinRUPHkxasAzDzccGgrWMWpGXFNFnL3w5CqPLeU76-5lVYQqOHWY10hvVXHQ7sgaG-483A3E6C123FonF45m7_2ooAstJDe49MeLTQKrsI1bS_KvqpYvFSPtcPzCZhkkk",
  "dp": "jmTnEq2qqa8oaymjhJSCnsvUXnMqC2AneQjRqkFqQz-vZ2VKPKNBpVXYvIf5b2-L3tM30W2zd2InDyRWUX17T510KwPTASTOnTqAmYChGiq8Xxd1hcrtSvXldBakC6saxwI_TzGYY2MVz22nCvCXH4qjSxOrf3pHFU",
  "dq": "R9FvU880VzEtkX13-5-WhsE4DjHmned2Iul3rifBdfLpq-P-iWPBbGag9wzQ1c-J7SzdCqekEJDv5yd2C7rn26kpzwBh_rml8zscAk1gsun7CJGaYz-7-sGWylJGShFazP52h1rCJ0YUeaQM3rIpY77_oLAhpDAOhLk",
  "qi": "8StC7ZknW6PHTjkwttQQOPLmRFwirRlfAViuD8BW9Cr7_7F20qUZCqzmHTYumwGFIH1LVRep7anlWeAJJCZ_1b3fq_al4qH3Pe-EKilH6IMazuRtLURCohThrExDbF5dYbscDnFwRUWErZ4N1Be0bmxYuPqwKd9Q2wMo0"
}

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

The following is generated to complete the signing operation:

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

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

Figure 225: JWS Protected Header JSON

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

dPpMqwRZxFyi1UfcDAaf0m99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIA
kIvn1qskChirjKvY9ESZU5CP4JjvfjyFS-nqjJxYoA5ztWoYFk2cZNIPXjcJSQ
wXPO9te-v4VSqgDoakHqFxYog4N6Cz11Kph1UsYDSI67_bLL7e1g_vkjFmp5
_W515LuUYGMe6hXQLaIUXf9EmV2JmvTMuZ-vBOWySniy1Epo72CRTvmttrf5
ARo0S6NllY3KtUxeP-SOmD-LeywW9S1kohYzMVAZDD0rVbV7KVRHpeYNaK75KE
QqDCEEKs_rskZ5-Qtt_nle3WTh1mEYA
```

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
.
eyJpc3MiOiJob2JiaXRvb15leGFtcGxlIiwiZXhwIjoxMzAwODE5MzgwLCJodHRwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0
.
dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIAkIvnlqskChirjKV9ESZ2NUCP4JjvyfPS-nqjjXyoA5ztWOyFk2cZNIPXjcJXSQwXP09tEe-v4VSqgD0aKHqPxYoq4N6Cz1llKph1U1sYDSI67_bLL7elg_vkjfM5p _W515LuUYGMe6hxQIaUXf9EwV2JmvTMuZ-vBOWy0Sniy1EFo72CRTvmtrIf5AR0o5MNLiy3KtUXeP-SOmD-LEYwW9SlkohYzMVAZDORVbv7KVRHpeYNaK75KEQqdCEEks_rskZS-Qtt_nlegTWh1mEYaA

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.

0RHSNYwN-6-2QBGsYTZLSQ

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-bi13mwy7RoYhjTkBE6P?7sMMXXx4g
zMedipiJHVQveyI-zkZV79matpgevAJWrXzOUsYGTtwoSN6gtUVt1Laivjvb21O0u14YxSHV-ByKlkyeeTf_fuYJxHoKQL9P424sKx2WGYb4zsBIPF4ssl_e51R7nany-25_UmC2urosNkoFz9cQ82MypZP8ggbQjyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dWVIY7qleKpXcV11WO_2FefEBqXxVijLeZivjNkzogCq3-I
apSjVFmJbXjpYLT8maawolyy1XXMunIpNcOY3n4KrXLrCcteX85m4IIHZa38s1hp56fPseMA-Jltmt-a9iEDt0zhtxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBK0Ywfk7JhLRDgOnJjiJLn7TI4Udp9dCmUXEN6zov23w15qJlENJtqnbplp
ymoowAHCT4e_Owbimlq0AEpTHUnA2iiINs9WTX_H_TXuPC8yDhh1smxS_X_x
pkIHkiIHWDLxO3BpqDTivpKkBYwgP2UZkcqxX2Fo_GnVrNwlK7Lgwx6FSQvDO0

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.

