

Internet Engineering Task Force (IETF)  
Request for Comments: 7520  
Category: Informational  
ISSN: 2070-1721

M. Miller  
Cisco Systems, Inc.  
May 2015

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

### Abstract

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

### Status of This Memo

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

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

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

### Copyright Notice

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

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

## Table of Contents

1. Introduction .....	5
1.1. Conventions Used in This Document .....	5
2. Terminology .....	6
3. JSON Web Key Examples .....	6
3.1. EC Public Key .....	6
3.2. EC Private Key .....	7
3.3. RSA Public Key .....	8
3.4. RSA Private Key .....	8
3.5. Symmetric Key (MAC Computation) .....	10
3.6. Symmetric Key (Encryption) .....	11
4. JSON Web Signature Examples .....	11
4.1. RSA v1.5 Signature .....	12
4.1.1. Input Factors .....	12
4.1.2. Signing Operation .....	12
4.1.3. Output Results .....	13
4.2. RSA-PSS Signature .....	15
4.2.1. Input Factors .....	15
4.2.2. Signing Operation .....	16
4.2.3. Output Results .....	17
4.3. ECDSA Signature .....	19
4.3.1. Input Factors .....	19
4.3.2. Signing Operation .....	19
4.3.3. Output Results .....	20
4.4. HMAC-SHA2 Integrity Protection .....	21
4.4.1. Input Factors .....	22
4.4.2. Signing Operation .....	22
4.4.3. Output Results .....	23
4.5. Signature with Detached Content .....	24
4.5.1. Input Factors .....	25
4.5.2. Signing Operation .....	25
4.5.3. Output Results .....	26
4.6. Protecting Specific Header Fields .....	27
4.6.1. Input Factors .....	27
4.6.2. Signing Operation .....	27
4.6.3. Output Results .....	28
4.7. Protecting Content Only .....	29
4.7.1. Input Factors .....	30
4.7.2. Signing Operation .....	30
4.7.3. Output Results .....	31
4.8. Multiple Signatures .....	32
4.8.1. Input Factors .....	32
4.8.2. First Signing Operation .....	33
4.8.3. Second Signing Operation .....	34
4.8.4. Third Signing Operation .....	36
4.8.5. Output Results .....	37
5. JSON Web Encryption Examples .....	39

5.1. Key Encryption Using RSA v1.5 and AES-HMAC-SHA2 .....	39
5.1.1. Input Factors .....	39
5.1.2. Generated Factors .....	41
5.1.3. Encrypting the Key .....	41
5.1.4. Encrypting the Content .....	42
5.1.5. Output Results .....	43
5.2. Key Encryption Using RSA-OAEP with AES-GCM .....	45
5.2.1. Input Factors .....	46
5.2.2. Generated Factors .....	47
5.2.3. Encrypting the Key .....	48
5.2.4. Encrypting the Content .....	48
5.2.5. Output Results .....	49
5.3. Key Wrap Using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2 ...	52
5.3.1. Input Factors .....	53
5.3.2. Generated Factors .....	54
5.3.3. Encrypting the Key .....	54
5.3.4. Encrypting the Content .....	55
5.3.5. Output Results .....	56
5.4. Key Agreement with Key Wrapping Using ECDH-ES and AES-KeyWrap with AES-GCM .....	59
5.4.1. Input Factors .....	59
5.4.2. Generated Factors .....	60
5.4.3. Encrypting the Key .....	60
5.4.4. Encrypting the Content .....	61
5.4.5. Output Results .....	63
5.5. Key Agreement Using ECDH-ES with AES-CBC-HMAC-SHA2 .....	65
5.5.1. Input Factors .....	66
5.5.2. Generated Factors .....	66
5.5.3. Key Agreement .....	67
5.5.4. Encrypting the Content .....	67
5.5.5. Output Results .....	68
5.6. Direct Encryption Using AES-GCM .....	70
5.6.1. Input Factors .....	70
5.6.2. Generated Factors .....	70
5.6.3. Encrypting the Content .....	71
5.6.4. Output Results .....	72
5.7. Key Wrap Using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2 .....	73
5.7.1. Input Factors .....	73
5.7.2. Generated Factors .....	74
5.7.3. Encrypting the Key .....	74
5.7.4. Encrypting the Content .....	75
5.7.5. Output Results .....	77
5.8. Key Wrap Using AES-KeyWrap with AES-GCM .....	79
5.8.1. Input Factors .....	79
5.8.2. Generated Factors .....	80
5.8.3. Encrypting the Key .....	80
5.8.4. Encrypting the Content .....	80
5.8.5. Output Results .....	82

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

## 1. Introduction

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

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

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

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

### 1.1. Conventions Used in This Document

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

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

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

All of the examples in this document have whitespace added to improve formatting and readability. Except for JWE Plaintext or JWS Payload content, whitespace is not part of the cryptographic operations nor the exchange results.

Unless otherwise noted, the JWE Plaintext or JWS Payload content does include " " (U+0020 SPACE) characters. Line breaks (U+000A LINE FEED) replace some " " (U+0020 SPACE) characters to improve readability but are not present in the JWE Plaintext or JWS Payload.

## 2. Terminology

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

## 3. JSON Web Key Examples

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

### 3.1. EC Public Key

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

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

```
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sYl_oJXu9
    A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
  "y": "AdymIHvOiLxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV5OhQHiraVy
    SsUdaQkAgDPrwQrJmbnX9cwlGfP-HqHZR1"
}
```

Figure 1: Elliptic Curve P-521 Public Key

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

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

### 3.2. EC Private Key

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

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

```
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sYl_oJXu9
    A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
  "y": "AdymIHvOiLxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV5OhQHiraVy
    SsUdaQkAgDPrwQrJmbnX9cwlGfP-HqHZR1",
  "d": "AAhRON2r9cqXX1hg-RoI6R1tX5p2rUAYdmpHZoC1XNM56KtscrX6zb
    KipQrCW9CGZH3T4ubpnoTKLDYJ_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-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHz_AJmSCpMaJMRBSFKrKb2wqV
wGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqkHHig3RBordaZ6Aj-
oBHqFEHYpPe7Tpe-OfVfHd1E6cS6M1FZcD1NNLYD5lFHpPI9bTwJlsde
3uhGqC0ZCuEHg8lhzwOHrtIQbS0FVbb9k3-tVTU4fg_3L_vniUFAKwuC
LqKnS2BYwdq_mzSnbLY7h_qixor7jig3__kRhuaxwUkRz5iaiQkqgc5g
HdrNP5zw",
  "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": "n4EPtAOCc9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rHVTeT
-O-XV2jRojdNhXJWTDvNd7nqQ0VEiZQHZAjMScPmaJMRBSFKrKb2wqV
wGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqkHHig3RBordaZ6Aj-
oBHqFEHYpPe7Tpe-OfVfHd1E6cS6M1FZcD1NNLYD5lFHpPI9bTwJlsde
3uhGqC0ZCuEHg8lhzwOHrtIQbs0FVbb9k3-tVTU4fg_3L_vniUFAKwuC
LqKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUkRz5iaiQkqgc5g
HdrNP5zw",
  "e": "AQAB",
  "d": "bWUC9B-EFRIO8kpGfh0ZuyGPvMNkVYWNtB_ikiH9k20eT-O1q_I78e
iZkpXxXQ0UTES2LsNRS-8uJbvQ-AlirkwMSMK1J3XTGgdrhCku9gRld
Y7sNA_AKZGh-Q661_42rINLRCe8W-nZ34ui_qOfkLnK9QWDDqpaIsA-b
MwWWSDFu2MUBYwkHTMEzLYGqOe04noqeqlhExBTHBOBdkMXiuFhUq1BU
6l-DqEiWxqg82sXt2h-LMnT3046AOYJoRioz75tSUQfGCshWTBnP5uDj
d18kKhyv07lhfsJdrPdM5Plyl21hsFf4L_mHCuoFau7gdsPfHPxxjVoc
OpBrQzwQ",
  "p": "3Slxg_DwTXJcb6095RoXygQCAZ5RnAvZlnolyhHtnUex_fp7AZ_9nR
a07HX_-SFfGQeutao2TDjDAWU4Vupk8rw9JR0AzZ0N2fvuIAmr_WCsmG
peNqQnev1T7IyEsnh8UMt-n5CafhkikzhEsrmdH6LxOrvRJlsPp6Zv8
bUq0k",
  "q": "uKE2dh-cTf6ERF4k4e_jy78GfPYUIaUyoSSJuBzp3Cubk3OCqs6grT
8bR_cu0Dm1MZwWmtDqDyI95HrUeq3MP15vMMON8lHTeZu2lmKvwqW7an
V5UzhMliZ7z4yMkuUwFWoBvyY898EXvRD-hdqRxHlSqAZ192zB3pVFJ0
s7pFc",
  "dp": "B8PVvXkvJrj2L-GYQ7v3y9r6Kw5g9SahXBwsWUzp19TVlgI-YV85q
lNIblrxQtD-IsXXR3-TanevuRPrt50B0diMGQp8pbt26gljYfKU_E9xn
-RULHz0-ed9E9gXLKD4VGngpz-PfQ_q29pk5xWHOJp009Qf1HvChixRX
59ehik",
  "dq": "CLDmDGduhylc9o7r84rEUVn7pzQ6PF83Y-iBZx5NT-TpnOZKF1pEr
AMVeKzFE141DlHHqqBLSM0WlsOFbwTxYWZDm6sI6og5iTbwQGIC3gnJK
bi_7k_vJgGHwHxgPaX2PnvP-zyEkDERuf-ry4c_Z11Cq9AqC2yeL6kdK
T1cYF8",
  "qi": "3PiqvXQN0zwMeE-sBvZgi289XP9XCQF3VWqPzMKnIgQp7_Tugo6-N
ZBKQCsmf3HaEGBjTVJs_jcK8-TRXvaKe-7ZMaQj8VfBdYkssbu0NKDDh
jJ-GtiseaDVWt7dch0cfwxgFUHpQh7FoCrjFJ6h6ZEpMF6xmujs4qMpP
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": "hJtXIZ2uSN5kbQfbtTNWbpdmhkV8FJG-Onbc6mxCcYg"
}
```

Figure 5: HMAC SHA-256 Symmetric Key

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

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

### 3.6. Symmetric Key (Encryption)

This example illustrates a symmetric key used for encryption.

