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

Abstract

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

Status of This Memo

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

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

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

Copyright Notice

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

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

1. Introduction .................................................... 5
   1.1. Conventions Used in This Document .......................... 5
2. Terminology .................................................... 6
3. JSON Web Key Examples ........................................... 6
   3.1. EC Public Key ............................................ 6
   3.2. EC Private Key ........................................... 7
   3.3. RSA Public Key ............................................ 8
   3.4. RSA Private Key ........................................... 8
   3.5. Symmetric Key (MAC Computation) ........................... 10
   3.6. Symmetric Key (Encryption) ................................ 11
4. JSON Web Signature Examples .................................... 11
   4.1. RSA v1.5 Signature ........................................ 12
       4.1.1. Input Factors ...................................... 12
       4.1.2. Signing Operation .................................. 12
       4.1.3. Output Results ..................................... 13
   4.2. RSA-PSS Signature ......................................... 15
       4.2.1. Input Factors ...................................... 15
       4.2.2. Signing Operation .................................. 16
       4.2.3. Output Results ..................................... 17
   4.3. ECDSA Signature .......................................... 19
       4.3.1. Input Factors ...................................... 19
       4.3.2. Signing Operation .................................. 19
       4.3.3. Output Results ..................................... 20
   4.4. HMAC-SHA2 Integrity Protection ............................ 21
       4.4.1. Input Factors ...................................... 22
       4.4.2. Signing Operation .................................. 22
       4.4.3. Output Results ..................................... 23
   4.5. Signature with Detached Content ........................... 24
       4.5.1. Input Factors ...................................... 25
       4.5.2. Signing Operation .................................. 25
       4.5.3. Output Results ..................................... 26
   4.6. Protecting Specific Header Fields ......................... 27
       4.6.1. Input Factors ...................................... 27
       4.6.2. Signing Operation .................................. 27
       4.6.3. Output Results ..................................... 28
   4.7. Protecting Content Only .................................. 29
       4.7.1. Input Factors ...................................... 30
       4.7.2. Signing Operation .................................. 30
       4.7.3. Output Results ..................................... 31
   4.8. Multiple Signatures ...................................... 32
       4.8.1. Input Factors ...................................... 32
       4.8.2. First Signing Operation ............................ 33
       4.8.3. Second Signing Operation ............................ 34
       4.8.4. Third Signing Operation ............................. 36
       4.8.5. Output Results ..................................... 37
5. JSON Web Encryption Examples ................................... 39
5.1. Key Encryption Using RSA v1.5 and AES-HMAC-SHA2 ...........39
  5.1.1. Input Factors ........................................39
  5.1.2. Generated Factors ......................................41
  5.1.3. Encrypting the Key .....................................41
  5.1.4. Encrypting the Content ................................42
  5.1.5. Output Results .........................................43
5.2. Key Encryption Using RSA-OAEP with AES-GCM ....................45
  5.2.1. Input Factors ..........................................46
  5.2.2. Generated Factors ......................................47
  5.2.3. Encrypting the Key .....................................48
  5.2.4. Encrypting the Content ................................48
  5.2.5. Output Results .........................................49
5.3. Key Wrap Using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2 ...52
  5.3.1. Input Factors ..........................................53
  5.3.2. Generated Factors ......................................54
  5.3.3. Encrypting the Key .....................................54
  5.3.4. Encrypting the Content ................................55
  5.3.5. Output Results .........................................56
5.4. Key Agreement with Key Wrapping Using ECDH-ES and AES-KeyWrap with AES-GCM ........................................59
  5.4.1. Input Factors ..........................................59
  5.4.2. Generated Factors ......................................60
  5.4.3. Encrypting the Key .....................................60
  5.4.4. Encrypting the Content ................................61
  5.4.5. Output Results .........................................63
5.5. Key Agreement Using ECDH-ES with AES-CBC-HMAC-SHA2 ........65
  5.5.1. Input Factors ..........................................66
  5.5.2. Generated Factors ......................................66
  5.5.3. Key Agreement ..........................................67
  5.5.4. Encrypting the Content ................................67
  5.5.5. Output Results .........................................68
5.6. Direct Encryption Using AES-GCM ..................................70
  5.6.1. Input Factors ..........................................70
  5.6.2. Generated Factors ......................................70
  5.6.3. Encrypting the Content ................................71
  5.6.4. Output Results .........................................72
5.7. Key Wrap Using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2 .....73
  5.7.1. Input Factors ..........................................73
  5.7.2. Generated Factors ......................................74
  5.7.3. Encrypting the Key .....................................74
  5.7.4. Encrypting the Content ................................75
  5.7.5. Output Results .........................................77
5.8. Key Wrap Using AES-KeyWrap with AES-GCM ........................79
  5.8.1. Input Factors ..........................................79
  5.8.2. Generated Factors ......................................80
  5.8.3. Encrypting the Key .....................................80
  5.8.4. Encrypting the Content ................................80
  5.8.5. Output Results .........................................82
5.9. Compressed Content ........................................... 84
  5.9.1. Input Factors ........................................... 84
  5.9.2. Generated Factors ........................................ 84
  5.9.3. Encrypting the Key ...................................... 85
  5.9.4. Encrypting the Content .................................. 85
  5.9.5. Output Results .......................................... 86
5.10. Including Additional Authenticated Data .................. 88
  5.10.1. Input Factors .......................................... 88
  5.10.2. Generated Factors ....................................... 89
  5.10.3. Encrypting the Key ..................................... 90
  5.10.4. Encrypting the Content .................................. 90
  5.10.5. Output Results ......................................... 91
5.11. Protecting Specific Header Fields ......................... 93
  5.11.1. Input Factors .......................................... 93
  5.11.2. Generated Factors ...................................... 94
  5.11.3. Encrypting the Key ..................................... 94
  5.11.4. Encrypting the Content .................................. 94
  5.11.5. Output Results ......................................... 95
5.12. Protecting Content Only ................................... 97
  5.12.1. Input Factors .......................................... 97
  5.12.2. Generated Factors ...................................... 98
  5.12.3. Encrypting the Key ..................................... 98
  5.12.4. Encrypting the Content .................................. 98
  5.12.5. Output Results ......................................... 99
5.13. Encrypting to Multiple Recipients ....................... 101
  5.13.1. Input Factors .......................................... 101
  5.13.2. Generated Factors ...................................... 101
  5.13.3. Encrypting the Key to the First Recipient .......... 102
  5.13.4. Encrypting the Key to the Second Recipient ....... 103
  5.13.5. Encrypting the Key to the Third Recipient ......... 105
  5.13.6. Encrypting the Content .................................. 106
  5.13.7. Output Results ......................................... 108
6. Nesting Signatures and Encryption .......................... 110
  6.1. Signing Input Factors ..................................... 110
  6.2. Signing Operation .......................................... 112
  6.3. Signing Output ............................................ 112
  6.4. Encryption Input Factors ................................... 113
  6.5. Encryption Generated Factors ............................... 113
  6.6. Encrypting the Key ......................................... 114
  6.7. Encrypting the Content ..................................... 114
  6.8. Encryption Output ......................................... 115
7. Security Considerations ....................................... 119
8. References ..................................................... 119
  8.1. Normative References .................................... 119
  8.2. Informative References ................................... 120
Acknowledgements ................................................ 120
Author’s Address .................................................. 120
1. Introduction

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

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

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

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

1.1. Conventions Used in This Document

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

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

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

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

2. Terminology

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

3. JSON Web Key Examples

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

3.1. EC Public Key

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

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

```json
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sYl_oJXu9A5RktKqjqvijyekWF-7ytDyRXyqCF5cj0Kt",
  "y": "AdymlHvOiIxXkEyYXQzRjJlZG1jY0mYX1JZaCJN34kfmC6pV5o5HiraVySsUdakAgDPewQrJmbnX9cwiGffP-HqHZR1"
}
```

Figure 1: Elliptic Curve P-521 Public Key

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

### 3.2. EC Private Key

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

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

```json
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sY1_oJXu9A5RkTKjqvjqyeKWF-7ytDyRXyCF5cj0Kt",
  "y": "Adym1HvoiLxKhEhayXQmNCvDx4h9htZaCJN34kfmC6pVoHbQHiraVySsUdAkAgDPrwqRJmbnX9cw1GfP-HqHZR1",
  "d": "AAhRON2r9cgXX1hg-RoI6R1tX5p2rUAYdmpHZoC1XNM56KtsrX6zbKipQrCW9GZH3T4ubpnoTKLDYJ_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-O-XV2jRojdNhxJWTDvNd7qQ0VEiZQHz_AJmScpMaJMRBSFKrKb2wqVwGU_NsYOYL-QtiWN21bzcEe6XC0dApr5ydQlrlHqkHHi3g3RBordaZ6Aj-oBHqFEHYpPe7Tpe-0fVFhd1E6cS6M1F2cD1NNLYD51FhpPl9bTwJlde3uhGgC0ZCuEHg81hzwOirtIqbS0FV_bb9k3-tVTU4f_g_3L_vn1UFAKwuCLgKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUkr5iaiQkqgc5gHdrNP5zw",  
  "e": "AQAB"  
}
```

Figure 3: RSA 2048-Bit Public Key

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

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

3.4. RSA Private Key

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

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


{
"kty": "RSA",
"kid": "bilbo.baggins@hobbiton.example",
"use": "sig",
"n": "n4EPtAOc9AlkeQHPzHStgAbg7bTZWUZdR8_KeKPEHld4rHTeT-
O-XV2jRoWnHxWJTDvNd7njqODVEiZQHz_AjMScpMaJMRBSFkrKb2wqV
wGu_NaYOL-QtiWJ2lbcb5e6X0qAp5yqDlhrqKhHih3RQJ67ca6Aj-
0BqHEHYYp7e7Tpe-OvFvHd1b6c86S861FZcDJ1NLYD51FhpP19bTwJ1sde
3uhGgC0ZCuEHg81hzwOHxtIQbS0FVbb9k3-tVUTU4f_3L_vniUFAKwC
LqKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUKz5ia1Qkqg5g
HdrNP5zw",
"e": "AQAB",
"d": "bWUC9B-EFRIO8kpGf02uyGPvMNKvYWMT_B_iikiH9k20eT-01q_I78e
izkpXxQ0UTEsLsNRS-8uJbVQ-A1irkwMSMK1I3XGdrzCkuqgRld
Y7sNA-AKZGh-Q6642rINLRc8eN-wn23ui_qofkLnK9QWdGQpaIsA-b
MwWWSDFu2MUBYWkHTEhZLYQo0e4noqeg1hExvBTHBOBkMXiuFhQg1BU
61-DqEiXqXq8x2sxt2h-LMnT3046A0YJoRioz75suQfGCshWTBnP5udj
18khv071hf5JdPm5qy1LhsFF84L_hMCuoFau7gdsPfHPxjV0c
OpBrQzwQ",
"p": "3Slxg_DwTXJcb06959RoxYggQCAZ5RnAvZ1n01yhHtnUex_fpp7AZ_9nP
a07HX-_SFgQeутa02TDjAWU4Vupk8rw9J0RazZ0ZFvUAMr_WCsmg
peNQnev1T7iEsnh8UMt-n5CahhkikzEsmnHDLxOrvRJlsPp6Zv8
buq0k",
"q": "uKE2dh-cT6ERF4k4e_jy78GfYUIuAyoSSJuBzp3Cubk30Csxg6grT
8bb_cuQDm1MZwntQdCy15fHurq3MP15vM0GN81HTe2u21mKvqWn7an
V5u2hMliZ74z4yKnuUwFwOaoyY89E.xlsx-hdqRxH1sqA1923BjPeF0j
7sPc",
"dp": "B8PVvXkkvjrj2L-GlYQ7v3y9r6Kw5g9SahXBwsWUzp19Tvtq-I-YV85q
1NB11rxQWd-I5XRX3-TanevuRPr50BodiMQp8pb26gljYFkU_EQxn-
-RULHzo-ed9E9gXLK4DxVngp-OpQ_q29pk5xWHojP09QF1hVChixRx
59ehik",
"dq": "CLDmDgdhylic9o7r84rEUVn7pq6F83Y-1BZw5NT-TpnoZKF1pEr
AMVeKZFL14HI6I5s0L5MW0isOFbwXTxYVWZdm6s16c51tbwQGC3gnjK
bi_7k_vJGnHxHxPa2X2vpN-zyEkeDEruf-ry4c_Zl1Cq9AgC2yeL6dkK
TlcYF8",
"qi": "3PiqvXQNozWee-sBv2q21289pX9CQF3VWqPzKKnqQp7_Tuq6o-N
ZBKClQsf3HaEGBjTVjJs_JcKb-TRXvaKe-72Maqj8VfBdykssbuONKDDh
jJ-GtisesAVT7dcH0cfwXqFUPhQ7FoCrjfJ6h6ZEpMF6xmujsq4Mpp
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.

```
{
  "kty": "oct",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037",
  "use": "sig",
  "alg": "HS256",
  "k": "hJtXIZ2uSN5kbQfbtTWNbpmhkV8FJG-Onbc6mXaCcYg"
}
```

Figure 5: HMAC SHA-256 Symmetric Key

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

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

This example illustrates a symmetric key used for encryption.