```
{
  "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.

SZI4IVKhmpaz1_pJQXX3mHv1ANnu04Wf9-utWYUcKrBNgCe20FMf66cSJ8k2Q kxaQ03_r60MEgE9ofomwtky3GFxMeGRjtpMt_90AvVLsAXB0_UTCBGyBq3C2bWLX qZlfJAAoJRUPRk-BimYzy81zVBuIhc7HsQePCpu33SzMsFHyjn41p_idrzJg_1z TNGKdt8zdnUPauKTKN6H1D4fuzvDYfIDAfqGPyL5svRwbiXpDgokEszM-9C hMPgW1QHzuZ_u13bvrJwr7nuGZs4cUScy3n8yE3AHCLurqst-A9mz1X38xEa u1V1814Fg9tLejdpQzJPBgqeHQBj41wGD5Ee0dQ-Mtz4NnhkIWX-YKBb_Xo2 zI3Q_1sYjKUuis7yWW-HTr_vqvFt0bj7WJf2zB0T3dvsoGaTvPH2dyWwumUr lX4gmPUzBdwTO6ubfYSDUEE5poyoD_OtWeUSyCYB0-k-M7tXg26qJo21gYjLf hnm9zy-W19sOCZGuzgFjPahwXHpvn8_t-O_ES9kogjJLxS1IIMU9YXmnn2MyNc 9EIwnogsCg-hYuvzyF0sIrulkmi194_SL1xgMl7o03phcTMxtlMizR88NKU1WkB siXMCjy1Noue7MD-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:

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6I1ExMjhHQ00ifQ.
a0JHRoITfpX4qRewImji1Stn8m3CPxBV1ueYVhjUrCyrB9j17YhcRjyphDOOS4
E7rXbr2Fn6NyQ-A-qgj0TFFxNjVOGrG-bi13mw7RoYhTkBE6Cp7sYMXXx4g
zMedpiJHQVeyI-zkZV7A9matgevAJwRzrU0yeYGTtwoSN6gtUVT1Laivjvb21
O0u14YxSVH-ByK1kyeKoKLQl9P424sKx2WGYb4zsbBIPF4ssl_e5I
R7nany-25_Umc2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzUDU
F_5JCIdl-Qv6H5dMVI7qYleKpXcV11WO_2FefEBqXxVjIjLeZivjNkzogCq3-I
ap5jVFqMjBxjyLT8muawaojly1XMMuinNpOCOY3n4KkrXLRccfteX85m4IHHM2
a38s1hpr56ffPseMA-Jltm-t-a91EDtOzhttxZ8AXy9tsCAZV2XBNWG8c3kJUsAa
mBK0Ywfk7JhLRDgOnJjIjLhn7TI4UtDp9dCmUXEN60v23W15qJIEHN5Jnb1p
ymooeWAhCT4_eOwhimIg0AEpTHUda2iiINs9WTX_H_TxuPC8yDh11smxS_X_x
pk1HkiIHWDLx03BpQTvPsKBYwqP2UZkcxqX2Fo_GnVrNwlK7LqYW6FSqvDO
0.
GbX119kXz0sxXPMa
.
SZI4IVKHmwpaz1_pJQXX3mHv1ANnOU4Wf9-utYWYucKrBNGe2OFMf66cJ8k2Q
kxaQ03_r60ME9ofomwtky3CGFmEGRjtpMt9A0vVLsAXBO_UTCDBGyBq3C2bWLY
qZlf1yAAoJRUrPKr-BmY2ZV8zEuThc7HsQePCpu330MsPFijn4IP_idrJz_gL
TNgKDt8zdnuPauKTKDNH1D4dFuxzDYFfIAfQPyL5sVRwbiXpDgokEsZM-9C
hMPqW1QNzuX_zu13bvrJwr7nuG2s4cUScY3n8yE3AHCLurzgIs-A9mx1k38xExa
ulV18l4Fg9tLejekAuQ2JfbebHEQBj4e1wGD5E0dQ-Mtz4NhXkIw-xYKbb_Xo2
zi3Q_1sYjKuUs7ywW-HT-qrqFvT0bj7Wf2vzB0T3dvsoGaTvPH2dyWwumUr
lx4gmPUzBdwo6ubfYSDUEEZ5pY0d_OtWeUSycCYBKO-aM7tXg26qJo2lgyJjEf
hn9zy-W19sOCZGuGzFJPhawXHpvnj_t-0_ES9k6kogjLJxSlI1MU95XmmwZMYnc
9ElwnqcsGg-hVuvzyU0airuktmI94_SLlXgML7o3phcTMx1MizR88NKUIWkB
siXMCjy1Noue7MD-SDp5dMm
.
KnIKEhN8U-3C9s4gtSpjSw