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

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

Figure 6: AES 256-Bit Symmetric Encryption Key

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

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

### 4. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

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

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

Figure 7: Payload Content Plaintext

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

```
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXlzigZG9vci4gWW91IHN0ZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBZZWVwIHlvdXlzigZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcmUgeW91IGlpc2h0IGJlIHN3ZXB0IG9mZiB0by4
```

Figure 8: Payload Content, base64url-encoded

#### 4.1. RSA v1.5 Signature

This example illustrates signing content using the "RS256" (RSASSA-PKCS1-v1\_5 with SHA-256) algorithm.

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

##### 4.1.1. Input Factors

The following are supplied before beginning the signing operation:

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

##### 4.1.2. Signing Operation

The following is generated to complete the signing operation:

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

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

Figure 9: JWS Protected Header JSON

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhkbXBsZSJ9
```

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).

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhkbXBsZSJ9
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXlzigZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwVWVhZlvdXlzigZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcmUgeW91IGlpc2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 11: JWS Signing Input

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

```
MRjdkly7_-oTPTS3AXP4liQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmK
ZopdHMlO2U4mwzJdQx996ivp83xuglII7PNDi84wnB-BDkoBwA78185hX-Es4J
IwmDLJK3lfWRa-XtL0RnltuYv746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8w
WlKt9eRo4QPocSadnHFXfxt8Is9UzpERV0ePPQdLuW3IS_de3xyIrDaLGdjlUP
xUAhb6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_f
cIe8u9ipH84ogoree7vjbU5y18kDquDg
```

Figure 12: JWS Signature, base64url-encoded

#### 4.1.3. Output Results

The following compose the resulting JWS object:

- o JWS Protected Header (Figure 9)
- o Payload content (Figure 8)
- o Signature (Figure 12)

The resulting JWS object using the JWS Compact Serialization:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAAG9iYml0b24uZXh0bXBzSj9

.SXTigJlZIGEGZGFuZ2Vyb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIGZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBBrZWVwIHlvdXIGZmVldCwgdGhlcmXigJlZIG5vIGtub3dpbmcd2hlcmUgeW91IGlpZ2h0IGJlIHNOZXB0IG9mZiB0by4

.MRjdkly7\_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmKZopdHM1O2U4mwzJdQx996ivp83xuglII7PNDi84wnB-BDkoBwA78l85hX-Es4JIwmDLJK3lfWRa-XtL0RnltuYv746iYTh\_qHRD68BNTluSNCrUCTJdT5aAE6x8w1Kt9eRo4QPocSadnHXXfxt8Is9UzpeRV0ePPQdLuW3IS\_de3xyIrDaLGdjlupxUAhb6L2aXic1U12podGU0KLUQSE\_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf\_fcIe8u9ipH84ogoree7vjbU5y18kDquDg

Figure 13: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlZIGEGZGFuZ2VyYb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBzZWVwIHlvdXIgZmVldCwgdGhlcmXIgJlZIG5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNOZXB0IG9mZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvImJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "MRjdkly7_-oTPTS3AXP4lIQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmKZopdHM1O2U4mwzJdQx996ivp83xuglII7PNDi84wnB-BDkoBwA78185hX-Es4JIwmDLJK3lfWRa-XtLORnltuYv746iYTh_qHRD68BNtlusNCrUCTJdT5aAE6x8wW1Kt9eRo4QPocSadnHFXfxt8Is9UzpeERV0ePPQdLuW3IS_de3xyIrDaLGdjluPxUAhb6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_fcIe8u9ipH84ogoree7vjbU5y18kDquDg"
    }
  ]
}
```

Figure 14: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlZIGegZGFuZ2VyY3VzIGJlc2luZXNzLlCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgc29h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlZIG5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNOZXB0IG9m
    ZiB0by4",
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbn
    NAAG9iYml0b24uZXhhbXBsZSJ9",
  "signature": "MRjdkly7_-oTPTS3AXP4liQIGKa80A0ZmTuV5MEaHoxnW2
    e5CZ5NlKtainoFmKZopdHM1O2U4mwzJdQx996ivp83xuglII7PNDi84w
    nB-BDkoBwA78185hX-Es4JIwDLJK3lfWRa-XtL0RnltuYv746iYTh_q
    HRD68BNTluSNCrUCTJdt5aAE6x8wW1Kt9eRo4QPocSadnHFXxt8Is9U
    zpERV0ePPQdLuW3IS_de3xyIrDaLGdjlUPxUAhb6L2aXic1U12podGU0
    KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_fcIe8u9ipH84ogore
    e7vjbU5y18kDquDg"
}
```

Figure 15: Flattened JWS JSON Serialization

## 4.2. RSA-PSS Signature

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

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

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

### 4.2.1. Input Factors

The following are supplied before beginning the signing operation:

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

#### 4.2.2. Signing Operation

The following is generated to complete the signing operation:

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

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

Figure 16: JWS Protected Header JSON

eyJhbGciOiJIUzI1NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBzZSJ9

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).

eyJhbGciOiJIUzI1NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9

SXTigJlzIGEgZGFuZ2Vyb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcb3V0IH  
1vdXIGZG9vci4gWW91IHN0ZXAgb250byB0aGUgc9hZCwGyW5kIGlmIHlvdSBk  
b24ndCBBrZWVwIHlvdXIGZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcb2hlcm  
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4

Figure 18: JWS Signing Input

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

cu22eBqkYDKgIlTpZDXGvaFfz6WGoZ7fUDcfT0kkOy42miAh2qyBzklxEsnk2I  
pN6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDItllVo6\_1OLPpcbUrhiUSMxbbXU  
vdvWXzg-UD8biiReQFlfz28zGWVsdinaUF8ZnyPEgVF442ZdNqiVJRmBqrYRX  
e8P\_ijQ7p8Vdz0TTrxUeT3l1m8d9shnr2l1fJT8ImUjvAA2Xez2Mlp8cBE5awDzT  
0qI0n6uiPlacN\_2\_jLAeqTlqRHTfa64QqUmFAAjjVKPbByi7xho0uTOcbH510a  
6GYmJUAFmWjwZ6od4ifK08DYM-X72Eaw

Figure 19: JWS Signature, base64url-encoded



### 4.2.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JWS Compact Serialization:

eyJhbGciOiJQUzZM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXh0bXBzSj9

.

SXTigJlZIGEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIGZG9vci4gWW91IHN0ZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBzZWVlIHlvdXIGZmVldCwgdGhlcmXigJlZIG5vIGtub3dpbmcd2hlcmUgeW91IGlPZ2h0IGJlIHN3ZXB0IG9mZiB0by4

.

cu22eBqkYDKgIlTpZDXGvaFfz6WGoZ7fUDcfT0kkOy42miAh2qyBzk1xEsnk2Ipn6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDIt1lVo6\_10LPpCbUrhIUSMxbbXUvdvWXzg-UD8biiReQF1fz28zGWVsdinAUf8ZnyPEgVFf442ZdNqiVJRmBqrYRXe8P\_ijQ7p8Vdz0TTTrxUet3lm8d9shnr21fJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiPlACN\_2\_jLAEQTlqRhtfa64QQSUMFAA\_jVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw

Figure 20: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLlCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
    ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2
        dpbnNAAG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "cu22eBqkYDKgIlTpzDXGvaFfz6WGoZ7fUDcfT0kkOy
        42miAh2qyBzklxEsnk2IpN6-tPid6VrklHkqsGqDqHCdP6O8TTB5
        dDDItllVo6_1OLPpcbUrhiUSMxbbXUvdvWXzg-UD8biiReQFlfz2
        8zGWVsdinaUf8ZnyPEgVFf442ZdNqiVJRmBqrYRxe8P_ijQ7p8Vd
        z0TTrxUeT3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0q
        I0n6uiPlaCN_2_jLAeQTlqRHtfa64QQSUMFAAjVKPbByi7xho0uT
        OcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw"
    }
  ]
}
```

Figure 21: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLlCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
    ZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbn
    NAAG9iYml0b24uZXhhbXBsZSJ9",
  "signature": "cu22eBqkYDKgIlTpzDXGvaFfz6WGoZ7fUDcfT0kkOy42mi
    Ah2qyBzklxEsnk2IpN6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDItllV
    o6_1OLPpcbUrhiUSMxbbXUvdvWXzg-UD8biiReQFlfz28zGWVsdinaUf
    8ZnyPEgVFf442ZdNqiVJRmBqrYRxe8P_ijQ7p8Vdz0TTrxUeT3lm8d9s
    hnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiPlaCN_2_jLAeQT
    lqRHtfa64QQSUMFAAjVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD
    4ifKo8DYM-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:

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

#### 4.3.2. Signing Operation

The following is generated before beginning the signature process:

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

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

Figure 23: JWS Protected Header JSON

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXh0bXBsZSJ9
```

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).

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hnbXBsZSJ9
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIGZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrc2VwIHlvdXIGZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcm
UgeW91IGlpc2h0IGJlIHNOZXB0IG9mZiB0by4
```