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

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

Figure 6: AES 256-Bit Symmetric Encryption Key

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

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

4. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

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

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

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

SXTigJ1lzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH1vdX1gZG9vc3ci4gWW91IHh0cHJvZ3M9Y29tOjQwOTk2MDA1MzU0MDM2MzEwNzEsIHZhbHVlOjEwMDA0OjQ0MDg5MDAwMDgwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9

Figure 10: JWS Protected Header, base64url-encoded

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

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywyg229pbmcb3V0IH1vdXIgZG9vcgi4gWW91IHNOZXAgb250byB0aGUgcm9hZ2VwIHlvdSBk
b24ndCBzZWVvIHlvdXIGZmVldCBrZWVwIHlvdGh1cmXigJlzIG5vIGtub3dpbmcb2hlcmUg

Figure 11: JWS Signing Input

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

Figure 12: JWS Signature, base64url-encoded

4.1.3. Output Results

The following compose the resulting JWS object:

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

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZ
hhbXbsZSJ9.
.SXTigJ1zIGEzGFuZ2Vyb3VzIGJ1c2luZXNzLCBgcmsg9kJbywgZ29pbmcgb3V0I
HlvdX1gZG9vci4gWW91IHN0ZXAgb250byB3aWUgcmsg9hZCg9aW5kIHBocmV0
b3dpbmcgd2hlcmUgeW91IHZpb2h0JGJ1IHN3ZXBXOIG9mZiB0by4.
.MRjdkly7_-oTPTS3AXP41iqIQKaa80A0zmTuV5MEaHoxxw2e5CZ5N1KtaineFoMk
ZopdHM1O2U4mwzDQx996ivp83xugl1I7PNDi4wnB-BDkoBwaA78185hX-Es4J
IwmDLJK3lfWRa-XtLORnluYv746iyTh_qHRD68BntuSNCrUCTJDT5aAE6x8w
W1Kt9eRo4QpocSadhHFXnt8Is9uzpERV0ePPQdLuW3IS_de3xyIrDaLgdjluP
xUAhb6L2aXic1u12pduQ0KLUQSE_oI-ZnmKJ3F4uO2Dnd6QZWJushZ41Axf_f
cIe8u9ipH84ogoree7vjB5y18kDquDg

Figure 13: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJ1zIGEzGFuZ2Vyb3VzIGJ1c2luZXNzLCBgcmsg9kJbywgZ29pbmcgb3V0I
    HlvdX1gZG9vci4gWW91IHN0ZXAgb250byB3aWUgcmsg9hZCg9aW5kIHBocmV0
    b3dpbmcgd2hlcmUgeW91IHZpb2h0JGJ1IHN3ZXBXOIG9mZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZ
        hhbXbsZSJ9",
      "signature": "MRjdkly7_-oTPTS3AXP41iqIQKaa80A0zmTuV5MEaHoxxw2e5CZ5N1KtaineFoMk
        ZopdHM1O2U4mwzDQx996ivp83xugl1I7PNDi4wnB-BDkoBwaA78185hX-Es4J
        IwmDLJK3lfWRa-XtLORnluYv746iyTh_qHRD68BntuSNCrUCTJDT5aAE6x8wW1Kt9eRo4Qpo
        cSadhHFXnt8Is9uzpERV0ePPQdLuW3IS_de3xyIrDaLgdjluPxAhb6L2aXic1u12pduQ0KLUQSE_oI-ZnmKJ3F4uO2Dnd6QZWJushZ41Axf_fceIe8u9ipH84ogoree7vjB5y18kDquDg"
    }
  ]
}
```

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

```
{
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHRlc3QgQXV0aW9uIGFydGguY29tIHN0cmluZw==",
    "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6Imlub3R5ZS1JbnRFMExFQjEuRjgwMjMxMDQ4OTY3OSB0cmFjdGVzc1RldiJ9",
    "signature": "MRjdkly7_-oIPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2HeCZ5N1Ktai0FmKZopdHM1O2U4mwzJdQx996ipv83xugl1I7PNDI84wNBDkoBwA78185hX-Es4J1wmdLJK3lfWRA-XtLRnltuYv746iYTh_qHRD68BNt1uSNcRUCTJdt5aAE6x8W1Kt9eRo4QPocSadnHXFxnt8Is9UzpeERV0ePPqdLuW3IS_de3xyIrDaLGdj1uPxUAbh6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Afxf_cFrw8u9ipH84ogoree7vjbU5y18kDquDg"
}
```

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 1.1.1.

4.2.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.

- RSA private key; this example uses the key from Figure 4.

- "alg" parameter of "PS384".
4.2.2. Signing Operation

The following is generated to complete the signing operation:

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

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

Figure 16: JWS Protected Header JSON

```
eyJhbGciOiqJQUsM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZX
hhbXbsZSJ9
```

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).

```
eyJhbGciOiqJQUsM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZX
hhbXbsZSJ9
.SXTigJlzIGe4zGFuZ2Vyb3VzIGJ1c2luZXNzLCB0byB0aGUgYWJjIGFyZSBk
b24ndCB0aGUgaW4gZmFzdGluZyBtZXJyaXRpb24gd2Vic29yZSBNZXR0
b3JrcGluZzEgYSB0byBlbnRpdGllcl0iQ24iLCJiI0t0cmlwdGlvbiIsI
zdHc3Ryb24tY29tZSB0byB0aGUgaXMgdGhlIHdpdGggaW4gb24gd2Vic2
9yZSBNZXR0b3JrcGluZzEgYSB0byBlbnRpdGllcl0iQ24iLCJiI0t0cml
wdGlvbiIsIzdhc3Ryb24tY29tZSB0byB0aGUgaXMgdGhlIHdpdGggaW4g
d2Vic29yZSBNZXR0b3JrcGluZzEgYSB0byBlbnRpdGllcl0iQ24iLCJiI0
```

Figure 18: JWS Signing Input

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

```
cu22eBqkYDKgI1TpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
pN6-tPid6VrkhkGgqGdqHcDp6O8TTB5dDItllVo6_1OLFpbcUruiUSMxbXU
vdvWxZg-UDBbiReQ1fz28zGWsdiNAUf8ZnyPeGFv442DNgviVJRmBqrYRX
e8P_iq7p8Vdz0TTrxUEt3lm8d9shnr21fJ78ImUjvAA2Xez2Mlp8cB65awDzT
0q10n6uiPlaCN_2_jLa eqT1qRhtfa64QQSUmFAajVKBpvYi7xho0uT0cbH510a
6GYmJUAfmWjW6oD4ifko8DYM-X72Eaw
```

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

The following compose the resulting JWS object:

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

The resulting JWS object using the JWS Compact Serialization:

```
eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9.
SXTigJlzIGEGzGFuZ2Vyb3VzIGJlczI2luZXNzLCBGcm9kbywgZ29pZXcb3V0IHY1dVXIGzG9vc14gWW91IHt0aGUqcm9hZCwgYW5kIGl1dvdSBk
b24ndCBzZWVwIHlvdXIgZ2VldCgGlcmXigJlzIG5vIGtub3dpbmcgd2hlcmlJgeW91IGl1d0dHALJH3XBHIG9mZ1Boby4
.cu22eBqkDYKgILRtpzDXGvaFfz6WGoz7fUDcfT0kko42miAh2qyBzk1xEsnk2I
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vybe3VzIGJ1c2luZXNzLCBGcm9kJbywg
  Z29pbmcgb3V0IhvldXIGzG9vc14gWW91IHN0ZXAgb25oby80aGUgcm9h
  ZCwgYW5kIGlmlHVldSBk24ndCBrZWVwIhvldXIg2mVldCwgGhlcmXi
  gJ1zIG5vIGtub3dpbmcgd2h1cmUgeW91IGlpZ2h0IGlHN3ZXBOIG9m
  ZiBoby4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJQUzIwMCIsImtpZCI6I0xhZ3JvclJhZ2dpbnNAaG9iYm1b
      24uZzWhbXZSJSJ9",
      "signature": "cu22eBqyYDKgI1TpdXGvaFFz6WGo77fUDcfT0kk0y
      42miAh2qyBzk1xEnsk2iPn6-tPid6Vrk1HKqQsGqQdHCDP608T
      fDDIt1lVo6_10LPcpcbUrhIUSMxbbXUvdvWXzg-UD8iieReQ1fxz
      8zGWsdiNAlF82nyPQ3vF4422dNqiVJRmBqrYRx8P_i2q7p8Vd
      z0TTTrxUeT3lm8d9shnr2lJT8ImUvAAe2zx2M1p8cBE5awDzToq
      I0nuii1aCN_2_jLAmiQ1qRHa64OQSUMFAajVKBpByi7xho0u
      2CebH510a6GymJUaMwJwZ6oD4ifKo8DYM-X72Eaw"
    }
  ]
}
```

Figure 21: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vybe3VzIGJ1c2luZXNzLCBGcm9kJbywg
  Z29pbmcgb3V0IhvldXIGzG9vc14gWW91IHN0ZXAgb25oby80aGUgcm9h
  ZCwgYW5kIGlmlHVldSBk24ndCBrZWVwIhvldXIg2mVldCwgGhlcmXi
  gJ1zIG5vIGtub3dpbmcgd2h1cmUgeW91IGlpZ2h0IGlHN3ZXBOIG9m
  ZiBoby4",
  "protected": "eyJhbGciOiJQUzIwMCIsImtpZCI6I0xhZ3JvclJhZ2dpbn
      NAaG9iYm1b24uZzWhbXZSJSJ9",
  "signature": "cu22eBqyYDKgI1TpdXGvaFFz6WGo77fUDcfT0kk0y
      42miAh2qyBzk1xEnsk2iPn6-tPid6Vrk1HKqQsGqQdHCDP608T
      fDDIt1lVo6_10LPcpcbUrhIUSMxbbXUvdvWXzg-UD8iieReQ1fxz
      8zGWsdiNAlF82nyPQ3vF4422dNqiVJRmBqrYRx8P_i2q7p8Vd
      z0TTTrxUeT3lm8d9shnr2lJT8ImUvAAe2zx2M1p8cBE5awDzToq
      I0nuii1aCN_2_jLAmiQ1qRHa64OQSUMFAajVKBpByi7xho0u
      2CebH510a6GymJUaMwJwZ6oD4ifKo8DYM-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

eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSIsInIi01XVerticalEllipsis
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).

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZX
hhbXBsZSJ9.
SXTigJlzIGEgZWFyZWVy3VzIGJ1c21uZXNzLCBhcm5hbmcgd2hlcmUg
```

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-JZGYN1aAjp2kgaluUIIUnC9qv
u9Plon7KRTZoNEluTVa2cmLLeJAqy3mtPBu_u_sDDyYjnAMDxXPrn7XrT01w-kv
AD890j18e2puQens_IEXBPABlsbEPX6sFY80cGDqoRuBomu9xQ
```

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:

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZX
hhbXBsZSJ9.
SXTigJlzIGEgZWFyZWVy3VzIGJ1c21uZXNzLCBhcm5hbmcgd2hlcmUg
```

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

```json
{
  "payload": "SXTigJ1zIGEgZGFuZ2VybzVzIGJ1c2luZWNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXigZG9vci4gWW91IHJvZ2V0byB0aGUgcm9h
ZCwgYW5kIGlmIHlvdSBkb24ndCBraZWF0aW9uIHlvdXIgZmVldC9m
XjJ1IG5vIGlub3dpbmcgdmUgeW91IG1pZ2h0IGJlIHN3ZXB0IjQm
ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJFUzUxMiveJhZ2dpbnNAaG9iYmloZ24uZXhhbXBsb3JJC9",
      "signature": "AE_R_YZCChjn4791jSQCrdPZCNyqXCTZHO-JZYNl
aAjP2kqalUIUIN9c9vbu9Plon7KRTzoNEuT4Va2cmL1eJAcY3mt
PBu_u_sDDyYjnAMDxXPn7XrT0lw-kvAD890j18e2puQens_IEKBp
HAB1sbEPX6sFY80cGDqoRuBom9xQ2"
    }
  ]
}
```

Figure 28: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```json
{
  "payload": "SXTigJ1zIGEgZGFuZ2VybzVzIGJ1c2luZWNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXigZG9vci4gWW91IHJvZ2V0byB0aGUgcm9h
ZCwgYW5kIGlmIHlvdSBkb24ndCBraZWF0aW9uIHlvdXIgZmVldC9m
XjJ1IG5vIGlub3dpbmcgdmUgeW91IG1pZ2h0IGJlIHN3ZXB0IjQm
ZiB0by4",
  "protected": "eyJhbGciOiJFUzUxMiveJhZ2dpbnNAaG9iYmloZ24uZXhhbXBsb3JJC9",
  "signature": "AE_R_YZCChjn4791jSQCrdPZCNyqXCTZHO-JZYNl
aAjP2kqalUIUIN9c9vbu9Plon7KRTzoNEuT4Va2cmL1eJAcY3mt
PBu_u_sDDyYjnAMDxXPn7XrT0lw-kvAD890j18e2puQens_IEKBp
HAB1sbEPX6sFY80cGDqoRuBom9xQ2"
}
```

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

```
eyJhbGciOjIuIzI1NiIsImtpZCI6ICIxOGMsYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9
```

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).

```
eyJhbGciOjIuIzI1NiIsImtpZCI6ICIxOGMsYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9.
SXTigJlzIGEqG6Fz2Vyb3VzIGJlci2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IGlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUGEW91G1pZ2h0IGJ1HN3XB0IG9mZiB0by4
```

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

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

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:

```
eyJhbGciOiJIUzI1NiIsInp7cCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVIzJmXNGJnZmNyb3NzIiwidHlwZSI6IkpvaG4gRG9lIiwiaWRldCI6Im9vcmVzcG9zdCJ9
```

```
.SXTigJlzIGEgZGFuZ2Vvb3VvIGZvcmQgY2FjaGVsZ2VuZGluZyB0byBhbmQgaXMgYW5nIHN0b3JlIHRvIHNldC1uaXN0IGludC1ucyB0byB0aGUgZnkgc2VjcmV0b3JlIGFubm90b24gaXMgdG8gZGFuZ3V0IGZvb21lbnQgb24gYW5nYW5zZW4gY2xvYmF4IG5hbWUgaW50ZXJtcGFja24gd2l0aCBtb3N0IGF0IHRoZSBzaWduZWNhdGFyc3RldCBhbmQgaXMgcmVzdG9taXR5IG9mIHNldCBwaWItc2Vjb250ZWN0aW9uIG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzdG9taXR5IG9mIHRvIG1hbmQgcmVzlJZW5jZU1pY2hsaW5rYXJ5IiwiY29uc3RyYW5jaXBzIiwiZm9ybWF0aW9uIiwidHJhY2tlbiI6M10.de
```

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

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
Z29pbmcbg3VOIHlvdXigZG9vc14gWW91IHN0ZXAgb250byB0aGUgcm9h
ZCwgYW5kIGlmIHlvdSBkb24ndCBzZWVwIHlvdXigZmVldCBrZC9u
X1giGh1cmXi
jGJlzIG5vIGtub3dpbmcgd2h1cmUgeW91IGlpZ2h0IGJ1IHN3ZXB0IG9m
ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LmNjOntz
MxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
0"
    }
  ]
}
```

Figure 35: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
Z29pbmcbg3VOIHlvdXigZG9vc14gWW91IHN0ZXAgb250byB0aGUgcm9h
ZCwgYW5kIGlmIHlvdSBkb24ndCBzZWVwIHlvdXigZmVldCBrZC9u
X1giGh1cmXi
jGJlzIG5vIGtub3dpbmcgd2h1cmUgeW91IGlpZ2h0IGJ1IHN3ZXB0IG9m
ZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LmNjOntz
MxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
0"
}
```

Figure 36: Flattened JWS JSON Serialization

4.5. Signature with Detached Content

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

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

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 7, encoded using base64url [RFC4648] to produce Figure 8.

- Signing key; this example uses the AES symmetric key from Figure 5.

- Signing algorithm; this example uses "HS256".

4.5.2. Signing Operation

The following is generated before completing the signing operation:

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

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

Figure 37: JWS Protected Header JSON

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9

Figure 38: JWS Protected Header, base64url-encoded

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

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9
 . SXTigJlzIGEgZGFuZ2VvbmFjdG9wIGFubm90ZWN0aW9uc2VydGlmaWNhdGlvbiBhZWhlc3RydWQgdG9wIGJlbm9sZSB0aGUgb24gaGFzIGZvcmVuY3kgc2hhZ2UgdG8gdGhlIG5ld3Mgb3VwZGF0ZS4gYW5kIGRldmVyb2plY3RzIG1pbmRlcjwtd2Vic29ybS50YWRtaW4ucG5n

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

```
s0h6KThzkfBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

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
```

```
s0h6KThzkfBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 41: General JWS JSON Serialization

The resulting JWS object using the general JWS JSON Serialization:

```json
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

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

```json
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTk5NDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTspqMDA7g1Md7p0"
}
```

Figure 43: Flattened JWS JSON Serialization

### 4.6. Protecting Specific Header Fields

This example illustrates a signature where only certain Header Parameters are protected. Since this example contains both unprotected and protected Header Parameters, only the general JWS JSON Serialization and flattened JWS JSON Serialization are possible. Note that whitespace is added for readability as described in Section 1.1.

#### 4.6.1. Input Factors

The following are supplied before beginning the signing operation:

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

#### 4.6.2. Signing Operation

The following are generated before completing the signing operation:

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

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

Figure 44: JWS Protected Header JSON
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.SXTigJlzIGEgZGFuZ2VybjIc2luZXNzLCBGcm9mbywgZ29pbmcgb3V0IGhlbGUsIG9mZiB0aGUgcm9hZCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IGZ2h0IGJlIHN3ZXB0IG9mZiB0by4

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": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
  Z29pbmcgb3VOIH1vdXIGZ9vc14gWW91IHN0ZXAgb250byB0aGUgcm9h
  ZCwgYW5kIGlmaH1vdSBkb24ndCBrZWVwI1vdXIGzVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcUgeW91IGlpZ2h0IGJ1HN3ZXBOIG9m
  Zib0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSo20"
    }
  ]
}
```

Figure 49: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
  Z29pbmcgb3VOIH1vdXIGZ9vc14gWW91IHN0ZXAgb250byB0aGUgcm9h
  ZCwgYW5kIGlmaH1vdSBkb24ndCBrZWVwI1vdXIGzVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcUgeW91IGlpZ2h0IGJ1HN3ZXBOIG9m
  Zib0by4",
  "protected": "eyJhbGciOiJIUzI1NiJ9",
  "header": {
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSo20"
}
```

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.