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

```json
{
    "recipients": [
        {
            "encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBVlueY1VhjurCyrbg3i7yRChRYjphDOOS4E7txbr2F6n6NyQq-A-gqTXFqXnjvO
GrG-bi13myw7RoYjh9j3kBEC6G7s5MyMXXX4gzMepyHJqVey-zkZV7A9matgpevAFWrzOuySGITtw0SN6gtUVt1Laivvbj2100u4YxS
HV-BxYkyeetRPhi_yJxHoKlQ19424sX2fWGy42zsBIF44s1_e5IR7nany-25_Umc2urosNg0Fz9cQ82MyzF8gqBqQyPN-Fpp42-5
o6yV64x6yzDUF_5JCIdl-Qv6H5dMVtY7q1eKpxcV1lW0_2FeEBq
XxXvJlZivjNkzogCq3-lApSjVFnMjBxjpYLT8muawolyy1XXM
uinIpNcOY3n4KRrXLrCcteX85m11HMzMa38s1Hpr56FPPsEMa-J1
tmt-a9iEDtohxxz8Ax9y9tcCAZV2XBNW8Gc3kJusAamBK0Ywk7J
hLRDgoJnjJLln7TI4Xdp9dCmUXEN6z0v23W15qJEJXcJqnb1p
ymceoWAHCT4e_Owm1qg0AEpTHUdA2ilLns9WTX_h_TXuPC8yDDh
1lsmxS2_xxpkIHKiiHWDD10Lx03BpqDTivkkBYWyqP2UZkcxqx2F0_
GnVrNw1K7Lgxw6FSQyD00"
        }
    ],
    "protected": "eyJhJbcQo1JSSJuTEt0FFUCIsImN0eSI6IkxPvXCIrImVuYy
I61kExMjhHQ00ifQ",
    "iv": "GbXl19kXz0sxXPMa",
    "ciphertext": "SZI4IvKHmpwzal_pJQXX3mHvlANnOU4Wf9-utWYUcKrBN
qCe2OMf66cSj8kQqxaQoD36O7sG99omwtky3GfxWeGRjtpM90Av
VLeAXBO_UTCGBgyBq3C2bWLMxqZ1fJAAoJRUPRk-BimY1Y12VBu14h7Ys
QePcpu33SxMsFhjn41p_idrzg2L7NgKQD78dnUPauKTkDNOHID4fu
vzDYDIAfGPyPn5sVrwiXpDx0GzkEszM-9ChMPqW1QNhzuX_zu13Bjr
wr7nuGz2cUSy38nE3AHCLurgIs-A9mz1X3Xeauv1814F9tLej
kAuQzJPbqfHQBj4e1wG5e0dQ-Mtz4NnhkIWX-YKbw_Xo2Z15Q_1sYj
KUuis7yWW-HTR-vqvFt0j7WjWf2vzB7T36dvsogkPHT2dyWwumUr1x
ghmuPzB5bPvC05_0tMeUSYcCYBKD-aM7tXq26qJ21qY
jLFnh9zym-W19sOC2GuzgFJPhawXqnvj_t-0_ES9kcojGLs81M1U9Y5
Xmnw2MvNc9EiwonsCg-hVuVzyP0sIrutkmI94_SLxkgMr7o3phecTMr
x1MrizR88KnU1WkbSiXMcy1Noue7MD-ShDp5dmM",
    "tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

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