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-JZGYNlaAjP2kqaluUIIUnC9qvb
u9Plon7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDDyYjnAMDxXPn7XrT0lw-kv
AD890jl8e2puQens_IEKBpHABlsbEPX6sFY8OcdGdQoRuBomu9xQ2
```

Figure 26: JWS Signature, base64url-encoded

#### 4.3.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JWS Compact Serialization:

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hnbXBsZSJ9
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIGZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrc2VwIHlvdXIGZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcm
UgeW91IGlpc2h0IGJlIHNOZXB0IG9mZiB0by4
.
AE_R_YZCChjn4791jsQCrdPZCNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvb
u9Plon7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDDyYjnAMDxXPn7XrT0lw-kv
AD890jl8e2puQens_IEKBpHABlsbEPX6sFY8OcdGdQoRuBomu9xQ2
```

Figure 27: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNoZXAgb250byB0aGUgcmlh
ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgZGhlcmXi
gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNoZXB0IG9m
ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2
dpbnNAAG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "AE_R_YZCChjn4791jSQCrDPZCNYqHXCTZH0-JZGYNl
aAjP2kqaluUIIUnC9qvbu9Plon7KRTzoNEuT4Va2cmLleJAQy3mt
PBu_u_sDDyYjnAMDxXPn7XrT0lw-kvAD890jl8e2puQens_IEKBp
HABlsbEPX6sFY8OcGDqoRuBomu9xQ2"
    }
  ]
}
```

Figure 28: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNoZXAgb250byB0aGUgcmlh
ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgZGhlcmXi
gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNoZXB0IG9m
ZiB0by4",
  "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbn
NAAG9iYml0b24uZXhhbXBsZSJ9",
  "signature": "AE_R_YZCChjn4791jSQCrDPZCNYqHXCTZH0-JZGYNlaAjP
2kqaluUIIUnC9qvbu9Plon7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sD
DyYjnAMDxXPn7XrT0lw-kvAD890jl8e2puQens_IEKBpHABlsbEPX6sF
Y8OcGDqoRuBomu9xQ2"
}
```

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:

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

#### 4.4.2. Signing Operation

The following is generated before completing the signing operation:

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

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

Figure 30: JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
```

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).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
.
SXTigJlzigEGZGFuZ2Vyb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXlzigZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBzZWVwIHlvdXlzigZmVldCwgZGhlcmXigJlzig5vIGtub3dpbmcdgd2hlcmUgeW91IGlpc2Z2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 32: JWS Signing Input

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

```
s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 33: JWS Signature, base64url-encoded

#### 4.4.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JWS Compact Serialization:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
.SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXlzigZG9vci4gWW91IHN0ZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBrcZWVwIHlvdXlzigZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 34: JWS Compact Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXIGZG9vci4gWW91IHNoZXAgb250byB0aGUgcmlh
ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIGZmVldCwgdGhlcmXi
gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNoZXB0IG9m
ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
RkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p
0"
    }
  ]
}
```

Figure 35: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
Z29pbmcgb3V0IHlvdXIGZG9vci4gWW91IHNoZXAgb250byB0aGUgcmlh
ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIGZmVldCwgdGhlcmXi
gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNoZXB0IG9m
ZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
RkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0"
}
```

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:

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

#### 4.5.2. Signing Operation

The following is generated before completing the signing operation:

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

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

Figure 37: JWS Protected Header JSON

eyJhbGciOiJIUzU1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiIiZmQ2LWVlZjMxNGJjNzAzNyJ9

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).

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliIiwiaXNzazNyJ9

SXTigJlzIGEgZGFuZ2Vyb3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcb3V0IH  
1vdXIGZG9vci4gWW91IHN0ZXAgb250byB0aGUgc9hZCwGyYW5kIGlmIHlvdSBk  
b24ndCBBrZWVwIHlvdXIGZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcd2hlcm  
UqeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4

Figure 39: JWS Signing Input

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

```
s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 40: JWS Signature, base64url-encoded

#### 4.5.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JWS Compact Serialization:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
.s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 41: General JWS JSON Serialization

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

Figure 42: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOW
    ItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0"
}
```

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:

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

##### 4.6.2. Signing Operation

The following are generated before completing the signing operation:

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

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

Figure 44: JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiJ9
```

Figure 45: JWS Protected Header, base64url-encoded

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

Figure 46: JWS Unprotected Header JSON

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

```
eyJhbGciOiJIUzI1NiJ9
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrc2VwIHlvdXIgZmVldCwgZGhlcmXigJlzig5vIGtub3dpbmcd2hlcm
UgeW91IGlpc2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 47: JWS Signing Input

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

```
bWUSVaxorn7bEFldjytBd0kHv70Ly5pzbomzMWSOr20
```

Figure 48: JWS Signature, base64url-encoded

#### 4.6.3. Output Results

The following compose the resulting JWS object:

- o JWS Protected Header (Figure 45)
- o JWS Unprotected Header (Figure 46)
- o Payload content (Figure 8)
- o 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": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgcm9h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNOZXB0IG9m
    ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "bWUSVaxorn7bEFldjytBd0kHv70Ly5pvbomzMWSOr2
        0"
    }
  ]
}
```

Figure 49: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgcm9h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNOZXB0IG9m
    ZiB0by4",
  "protected": "eyJhbGciOiJIUzI1NiJ9",
  "header": {
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "bWUSVaxorn7bEFldjytBd0kHv70Ly5pvbomzMWSOr20"
}
```

Figure 50: Flattened JWS JSON Serialization

#### 4.7. Protecting Content Only

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

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

#### 4.7.1. Input Factors

The following are supplied before beginning the signing operation:

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

#### 4.7.2. Signing Operation

The following is generated before completing the signing operation:

- o 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).

```
.
SXTigJlZIGEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBzZWVwIHlvdXIgZmVldCwgdGhlcmXigJlZIG5vIGtub3dpbmcd2hlcm
UgeW91IGlpZ2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 52: JWS Signing Input

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

```
xuLifqLGiblpv9zBpuZczWhNjlgARaLV3UxvxhJxZuk
```

Figure 53: JWS Signature, base64url-encoded

#### 4.7.3. Output Results

The following compose the resulting JWS object:

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

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

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXlzigZG9vci4gWW91IHNN0ZXAgb250byB0aGUgc29h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXlzigZmVldCwgdGhlcmXi
    gJlzig5vIGtub3dpbmNpd2hlcmUgcW91IG1pZ2h0IGJlIHNN3ZXB0IG9m
    ZiB0by4",
  "signatures": [
    {
      "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "xuLifqLGiblpv9zBpuZczWhNjlgARaLV3UxvxhJxZu
        k"
    }
  ]
}
```

Figure 54: General JWS JSON Serialization

The resulting JWS object using the flattened JWS JSON Serialization:

```
{
  "payload": "SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIGZG9vci4gWW91IHN0ZXAgb250byB0aGUgc29p
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIGZmVldCwgZGhlcmXi
    gJlzig5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
    ZiB0by4",
  "header": {
    "alg": "HS256",
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  "signature": "xuLifqLGiblpv9zBpuZczWhNjlgARaLV3UxvxhJxZuk"
}
```

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:

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

- o JWS Protected Header; this example uses the header from Figure 56, encoded using base64url [RFC4648] to produce Figure 57.
- o JWS Unprotected Header; this example uses the header from Figure 58.

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

Figure 56: Signature #1 JWS Protected Header JSON

```
eyJhbGciOiJSUzI1NiJ9
```

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

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

Figure 58: Signature #1 JWS Unprotected Header JSON

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

```
eyJhbGciOiJSUzI1NiJ9
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIGZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBzZWVwIHlvdXIGZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcm
UgeW91IG1pZ2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 59: JWS Signing Input

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

```
MISjqtVlOpa7lKE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53uoimicltcMdSg-qpt
rzZc7CG6Svw2Yl3TDIqHzTUrL_lR2ZFcryNFiHkSwl29EghGpwkpxaTn_THJTC
glNbADkolMZBCdwzJxwqZc-1RlpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqjy
tKSeAMbhMBdMma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZVlaIZsv33uPUqB
BCXbYoQJw7mxPftHmNlGoOSMxR_3thmXTCm4US-xiNOyhb8afKK64jU6_TPt
QHiJeQJxz9G3Tx-083B745_AfYOnlC9w
```

Figure 60: JWS Signature #1, base64url-encoded

The following is the assembled first signature serialized as JSON:

```
{
  "protected": "eyJhbGciOiJSUzI1NiJ9",
  "header": {
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "MISjqtVlOpa7lKE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53u
oimicltcMdSg-qptrzZc7CG6Svw2Yl3TDIqHzTUrL_lR2ZFcryNFiHkS
wl29EghGpwkpxaTn_THJTCglNbADkolMZBCdwzJxwqZc-1RlpO2HibUY
yXSwO97BSe0_evZKdjvvKSgsIqjytKSeAMbhMBdMma622_BG5t4sdbuC
HtFjp9iJmkio47AIwqkZVlaIZsv33uPUqBBCXbYoQJw7mxPftHmNlGo
OSMxR_3thmXTCm4US-xiNOyhb8afKK64jU6_TPtQHiJeQJxz9G3Tx-0
83B745_AfYOnlC9w"
}
```

Figure 61: Signature #1 JSON

#### 4.8.3. Second Signing Operation

The following is generated before completing the second signing operation:

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

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

Figure 62: Signature #2 JWS Unprotected Header JSON

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

```
.
SXTigJlzigEGZGFuZ2VyY3VzIGJlc2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBk
b24ndCBzZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzig5vIGtub3dpbmcd2hlcm
UgeW91IGlPZ2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 63: JWS Signing Input

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

```
ARcVLnaJJJaUWG8fG-8t5BREVAuTY8n8YHjwD0lmuhcdCoFZFFjfISu0Cdkn9Yb
dlmi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrqcI3Jkl2U5IX3utNhODH6v7
xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD
```