```
{
  "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).

```
SXTigJ1zIGEgZGFuZ2Vyb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IGhhdmUgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCBlYW0iIGJlIHN3ZXB0IHNldmEgYXJ0IHNpZ25hdGF0ZSB0byBzaXRlIGJlbmNoeSB0aGUgcmFiamZlZCB0byB0aGUgYmFzZSB0aGUgNmUgdGhlIGZvciBkbyBzdXJ2ZXItd2lkdGgu4K:
```

Figure 52: JWS Signing Input

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

```
xuLifqLGiblpv9zBpuZczWhNjlgARaLV3UxvhJxZuk
```

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": "SXTigJlzIGEgZGFuZ2Vyby3VzIGJ1c2luZXNzLCBGcm99bywgZ29pbmcgb3V0IHlvdXIgZG9vc2VjcmVhcnl0cyB2b3Jrc2VzIGhhcmNvbnRlbnQsIDAgZmFyZm90aWdpbmcgY29tbW92aWVzIHN0ZXJjaGljZWJlc2FydCBjYW4gbG9vbid0aGVyIGFyc2UrZ2V0IHlvdXMgYW5kIGJ1c2luZ3MgY29tcG9uZW50IGFkZ2V0IG1hL2Rva2V0IHlvdXIgZmVlY3Qu",
    "signatures": [
        {
            "header": {
                "alg": "HS256",
                "kid": "018c0ae5-4d9b-471b-bfd6-ee7314bc7037"
            },
            "signature": "xuLifqLGiblpv9zBpuZczWhNj1gARaLV30vvhJxZu"
        }
    ]
}
```

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

```
{
    "payload": "SXTigJlzIGEgZGFuZ2VybxZVzIGJ1c2luZWNzLCBGcm9kbywgZ29pbmcgb3VOIHlvdXJgZG9vc14qWW91IHNOZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIg2mVldCwgGh1cmlkGjJlIG5vIGtub3dpbmcg2h1cmUgeW91IGlpZ2hh0IGJ1IHN3ZXBiIG9mZiB0by4",
    "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
    },
    "signature": "xulifqLGiblpv9zBpuZczWhNj1gARaLV3UxvzhJxZuk"
}
```

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

eyJhbGciOiJSUzI1NiI=

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).

eyJhbGciOiJSUzI1NiI=

.SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZWNzLCBGcm9kbywgZ29pbmcgb3V0IHN0ZXAgZGV2
lvdXIgZG9vci4gWW91IHN0ZXAgY29tcG9uZyB0aGUgc2VjdGFuY2VvdXQgd2l0aCBhbmQg
bmlzdGlvbl91dGlsaXR5b3JhbmcgZ2V0IHJlc3Vtb3VuZGVyIG9mZiB0aGUgYmVzdCBi
YXNlZSBpbi4=

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

```
MIisjqtVlopa71KE-Mss8_Nq2YH4FHgicxqrrgi5NvyG53uoimir1tcdMsq-gqtrrzZc7CG6Swv2Y13TD1qHzTURL_1R22FcryNFiHkSw129EghGpwkpxaTn_THJTCglNbADko1MZBCdzwJxwqZc-1Rlp2HibUYyXSwo97BSe0_evZKdjvVKsqsIqjytkS6eAMbhMBdMma622_BG5t4sdbuCHTfj9iJnko47AIwqkZV1aIZsv33uPUQBCXbYoQJw7mxPftHmN1GoOMxR_3thmXTCm4US-xIN0yhbm8afKK64jU6_TPQHiJeQJxz9G3Tx-083B745_AfYOlnlC9w
```

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": "MIisjqtVlopa71KE-Mss8_Nq2YH4FHgicxqrrgi5NvyG53uoimir1tcdMsq-gqtrrzZc7CG6Swv2Y13TD1qHzTURL_1R22FcryNFiHkSw129EghGpwkpxaTn_THJTCglNbADko1MZBCdzwJxwqZc-1Rlp2HibUYyXSwo97BSe0_evZKdjvVKsqsIqjytkS6eAMbhMBdMma622_BG5t4sdbuCHTfj9iJnko47AIwqkZV1aIZsv33uPUQBCXbYoQJw7mxPftHmN1GoOMxR_3thmXTCm4US-xIN0yhbm8afKK64jU6_TPQHiJeQJxz9G3Tx-083B745_AfYOlnlC9w"
}
```

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": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhcdCoFZFFjfISu0CdK9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFMBObLftVpNcrqI3Jk12U5IX3utNhODH6v7Xgy1Qahsn0fyb4zSAkje8bAWz4vlfj5pCMYxvm4fgV3q7ZYhm5eD"
}
```

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

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxY1ljZmQ2LW
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).

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxY1ljZmQ2LW
V1ZjMxNGJjNzAzNyJ9.

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

s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0

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

```
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYili2mQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7gLmd7p0"
}
```

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:

```json
{
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9yZGVkZ2V0
ZmVjdW50eSB2YWxldmUgZGF0YSB0aGlzIGFsbCBwdGlvbiBiYWtlIGFib3Ig
Y29udGVudCB0bywgb24gdG8gewp0byBtdWx0aWwu",
    "signatures": [
        {
            "protected": "eyJhbGciOiJSUzI1NiI6IkluczIsImtpZCI6IjAxOGMwYWU1LT
RkOWUtNDc5YzBlOWItZTItZjBjNzQwMzEzNGI1OSJ9",
            "signature": "s0h6KThzkfBBBkBspWlhb84VsJZFTsPpqMDAg7g1Md7p
0"
        },
        {
            "protected": "eyJhbGciOiJ0a2ViY2FsbG93IiwiLi46Xy5zIGF0dCIsIi46
Xy5zIGF0dCIsIi46Xy5zIGF0dCI6IjAwMDIwMzI5OTYyMzAxNjQxNTgzMzQw
MTI5NzQ1ODUwMzE5OCJ9",
            "signature": "ARcVLnaJJaUWG8fG-8t5RElJ1gYn8HjwD0muhc
dCoFZFfjFSu0Cdkn9Ybd1mi54ho0x924D7zdc0sK7Zkhc7AFM80b
LfrTNCrqc13jkl2U51X3utNhODH6v7xgy1Qahsn0fyb4zSAjke8b
AWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD"
        },
        {
            "protected": "eyJhbGciOiJ0a2ViY2FsbG93IiwiLi46Xy5zIGF0dCIsIi46
Xy5zIGF0dCIsIi46Xy5zIGF0dCI6IjAwMDIwMzI5OTYyMzAxNjQxNTgzMzQw
MTI5NzQ1ODUwMzE5OCJ9",
            "signature": "s0h6KThzkfBBBkBspWlhb84VsJZFTsPpqMDAg7g1Md7p
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": "maxhbsmBtdQ3CNrKvprUE6n91YcregDMLYNeTAWclj8NnPU9XIYegTHVHQjxKDSHP21-F5j5SppG1wgAQzynhXvXnYvncM7Tf9gKxqNx_xAHx6f3y7s-M9F5Psc26h6kAKR4I00h9V9ly99M9P41fBUp95f5sJ9W5U9whHAllr-hoqQGpjIeI1deH7w-3ZHu3c6D0pu_LJI16hK59wbaUAmAcr58bd2p9baY7AsgsjQUBtYjA1NIHSOIlXprUdJZKUMAzVQWOPkfa60P1IQoypbAdjuM24A3J3BNaSYsE2huaTeXvZB4eZOAIyih2e_VOKVMsDrJYAVoGlVMQ",
  "e": "AQAB",
  "d": "Kn9tg0hFfTVi8uP5b9TnwyHwG5k6RE0uFdlpCGnJN7Z1E1963R7wyBoQ1PLAmpIBNTztfrheoAniVRVINCIGxAXw_1Qs461ixDtp4nETpncQksY05jMAj7-CL8lvhYyNFvYsMoVaPZMYT9T963hNMoAw7USJ_hL96Oe1mY0vHTI3FucjSM6Nff4oIEN43r2fspeEPQgrd6fphC99aq-geP1GFULmnrmdm-P8q8rkV3K1lNATExQagTtgz80s-3VD0fgQfQfB1PNniuPU0x8OpI9KDIfu_acc6dfg14nsNaJq exacerbates the GPH2afqHqSy_Fd2vPjz85bQQ",
  "p": "2DwQmZ4ZFoNq8mU3bMkRf5Ek2mizA5xE2JmInUE3sdTYKSLtaEoekX9vBzUWxHvM6UnKCIJ_21nXk8Z0ayLYHLO_2G1aXf9-unynEpUHS7HHTkIlpYACox12gVjljoxAdWNn3iEFrjzLZGSt1OH-a3Q1DQoJOJ2VFmU",
  "q": "te8L7Y4-W7IyaiqH1ExujjMqoTAleTeRbvoVLQnfLY2xIINrwDwiQ93_VF099ajPESlJa2n-6iKIE-q7mtCPOzKcFvVUyfqhRJ_XY2kfxenJINb91h2HMv5p1sk2pEiS-GPcHC6gRLko1q-idn_qyusfWVw7WAx1SVQfK8d6Et0",
  "dp": "UfyKcL_or492Vc0PzwLPrlbgL3-25wL48mISwpbOyIgd2xKTHQmyjPFA1Zg-zf9RmgJXKdRsf9rdxPTasL1WyDeC5c12S5FKdK317JVRDol1nX7x2Kdh8ErcweW4_8zXiTulT_iKZXZNU5lvMqjWBw2eXtl1psfoI0rYU",
  "dq": "IEGOcQgfpepH8FwD7mUFyRxdoOxJBCoGChY6YuKuIHGC_p8Le9MbpKFESzEaL1NlEHsf6obG615z_ayU12j2IoQ28znoUrpa9fVYNot87ACfzIG7QMu7RiPaderzi03tkVXXAdbau_9vs5rS-7HMtxkVrxSUUvY14TkX1HE",
  "qi": "K1z1ZQOYoAfZCRsl0tOVRecogoVqAnYhqgjIRgz-Msz4sCcRkmx5V2lXyy6R7El-AajgjJa1kjieGlxTTTHHD8Igaa6fGbm2Ar5uR1ihQSpSc7G17CF12sBkM7TQMN6EshYyZfXW80mIO8M6Rzuh0beLF6G9mKcIyPrBxx2BQ_mmM"
}

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

5.1.2. Generated Factors

The following are generated before encrypting:

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

```
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-_BgLOXMozKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WC1nQePFvG2K-pv51Wc9BRIazDrn50RcRai__3TDON395H3c62tIouJ4XaRvYHFj2T22G
Xfz8YA1mcc91Tfk0WXCF5Xbb71C1Q1DDH151t1pH77f2ff7xiSxh9oSevYrcG
TSLUeeCt36r1Kt30Sj7EYBQXo21N7IxbyhMAfgIe7Mv1rOT01518NQqeXXW8V1
zNmoxaGMny3YnGir5If6Qt2nBq4qDapnaAuuGUGEce1LO1wx1BpyIfqvfjOh
MBs9M8XLZ23Fg47x1GsMXdfuY-4jaqVw
```

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

The following is generated before encrypting the Plaintext:

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

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

Figure 77: JWE Protected Header JSON

eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUJlc3JhbmluMiOiJBJTMTI4Q0JDLUhTMjU2In0

Figure 78: JWE Protected Header, base64url-encoded

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

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

produces the following:

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

```
0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvG24_FNVSiGc_r
aa0HnLQ6s1P2sv3Xz11pl1_o5wR_RaSzs8Z-wnI3Jvo0mkpEEn1Dm2vDx_k80
WzJu7eZvEtiWkdyVzFhPpiyQU28GLOpRc2VvbVbK4dQKpDNTjPPEmRqcaGeTWZV
yeSUvf5sk59yJZxRuSVWF6KrNtmRd28R4mD0jHSrM_s8uwFct4r5G8TKaI0
zT5CbL5Q1w3sRc7u_hg0yKVOiRytEAes3v2kcfLkP6nbXxdC_PkMdNS-OhP78T2
O6_7uInMGhFeX4ctHG7VelHGiT93JfWDEQi5_V9UN1rhXNRyu-0fVMkZAKX3VW
i71zA6BP430m
```