```json
{
  "encrypted_key": "a0JHRoITfpX4qRewImjl1Stn8m3CPxBV1ueY1vhjurCyrBg3I7YhcRYjphDoDOS4E7rXbr2Fn6NqYQ-a-gqT0FxqjVOGr-bi13mwy7R0yHjTbEBC6Fp7aMYMX4x4gzMepd1jHqVeyI-zkZ7V9amatpgevAJWrXzOUysYTtowoSN6gtUv1Laivjvb2100u14YxSHV-ByKlyeettRpf_uYjxHoKLQL9P424sKxZ2WGYb4zbsBIPF44ss1_e5IR?nany-25_UmC2urosNkoFz9CQ82Myp2P8ggbQJyPN-Fpp4Z-5o6yV64x6yzDFU_5JC1d1-Qv6H5dMV1Y7qleRpxCv11wO_2FefEBqxXxv1IjLeZivjNkzogCq3-IapSFVnMJbXjyjLYT8muaa0Ly1XMMuinIpNcOY3n4KrXtCcCteX85mdIIHMZa38s1Hpr56fFpeMA-Jlmtt-a9iEDfozhtxxz8AXy9tsCAZV2XBEWNG8c3kJuAsAmBKOYwfk77JhLrdgOnJj1JLhn7T14UXdP9dCmUXEN6ozv23W15qJIEXNJtqnlpmooeWAHCT4e_Owbim1g0AEPThUda2ii1ns9WHTX7UuPC8ydDHi1sxxS_X_xpkIHk1IHWDOlX30BpqDTivpKkBywqP2UzkcxqX2F0_GnVrNw1k7Lgqw6FSQvD00",
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6In0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ",
  "iv": "GbXi1l9kXz0sxxPmA",
  "ciphertext": "SZI4IvKHMwpazl_pJQXX3mHvl1ANnO4UWF9-wtWYUcKrBNgCe2OFM66cSJ8k2QkxqaQD3_R60MG9e9ofomwtky3GFxMeGRjtpM9OAvVLsAXB0-UTCgDyBq3C2bWLXqZ1fJAAoJRU3Rk-BimYzY81zVBu1c78sQePcPru33SzMsFhjn41P_idrJz_g1ZTNgKtD8s2nUPaUkTKDNOH1DD4fuvrDYsDiaQpFyL5sVRwb1XPxdOgkEszM-9ChMPqW1QNhzux_X_2u13brWx7nuGZsQ4UScY3n8E3AHCLurJg-A9mz1X38xEauiV1814Fg9tLejdxAqUZjFbqehQ3B3e41WGD5EedQ-Mtc4NNhki1Wx-YKbb_Xo2zi1Q_I8zYKUus7yWW-HTR_vvqvtcbj7W7JF2vzcB0T33dvsoGaTveHZ2yWwumUrl1x4gmpU2BzD06ubYSDU8zHsy0d_DtWeUSyCYYBD-aM7Xq246qJo2lgYjLfhn9zy-W19s0CZGuzgF3phawXHpvn_t-t-0_ES96kogJLxM1U9SYXmnowz2MyNcEifwmoosG-hVvzyP0sIrutkmI94_SL1x8gM17o03phcTmx1t1MiZrB88NKU1WksbixWMCjy1Noe7MD-ShDP5dmM",
  "tag": "KnIKeHn8U-3C9s4gtSpjSw"
}
```

Figure 238: Flattened JWE JSON Serialization
7. Security Considerations

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

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

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

8. References

8.1. Normative References


8.2. Informative References


Acknowledgements

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

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

Author’s Address

Matthew Miller
Cisco Systems, Inc.

EMail: mamille2@cisco.com