Figure 64: JWS Signature #2, base64url-encoded

The following is the assembled second signature serialized as JSON:

```
{
  "header": {
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "ARcVLnaJJJaUWG8fG-8t5BREVAuTY8n8YHjwD0lmuhcdCoF
ZFFjfISu0Cdkn9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrq
cI3Jkl2U5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCM
Yxxm4fgV3q7ZYhm5eD"
}
```

Figure 65: Signature #2 JSON

#### 4.8.4. Third Signing Operation

The following is generated before completing the third signing operation:

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

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

Figure 66: Signature #3 JWS Protected Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
```

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).

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9
.
SXTigJlzlGEgZGFuZ2VyY3VzIGJlc2luZXNzLzCBGcm9kbywgZ29pbmcgb3V0IHlvdXlZG9vci4gWW91IHNOZXAgb250byB0aGUgc9hZCwgYW5kIGlmIHlvdSBkb24ndCBZbWVlZlvdXlZmVldCwgdGhlcmXigJlzlZIG5vIGtub3dpbmcd2hlcmUgeW91IGlPZ2h0IGJlIHNOZXB0IG9mZiB0by4
```

Figure 68: JWS Signing Input

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

```
s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0
```

Figure 69: JWS Signature #3, base64url-encoded

The following is the assembled third signature serialized as JSON:

```
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTkOW
    ItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p0"
}
```

Figure 70: Signature #3 JSON

#### 4.8.5. Output Results

The following compose the resulting JWS object:

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

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

The resulting JWS object using the general JWS JSON Serialization:

```
{
  "payload": "SXTigJlZIGegZGFuZ2VyY3VzIGJlc2luZXNzLlCBGcm9kbywg
    Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHNoZXAgb250byB0aGUgcm9h
    ZCwgYW5kIGlmIHlvdSBkb24ndCBrc2VwIHlvdXIgZmVldCwgdGhlcmXi
    gJlZIG5vIGtub3dpbmcd2hlcmUgeW91IG1pZ2h0IGJlIHNoZXB0IG9m
    ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiJ9",
      "header": {
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "MIsjqtVlOpa7lKE-Mss8_Nq2YH4FGhiocsqrgi5Nvy
        G53uoimicltcMdSg-qptrzZc7CG6Svw2Y13TDIqHzTUuL_lR2ZFc
        ryNFihkSwl29EghGpwkpxaTn_THJTCglNbADkolMZBCdwzJxwqZc
        -lRlpO2HibUYyXSw097BSe0_evZKdjvKSGsIqjytKSeAMbhMBdM
        ma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZVlaIZsv33uPUqBBC
        XbYoQJw7mxPftHmNlGoOSMxR_3thmXTCm4US-xiNOyhb8afKK6
        4jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOnlC9w"
    },
    {
      "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwD0lmuhc
        dCoFZFFjfiSu0Cdkn9Ybdmi54ho0x924DUz8sK7ZXkhc7AFM8Ob
        LfTvNCRqci3Jkl2U5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8b
        AWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD"
    },
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
        RkOWItNDcxYiliZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspWlh84VsJZFTsPPqMDA7g1Md7p
        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:

- o Plaintext content; this example uses the content from Figure 72.
- o RSA public key; this example uses the key from Figure 73.

```

o "alg" parameter of "RSA_5".

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

{
  "kty": "RSA",
  "kid": "frodo.baggins@hobbiton.example",
  "use": "enc",
  "n": "maxhbsmBtdQ3CNrKvprUE6n9lYcregDMLYNeTAWcLj8NnPU9XIYegT
    HVHQjxKDSHP2l-F5jS7sppG1wgdAqZyhnWvXhYNvcM7RfgKxqNx_xAHx
    6f3yy7s-M9PSNCwPC2lh6UAkR4I00EhV9lrypM9Pi4lBUop9t5fS9W5U
    NwaAllhrd-osQGPjIeIildeHTwx-ZTHu3C60Pu_LJl16hKn9wbwaUmA4c
    R5Bd2pgbaY7ASgsjCUbtYJaNIHSoHXprUdJZKUMazV0WOKPfA6OPI4oy
    pBadjvMZ4ZAj3BnXaSYsEZhaueTXvZB4eZOAjIyh2e_VOIKVMsnDrJYA
    VotGlvMQ",
  "e": "AQAB",
  "d": "Kn9tgoHfiTVi8uPu5b9TnwyHwG5dK6RE0uFdlpCGnJN7ZEi963R7wy
    bQ1PLAHmpIbNTztfrheoAniRVlNCIqXaW_qS46lxiDTp4ntEPnqcKsyO
    5jMAji7-CL8vhpYYowNFvIesgMoVaPRYMYT9TW63hNM0aWs7USZ_hLg6
    OelmY0vHTI3FucjSM86Nff4oIENt43r2fspgEPGRrde6fpLc9Oaq-qeP
    lGFULimrRdndm-P8q8kvN3KhlNAtEgrQAgTTgz80S-3VD0FgWfgnblPN
    miuPUx08OpI9KDIfu_acc6fg14nsNaJqXe6RESvhGPH2afjHqSy_Fd2v
    pzj85bQQ",
  "p": "2DwQmZ43FoTnQ8IkUj3BmKRf5Eh2mizZA5xEJ2MinUE3sdTYKSLtaE
    oekX9vbBZuWxHdVhM6UnKCJ_2iNk8Z0ayLYHL0_G21aXf9-unynEpUsH
    7HHTklLpYAZ00x1ZgVljoxAdWNn3hiEfrjZLZGS7lOH-a3QQlDDQoJOJ
    2VFmU",
  "q": "te8LY4-W7IyaqH1ExujjMqkTAlTerbv0VLQnfLY2xINnrWdwiQ93_V
    F099aPlESeLja2nw-6iKie-qT7mtCPozKfVtUYfz5HrJ_XY2kfexJINb
    9lhZHMv5plskZpeIS-GPHCC6gRlKolq-idn_qxyusfWv7WAxlSVfQfk8
    d6Et0",
  "dp": "UfYKcL_or492vVc0PzwLSplbg4L3-Z5wL48mwiswbpzOyIgd2xHTH
    QmjJpFAIZ8q-zf9RmgJXkDrFs9rkdxPtAsLlWYdeCT5c125Fkdg317JV
    RDolinX7x2Kdh8ERCreW8_4zXItuTl_KiXZNU5lvMQjWbIw2eTx1lpsf
    lo0rYU",
  "dq": "iEgcO-QfpepdH8FWd7mUFyrXdnOkXJBCogChY6YKuIHGc_p8Le9Mb
    pFKESzEaLlN1Ehf3B6oGB15Iz_ayUlZj2IoQZ82znoUrpa9fVYNot87A
    CfzIG7q9Mv7RiPaderZi03tkVXAdaBau_9vs5rS-7HMTxkVrxSUVJY14
    TkXlHE",
  "qi": "kC-lzZOqoFaZCr5l0tOVtREKoVqaAYhQiqIRGL-MzS4sCmRkxm5vZ
    lXYx6RtElN_AagjqajlkjieGlXTTThHD8Iga6foGBMaAr5uRlhGQpSc7
    Gl7CF1DZkBJMTQN6EshYzZfxW08mIO8M6Rzuh0beL6fG9mkDcIyPrBXx
    2bQ_mM"
}

```

Figure 73: RSA 2048-Bit Key, in JWK Format



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

#### 5.1.2. Generated Factors

The following are generated before encrypting:

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

3qyTVhIWt5juqZUCpfRqpvaubB956MEJL2Rt-8qXKSo

Figure 74: Content Encryption Key, base64url-encoded

bbd5sTkYwhAIqfHsx8DayA

Figure 75: Initialization Vector, base64url-encoded

#### 5.1.3. Encrypting the Key

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

laLxIOj-nLH-\_BgLOXMozKxmy9gfffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePF  
vG2K-pvSlWc9BRIazDrn50RcRai\_\_3TDON395H3c62tIouJJ4XaRvYHFjZTZ2G  
Xfz8YAIccc91Tfk0WXC2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcG  
TSLUeeCt36rlKt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8Vl  
zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEEcelIO1wx1BpyIfgvfjOh  
MBs9M8XL223Fg47xlGsMXdfuY-4jaqVw

Figure 76: Encrypted Key, base64url-encoded

#### 5.1.4. Encrypting the Content

The following is generated before encrypting the Plaintext:

- o 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

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YWlwbGUlLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 78: JWE Protected Header, base64url-encoded

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

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

produces the following:

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

```
0fys_TY_na7f8dwSfXLIyDHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_r
aa0HnLQ6s1P2sv3Xz1lp1l_o5wR_RsSzs8Z-wnI3Jvo0mkpEEenlDmZvDu_k8O
WzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc2VbVbK4dQKPdNTjPPEmRqcaGeTWZV
yeSUvf5k59yJZxRuSvWff6KrNtmRdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaIO
zT5CbL5Qlw3sRc7u_hg0yKVOiRyteAEs3vZkcfLkP6nbXdc_PkMdNS-ohP78T2
O6_7uInMGhFeX4ctHG7VelHGiT93JfWDEQi5_V9UNlRhXNrYu-0fVMkZAKX3VW
i7lzA6BP430m
```

Figure 79: Ciphertext, base64url-encoded