Figure 79: Ciphertext, base64url-encoded

```
kvKuFBXHe5mqr41qgobAUG
```

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

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

```json
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.

laLxI0j-nLH--_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzxq0V7yk1WClnQePFvG2K-pvSlWC9BRiazDrn50RcRai__3TDON395H3c62tIouJ4XaRvYHFj2TZzGxfz8YAlmcc91Tfk0WXC2F5Xb771ClQIDH151t1pH7T7f7fxI6xh9oSeWYrcGTSLLveCt36r1Kt3OSj7EyBQXoZ1N7IxbhMAfge1e7Mv1rOTI1518NQqeXXW8V1zNmoxAGMny3YnGir5Wf6q7nBq4qDaPdnaAuUGEceclIO1wx1BpyfvgfjOhMBs9M8XL223Fg47x1GsMxdfuy-Y-4jaqVw

bbd5sTkYwhAIqfHsx8DayA

0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xz11p11_o5wR_RsSrzS8Z-wnI3Jvo0mkpEEnlDm2vDu_k80WzJv7eZVEqlWkDyvVzFpipyQU28GLOpRc2VvBbK4dQPdNTjPEmRqcaGeVZVyeSUvf5k59yJzRvWf6wKrMntmRdZ8R4mDOjHsMrM_s8uwIFcq4r5GXBTKaI0ztSCbLsQ1w3sRc7u_hg0yKVOiRyTEaeEs3v2kcfLkP6nbXdc_PkMdNS-ohP78T2O6_7uInMGhFeX4tC7GVe1HGt793fWDEQi5_V9UN1rhXNrrYu-0fVMKZAKX3VVi7lzA6Bp430m

kvKuFBXHe5mQr41qgobAug
```

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

```
{
  "recipients": [
    {
      "encrypted_key": "laLxI0j-nLH-_BgLOXMozxky9gyffy2g7dvqzfTihJBuuzxg0V7yk1WClnQePFvG2K-pv5lWC9BR1azDrn50RcRai_3TDON395H3c6t1ouJj4XaRvYHFj2Z2zGXfas8YAlmcc91Tfk0WX
      C2F5Xbb71C1Q1DDH151t1pH77f2ff7xiSxh90SewYrcGTSLUueetC36r1Kt3OsJ7EyBQXo21N7IxbyhMAfgIe7Mv1rOTOI518NOqe1vW8
      V1zNmoxaGMny3YnGir5Wf6Q2t2nBq4qDaPdncAuUUGEceci1OI1wx
      1BypIfgYfvoMBS9M8XLZ23Fg47x1GsmXdfuY-4jaqVv
    }
  ],
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdg9uLmV4YW1wbGUiLCJ1bnMiOiJJBMTI4Q0JDLUhTMjju2In0",
  "iv": "bbd5SkYNw3IgfHs8DayA",
  "ciphertext": "0fys_TY_na7f8dWSfXLiYdHaA2DxUj67ieF7fcVbIR62
      JhJvGZ4_FNVSiGc_raa0HnLQs1P2sv3Xz1p1l_05rRsSrS8Z-wn
      I3Jvo0mkpEEnlDmZvDu_k8OWzJv7e2VeEqiWKdyVzPhPpiyQU28LQpc
      2VbVbK4QKpDNTjPPEmRqcaGETZVYseSUv5k59yJxzRuSvWFf6KrNtm
      RdZ8R4mDOjHSrM_s8uw1Fcq4rS5X8TkaI0zT5GbL5QW3sRc7u_hg0y
      KV0iRytEAEs3v2kcfLkP6nbXdC_PkmNdNS-ohP78T2067uInMGrFeX4c
      tHG7ve1HGiT93JfWDEQ15_V9UN1rhXNrYu-0fVMzZARX3VWl7zA6BP4
      3m0",
  "tag": "kvKuFBXHe5mQr41qgobAUG"
}
```

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

```json
{
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm4YW1wbGU1LCJ1bmMiOiJJBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "laLxI0j-nLH-_BgLOXMoKxmy9gffy2gTdvqzfTihJBuuzg0V7yk1WClnQePFvG2K-pvS1Wc9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4xArvYHFj2TZ2GxFz8YAIgcc91Tfk0WXC2F5Xbb71C1Q1DDH151t1pH77f2ff7xiSxh9oSewYrcGTSLUEecT36r1Kt3O8j7EyBQoxZlN7IxbxyhMAfgIe7Mv1rOTOI5I8NQqeXXW8VlzNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdauuGUEeceleI0lwx1BpyIfgvfj0hMBs9M8XL223Fg47X1GsMXdfuY-4jaqVw",
  "iv": "bbd5sTkYwhAIqfHSx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSFXLiyDhaA2DxUj67ieF7fcVbIR62JhJvGZ4_FNVSigC_raaOHnLQ6s1P2sv3Xz1l1p1_o5wR_RsSrzs8Z-wnI3JvO0mpkEEnlDmzVu_k80WzJv7eZEiqiWKdyVzhPpiyQU28GLOpRc2VbVbK4dQKpO7jPPEmRqcaTeWZtVeSUv5f59yJZxRuSVWf6KrNtmRz8R4mdOjHsrM_s8uwIFcqt4r5GX8TkaI0zT5CbL51w3sRc7u_hg0yKVoIbytEAE3v2kcfLkP6nhXdC_PkmDNS-obP7872O6_7uInMGrhFeX4cTHG7VehHGI793JfWDEq15_V9UN1rhXNrYu-0fVMk2AXX3Vwi71zA6BP430m",
  "tag": "kvKuFBXHe5mQr41qgobAUg"
}
```

Figure 83: Flattened JWE JSON Serialization

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

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

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

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

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

The following are supplied before beginning the encryption process:

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

```json
{
    "kty": "RSA",
    "kid": "samwise.gamgee@hobbiton.example",
    "use": "enc",
    "n": "wbduxI55Vaanz2XPYz5hdmv2XhvgAhoxukAnzf2-5zVUxa6prHRr
        I4pqP1AhqojRl7ZFyTww5MN6rg2pAHiJlhV4yS9w0iBioZB11XP2e-C-Fy
        XJGcTy0h4kQWlrFh7m426W7Yv0Yr4Rga90uLGF6GqGr2Laroh1WCPnk
        Nrg71s2CunZSQBIPGjx0km4y2t1_vWg6nL2G9p1xYj5YlBdxz30ExeSt
        sqo571utNfoUTU8E4qd3J3U1DIt0vKpGswM1mnmmJw7sXRTtBcivR4M
        5q2tdw+74WuR4779ubD5nAlMv2S66-RCpFAzWSKxtBDnFJSDGIU
        e77zizj1nms0Xg_yPub_U01inh0e8c5FCf1hACpWg8schr0BeNqhibD0
        FskYpUc2L2C5JA2TaPF2da67dg1TTs_CFupf32kWGrCEIlgrpxKhcWNYQb
        868-HozjH2c7qtaU8Z6rV5t6uTa-TpkcjJfNcF1I3h3b8mb-H_ox3sFjgB
        SAjLKyoeqFkTPvXh0d90knwq6f6V9q6UC18-T01jMFrFVWXuXnfh0
        OnzW6hHSSZdJc9WcuV9szUVn54slQ9w1cYWF3g5qFDxQKis99gcaDa
        1CAwM3yEBizuNec5d3artHdb1xEB_HcH5eYbghbMjFgasvKn0aRsnT
        yC0xhWb1s0lZ2E",
    "e": "AQAB",
    "alg": "RSA-OAEP",
    "d": "n7fzJc3_WG59VEOBTKayzu5MM780OJquzJn_Kb8h1LOG2520aAT4Bx
        cc0xQn5oZe5uScIw9g91Oc07jxVpcmzqjaz2g1nirjcWZ-oBtVkJ7gCAWq
        -B3qBHF3i3lKbko6rzjHajjcy3ySHbysy4_WerrXg4MDNE44Y0yj68TcXZ
        2LYQ80rukF5t7xWm0i1e1x5GtVnQn5n6tXEUcvwefmn4v0eEogLx9E
        A-KMc4AjT1sXzQXIQVH4W8G7V_7wV_Hr2yU7mYcHc2hKrvp9E700ok8076
        Dkh0s8v40U2Lwa101UX98mkogwQ58A_Y21BYbVx1_s14PspEqbgh-nqij
        h1lf0G0G4Dn5flhLnc6wItw7CPzLtInMazeCWAG7ZlFlv-R9flIv9j26r7r-
        MSh9sbzuiiHNgrGJd_jfR1MHa0184fFK16bcq11JwXVPvhNZ001yD
        F-1L1QnqUYSePF6X3a2SO6kgBRiquE6EvLsYIdpJq3kJDsGlL8M01L
        oomgjXuULw_GWE0G0u2gpllyzm-90Q0UyhEf1uhSR8aJAQWAIFIMwH5W
        _IQT917-yirndr_2Fw9g_iLUGMsGza7aOGzzFp1jry6-z_Ty0uBG0-28
        S_aWvjiyUcAlp8AUy4KjBZ-7CWH32fGWK48j1t-zwomrwjL_mhnsPbG0c
        9SwagRzi-K8gE",
    "p": "7_2v3OQzZ1PFCyHyfYLkABQ3XP85Es4hCdwCkbDeltaUXgY919etKgh
        vM4Hk0rvb01kYVULFmXKCDtpi-zLCKAdXKx4K3PtSbtzld_X39nsls
        a_Q2WpxBk_IRF7YfdKUdMz94pUHUGFgj7nr6NNxxpHhSHNFElzD_AC3m
        Y46J961Y2LRnreVwAGNW53p07Db8yD_92pa97vq2Odgqybh9q6uma-
```
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.
Performing the key encryption operation over the CEK (Figure 85) with the RSA key (Figure 84) produces the following Encrypted Key:

\[ \text{Encrypted Key} \]

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.

\[
\begin{json}
{ 
    "alg": "RSA-OAEP", 
    "kid": "samwise.gamgee@hobbiton.example", 
    "enc": "A256GCM"
}
\end{json}
\]

Figure 88: JWE Protected Header JSON

eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdvYmJpdG9uLmV4YW1wbGlhbmMiLCJleHAiOjE2MzAwMDAxMDMwMSwiYWN0aW9uIjp7ImVuZFwiOiJBMjU2R0NNIn0

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_t2m1GXk1SggBdpACm6UJuJowOHCSytjqYqRL-I-soPlwqMUF4UgRWWeaOGNw6vGW-xyM011TyrxXfVzIIaRdhYtEMRBvBWBwEwP7ua1DRfvaOjgZv6Ifa3brcAM64d8p51hhNcizPersuhw5f-pGYzseva-TUaL8iWntcs-Swy7SqmRkfhdJwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML7Cc2Gxgv17zqWo0YIEc7aCf1LG1-8BboVWFdZKLK9vNoycrYHumwzKluLWElbSVmaPpOs1Y2n525DxDfWaVFUfKQxmMF56vn4B9QmpWAbnypNimbM8zV0w
```

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:

```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpcc2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YW1wbGUIClJbmMiO1JBMjU2R0NNIN0
```

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

```
{"recipients": [
{
"encrypted_key": "rT99rwrBTbTI7IJM8fU3Eli7226HEB7IChCxNu
h71Ciud48LxeolRdttFF4nxQibeY015S_PJsAXzWXxtDePz9hk-Bb
tsTBqC2uSoDwjC9NhNupNNu9HIVftDyucvI6hvALeZ0GnhNV4
v1zx2k7Q1D89mAzfW-_kT3RkuorPD-CpBENfIHX1Q58-Aad3FzM
uo3FnhbEUyXakLXYaI5BUXQsupM4A1GD4_H4Bd7V3u9h8Gkq8B
px1KdUV9ScfJQTcYm6eJBz3aSwIA4T3-dwWpuBohRQXBoJsZ13
asnuHTMtp2RIIIfux5BC6huIVmY7kzV7W7aIUrpYm_3H4zYvyMeq
5pGqFmW2k8zp087TR1z7p2fPYDSXZyS0CfKKkMozT_qiCWZTSz
4duYnt8hS429sGthXn9uDqd6wycMagnQfOTS_lycTwmY-agQVWDKh
jYNRf03NiwrRbt5BE-tOdfWcASQj3uuAgPrO2AWBe38UjQb01lVx
1SpvyyZ3Wfc7WOJYaTa7A8Drn6MC6T-xDmMuxCO77S2rscw51QQU
06MvZ1Fot0UvfKba03cxA_nIBh1Mjy2OTXQmmpDTr6Cbo8a
KaOnx6ASE5Jx9p9bPnmO0KH35j_Q1rQhDWUN6A2Gg8fIayJ69x
EdHAvCRGrn3ovoEI2ozDRs"
}
],
"protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpIjoiCI6InNhXdbpc2UuZ2
FtZ1ZGhrYmJpdG9uLmV4YW1wbGUlCJlbnMiOiJBjMjU2R0NNIn0",
"iv": "-nBoKLH0yK1LZPSI9",
"ciphertext": "54k2cnGn8rSSw3IDol7uyySkqeS_t2m1GXklS9BdpACm6
UJuJowOHC5ytjqYpLj-1-s0PlwqMlM4UgVRWeagOGNw6vwW-xZM01LTVx
rXFVZlIaRdhHtEMRBBwWbEw7u1DRfvaOjgZv6Ifa3brcAM648p1h
hNciPvshuh5f-pGYzsev-TuAL81Wntcc-sSwy7QSqmKphDjvwbuxz
6kFovEjgj64X1It5s76GI0p5fnbYLGALQ1uML7Cc2GxgY7zWqWoY1Ec7a
Cf1LGl-8BBoOvVWfdZKL9vNoycrYHumwzKlulWcBvMpsO1Y2n52Dx
DfWv4FUFkQxM5f6vn4B9QMPwAbnypNimbM8zV0w",
"tag": "UCGiqJxhBII3IFVdPaIHVa"
}
```

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

```
{
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpdGtc2Ui49
  Ft2Z2V1QGhvYmJpdG9uLmV4YW1wbGU1LCJhbmMiOiJbMjU2RNINi0",
  "encrypted_key": "rT99rw8BT7I71JM8U3EEl7226HEB7IcoNcXWuh71C
  iud48lXloe1Rt8FF4nzQ1beY015S_PJaAXzwSTDeFZ9hkBtstTBqC2U
  aPDwC9NhNun9u9H1VFt8yuc8v16hvALez60GnhNV4v1zxi3k701D89
  mAfzw-_kT3tkuorpDU-CpBENf1HX1Q58-Aaad3FZMuoS9buEP2yXaKL
  XYa15BUXQsupM4A1GD4-H4Bd7V3u9h8Gkg8BpxKdUV9ScfJQTCy6neJ
  Bz3aSwIAK4T3-dwWp8b0fRQXBoJsJzSlsasuHtVmt2pKlIffux5BC6huI
  vmY7kz7W7aIUrpyYm_3H4zYyyMeqSgPqFmW2k8z0p878TR1x7pZfFyD
  SXZyS0CfKkKmMozT_qiCwZTS4duYnt8hs4Z9sGt45y9n5uD96wycMcagnQ
  fOTs_lycTwmY-aqWVDK1hjYNRF03INwRtb5BE-t0dFWCASQj3uuanPGrO
  2AWEbe38ujQbo1vXn1SypYLV3Wfc7WOJYaTa7A8DRn6MC6T-xDrmmXcG0
  7S2rscw51QQU06MvZT1FotOvufKBA03cxA_nIBlHLMjY2kOTxQmpD
  Tr6Cbo8aKaOnx#ASE5Jx9paBpnNmoOKH35j_QlrQhDWUN6A2G8iFayF
  69xEdHACGZgozN3woE1ozDRs",
  "iv": "-nBoKLH0YkLZFSf9",
  "ciphertext": "o4k2cnGN8rSSw31DolYuySkqes_t2m1GXk1sgBdpACm6
  UJuJowOHc5ytjqYgRL-I-soPlwqMuf4ugRWWeaOGNw6vGw-yxM01ITyX
  rXVizIAaRhYe9kMRBvWbEwP7u1a1DrfvaOjg2v6Fam3bAcMA64d851h
  hnczPersuwv5-pgYzseva-TUaL8iWnctt-sSv75QmrKkhJdvjbx0fz
  6kFovEqj64XII5sTEGlp5fnbYGlL1QULM7C2Xgy17qzW00vIEcSa
  CFlL1G-8BboWFd4KLKv9noyrCHumwXKvLWEb5VmaP0sly2z525Dx
  DfWaVFUFXQxMF56vn4B9QMnNAbnypNimbM8zV0w",
  "tag": "UCGIqJhXhB13FVdPa1HHVA"
}
```

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-256-KeyWrap) key encryption algorithm with the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

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

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

5.3.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

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