```
kvKuFBXHe5mQr4lqgobAUg
```

Figure 80: Authentication Tag, base64url-encoded

### 5.1.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLmV4YWlwbGUlLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.
.
laLxIOj-nLH-_BgLOXMozKxmy9gffY2gTdvqzfTihJBuuzxg0V7yk1WClNqePF
vG2K-pvSlWc9BRiAzDrn50RcRai__3TDON395H3c62tIouJJ4XaRvYHFjZTZ2G
Xfz8YAIccc91Tfk0WXC2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcG
TSLUeeCt36rlKt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8Vl
zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEEcelIOlwx1BpyIfgvfjOh
MBS9M8XL223Fg47xlGsMXdfuY-4jaqVw
.
bbd5sTkYwhAIqfHsx8DayA
.
0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_r
aa0HnLQ6s1P2sv3Xz1lp1l_o5wR_RsSzs8Z-wnI3Jvo0mkpEEEnlDmZvDu_k8O
WzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc2VbVbK4dQKPdNTjPPEmRqcaGeTWZV
yeSUvf5k59yJZxRuSvWff6KrNtmRdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaIO
zT5CbL5Qlw3sRc7u_hg0yKVOiRyteAEs3vZkcfLkP6nbXdc_PkMdNS-ohP78T2
O6_7uInMGHFeX4ctHG7VelHGIt93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VW
i7lzA6BP430m
.
kvKuFBXHe5mQr4lqgobAUg
```

Figure 81: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "laLxIOj-nLH-_BgLOXMozKxmy9gffY2gTdvqzf
        TihJBuuzxg0V7yk1WClNqEPFvG2K-pvSlWc9BRIazDrn50RcRai_
        _3TDON395H3c62tIouJJ4XaRvYHFjZTZ2GXfz8YAIccc91Tfk0WX
        C2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcGTSLUeeCt
        36r1Kt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8
        VlzNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEEcelIOlwx
        1BpyIfgvfjOhMBS9M8XL223Fg47xlGsMXdfuY-4jaqVw"
    }
  ],
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW
    5zQGhvYmJpdG9uLmV4YW1wbGUlLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In
    0",
  "iv": "bbd5sTkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSfXLIyDHaA2DxUjd67ieF7fcVbIR62
    JhJvGZ4_FNVSiGc_raq0HnLQ6s1P2sv3Xz1lp1l_o5wR_RsSzs8Z-wn
    I3Jvo0mkpEEEnlDmZvDu_k8OWzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc
    2VbVbK4dQKPdNTjPPEmRqcaGeTWZVyeSUvf5k59yJZxRuSvWff6KrNtm
    RdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaI0zT5CbL5Qlw3sRc7u_hg0y
    KVOiRytEAEs3vzkcfLkP6nbXdC_PkMdNS-ohP78T2O6_7uInMGhFeX4c
    tHG7VelHGIt93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VWi71zA6BP4
    30m",
  "tag": "kvKuFBXHe5mQr4lqgobAUg"
}
```

Figure 82: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW
    5zQGhVYmJpdG9uLmV4YWlwbGUuLmV4YWlwbGUuLmV4YWlwbGUuLmV4
    0",
  "encrypted_key": "laLxIOj-nLH-_BgLOXMozKxmy9gffY2gTdvqzfTihJ
    Buuzxg0V7yk1WClNqEPFvG2K-pvSlWc9BRIazDrn50RcRai__3TDON39
    5H3c62tIouJJ4XaRvYHFjZTZ2GXfz8YAIccc91Tfk0WXC2F5Xbb71ClQ
    1DDH151tlpH77f2ff7xiSxh9oSewYrcGTSLUeeCt36r1Kt3OSj7EyBQX
    oZlN7IxbyhMAfGie7Mv1rOTOI5I8NQqeXXW8VlzNmoxaGMny3YnGir5W
    f6Qt2nBq4qDaPdNaAuUGUGeeceLI0lwx1BpyIfgvfjOhMBS9M8XL223F
    g47xlGsMXdfuY-4jaqVw",
  "iv": "bbd5sTkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSfXLIYdHaA2DxUjd67ieF7fcVbIR62
    JhJvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xz1lp1l_o5wR_RsSzs8Z-wn
    I3Jvo0mkpEEnlDmZvDu_k8OWzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc
    2VbVbK4dQKPdNTjPPEmRqcaGeTWZVyeSUvf5k59yJZxRuSvWff6KrNtm
    RdZ8R4mDOjHSrM_s8uWIFcqt4r5GX8TKaI0zt5CbL5Qlw3sRc7u_hg0y
    KVOiRytEAEs3vZkcfLkP6nbXdc_PkMdNS-ohP78T2O6_7uInMGhFeX4c
    tHG7VelHGIt93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VWi7lza6BP4
    30m",
  "tag": "kvKuFBXHe5mQr4lqgobAUg"
}
```

Figure 83: Flattened JWE JSON Serialization

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

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

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

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

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

### 5.2.1. Input Factors

The following are supplied before beginning the encryption process:

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

```
{
  "kty": "RSA",
  "kid": "samwise.gamgee@hobbiton.example",
  "use": "enc",
  "n": "wbdxI55VaanzXPY29Lg5hdmv2XhVqAhoxUkanfzf2-5zVUxa6prHRr
    I4pPlAhoqJrLzfyTWWd5mmHRG2pAHIlh0ySJ9wi0BioZBl1XP2e-C-Fy
    XJGcTy0HdKQWlrfhTm42EW7Vv04r4gfao6uxjLGwfpGrZLarohiWCPnk
    Nrg71S2CuNZSQBIPGjXfkmIy2tl_VWgGnL22GplyXj5Y1BLdxXp3XeSt
    sqo57lutNfoUTU8E4qdzJ3U1DitoVkPGsMwlmnnJiwA7sXRitBCivR4M
    5qnZtdw-7v4WuR4779ubDuJ5nalMv2S66-RPcnFAzWSKxtBDnFJJJGDIU
    e7Tzizjglnms0Xq_yPub_UOlWn0ec85FCftlhACpWG8schrOBeNqHBOD
    FskYpUc2LC5JA2TaPF2dA67dglTTsC_FupfQ2kNGcElLgprxKHcVWYQb
    86B-HozjHZcqttauBzFNV5tbTuB-TpkcvJfNcFLlH3b8mb-H_ox35FjqB
    SAjLKyoeqfKTpVjvXhd09knwgJf6VKq6UC418_TOljMVfFTWXUxlnfhO
    OnzW6HSSzDlc9WrCuVzsUMv54szidQ9wflcYWF3g5qFDxDQKis99gcDa
    iCAwM3yEBIzuNeeCa5dartHDb1xEB_HchSeYbghbMjGfasvKn0aZRsnT
    yC0xhWBlsolZE",
  "e": "AQAB",
  "alg": "RSA-OAEP",
  "d": "n7fzJc3_WG59VEObTkayzuSMM7800JQuZjN_KbH8l0ZG25ZoA7T4Bx
    cc0xQn5oZE5uSCIwg9l0Ct0JvxPcpmqzaJZglnirjcWZ-oBtVk7gCAWq
    -B3qhfF3izlbkosrzjHajIcY33HBhsy4_WerrXg4MDNE4HYojy68TcxT
    2LYQRxUOCf5TtJXvM8olexlSGtVnQnDRutxEUCwiewfmmrfveEogLx9E
    A-KMgAjTiISXxqIXQhWUQXlG7v_mV_Hr2YuImYcNcHkRvp9E7ook0876
    DhkO8v4UOZLwAl0lUX98mkoqwc58A_Y2lBYbVx1_s5lpPsEqbbH-nqIj
    hlflL0gdNfihLxnclWtW7pCztLnImZAyeCWAG7ZIfv-Rn9fLlV9jZ6r7r
    -MSH9sqbuziHN2grGjD_jfRluMHa0l84fFKl6bcqNlJWxPVhzNZo0lyD
    F-1LiQnqUYSepPf6X3a2S0dkqBriqueE6EvLuSYIDpJq3jDIsgoL8Mo1L
    oomgiJxUwL_GWEOGu28gplyzm-9Q0U0nyhEfluhSR8aJAQWaiFImWH5W
    _IQT9I7-yrindr_2fWQ_ilUgMsGzA7aOGzZfPljRy6z-tY_KuBG00-28
    S_aWvjyUc-Alp8AUyKjBZ-7CWH32fGWK48jlt-zomrwjL_mnhsPbGs0c
    9WsWgRzI-K8gE",
  "p": "7_2v3OQZzlPFchYyFLABQ3XP85Es4hCdwCkbDeltaUXgVy9l9etKgh
    vM4hRkOvbb0lkYVuLFmxIkCDtpi-zLCYAdXKrAK3PtSbtzld_XZ9nlsY
    a_QZWpXB_IrtFjVfdKUdMz94pHUhFGFj7nr6NNxfpiHSHWFE1zD_AC3m
    Y46J96lY2LRnreVwAGNw53p07Db8yD_92pDa97vqcZOdgtYbH9q6uma-
```

```

RFNnO1AoiJhYZj69hjmMRXx-x56HO9cnXNbmzNSCFCKnQmn4GQLmRj9s
fbZRqL94bbtE4_e0Zrpo8RNo8vxRLqQNWiy85fc6BRgBJomt8QdQvIgp
gWCv5HoQ",
"q": "zqOHk1P6WN_rHuM7ZF1cXH0x6RuOHq67WuHiSknqQeefGBA9PWS6Zy
KQCO-O6mKXtcgE8_Q_hA2kMRcKOcvHil1hqMCNSXlflM7WPRPZu2qCDc
qssd_uMbP-DqYthH_EzwL9KnYoH7JQFxxmcv5An8oXUtTwk4knKjkIYG
RuUwfQTus0wlNfjFayxOOiAQ37ussIcE6C6ZSsM3n41UlbJ7TCqewzVJ
aPJN5cxjySPZPD3Vp01a9YgAD6a3IIaKJdIxJS1ImnfPevSJQBE79-EX
e2kSwVgOzvt-gsmM29QQ8veHy4uAqca5dZzMs7hkkHtwlzojHV90epQJ
JlXXnH8Q",
"dp": "19oDkBlAXelMIxQFm2zzTqUHAzCIR4xNIGEPNoDt1jK83_FJA-xn
x5kA7-1erdHdms_Ef67HsONNV5A60JaR7w8LHnDiBGnjdaUmmuO8XAXQ
J_ia5mxjxNjS6E2yD44USo2JmHvzeeNczq25elqbTPLhUpGolIZuG72F
ZQ5gTjXoTXC2-xtCDEUZfaUNh4IeAipfLugbpe0JAF1FfrTDAMUFpC3i
XjxqzbEanflwPvj6V9iDSgjj8SozSM0dLtxvu0LIeIQAEgT_yXcrKGm
pKdSO08kLBx8VUjkbv_3Pn20Gyu2YEuwPflM_H1NikuxJNKFGmnAq9Lc
nwwT0jvoQ",
"dq": "S6p59KrlmzGzaQYQM3o0XfHCGvfqHLYjCO557HYQf7209kLMCfd_1
VBEqeD-1jjwELKDjck8kOB15UvohK1oDfSP1DleAy-cnmL29DqWmhgW
lip0CCNmksmDSlqkUXDi6sAaZuntyukyflI-qSQ3C_BafPyFaKrt1fg
dyEwYa08pESKwwWisY7KnmoUvaJ3SaHmohFS78TJ25cfc10wZ9hQNOri
ChZlkiOdFctxDqdmCqNacnhgE3bZQjGp3n83ODSz9zwJcSUvODlXBpc2
Aych6Ci5yjbxt4Ppox_5pjm6xnQkiPgj0lGpsUssMmBN7iHVsre7N2iz
nBNCEOUIQ",
"qi": "FZhClBMywVVjnuUud-05qd5CYU0dK79akAgy9oX6RX6I3IIIPckCc
iRrokxglZn-omAY5CnCe4KdrnjFOT5YUZE7G_Pg44XgCXaarLQf4hl80
oPEf6-jJ5Iy6wPRx7G2e8qLxnh9cOdf-kRqgOS3F48Ucvw3ma5V6KGMw
QqWFeV31XtZ815cVI-I3NzBS7qltpUVgz2Ju021eyc7IlqgzR98qKONl
27DuEES0aK0WE97jnsyO27Yp88Wa2RiBrEocM89QZIlseJiGDizHRUP4
UZxw9zsXww46wy0P6f9grnYp7t8LkyDDk8eoI4KX6SNMNVcyVS9IWjlq
8EzqZEKIA"
}

```

Figure 84: RSA 4096-Bit Key

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

### 5.2.2. Generated Factors

The following are generated before encrypting:

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

```
mYmfsggkTAm0TbvltFh2hyoXnbEzJQjMxmglN3d8xXA
```

Figure 85: Content Encryption Key, base64url-encoded

```
-nBoKLH0YkLZPSI9
```

Figure 86: Initialization Vector, base64url-encoded

### 5.2.3. Encrypting the Key

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

```
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi
beYO15S_PJsAXZwSxtDePz9hk-BbtsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyu
cvI6hvALeZ6OGnhNV4vlzx2k701D89mAzw-_kT3tkuorpDU-CpBENfIHx1Q58
-Aad3FzMuo3Fn9buEP2yXakLXYa15BUXQsupM4AlGD4_H4Bd7V3u9h8Gkg8Bpx
KdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzSlasnuHtVMt2pK
IIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zp0878TR1zx7
pZfPYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
fOTs_lycTWmY-aqWVDKhjYNRf03NiWrtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3
8UjQb0lvXnlSpyvYZ3Wfc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8aKaOnx6ASE5
Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR
s
```

Figure 87: Encrypted Key, base64url-encoded

### 5.2.4. Encrypting the Content

The following is generated before encrypting the Plaintext:

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

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

Figure 88: JWE Protected Header JSON

```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpY2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YWlwbGUlLCJlbmMiOiJBMjU2R0NNIn0
```

Figure 89: JWE Protected Header, base64url-encoded



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

- o CEK (Figure 85);
- o Initialization Vector (Figure 86); and
- o JWE Protected Header (Figure 89) as authenticated data

produces the following:

- o Ciphertext from Figure 90.
- o Authentication Tag from Figure 91.

```
o4k2cnGN8rSSw3IDolYuySkqeS_t2mlGXklSgqBdpACm6UJuJowOHC5ytjqYgR
L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw
P7ualDRfvaOjgZv6Ifa3brcAM64d8p5lhhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sSwy7SQmRkfhDjwbz0fz6kFovEgj64XlI5s7E6GLp5fnbYGLa1QUiML
7Cc2Gxgvi7zqWo0YIEc7aCflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV
maPpOsly2n525DxDfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw
```

Figure 90: Ciphertext, base64url-encoded

```
UCGiqJxhBI3IFVdPalHHvA
```

Figure 91: Authentication Tag, base64url-encoded

#### 5.2.5. Output Results

The following compose the resulting JWE object:

- o JWE Protected Header (Figure 89)
- o Encrypted Key (Figure 87)
- o Initialization Vector (Figure 86)
- o Ciphertext (Figure 90)
- o Authentication Tag (Figure 91)

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbnRlc2UuZ2FtZ2VlQGVhYmJpdG9uLmV4YWlwbGUuLmJmMiOiJBMjU2R0NNIn0.
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi
beYO15S_PJsAXZwSxtDePz9hk-BbtsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyu
cvI6hvALEz6OGnhNV4vlzx2k701D89mAzfw-_kT3tkuorpdU-CpBENfIHx1Q58
-Aad3FzMuo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8Bpx
KdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzSlasnuHtVMt2pK
IIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zp0878TRlZx7
pZfPYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
fOTs_lycTWmY-aqWVDKhjYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3
8UjQb0lvXn1SpyvYZ3Wfc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8aKaOnx6ASE5
Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR
S
-nBoKLH0YkLZPSI9
o4k2cnGN8rSSw3IDolYuySkqeS_t2mlGXklSgqBdpACm6UJuJowOHC5ytjqYgR
L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw
P7ualDRfva0jgZv6Ifa3brCAM64d8p5lhhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sSwy7SQmRkfhdjwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML
7Cc2GxgvI7zqWo0YIEc7aCflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV
maPpOsly2n525DxDfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOW
UCGiQJxhBI3IFVdPalHHvA
```

Figure 92: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "rT99rwrBTbTI7IjM8fU3Eli7226HEB7IchCxNu
        h7lCiud48LxeolRdtFF4nzQibeYOl5S_PJsAXZwSxtDePz9hk-Bb
        tsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyucvI6hvALeZ6OGnhNV4
        vlzx2k7O1D89mAzfw-_kT3tkuorpdU-CpBENfIHX1Q58-Aad3FzM
        uo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8B
        pxKdUV9ScfJTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzS1
        asnuHtVMt2pKIIifux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq
        5pGqFmW2k8zpO878TR1Zx7pZfPYDSXZyS0CfKKkMozT_qiCwZTSz
        4duYnt8hS4Z9sGthXn9uDqd6wycMagnQfOTs_lycTWmY-aqWVDKh
        jYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe38UjQb0lvXn
        lSpyvYZ3Wfc7W0JYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
        06MvZTlFOt0UvfukBa03cxA_nIBiHLMjY2kOTxQMmpDPTr6Cbo8a
        KaOnx6ASE5Jx9paBpnNmOOKH35j_QlRQhDWUN6A2Gg8iFayJ69xD
        EdHAVCGRzN3woEI2ozDRs"
    }
  ],
  "protected": "eyJhbGciOiJIJSU0Ett0FFUCIsImtpZCI6InNhbXdpc2UuZ2
    FtZ2VlQGhvYmJpdG9uLmV4YWlwbGUlLCJlbmMiOiJBMjU2R0NNIn0",
  "iv": "-nBoKlH0YkLZPSI9",
  "ciphertext": "o4k2cnGN8rSSw3IDolYuySkqeS_t2m1GXklSgqBdpACm6
    UJuJowOHC5ytjqYgRL-I-soPlwqMuf4UgRWWeaOGNw6vGW-xyM01lTYx
    rXfVzIIaRdhYtEMRBvBWbEwP7ua1DRfvaOjgZv6Ifa3brcAM64d8p5lh
    hNcizPersuhw5f-pGYzseva-TUaL8iWnctc-sSwy7SQmRkfhDjwbz0fz
    6kFovEgj64XlI5s7E6GLp5fnbYGLa1QUiML7Cc2GxgvI7zqWo0YIEc7a
    CflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSVmaPpOslY2n525Dx
    DfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw",
  "tag": "UCGiqJxhBI3IFVdPalHHvA"
}
```