```
{
  "keys": [
  {
    "kty": "oct",
    "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
    "use": "enc",
    "alg": "A128GCM",
    "k": "XctOhJAkA-pD9Lh7ZqW_2A"
  },
  {
    "kty": "oct",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "use": "enc",
    "alg": "A128KW",
    "k": "GZy6sIZ6w19NJ0KB-jnmVQ"
  },
  {
    "kty": "oct",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "use": "enc",
    "alg": "A256GCMKW",
    "k": "qC57U_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
  }
  ]
}
```

Figure 95: Plaintext Content
5.3.2. Generated Factors

The following are generated before encrypting:

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

```
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:

```
d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdgtURtmeDV1g
```

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": "8Q1SzinasR3xchYz6ZcHA",
  "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.
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:

eyJhbGciOiJQQkVTMi1IUzUzMitBMjU2S1ciLCJwMnMiOiI4UTFTemmluYXNSM3hjaF16N1paY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pb24iLCJlbnMiOiJBMTI4Q0JDLUhTMjU2In0
.
    d3qNhUWFqheyPp4H8sjoOWsDYajoej4c5Je6r1UtFPWdgtURtmeDV1g
    VB1cZVHNoLiR3F4V82uoTQ
    23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIX86vMXqI6IRsfywCRFzLxcZBRnTvG3nhzPkJKDD7FMyxHUpDjEYCNA_XOmzg8yZR9oyjo61TF6s149F22EhzgFQCLQ_6h5EVg3vR75_hkBnuoqoM3dwejXBtIodN84PeqMb6asmas_dpSz7H10fC5n9xiz424qivBiYL1dF6exVmL93R3foOJbmk2GBQZL_SEG1lv2cQsBgeprArSaQ7bq99tT80c0H8ItJvgV08AtzXFSx9qKvC92KLKdPOMT1VJKqtV4Ru5LEVpBZXBnZrtViSogyg6AiUwaS-rCrC_D_ePOGSuxvgtrokAKYPqmUXeRdjFJwafkYEkiuDCV9vWGAi1DH2xTafhJwcmywIyzi4BqRpmdn_N-z15tuJYYuvKhjYv6ihbsV_k1hJGPAXJ6UpmwC4PTQ2izEm0TuSE8oMKdTw8V3kobXZ77ulMwDs4p
.
0HlwodAhOCILG5SQ2LQ9dg

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

```
{
  "recipients": [
    {
      "encrypted_key": "d3qNhUWFqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdgtURtmeDV1g"
    }
  ],
  "protected": "eyJhbGciOiJQQkVTMi1IUzUz8l8jIiwiaF8jIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "iv": "VBiCzVHNoLiR3F4V8uTQ",
  "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXysTHAJLZ2nsnGIv86vMXqIi61RsfywCRFzLxEczBRnTvG3NhzPp0GDD7FMyXhUhpDjEYlCNA_X0mg8y79oyj061FF614q9FZ2EhzgFQCL0_6h5EVq3vR75_hkBsnuoqoM3dwejXBlIodN84PeqMb6asmas_dbSz7H10FC5ni9xiz424giB1YLldF6exVml93R3foOJbmk2GBQZL_SEGl1v2cQsBegprARsaQ7Bq9ttT80coH81tBj9v08AtzXFFsX9qKvC982KLDkFQMT1VJKkqtV4Ru5LEvPzXhZrvtViSogyg6AiwaS-rCrcD_eP0GuxvxqroAhKYQpmXUeRdjJwafkYEikuDCV9wGAI1DH2xTafhJwcyiwIyzi4BqRpmdn_N-zl5tuJYyuVhjKv6ihsb_v_k1hJGPgAXJ6wupaMwC4PTQ2izEm0tuSE8oMKdTw8V3kobXZ77ulMws4p",
  "tag": "0HlwodAhOCILGSSQ2LQ9dg"
}
```

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

```
{
  "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU21ciLCJwMnMiOiI4UTFTemluYXNSM3jaF6N1paY0hBiIwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pz241LCJlbnMiOi1JBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "d3qNhUWfqheyPp4H8sjOwSaYajoej4c5Je6r1UFpWdgtURtmeDVlg",
  "iv": "VBizVHNoLiR3F4V82uoTQ",
  "ciphertext": "Z3i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjrUHXysTHAJLZ2nsnGIX86vMXqi6IRsfywCRFzLxEc2BRnTvG3hzhPkoGDD7FMyXuUHpDjEYCNA_Xomzug8yZ9oyjo61TF6si4q9F2 EzgFQClO_6h5EVg3v975_hkBsnuoqoM3dwejXBiTiodN84PeqMb6asmas_dpsSsz7H10fc5ni9Iz424givB1YldF6exVmL93R3f0o0Jbmk2GBQZL_SEG1lv2cQsBgeprARsaQ7Bq9t9T80coH8itBjgV08AtxXFSsx9qKvC982KLdpQMT1VJKkqtV4Ru5LEVpBZXnZrtViSoyyg6AiwaS-rCrcd_ePOGuxvgtrokAKYPqmXUE RdjFJwafyYekiUDCV9wWGAiDH2XTahJwcmwywIzyi4BqRpmnd_N-z15tuJYuvKvh1i6hsV_k1hJGFGAkJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V3kobXZ77ulMwDs4p",
  "tag": "0HlwodAhOCILG5SQ2LQ9dgy"
}
```

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.
- 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": "YU4rRZdVqmRtWos2OpDE_T5fsNIOdcG8G5FWPrTPMyxpzsSOGaQLpe2FpxBmu2",
    "y": "A8-yxCHxkFBz3hKZf1IjUYMjUhsEveZ9THuwFjH2sCNdtksRU7D5-SkgFLIETP",
    "d": "iTx2pk7w-GgJkHcEkFQb2EFyYc07RugmaW3mRrQAOUUpommT0IIdnYK2x1zh-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.

Nou2ueKp70ZXDboq9UrRwg

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.

```json
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx5l0xowrd_oYzBmaz-GKFZu4xFFFkbYoGwtEK6iuEDsQ6wNdNg3",
    "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEwZ0yQTA-JdaY8tb7E0",
    "d": "D5H4Y_5PSKZvhfVFbcCYJ0tcGZygRgfZkpsBr59Icmmhe9sW6nkZ8WfwhinUFWJg"
  },
  "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:

eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncm1ucHlNPWVyb2tAdH
Vja2Jvcm91Z2guZxhhXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6I1At
Mzg0IiwieCI6InVCbzRrsFB3Nmntiag1bDB4b3dy2F9vWXpCbfWE6LudLRlp1NH
hBRkZrY1lpV2dldEVLNnl1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcDVTR2haVMy
ZmFYdW5LW5S1UytW84S3BvWXJGRHI1eVBOVnRXNFBnRXdTa31RVEEtSmRhWT
h0YjdfMCJ9LCJlbnMiOiJBMTI4R0NNIn0

0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2

mH-G2zVqgztUtnW_

tkZuO09h95OgHjmkkrfLBisku8rGf6nzVxhRM3sV0hxXg5N576oID71pnAi_cP
WJRCjSpaAUZ5dOR3Spy7QuEkmKx8-3RCmHYMzSxaEwDdxta9Mn5B7cCBoJKB0
IgEnj_qfolhi-i-eKupOZ8aLTZGHfpl05jMWbKkTe2yK3mjF6SBAsgicQDVckc
Y9BLluzx1Rmc3ORxAm0JaHPB93ycdSDGgpgBWMVrNU1ErjkjcMqMo7_wtCex3w0
3XdLkjXiuEr2hWgEP-nkUZTPU9EoGSPj6fAS-bSz87RPCrxeZdj_iVyc6QwcaAu
07WNhjzJEPC4jVntrRjK53NgP5qP991324080Uqj4ioYezbS6vTP1Q

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

```json
{
    "recipients": [
        {
            "encrypted_key": "0DJjBXri_kBc46IcU5__Jk9BqaQeHdv2"
        }
    ],
    "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieCI6IiIsImVwayI6IiIsImNydiI6IlMtMzg0IiwieC...

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

```json
{
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVYisImtpZCI6InBlcmVncmluRvb2tAdHVja2Jvcm91Z2guZXhhXBSZISIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAtMzg0IiwieCI6InVCbzRrSFBybHltaeFRb2Nlcy1hMjUmd3d3IjwiLCJ5Ijoic3AzcDVTR2haVkMy2mFYdW1JLWUtS1uyTW84S3BvWXJGRHI1eVB0VnRXNFbRxT31RVEEtSmRhWTh0YjdFMCJ9LCJlbmMiOiJJBMTI4RONNIn0",
  "encrypted_key": "0DJjBXri_kkC461kU5_Jk9BqaQeHdv2",
  "iv": "mH-G2zVqgzUtunK_
  "ciphertext": "tk2uOO9h950gHJmkkrlBisr8Gf6nzVxhRM3sVOhXg5NJ76oID71pnAl_CPWJRCJspAaUZ5dOR3Spy7QuEkmKX8-3RCMHsYMz5XaEwDxTa9Mn5b7cCBOJKBl0gEnj_qfolihIi-uEkuP0Z8alTlGHfp105jMwbKke2yK3mjF6SBAsgicDVCkcy9BLlux1RmC0RXaMOJaHPB93Y cdSDGpgpBWfMrNU1ErkqMcMoT_wtCex3wO3XdlkXJuEr2hWgeP-nkUZTPU9EoGSpj6fAS-bSz87RCPrxZdj_iVvC6QWcqAu07WNhjzJEPc4jVn tRj6K53NgPQ5p9913Z408Ougj4icYeZbS6vTP1Q",
  "tag": "WuGzxmcreYjpHGoa17EBq"
}
```

Figure 119: Flattened JWE JSON Serialization

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

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

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

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

The following are supplied before beginning the encryption process:

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

```json
{
  "kty": "EC",
  "kid": "meriadoc.brandybuck@buckland.example",
  "use": "enc",
  "crv": "P-256",
  "x": "Ze2loSV3wrrcKUN_4zhwGhCg03Xhu1td4QjeQ5wIVR0",
  "y": "HlLldXARY_f55A3fnnQbPcm6hgr34Mpq8p-nuzQCE0Zw",
  "d": "r_kHyZ-a06rmxM3yESK84rlotSg-aQcVStkRha-iCM8"
}
```

Figure 120: Elliptic Curve P-256 Key

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

5.5.2. Generated Factors

The following is generated before encrypting:

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

yc9N8v5sYyv3iGQT926IUg

Figure 121: Initialization Vector, base64url-encoded

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

The following is generated to agree on a CEK:

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

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "mPUKT_bAWGH1hg0TpjjqsP1rXWQu_vwVOHHtNkdYoA",
  "y": "8BQAsImGeAS46fyWw5MhYfGT0IjBpFw2SS34Dv4Irs",
  "d": "AtH35vJsQ9SGjYf0sJayXQKtPH3FjZHmEtSk6N8cM"
}
```

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_bAWGH1hg0TpjjqsP1rXWQu_vwVOHHtNkdYoA",
    "y": "8BQAsImGeAS46fyWw5MhYfGT0IjBpFw2SS34Dv4Irs"
  },
  "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:

eyJhbGciOiJFQ0RILUVTTiw2IjoibWVyaWFkb2MvYnJhbmrR5YnVja0Blw
NrbGFuZC5leGFtGlcIiwizXBrIjpm7nt0eSI6IkVDIiw3Y2IjoicGUyNTYi
LCJ4IjoibVBVS1RFykFXR0hJaGcwVHBqanFWc1AxclhUXVfnddW0tIvE5rZF
lvQSIisinkoi4QlFbc01tR2VBUzQ2n1XdzVNaF1mR1RUME1qQnBgdzJTUzMT
RHY0SXJzn0sImMuVYi61ExMjhDQkMSFMyNTYiFQ
.
.
yc9N85sYyv3iGQT926IUg
.
BoDlwPnTpyq-ivjmQvAYJLb5Q61-F3LIqQomlz87yW4QPKbWE1zSTEFjdFhU9
IPrOSld4m14m7IDFwax-1XvHtdl4t4W14RXGMesDlqAYtskTTmmzNA-q4F_e
vAPumw1o-2ZG45Mnq4uhMfM_9rBtWolqZSF3xGNNKpOMKFI1818wjrli7-
IXyjrlKQsbbhqRzkv81cy6aH14j0j3C-AR21e1r7URuhArM79-y8soZU01lwI
-sD5PZ314NDDCe1iXkXoRAExsKJMySMPOeRb2N1i5UL4mYpvdKIdwyzdG65KqVw7
MsFfI_K767G9C9Ampz3gKZD0DyU1mn0W5Lmyx_yJ-3AROq8p1WZBfG-ZyJ6l
95_JG2m9Csg
.
WCCKna-x4BeB9hIDIFuhg

Figure 128: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{"protected": "eyJhbGciOiJFQ0RILUVTTiw2IjoibWVyaWFkb2MvYnJhbmrR5YnVja0BlwNrbGFuZC5leGFtGlcIiwizXBrIjpm7nt0eSI6IkVDIiw3Y2IjoicGUyNTYiLCJ4IjoibVBVS1RFykFXR0hJaGcwVHBqanFWc1AxclhUXVfnddW0tIvE5rZFlvQSIisinkoi4QlFbc01tR2VBUzQ2n1XdzVNaF1mR1RUME1qQnBgdzJTUzMT0RHY0SXJzn0sImMuVYi61ExMjhDQkMSFMyNTYiFQ",
"iv": "yc9N85sYyv3iGQT926IUg",
"ciphertext": "BoDlwPnTpyq-ivjmQvAYJLb5Q61-F3LIqQomlz87yW4QPKbWE1zSTEFjdFhU9IPrOSld4m14m7IDFwax-1XvHtdl4t4W14RXGMesDlqAYtskTTmmzNA-q4F_evAPumw1o-2ZG45Mnq4uhMfM_9rBtWolqZSF3xGNNKpOMKFI1818wjrli7-IXyjrlKQsbbhqRzkv81cy6aH14j0j3C-AR21e1r7URuhArM79-y8soZU01lwIsD5PZ314NDDCe1iXkXoRAExsKJMySMPOeRb2N1i5UL4mYpvdKIdwyzdG65KqVw7MsFfI_K767G9C9Ampz3gKZD0DyU1mn0W5Lmyx_yJ-3AROq8p1WZBfG-ZyJ6l95_JG2m9Csg",
"tag": "WCCKna-x4BeB9hIDIFuhg"
}
```

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

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

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

5.6.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

5.6.2. Generated Factors

The following is generated before encrypting:

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

refa467QzzKx6QAB

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

The following is generated before encrypting the content:

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

```
{
  "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_ELQPGaYyeAB6HYGcR55919TYnSovc23XJoBcW29rH8yZOZ7Y
hLpTibjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUYkshtrMmIUAYmUnd9zM
DB2n0cRDHAAzFveJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nce4h3-FtBP-c_
BIwCPTjb9o0BbdcRDEEMJMyZBH8ySMVilgPD9yxi-aQpGbSv_F9N4IZAxscj5
g-NJsnUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHeJHXe45Uz9aKZRSIn
ZI-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios61EKgyIQ4CWao8PfMj8TTnp
```

Figure 134: Ciphertext, base64url-encoded

```
vbb32Xv1lea20tmHAdccRQ
```

Figure 135: Authentication Tag, base64url-encoded
5.6.4. Output Results

The following compose the resulting JWE object:

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

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

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOCD2ZTEzLTQ1Y2YtODY3Mi02MTFmNi1ONi0zMDJzYjM2MDMtMDQyZDIyMDQ2OTQ2OCJkY2Y3ODUzMWQ1YmIiLCJlcyI6eyJpc3MiOiJieSIsImljYyI6IjgwNzA1MDY5NjI1ZjYiLCJuYW1lIjoiTG93IiwiY2xpZW50X2lkIjoiZGVmYXVsdCJ9
```

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

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTEzLTQ1Y2YtODY3Mi02MThdIWNVNTIOM2EiLCJ1bmNoIiJ9",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPغایAB6HYGcR55919TynSovc23XJ
oBcW29rHP8yZOZG7YhLpT1bJFuvZPjQS-m0IFtVcXkZXdH_l_r_FrdbY9
HRUYkshtnrMmIUayGmUn9zMDB2n0cRDIHafVfeJUDxkUwVAE7_YGRPdc
qMyiBoCO-FBde-Nceh4h3-FtBP-c_BlwCPTjb9o0SbdcdREEMJMyZBH8
ysWMVi1gPD9yxi-aQqGbSv_F9N4IZAxsclj5g-NJsUPbik29-s7LJAGb1
5wEBtxPhvCggy53CoiKLHHeJXhex45U9aKZRSISnZI-wjsY0yu3cT4_
qO3i0-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".
{  "kty": "oct",  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",  "use": "enc",  "alg": "A256GCMKW",  "k": "qC57l_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstis8" }

Figure 138: AES 256-Bit Key

5.7.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 139.

- Initialization Vector for content encryption; this example uses the Initialization Vector from Figure 140.

UWxARpat23nL9ReIj4WG3Dleee9I4r-Mv5QLuFXdy_rE

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.

KkYT0GX_2jHlfqN_

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.

lJf3HbOApxMEBkCM0oTnnABxs_CvTWUrnZQ2E1LvYNok

<table>
<thead>
<tr>
<th>Figure 142: Encrypted Key, base64url-encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>kFpdvQ3T3H6vnewt--ksw</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Figure 143: Authentication Tag from Key Wrapping, base64url-encoded</th>
</tr>
</thead>
</table>

5.7.4. Encrypting the Content

The following is generated before encrypting the content:

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

```json
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "kFpdvQ3T3H6vnewt--ksw",
  "iv": "KkYT0GX_2jH1fqN_",
  "enc": "A128CBC-HS256"
}
```

<table>
<thead>
<tr>
<th>Figure 144: JWE Protected Header JSON</th>
</tr>
</thead>
</table>

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

- CEK (Figure 139);
- Initialization Vector (Figure 140); and
- JWE Protected Header (Figure 145) as authenticated data

produces the following:

- Ciphertext from Figure 146.
- Authentication Tag from Figure 147.

Figure 146: Ciphertext, base64url-encoded

DKW7jrb4WaRSNfbXVP1T5g

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

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

eyJhbGciOiJBMjU2R0NNSlciLCJraWQiOiIxOGVjMdh1MS1iZmE5LTrkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0cikiOiJrZlBkdVZRM1QzSDZ2bmV3dC0ta3N3IiwieXYiOiJLa1lUMEdyYxjQSGxmcU5fIiwiZGEiOiJzZVYyOWE1M0YiLCJyZWRvIjoiODAiLCJyb20iOiJODAiLCJleHAiOiJCIzI6NDIsIjI6MiwgI2h0dHA6Ly9jbi4vd3d3LmN0cy8xNjQ2MjM5OTgifQ

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

```json
{
    "recipients": [
      {
        "encrypted_key": "1Jf3HbOApvxMEBkCMoOtBnABxa_CvTWUmZQ2E1L
        vYNok"
      }
    ],
    "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDh1MS
    liZmE5LTRkOTU5YjIwNS0yYjRkZDFkNDBmMWQiLCJ0YWciOiJrZ1BkdV
    ZRM1QzSDZ2bV3dC03N3iwiiaXYiOiJLaallUMEdYXzJqSGxmcU5fIi
    wiZW5jIiojQTEyOENCQy1IUzI1NiJ9",
    "iv": "g6NjyEFMm_vm8gj6FwoFQ",
    "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFm10Ro7w7G1QiaZpI80aiVgD8E
    qoDZHyFKFBupS8iaEeVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhzWyW
    tZKX0gxKdyHlgvGmZCzlJqcpDiF8q2_62EVabr2uSc2oxFmFuIQ
    HLCqAHz51449kz7ewzZaGV3eFghpco8o4DijxaG5_7kp3h2caJRfD
    gymuxUbgWgLqaeNQaJtvJmSMFuEOSAzw9Hdeb6yhdTynCRmu-kgtO5Dec
    41T2OMZKpnxc_F1_4yDJFcbq5CiDSmA-psB2kJtxAj4UPI61oONK7z
    zFIu4gBfjJCndszfvdvG7h8GwGjY98QhrKEnk7xKZ3KCr0_qR1B-gxpNk3
    xWU",
    "tag": "DKW7jrb4WaRSNfbXVP1T5g"
}
```

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

```json
{
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJpdiI6IiIiLCJraWQiOiJrZlBkdVZRM1QzSD22bmV3dCoa3N3lwiZW5jIjoicmF0aUJlRUYiLCJ0YWciOiJrZlBkdVZRM1QzSD22bmV3dCoa3N3lwiZW5jIjoiQTEyOENCQy1IUzI1NiJ9",
  "encrypted_key": "lJf3HbOApxMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNo",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJy_IQ_byKFMt0Ro7w7GlQia2pI80aiVgD8EGoD2HyFKFBupS8iaEeViMgGmWmsuJKuosVgzR3YfzoMd3GxEm3VhxNhWyWtZXG99x6GtvBqczLjqlqcpDf8q2_62EVAbr2uSc2oaaxFmFuIQHLcqAhxy51449xtz7ewZaGV3eFqhtco8o4Di6XaG5_7kp3h2cajRfDgymuxUbwgLqaeNQaJtvJSMFUeOSAz9Hdeb6yhdTynCMru-kqt05Dec41T2OM2Knxc_F1_4yDYFcqb5CdSmA-psB2k0JxJx4j14UPI61oONK7zHFvU4gBfjJCndsZfdvG7h8Gj98QhrKEnR7xKZ3KCr0_qR1B-gxpNk3xWU",
  "tag": "NvBveHR_vonkvlfnUrMBQ"
}
```

Figure 150: Flattened JWE JSON Serialization

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

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

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

#### 5.8.1. Input Factors

The following are supplied before beginning the encryption process:
- Plaintext content; this example uses the content from Figure 72.
- AES symmetric key; this example uses the key from Figure 151.
- "alg" parameter of "A128KW".
- "enc" parameter of "A128GCM".
{  
  "kty": "oct",  
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",  
  "use": "enc",  
  "alg": "A128KW",  
  "k": "GZy6sIz6w19NJOKB-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_glx1w

Figure 152: Content Encryption Key, base64url-encoded

Qx0pmsDa8KnJc9Jo

Figure 153: Initialization Vector, base64url-encoded

5.8.3. Encrypting the Key

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

CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx

Figure 154: Encrypted Key, base64url-encoded

5.8.4. Encrypting the Content

The following is generated before encrypting the content:

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


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

Figure 155: JWE Protected Header JSON

```
eyJhbGciOiJBMTI4SlciLCJraWQiOiI4MWIyMDk2NS04MzMzMyLTQzZDktYTQzQy00MjE2MGFkOTBhYzgiLCJlbmMiOiJBMTI4R0NNIn0
```

Figure 156: JWE Protected Header, base64url-encoded

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.

```
AwliP-KmWgs2Z37BvzCefNen6VTbRK3QMA4TkvRkH0tPlbTdhtFJgJxeVmJkJLD6
1AIhnWGetdg11c9ADsnWgL56NywxwSYjU1ZEHcGkd3EKU0vjH19gT1b90qSYFfe
F01WkcTtjbYKscsiNJoKcIp1yeM030muiYSoYVJSpf7ej6zaYcMv3WwdxDF18RE
wOhNImk2X1d2Xq6BR53TSFkyT7PwVLuuq-1GwtGH1Qeg7gDT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJfqqFvETUXLAF-KjcbTS5dNy6egwYtOt8EIHK-oEsKYtZRa
a827MOZ7UGxGImvEmxrGCFeJa14slv2-gaqK0kETHkaSqdYw0FPkQZF
```

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:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbnMiOiJBMTI4R0NNIn0

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

```json
{
  "recipients": [
    {
      "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJ1bmMiOiJBMTI4R0NNIn0",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tP1bTdhTfJgJxeVmJKL61A1hnWGetdq11c9ADsnWgL56NywxSYujU1ZEHGk
d3EkU0vjHi9gTi1b90qSYFe0LwkcTtjbyKCSiNjQkIplyeM030muYSoJyVSpf7e6zaYcMv3WwdxF18REwOhNImk2Xl2dJXq6BR537SFkY7
PwVLug-1GwtGH1Qeg7gDT6xW0JqHDpN_h-puQsmthc92g0ojmJfqqFVE
UXLAF-KjceBTS5dNy6egwkYt08ElhK-oESYyTZaa87Z0M70GxGIMv
EmxrGCpJa14slv2-gaqK0kETHkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI_NKvn72b1Lw"
}
```

Figure 160: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```json
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJ1bmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tP1bTdhTfJgJxeVmJKL61A1hnWGetdq11c9ADsnWgL56NywxSYujU1ZEHGk
d3EkU0vjHi9gTi1b90qSYFe0LwkcTtjbyKCSiNjQkIplyeM030muYSoJyVSpf7e6zaYcMv3WwdxF18REwOhNImk2Xl2dJXq6BR537SFkY7
PwVLug-1GwtGH1Qeg7gDT6xW0JqHDpN_h-puQsmthc92g0ojmJfqqFVE
UXLAF-KjceBTS5dNy6egwkYt08ElhK-oESYyTZaa87Z0M70GxGIMv
EmxrGCpJa14slv2-gaqK0kETHkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI_NKvn72b1Lw"
}
```

Figure 161: Flattened JWE JSON Serialization
5.9. Compressed Content

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

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

5.9.1. Input Factors

The following are supplied before beginning the encryption process:

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

5.9.2. Generated Factors

The following are generated before encrypting:

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

bY_BDcIwDEVX-QNU3QEOrIA4pq1DokYxchxVvbEDGzIjbioOSJwc-f___HPj8u8KVFPvtAp1VE1-wZo0YjNzo3C7R5v72pV5f5X382vWjYQpqZKAyjziZOr2B7kQPSy6oZIXUnDYbVKNa4jNXi2u0yB7t1qSHTjMODf9qvrDzfTIQXnyQRuUYya4zIWG3vT0dir0v7BRHFYWq3k1k1A_gSDJqtcBF-GZxw8

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

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

5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi

Figure 165: Encrypted Key, base64url-encoded

5.9.4. Encrypting the Content

The following is generated before encrypting the content:

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

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

Figure 166: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTYzgiLCJhbCI6MCwiMyI6NDc0NTQ5MzMiLCJzIjoiMDc0NTQ5MzMiLCJzIiwiMDc0NTQ5MzMiLCJ0aCI6OTIwMzg4MDU5O

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.