Figure 93: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJIU0EtT0FFUCIsImtpZCI6InNhbmxdpc2UuZ2
    FtZ2VlQGhVYmJpdG9uLmV4YWlwbGUlClJlbmMiOiJBMjU2R0NNIn0",
  "encrypted_key": "rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lC
    iud48LxeolRdtFF4nzQibeYOl5S_PJsAXZwSxtDePz9hk-BbtsTBqC2U
    sPOdwjC9NhNupNNu9uHIVftDyucvI6hvAlEz6OGnhNV4v1zx2k7OlD89
    mAzw-_kT3tkuorpdU-CpBENfIHx1Q58-Aad3FzMu03Fn9buEP2yXakL
    XYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8BpxKdUV9ScfJQTcYm6eJE
    Bz3aSwIaK4T3-dwWpuBOhROQXBosJzSslasnuHtVMt2pKIIfux5BC6huI
    vmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zp0878TRlZx7pZfPYD
    fXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
    sOTs_lycTWmY-aqWVDKjYjYnRf03NiwrTb5BE-tOdFwCASQj3uuAgPgrO
    2AWBe38UjQb0lvXn1SpyvYZ3Wfc7W0JYaTa7A8DRn6MC6T-xDmMuxCOG
    7S2rscw5lQQU06MvZTlF0t0UvfukBa03cxA_nIBIhLMjY2kOTxQMmpDP
    Tr6Cbo8aKaOnx6ASE5Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ
    69xDEDHAVCGRzN3woEI2ozDRs",
  "iv": "-nBoKLH0YkLZPSI9",
  "ciphertext": "o4k2cnGN8rSSw3IDolYuySkqeS_t2mlGXklSggBdpACm6
    UJuJowOHC5ytjqYgRL-I-soPlwqMUF4UgRWWeaOGNw6vGW-xyM01lTYx
    rXfVzIIaRdhYtEMRBvBWBewP7ualDRfva0jgZv6Ifa3brCAM64d8p5lh
    hNcizPersuhw5f-pGYzseva-TUaL8iWnctc-sSwy7SQmRkfhdJwbz0fz
    6kFovEgJ64XlI5s7E6GLp5fnbYGLa1QUiML7Cc2Gxgvi7zqWo0YIEc7a
    CflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSVmaPpOslY2n525Dx
    DfWaVFUFkQxMF56vn4B9QMPwABnypNimbM8zVow",
  "tag": "UCGiQJxhBI3IFVdPalHHvA"
}
```

Figure 94: Flattened JWE JSON Serialization

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

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

A common use of password-based encryption is the import/export of keys. Therefore, this example uses a 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:

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

```
{
  "keys": [
    {
      "kty": "oct",
      "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a",
      "use": "enc",
      "alg": "A128GCM",
      "k": "XctOhJAKA-pD9Lh7ZgW_2A"
    },
    {
      "kty": "oct",
      "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
      "use": "enc",
      "alg": "A128KW",
      "k": "GZy6sIZ6wl9NJOKB-jnmVQ"
    },
    {
      "kty": "oct",
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
      "use": "enc",
      "alg": "A256GCMKW",
      "k": "qC57l_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
    }
  ]
}
```

Figure 95: Plaintext Content

```
entrap_o\xe2\x80\x93peter_long\xe2\x80\x93credit_tun
```

Figure 96: Password

### 5.3.2. 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 97.
- o 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:

- o Salt input; this example uses the salt input from Figure 99.
- o 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:

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

produces the following Encrypted Key:

```
d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdgtURtmeDVlg
```

Figure 100: Encrypted Key, base64url-encoded

#### 5.3.4. Encrypting the Content

The following is generated before encrypting the content:

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

```
{
  "alg": "PBES2-HS512+A256KW",
  "p2s": "8QlSzinAsR3xchYz6ZZcHA",
  "p2c": 8192,
  "cty": "jwk-set+json",
  "enc": "A128CBC-HS256"
}
```

Figure 101: JWE Protected Header JSON

```
eyJhbGciOiJIQQkvVTMiIiwzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3
hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 102: JWE Protected Header, base64url-encoded

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

- o CEK (Figure 97);
- o Initialization Vector (Figure 98); and
- o JWE Protected Header (Figure 102) as authenticated data

produces the following:

- o Ciphertext from Figure 103.
- o Authentication Tag from Figure 104.

```
23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIX86vMXqIi6IR
sfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpDjEYCNA_XOmzg8yZR9oyjo6l
TF6si4q9FZ2EhzgFQCLO_6h5EVg3vR75_hkBsnuoqoM3dwejXBtIodN84PeqMb
6asmas_dpSsz7H10fC5ni9xIz424givB1YLldF6exVmL93R3fOoOJbmk2GBQZL
_SEGllv2cQsBgeprARsaQ7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKd
PQMTlVJKkqtV4Ru5LEVpBZXBNzrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrok
AKYPqmXUeRdjFJwafkYEkiuDCV9vWGAILDH2xTafhJwcmwIyzi4BqRpmdn_N-
zl5tuJYyuvKhjKv6ihbsV_klhJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V
3kobXZ77ulMwDs4p
```

Figure 103: Ciphertext, base64url-encoded

```
0HlwodAhOCILG5SQ2LQ9dg
```

Figure 104: Authentication Tag, base64url-encoded

#### 5.3.5. Output Results

The following compose the resulting JWE object:

- o JWE Protected Header (Figure 102)
- o Encrypted Key (Figure 100)
- o Initialization Vector (Figure 98)
- o Ciphertext (Figure 103)
- o Authentication Tag (Figure 104)



The resulting JWE object using the JWE Compact Serialization:

```

eyJhbGciOiJQQQkVIMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3
hjaFl6NlpaY0hBIiwicDJJiJo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0
.
d3qNhUWfqheyPp4H8sjOWSDYajoej4c5Je6rlUtFPWdgtURtmeDVlg
.
VBiCzVHNoLiR3F4V82uoTQ
.
23i-TblAV4n0WKVSSGcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIX86vMXqIi6IR
sfywCRfZLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpDjEYCNA_XOmzg8yZR9oyjo6l
TF6si4q9FZ2EhzhgFQCLO_6h5EVg3vR75_hkBsnuoqoM3dweJXbtIodN84PeqMb
6asmas_dpSsz7H10fC5ni9xIz424givB1YLldF6exVmL93R3fOoOJbmk2GBQZL
_SEGllv2cQsBgoprARsaQ7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKd
PQMTlVJKkqtV4Ru5LEVpBZXBNzrtViS0gyg6AiuwaS-rCrcD_ePOGSuxvgtrok
AKYPqmXUeRdjfJwafkYEkIUdCV9vWGAIldH2xTafhJwcmYwIyzi4BqRpmdn_N-
z15tuJYyuvKhjKv6ihbsV_k1hJGPGAxJ6wUpmWc4PTQ2izEm0TuSE8oMKdT8V
3kobXZ77ulMwDs4p
.
0HlwodAhOCILG5SQ2LQ9dg

```

Figure 105: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlU
        tFPWdgtURtmeDVlg"
    }
  ],
  "protected": "eyJhbGciOiJIQQkVTMi1lUzUxMitBMjU2S1ciLCJwMnMiOi
    I4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOi
    Jqd2stc2V0K2pzb24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "iv": "VBiCzVHNoLiR3F4V82uoTQ",
  "ciphertext": "23i-TblAV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2
    nsnGIX86vMXqIi6IRsfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpd
    jEYCNA_XOmzg8yZR9oyjo6lTF6si4q9FZ2EhgzgFQCL0_6h5EVg3vR75_
    hkBsnuoqoM3dwejXBtIodN84PeqMb6asmas_dpSsz7H10fC5ni9xIz42
    4givB1YLldF6exVmL93R3fOoOJbmk2GBQZL_SEGllv2cQsBgoprARsaQ
    7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKdPQMTlVJKkqtV4Ru
    5LEVpBZXBNzrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrokAKYPqmXUe
    RdjFJwafkYEkiuDCV9vWGAilDH2xTafhJwcmYwIyzi4BqRpmDn_N-zl5
    tuJYyuvKhjKv6ihbsV_klhJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdT
    w8V3kobXZ77ulMwDs4p",
  "tag": "0HlwodAhOCILG5SQ2LQ9dg"
}
```

Figure 106: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "encrypted_key": "d3qNhUWfqheyPp4H8sjOWsDYajoej4c5Je6rlUtFPWdgtURtmeDVlg",
  "iv": "VBiCzVHNoLiR3F4V82uoTQ",
  "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsgIX86vMXqIi6IRsfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMxXhUHpDjEYCNA_XOmzg8yZR9oyjo6lTF6si4q9FZ2EhzgFQCLO_6h5EVg3vR75_hkBsnuoqoM3dwejXBtIodN84PeqMb6asmas_dpSsz7H10fC5ni9xIz424givB1YLldF6exVmL93R3fOoOJbmk2GBQZL_SEGllv2cQsBgeprARsaQ7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKdPQMTlVJKkqtV4Ru5LEVpBZXBNzrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrokAKYPqmXUeRdjfJwafkYEkiuDCV9vWGAi1DH2xTafhJwcmYwIyzi4BqRpmdn_N-zl5tuJYyuvKhjKv6ihbsV_klhJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdT w8V3kobXZ77ulMwDs4p",
  "tag": "0HlwodAhOCILG5SQ2LQ9dg"
}
```

Figure 107: Flattened JWE JSON Serialization

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

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

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

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

##### 5.4.1. Input Factors

The following are supplied before beginning the encryption process:

- o Plaintext content; this example uses the content from Figure 72.
- o EC public key; this example uses the public key from Figure 108.

- o "alg" parameter of "ECDH-ES+A128KW".
- o "enc" parameter of "A128GCM".

```
{
  "kty": "EC",
  "kid": "peregrin.took@tuckborough.example",
  "use": "enc",
  "crv": "P-384",
  "x": "YU4rRUzdmVqmRtWOs2OpDE_T5fsNIodcG8G5FWPrTPMyxpzsSOGaQL
    pe2FpxBmu2",
  "y": "A8-yxCHxkfBz3hKZfI1jUYMjUhsEveZ9THuWFjH2sCNdtkSRJU7D5-
    SkgaFLlETP",
  "d": "iTx2pk7wW-GqJkHcEkFQb2EFyYcO7RugmaW3mRrQVAOUiPommT0Idn
    YK2xDlZh-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.

Nou2ueKlP70ZXDbq9UrRwg

Figure 109: Content Encryption Key, base64url-encoded

mH-G2zVqgzUtW\_

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.