HbDtosdai0YziSx25KKEeTxmlnh8L8jKMfNc1k3zmM16VB8hry57tD261jXyezSPT0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGl18blyiMQmOn9J--XhhLYg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-wSE_o2B3jDbryBKhpYA7qi3AyijnCJ7BP9rr3U8kExCpG3mK420TjOw

Figure 168: Ciphertext, base64url-encoded

VILuUwuIxALvh5X-T7kmA

Figure 169: Authentication Tag, base64url-encoded

5.9.5. Output Results

The following compose the resulting JWE object:

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

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJibmMiOiJJBMTI4R0NNIiwielWiioiREVGIN0
.5vUT2W0tQxKWcekM_IzVQwkGgz1FDwPi
.5vUT2W0tQxKWcekM_IzVQwkGgz1FDwPi
```

Figure 170: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "5vUT2W0tQxKWcekM_IzVQwkGgz1FDwPi"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJibmMiOiJJBMTI4R0NNIiwielWiioiREVGIN0",
  "iv": "p9pUpq6XHY0jFEZI1",
  "ciphertext": "HbDtOsdai10yziSx25KEeTxmwhh8L8jKMFNc1k3zmMI6VB8hry57tD261jXyey
Spt0fdLVfe6Jf5y5-JaCap_JQcb5opbmT60uWGm18blyiMQmOn9J--XhhLyg0
m-BHqFDO5iTOwxFxFMUedx7Wcy8mxgDHj0aBMG6152PsM-w5E_o2B3jDrbrYBK
hpYA7qi3AyijnC7BF9rr3U8kxExCpG3mK420TjOw",
  "tag": "VILuUwuIxaLVmh5X-T7kmA"
}
```

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

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiIJBMTI4R0NNIiwiemlwIjoiREVGIn0",
  "encrypted_key": "5vUT2WotQxKWcekM_IzVQwkGgz1FDwPi",
  "iv": "p9pUq6XHY0jfEZI1",
  "ciphertext": "HbDtOsdai1oYziSx25KEeTxmwnh8L8jKMtNc1k3zmMl6V
  B8hry57tDZ61jXyezSPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT6OuWG
  mlBblyiMqmOn9J--XhhlYg0m-BHaqFD05iTOWxPxFMu6dx7WCy8mxgDH
  j0aBMG6152PsM-w5E_o2B3jDrbrYKhUpYA7qi3AyijnC7BP9rr3U8kxE
  xCpG3mK420TjOw",
  "tag": "VILuUwuIxaLVmh5X-T7kmA"
}
```

Figure 172: Flattened JWE JSON Serialization

5.10. Including Additional Authenticated Data

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

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

5.10.1. Input Factors

The following are supplied before beginning the encryption process:

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

Figure 173: Additional Authenticated Data, in JSON Format

NOTE: Whitespace between JSON values was added for readability.

5.10.2. Generated Factors

The following are generated before encrypting:

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

75m1ALsYv10pZTKPWrsqdq

Figure 174: Content Encryption Key, base64url-encoded

veCx9ece2orS7c_N

Figure 175: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc21vbilse30sInRleHQiLC10jAiXxbImZuIix7fS
widGV4dCIsIk1lcmlhZG9jIEJyYW5kewJ1Y2siXSxbIm4iLHt9LCJ0ZXh0Iix7fS
IkJyYW5kewJ1Y2siLCJNZXJpYyIsIk1yLiIsIkJdXSxbImlkYXkiLHt9LC
J0ZXh0IiwVEEgMjk4MiJdLFsiZ2VuZGVyIix7fS
widGV4dCIsIk0iXV1d

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_ZzH76Ta1kJmYfRFgOV9MiIpnx4X

Figure 177: Encrypted Key, base64url-encoded

5.10.4. Encrypting the Content

The following is generated before encrypting the content:

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

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

Figure 178: JWE Protected Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

Figure 179: JWE Protected Header, base64url-encoded

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

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

produces the following:

- Ciphertext from Figure 180.
- Authentication Tag from Figure 181.
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": "4Yi1Q_ZzH76TaIkJmYfRFgOV9MIPnx4X"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGRkOTFhYzgiLCJ1bmMiOiJBMTI4R0NNIn0",
  "iv": "veCx9ece2orS7c_N",
  "aad": "WyJ2Y2FyZCI6W1sidmVyc2lvbiIse30sInRleHQiLCI0LjAiXxbImZuIixfSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXxbIm4iLHt9LCJ0ZXh0InIxbIkJyYW5keWJ1Y2siLCJNZXJpYWNrYWxib3JkIiwiIiJ9",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0Ui8p74ScEUJx8ygMloEFWdCzeIa668C6t8qEFiqTUEyiNkOWDnozF14T_4NFqF-p2Mx8zkKXI7oPK8KNarFbxyLDvTICqBLa-v3uzXbdB89fzOI-Lv4Pj0PAQGhrqGvrjXAmKbgkft9Ce4WeyZw8MdbHbHc-V_KWZs1rsLNgyn_gJWd_ek6LQn5RhehVaqpqf9zr8B4aq3FXBw0xvCys35PhCdaggy2kfUf120kwKnWUbgXDV1C6xLI1qHhCwXDG59weHRDQeHyMROb1joV3X_bUTJDnKBOod7nLz-cj48JMx35nc2TpbQAkFV",
  "tag": "vOaH_Rajnpy_3hOtqzvZHRA"
}
```

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

```json
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFgOV9MIPnx4X",
  "aad": "WyJ2Y2FyZCIsW1sidmVyc2lvbIIs30sInRleHQiLCI0LjAiXSxbImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4iLHt9LCJ0ZXh0IixbIkJyYW5keWJ1Y2siLCJNZXJpYWtscyIiJdXSxbImJkYXkiLHt9LCJ0ZXh0IiwibmFtZSsiZ2VuZGluZ190ZXh0IiJdLFsiZ2VuZGluZ190ZXh0IiwiVHlwZXJpb25zLmpwZyIsIm5vbmUiXV0",
  "iv": "veCx9ece2orS7c_N",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Iyf3iPppGf3pji7wNzqteJ0Ui8p74SchQP8xygMloFRWtCNe1a6s6BcEtOp8qEFiqTUEyinKOWDNofbF4T_4NFqF-p2Mx8zkKkx7oPK8KNarFbxyIDvICNqBLba-v3uzXbdBB9fzOI-LVwPjOFAPQGhrgv1rjXAmKbgkft9cB4wEyyZw8MldBhc-c-V_KWzlrsLNygon_JWJd_e6fLq5n3RehVApq9F9xRb4aq3FXBoX0Cy535fCdaggy2kfUfgL0wKNuWbgXVD1C6HxL1lgHhgXxDG59weHRDqehyMRoBljOv3X_bUTJdnKBFQod7nLz-cj48JMr33nCZTpbQAkvFV",
  "tag": "vOah_Rajnpy_3hOtqvZHRd"
}
```

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-hbx3wnqhf5F1kEYos0sHsF0H

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.

L1bCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2DM3swKkjOwQyZtWsFL
YMj5YeLht_StAn21tHmQjuuNt64T8D4t6C7kc9OCCJ1IHAo1Uv4MyOt80MoPb8
fZYbNKnqplzYJgIL58g8N2v46OgyG637d6uuKPwhAnTgm_zWhqc_srOvgiLkzyF
XPq1hBAURbc3-8BqeRb48iR1-_5g5UjWVD31gILCN_P7AW8mImFvUNXBFJK3n0
WL4teUPs8yHLbWeL83oU4UAgL48x-8dDKH23JykiVSQju-f7e-1xreHWXzWL
Hs1NqBbre0dEwK3HX_xMOLjUz777Krppgoutpf5qaKg31-_xMINmf

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": "81b20965-8332-43d9-a468-82160ad91ac8"
}
```

Figure 191: JWE Shared Unprotected Header JSON

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "jJIcM9J-hbx3wnqhf5F1kEYos0sHsF0H"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRxnB2yLQOTqjCDKV3H30oss0w3uD9DPsqLL2DM3swKkjOwQy2tWsFLYMj5YeLht_StAn2itHmQJuuNt64T8D4t6C7kC9QCCJ1IAo1Uv4Myot80MoPb8fZYbNkgplzYjgIL58g8N2v460qyG63d6uuKpwhNtGm_zWhqc_srOvgILkzyFXPq1hBAURbc3-8BqeRb481Rl-__g5UjWVD31g1lCN_P7AW8miFvUNXBPJK3nOWL4teUPS8yHlbWel83o1U4UAgl48x-8dDkH23JykbVSQju-f7e-1xreHWxWzWLh1NqBbre0dEwK3HX_xMOLjUz77Krppgegoutpf5qaKg31_-xMINmf",
  "tag": "fNYLqpUe84KD451vDiaBAQ"
}
```

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

```json
{
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "encrypted_key": "jJIcM9J-hbx3wnghf5FlkEYos0sHsF0H",
  "iv": "WgEJzsS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQQTqjCDKV3H30ossOw3uD9DPsqLL2D
M3swKk0wQy2tWsFLYmj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC90
CCJ1IHao1Uv4MyOt80MoPb8fZyBNKqp1zYJgIL58g8N2v46OgyG637d6
uuKpwhAnTGM_zWhqc_srOvgiLkzyFXPq1hBAURbc3-8BqeRb48iR1-_5
q5UjWD31giLCN_P7AW8mIfvUNXBPJK3nOWL4teUPS8yHLbWeL83o1U
4UAgl48x-8dDkH23JykibVSQju-f7e-1xreHWWzWLHs1NqBbere0dElwK3
HX_xM0Lju77Krppgeoutp5qakg3l-_xMInmf",
  "tag": "fNYLqpu84KD451vDiaBAQ"
}
```

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.

<table>
<thead>
<tr>
<th>Figure 194:</th>
<th>Content Encryption Key, base64url-encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBooAFl30QPV3vkcZlXnzQ</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Figure 195:</th>
<th>Initialization Vector, base64url-encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>YihBoVOGsR1l7jCD</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Figure 196:</th>
<th>Encrypted Key, base64url-encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>244YfO_W7RMpQW81UjQrZcq5LSyqiPv</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

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:

```
{
  "recipients": [
    {
      "encrypted_key": "244YfO_W7RMpQW81UjQRZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoV0GsR117jCD",
  "ciphertext": "qtPIMMaOBrgASL10dNQhOa7Gqrk7Eallvwh7R4TTluq-arsVCPaeFwQfzrSS6oEUNwBtxEasE0vC6r7sphVziMCVJEuRJoAHFSP3 eqQbp4lcsDSqyXjw_l3svybhHUYGyQuTmUQEDjgjJfB0ifwHIsDsRPEBz1 NmqeifVPq5GTCWFO5K_MNIQURR2Wj0AH2Ck7J2fu2HJUHLF8ExF2Z41lnmsv Ju_mviFMyiikFnsZAudISO6073yPZtL04k_lIF7WDf rb2w70gLwDxz1pcxohPVOLQwpA3mFNRKdY-bQz424KX91fz1cne31N4-8Bkmj pw-0dQJkDLOGkC445Fb_K1tlDQXw2sBF",
  "tag": "e2m0V7jVK2VpCKXS-kyg"
}
```

**Figure 200: General JWE JSON Serialization**

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "encrypted_key": "244YfO_W7RMpQW81UjQRZcq5LSyqiPv",
  "iv": "YihBoV0GsR117jCD",
  "ciphertext": "qtPIMMaOBrgASL10dNQhOa7Gqrk7Eallvwh7R4TTluq-arsVCPaeFwQfzrSS6oEUNwBtxEasE0vC6r7sphVziMCVJEuRJoAHFSP3 eqQbp4lcsDSqyXjw_l3svybhHUYGyQuTmUQEDjgjJfB0ifwHIsDsRPEBz1 NmqeifVPq5GTCWFO5K_MNIQURR2Wj0AH2Ck7J2fu2HJUHLF8ExF2Z41lnmsv Ju_mviFMyiikFnsZAudISO6073yPZtL04k_lIF7WDf rb2w70gLwDxz1pcxohPVOLQwpA3mFNRKdY-bQz424KX91fz1cne31N4-8Bkmj pw-0dQJkDLOGkC445Fb_K1tlDQXw2sBF",
  "tag": "e2m0V7jVK2VpCKXS-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.
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:

```
dYOD28kab0Vvf40DgxVAJgHcSZICSoP8M51zjwj4w6Y5G4XJQsNNIBiqyvUUA
OcpL7S7-cFe7Pio7qV_QO6WmCSa-vhW6me4bWrBf7cHwEQJdXihidAYWVajJIA
KMXmVFMV61d1Rr076DFthg2-AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDQGbw-
wqjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_oxtr9c1N1ln2MbeYS
rRicJK5odvWgkpIdkMHo4LvdhRRvzoKzlic89jFWP1nBq_V4n5trGuExtp_d
bHcGlihqc_wGgho9fLMK8JOArYLCMDNQ
```

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

- Recipient JWE Unprotected Header from Figure 205.

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

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

```
{
"encrypted_key": "dYOD28kab0Vvf40DgxVAJxgHcSZICSOp8M51zjw4w6Y5G4XJQsNN1BiqyvUUAOcpL7S7-cFe7Pi07gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXIhidAYWVajJIaKMXvFRMV6iD1Rr076DFthg2_AV0_tSiV6xSEIFqt1xnYPfmP91tc5WJD0Gb-wqjw0-b-S1laS1lQVbuP78dQ7Fa0xAVzzjHX-xvyM2wxj_otxr9c1N1LnZMbeYSrRicJK5xodWgkpIdkMHo4LvhRRvzoKzlic89jFWP1nBq_V4n5trGuExtp_-dbHcGlihqc_wGgho9fLMK8J0ArYLcMDNQ",
"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": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I6818g2b8dviRhsE2xAn2DtMRb25Ma2CX",
"y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1EjI1pOMbw9lfzZ84pbfm",
"d": "1DKHfTv-PiifVw2VBHM_Z1VcwOMxk0yANS_lQHJcrDxVY3jhVCvZPwMxJKIE793C"
}
```

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

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

produces the following Encrypted Key:

ExInT0lo9Bq8MYF6-maw5tZlgoZThD1zWKsHixJuw_e1Y4gSSIId_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": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2
         DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1Ej1pOMb
         w9lfzZ84pbfm"
  }
}
```

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

```json
{
  "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw
    _elY4gSSId_w",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "Uzdvk3pi5wKCRC11zp5_r00jeqT-I68i8g2b8mva8diRhsE2x
    n2DtMRb25Ma2CX",
      "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjpLioy3ylEj1pO
    Mbw91fz84pbfm"
    }
  }
}
```

Figure 210: Recipient #2 JSON

5.13.5. Encrypting the Key to the Third Recipient

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

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

AvpeoPZ9Ncn9mkBn

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

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

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

produces the following:

- Encrypted Key from Figure 212.
- Authentication Tag from Figure 213.
a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTps1E

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

59Nqh1LLyTlVhfd3pgRGvw

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.

```
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "59Nqh1LLyTlVhfd3pgRGvw",
  "iv": "AvpeoPZ9Ncn9mkBn"
}
```

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

The following is the assembled third recipient JSON:

```
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTps1E",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1LLyTlVhfd3pgRGvw",
    "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.
The following is generated after encrypting the Plaintext:

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

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

Figure 220: JWE Shared Unprotected Header JSON

### 5.13.7. Output Results

The following compose the resulting JWE object:

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

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

```json
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf4ODgxAJxGhCcSICoOp8M51zj
wj4w6Y5O4xJQsNNiBgyvUAAOcpL57cS-cFe7Fio7gVQ06WmCSa-
vhW6m4eBrF7cHwEQJdXihtdAYWvajJIaRMMxMfVRV6i1IRr076
DFthg2r_AVo3iV6xSEIFq1txnYPpmP91tc5WJDQG-b-wqjw0-b-S
11aS12QnbeP78rQ7Fa0zAVzzjxHx-xvyM2wjJt_oxtr9c1NIzZMbe
YSrRicJK5xodvWgkPdIdMHo4LvdhRrvzocKzlic89jFWP1nBqV4n
5trGuExtz-_dbHcGl1hqc_wGgho9fLMK8JOArYLCdMNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5t2lgo2XThD1zWKsHi
xJuwe1Y4gSSID_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvkiep5wKCRc1izp5_r00jeqT-I68i8g2b8mva8d1Rhs
E2xAn2DmrRb5Ma2CX",
          "y": "VDrryFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7pjpLoy3y1Ej
IlpOMbw91fzZ84pbfm"
        }
      }
    },
    {
      "encrypted_key": "a7CclAejo_7JSuPB8zeagXXRam8dwCfmkt9-Wy
Tps1E",
      "header": {
        "alg": "A256GCMKw",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59Nqh1lYtVIgfD3pGrGw",
        "iv": "AvpeoPZ9Ncn9mkBn"
      }
    }
  ],
  "protected": 
  "eyJlbmMiOiJBMTI4Q0JDLuhTMjU2In0",
  "unprotected": 
  "cty": "text/plain"
}
```
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".

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

Figure 222: Payload Content, in JSON Format
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiZXhwIjoxMzAwODE5MzgwLCAiLCJodHRwOi8vZXhhbXBsZS5zZWN0b3N0LmNvbS9fcm9vdC90aWRldA==

Figure 223: Payload Content, base64url-encoded

{  "kty": "RSA",  "kid": "hobbiton.example",  "use": "sig",  "n": "knRPIBDXM6fcyv5i-QHQAQ-K8gsC3HJb7FYyYaw8xhBnJa-t8q0lDkWuLgQXYT-ffWxXJv5GRlZ4GUS21fMeEgTDyTRQ3tepgKFjMJGg6Iy6fkI1N6x2gEonsnlSfzA9GJwrRtmKpbklsw-hw11USAT-AIelNqBgcFv2E5W25S_GSGBoaROVdUYxqETDgmlz5cKV42jJDz-1h4oVB07bkc6LQdHhpJUuSyXh_Er20DxXz30Ky197pcixKTS-QKXmnm8ivyRCmuxz2z0PUnd2BKC5Oi4MwALhALaLZ22k8CsRdfy-7dg7z41R0D0zeEevtup4bX4AKral4rfw",  "e": "AQB",  "d": "ZLc_TIxpE9-w-n2vBA-HWvuYPtjvxxVXClJCFpJsdea8g9RMx34qEOEtqoCy2un3C3LjTj-mju5RAT8YSSc76YJd32ZVw0U08MBBGE-i0nvgobobNk75K7-xjTJ2U72eJor9kb7Z6ZkWDdq7HFyCDhUecYcHFVc7jIL6TibVhA0FONlqLjGwVYd0rybNGKifdnpeBwYHoMwY6HM1qvnEFgP7i20YzHUT535x6jj4VKcAD7zuFkhUaivsySEW7mxZM6fj1vdjJIy9L01fIz30Xv4ckogqKF5GONU6tNMmMgADg6jIvYElE1PrX11tbCHIC1b4RW-zrprHgAQ",  "p": "yKWyonIqvMRQlBOdTi1NiCbDNUUs2Rh-pBxAd_mIkweM74-0B2iSYvMhrs8horhonV7vxCagcBAATGW-hAafUehWJxWSH-3KccRM8toL4e0q7M-idRDOBKSoe722-2CV2x_2CY3R8qpp642R13WgQGDIM4MbUK2jcy9-c",  "q": "uNd4o15V30KDzf8vFjw589plv1QVQ3NEilrlnRUPHkkxaAzDzccGgrWMwpGxFFmN3l35CqPEluS7-51VYQqOHWV1IpQXHZ7sqaGu-483A3dENCLo23FOnF45m7_2ooAstJDe49MeITQKrSIB1SKvqYvfSPTczPzCzhkkk",  "dp": "jmtTbo2qq8aoyamjhJSCnsveUXnMQC2aneQJRQkFqFq-zV2PKPKNBPvKvYF15b2-L3tM3O2W2diNDyRUWX1T7V510KwPTABSTOnTqAmYChGibXkXkd1hcrxtVXldBakC6asawxI_TzGGY2MVXzc22NcCVXH4qjSxOrf3PfHFU",  "dq": "R9FUvU8BoVZcEtkX13-5-WusE4Dhmnnde21ul3rirrBdflpq-P-iWPBbGaq9wzQ1c-J7SzCdgQjEJzv5yd2c7rnnz6kpzwBh_kmL8szAc1qsunntCJCAYz7-sGWy11JShFazPF52B41rCJ0YuEaQMrIzPYYY_oLahpmDA0Hlk",  "qi": "S8tC7ZknR6HPlTjkwtqQOPIVnRfwirR1FAviDb8NW9CrV_7F2QoUZQozmHTYawmuGFII1LVrjRep7anleWaJjxC_1b3fq_al4gH3Pe-EKihH6IMazuRTLURcThrExDbF5dybsciDnFwRUWErZ41Ne0binxyYuPqyawKd9QzwMo0"
}

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.

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

Figure 225: JWS Protected Header JSON

eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9

Figure 226: JWS Protected Header, base64url-encoded

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

dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprzDWNwIA
kIvn1qskChirjKvY9ESZUcP4JjvyFS-nqjJxYoA5ztW0yFk2cZNIPXjcJXSQ
wXPOtE-e-v4VSqgD0aKHzFxYog4N6czllKph1UlsYDS167_bLL7elg_vkjdMp5
_W515LuUYGMeh6hxQ1aIXf9EmV2JmvTMuZ-vBOWy0Sniy1Efo72CRTvmtrIf5
AR0o5MNliY3KtUxeP-SomD-LEYwW9SlkohYzMVAZDD0vVb7KVHRpeYNaK75KE
QqdCEekS_rskZS-Qtt_nlegTWh1mEYaA

Figure 227: JWS Signature, base64url-encoded

6.3. Signing Output

The following compose the resulting JWS object:

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

```
eyJhbGciOiJQUzI1NiIsInR5cCI6IkFVX29yc3Qk
eyJpc3MiOiJ0cml0ZS10b3V0cmF0aW9ucyJ9
```

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.

```
ORHSNYwN-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:

```plaintext
a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueY1VhjurCyrBg317YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FXqNjV0Grg-bi13mwy7RoYjhTkBEC6p7sYMXXx4g
zMedpiJHQVeyI-zkZV7A9matpgevAJWrXzOUsYGTtwoSN6gtUVt1Laivjvb21
O0u14YxSHV-ByK1kyeetRp_fuYJxHoKLQL9F424sKx2WGYb4zsbVIPF4ssl_e51
R7nany-25_UmC2urosNkoFz9cQ82MypZP8ggbQyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIld1-Qv6H5dMWIY7qleKpXcVl1W0_2Fe6EbqXxvIjLeZivjNkzogCq3-I
ap8JFVnMjBxjpYLT8maawolyy1XXMuinIpNcY3n4KKrXLrCcteE85m4IHMZ
a38s1Hpr56fPPseMA-Jltmt-a9iEDtOzhtxZ8AXy9tsCAZV2XBNdNG8c3KJusAa
mBK0Ywfk7JhLrDgOnJjlJLhn7TI4UxDP9dCmUXEN6z0v23W15qJIEWXJtqnblp
ymooeWAHT4e_Owbimlg0AEpTHUdZlILNs9WTX_H_TXuPC8yDDh1smsxS_X_x
pkIHkiiWJDOLx03BpqDTivpKkBYwqP2UZkcxqX2F0_GnVrNwlK7Lgzw6FSQvD00
```

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.

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

Figure 232: JWE Protected Header JSON

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkJhHMQ0ifQ

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

- CEK (Figure 229);
- Initialization Vector (Figure 230); and
- JWE Protected Header (Figure 233) as authenticated data produces the following:

- Ciphertext from Figure 234.
- Authentication Tag from Figure 235.

```
SZI4IvKhmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYWucKrBNgCe2OFMf66cSJ8k2QkxaQD3_R6OMGE9ofomwtky3GFxMeGRjtpMt90AvVLsAXB0_UTCBGyBq3C2bWLXqZlfJAaoJRUPRk-BimYZY81zVBuIhc7HSpePCpu33SzMsFHjn41P_idrJz_glZTNgKDt8znU5auKTKDN0H1ID4fuzvDyFDIAfGPyL5sVRwbIXpDgOzKsM-9CyMPgW1QNhzux_Zu13bvrJwr7nuGZs4cUScY3n8yE3AHClurgl-A9mz1X38xEau1V1814Fg9tLejdkAuQzjPbqeHQBJe4t WD5E0dQ-Mtz4NhkIWx-YKBb_Xo2zi30_1sYjKuiis7ywW-HTr_vqvFt0bj7WJf2vzB0T2dvs0GaTvPH2dyWwumUr1x4gmPuZbdwTO6ubfYSDEUE5py0d_OtWeUSyCYBKDaM7txG26qJo21gYjLfhn9zy-W19sOCZGuzgFjJPhawXHpvunj_t-0_ES96kogjJLxS1IMU9Y5xmnwZMyNc9EIwnogsCg-hVuvzyF0siRuktmI94_SL1xgM170O3phcTMxt1MizR88NkU1WkBsixMCjy1Noe7MD-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:

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkpXVCIsImJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkpXVCIsImJhbGciOiJSU0EtT0FFUCIsInIi6IkpXVCJ9

---

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

```json
{
"recipients": [
{
"encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueY1VhjucRyBq3Y7yKhCRjyjhpDO0S4E7t=rbr2F6n6KyQq-A-qtg0FXqNgV0GrG-b113myW7roYyjtkEBCE6FP7sMyMXXx4gzmzmpipjHQVye-3kZV7A9matpgevAJWTx0UysYGTtowoSN6gUVT1lailjvbjz1O0u14YXSiHV-BvYkjetsRpf_fuYjxHoKLQ19F4y24sZk2tJgYb4zBIF4s1l_e5IIR7nany-25_UmC2urosNkoFz9cQ82MypZpFgqBQJyPN-FpF4z-5o6yV6x4yzmUUF5_5J1C1d1-Qv6H5dMVY7q1eKpxpV1lW0-2FeEBqXxXvIjLeZiVjNkwzogCq3-Iap5jVFNjBxjpYLT8mauaw0y1YXMunupnc0y3n4KkXRcCcteX8m4IHMZa38s1Hprn5fFPsEMa-J1tmt-a9iEdOtZhtxz8Xy9tscAZV2XBWGN8c3kJusAamBKOYwv7kJhLrDgOnJjJLhnn7TI4UxP9dCmUXEN6z0v2315qJExNqjfnlbpymoomewAHACT4e_0wibm1gQ0EpTHUdA2iiLNs9WGXH_1_tuPC8yDohilsmxsX_xpkIHkIIHWDODLx03BpqDtvkKBywqP2UZkcxqX2Fo_GnVrnw1K7Lgxw6FSQVd00"
},
"protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6I2I1kpxVClISIsImVuYyI6IkExMjhHQ00iifoQ",
"iv": "GbXl19kZx0sxXPMa",
"ciphertext": "SZI4IvKHmwpaz1_pJQXX3hv1ANnOU4Wf9-utWYUcKrBNQeCropu33xSfHj4n1p_fldrz_g12TNgDkt8zdnUfauKTDN0H1DD4fu_vzYDIFIAgfGPy5zVRwbiXpXxGokEszM-MchMPqW1QHxzU9u318brzrwr7nuGzS4cUSyc3n8yeJAHClurgis-A9mz1X38eaulV1814F9tLejdkaUqOzjPbqehQBBj4eIw5D5E0dq-Mtz4NnhkIWxy-YKBB_Xo2zI3Q_1sYjKUuis7yWW-HTR_vqvFtoIB7WjJf2vzB0Z1T3dvsoGaTvPH2yDwwUmUr14gmPuPZbwTD6ubFYSDEeZp1y0d_DtWeUSyCYSB-Da7tXq264qJo21qYjLfnhn9zy-W19scOCZGuzgFJPhawXhvnpn_jt-0_ES96kogjJLxSi1MUY5Xmnw2ZYnc9E1iwngosGg-hVvuzyp0sIrurktmI94_4LxkgMl7o3phcTmx1lMrz88NKn1UKBsiXMCJy1NoueMD-ShDp5dmM",
"tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

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

```json
{
    "encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBVlueY1VhjurCyrBg3I7YhCERYjphDOS4E7rXbr2Fh6NyQq-A-gqT0FxqjV0Gr-bi13mwy7RoyhjTBEK6F7aMYMXXx4gzMedplJHQVeyI-zkZV7A9matpgevAJWrXzOUysYTtwoSN6gtUV1laiyjvb2100u14YxSHV-ByKkiye8rP_fYujHoKLQL9P42sKx2WGYb4zsBIFP4ss1_e5IR?nany-25_UmC2urosNkoFz9Q82Myp2P8ggbQjyPN-Fpp4Z-5o6yV64x6yzDUF_5JCIdl-Qv6H5dmVjI7y1eRpXcV1lWO_2FefEBqXxViJLeZivjNkzogCq3-Iap8yF
nmJbBjypYLTT8maawo1y1XXuininPnCoY3n4KrXLrCcteX85m4IIHZMa38s1Hpr56FPPseMA-Jltmnt-a9iETdOzhtxxz8AXY9tsCAZV2XBWNG8s3okuAamBKOYwfkJ7jHLRdgOnnji1Lhn7TI4UxDp9dCmUXEN60zv23W15qJlEXNlqnb1ymooeWAHCT4e_Owbi1mg0AEpTHUda2iiLns9WHTXrUXpC8yDDhi1smxS_X_zpkI1HWDOLx03BpqDTipvKkBYeqpP2UZckxqX2F0_GnVrNw1K7Lgxw6FSQyD00", "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ", "iv": "GbX1i9kXz00sxXPMa", "ciphertext": "SZI4IvKHMwpazl_pJQXX3mHl1ANnOU45w9-utWYUCrKBNgCe20FMf66cSJ8k2QkxaQOD3_R360MG9ofonwtky3GFxMeGRjtptMt9OAvVLaXAB0-UTCBBGyBq3c2bWLXq2lfjAaoJRUPrk-BimY8Y81zV6u7c7HsQePCpu33ZsMsFhjn4lP_idrJz_g1ZTNqKdt8zdnUPaU7KRDQ1H1D4fzvDyRfDIAqFy6s2VRw8iXpXdGoEszM-9ChMPqW1QhzuX_Su13vrjrwr7nuGZs45UScY3n8E3AHCLurgls-A9mz1X38xEauV1814Fg9tLejdkaAuQzjFbqzHqBj4e1WGD5EeD-Mc49nNhk1Wx-YKBB_Xo2zi13Q_sfYJkU1us7yWW-HTR_vqvFtObj7WJF2vzB0T3dvsogATvPH2dyWwumUrlx4qgMPzUdW706ubUYSDUEz3spv0d_0tWeUSycyYBkD-aM7Xcgk26qJ02lqYSj1fhn9zy-W19sGczGuzgFJPhawXhpvN_t-0_ES96kogJLX1MU9YSXmnoz2sMcEfwnogcs-hVuvzyP0sIructmT94_Sb1xgM170o3phcTMx91nMrz8NNU1WKBsxiXMCy1noue7MD-ShPd5dmM", "tag": "KnIKEhN8U-3C9s4tSpjSw"
}
```

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