```
{
  "kty": "EC",
  "crv": "P-384",
  "x": "uBo4kHPw6kbjx5l0xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iuE
    DsQ6wNdNg3",
  "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEwZOyQTA-
    JdaY8tb7E0",
  "d": "D5H4Y_5PSKZvhfVFbcCYJOtcGZygRgfZkpsBr59Icmmhe9sW6nkZ8W
    fwhinUfWJg"
}
```

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

Performing the key encryption operation over the CEK (Figure 109) with the following:

- o The static Elliptic Curve public key (Figure 108); and
- o 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:

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

```
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx5l0xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6i
      uEDsQ6wNdNg3",
    "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4PgEwZOyQT
      A-JdaY8tb7E0"
  },
  "enc": "A128GCM"
}
```

Figure 113: JWE Protected Header JSON

```
eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2JvcmluZ2guZXhhbXBsZSI6ImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InVCbzRrSFB3NmtianglbnDB4b3dyZF9vWXpCbWF6LUdLRlp1NH
hBRkZrYllpV2dldEVLNml1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcdVTR2haVhMy
ZmFYdW1JLWU5SlUyTW84S3BvWXJGRHileVBOVnRXNFBnRXdaT3lRVEEtSmRhWT
h0YjdFMCJ9LCJlbmMiOiJBMTI4R0NNIn0
```

Figure 114: JWE Protected Header, base64url-encoded

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

- o CEK (Figure 109);
- o Initialization Vector (Figure 110); and
- o JWE Protected Header (Figure 114) as authenticated data

produces the following:

- o Ciphertext from Figure 115.
- o Authentication Tag from Figure 116.

```
tkZu009h950gHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID7lpnAi_cP
WJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsXaEwDdXta9Mn5B7cCBoJKB0
IgEnj_qfolhIi-uEkUpOZ8aLTZGHfpl05jMwbKkTe2yK3mjF6SBAsgicQDVCkc
Y9BLluzxlRmC3ORXaM0JaHPB93YcdSDGgpgBWMVrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXIuEr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu
07WNhJzJEPc4jVntRJ6K53NgPQ5p9913Z408OUqj4ioYezbS6vTP1Q
```

Figure 115: Ciphertext, base64url-encoded

WuGzxmcreYjpHGJoa17EBg

Figure 116: Authentication Tag, base64url-encoded

#### 5.4.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJFQ0RILUVTk0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm9lZ2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InVCbzRrSFB3NmtianglbdDB4b3dyZF9vWXpCbWF6LUdLRlp1NH
hBRkZrYllpV2dldEVlNmllRURzUTZ3TmROZzMlLCJ5Ijoic3AzcdVTR2haVkJmY
ZmFYdWlJLWU5SlUyTW84S3BvWXJGRHlleVBOVnRXNFBnRXdaT3lRVEEtSmRhWT
h0YjdFMCJ9LCJlbmMiOiJBMTI4R0NNIn0
.
0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2
.
mH-G2zVqgztUtnW_
.
tkZu009h95OgHJmkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID7lpnAi_cP
WJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsXaEwDdXta9Mn5B7cCB0JKB0
IgEnj_qfolhIi-uEkUpOZ8aLTZGHfpl05jMwbKkTe2yK3mjF6SBAsgicQDVCKc
Y9BLluzx1RmC3ORXaM0JaHPB93YcdSDGpgBWMVrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXIuEr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu
07WNhjzJEPc4jVntRJ6K53NgPQ5p9913Z408OUqj4ioYezbS6vTPlQ
.
WuGzxmcreeYjpHGJoal7EBg
```

Figure 117: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
    }
  ],
  "protected": "eyJhbGciOiJIJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdHVja2Jvcn9lZ2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAtMzg0IiwieCI6InVCbzRrSFB3NmtianglbdB4b3dyZF9vWXPcbWF6LUdLRlp1NHhBRkZrYllpV2dldEVLNml1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkMyZmFYdW1JLWU5SlUyTW84S3BvWXJGRHileVBOVnRXNFBnRXdaT3lRVEEtSmRhWTh0YjdFMCJ9LCJlbmMiOiJBMTI4R0NNIn0",
  "iv": "mH-G2zVqgztUtnW_",
  "ciphertext": "tkZu009h950gHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID7lpnAi_cPWJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsXaEwDdXta9Mn5B7cCB0JKB0IgEnj_qfolhIi-uEkUpOZ8aLTZGHfp105jMwbKkTe2yK3mjF6SBAsgicQDVckcY9BLluzx1RmC3ORXaM0JaHPB93YcdSDGgpgBWMVrNU1ErkjcMqMoT_wtCex3w03XdLkjXIuEr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu07WNhJzJEPc4jVntrJ6K53NgPQ5p99l3Z408OUqj4ioYezbS6vTPlQ",
  "tag": "WuGzxmcYjPHGJoal7EBg"
}
```

Figure 118: General JWE JSON Serialization



The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJIJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcm
    VncmluLnRvb2tAdHVja2JvcmluZ2guZXhhbXBsZSIsImVwayI6eyJrdH
    kiOiJFQyIsImNydiI6IlAtMzg0IiwieCI6InVCbzRrSFB3Nmtianglbd
    B4b3dyZF9vWXpCbWF6LUdLRlp1NHhBRkZrY1lpV2d1dEVLNml1RURzUT
    Z3TmROZzMiLCJ5Ijoic3AzcdVTR2haVkJmYmFYdW1JLWU5SlUyTW84S3
    BvWXJGRHileVBOVnRXNFBnRXdaT3lRVEEtSmRhWTh0YjdFMCJ9LCJlbm
    MiOiJBMTI4R0NNIn0",
  "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2",
  "iv": "mH-G2zVqgztUtnW_",
  "ciphertext": "tkZu009h950gHJmkrflBisku8rGf6nzVxhRM3sVOhXgz
    5NJ76oID7lpnAi_cPWJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzs
    XaEwDdXta9Mn5B7cCB0JB0IgeNj_qfolhIi-uEkUpOZ8aLTZGHfpl05
    jMwbKkTe2yK3mjF6SBAsGicQDVckY9BLluzx1RmC3ORXaM0JaHPB93Y
    cdSDGgpgBWMVrNUlErkjcmQMoT_wtCex3w03XdLkjXIuEr2hWgeP-nkU
    ZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu07WNhJzJEPc4jVn
    tRJ6K53NgPQ5p99l3Z408OUqj4ioYezbS6vTPlQ",
  "tag": "WuGzxmcYjPHGJoal7EBg"
}
```

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:

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

```
{
  "kty": "EC",
  "kid": "meriadoc.brandybuck@buckland.example",
  "use": "enc",
  "crv": "P-256",
  "x": "Ze2loSV3wrroKUN_4zhwGhCqo3Xhultd4QjeQ5wIVR0",
  "y": "HlLtdXARY_f55A3fnzQbPcm6hgr34Mp8p-nuzQCE0Zw",
  "d": "r_kHyZ-a06rmxM3yESK84rlotSg-aQcVStkRhA-iCM8"
}
```

Figure 120: Elliptic Curve P-256 Key

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

### 5.5.2. Generated Factors

The following is generated before encrypting:

- o 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:

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

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "mPUKT_bAWGHIhg0TpjjqVsPlrXWQu_vwVOHhtNkdYoA",
  "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs",
  "d": "AtH35vJsQ9SGjYfOsUxYXQKrPH3FjZHmEtSKoSN8cM"
}
```

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_mFRhKsKLEzPfshfXs9l6areCc
```

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

### 5.5.4. Encrypting the Content

The following is generated before encrypting the content:

- o 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_bAWGHIhg0TpjjqVsPlrXWQu_vwVOHhtNkdYoA",
    "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs"
  },
  "enc": "A128CBC-HS256"
}
```

Figure 124: JWE Protected Header JSON

```
eyJhbGciOiJFQ0RILUVTTiwiia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjojUC0yNTYi
LCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqanFWclAxcldXUXVfdndWT0hIdE5rZF
lvQSIsInkiOiI4QlFBc0ltR2VBuzQ2Zn1XdzVNaFlmR1RUMElqQnBGdzJTUzM0
RHY0SXJzIn0sImVuYyI6IkExMjhdQkMtSFMyNTYifQ
```

Figure 125: JWE Protected Header, base64url-encoded

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

- o CEK (Figure 123);
- o Initialization Vector (Figure 121); and
- o JWE Protected Header (Figure 125) as authenticated data

produces the following:

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

```
BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4OPKbWE1zSTEFjDfhU9
IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEsDIqAYtskTTmzmzNa-_q4F_e
vAPUmwlo-ZG45Mnq4uhMlfm_D9rBtWolqZSF3xGNNkpOMQKF1Cl8i8wjzRli7-
IXgyirlKQsbhhqRzkv8IcY6aHl24j03C-AR2le1r7URUharM79BY8soZU0lzwI
-sD5PZ3l4NDCCei9XkoIAfsXJWmySPoerB2Ni5UZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K767G9C9Azp73gKZD0DyUnlmm0WW5LmyX_yJ-3AR0q8p1WZBfG-ZyJ6l
95_JGG2m9Csg
```

Figure 126: Ciphertext, base64url-encoded

```
WCCKNa-x4BeB9hIDIfFuhg
```

Figure 127: Authentication Tag, base64url-encoded

#### 5.5.5. Output Results

The following compose the resulting JWE object:

- o JWE Protected Header (Figure 114)
- o Initialization Vector (Figure 110)
- o Ciphertext (Figure 115)
- o 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:

```
eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwizXBrIjp7Imt0eSI6IkVDIiwia3J2IjojUC0yNTYi
LCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqanFWclAxcldhXUXVfdndWT0hIdE5rZF
lvQSIIsInkiOiI4QlFBc0ltR2VBuzQ2ZnlXdzVNaFlmRlRUMElqQnBGdzJTUzM0
RHY0SXJzIn0sImVuYyI6IkExMjhDQkMtSFMMyNTYifQ
.
.
yc9N8v5sYyv3iGQT926IUg
.
BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4OPKbWE1zSTEFjDfhU9
IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEsDIqAYtskTTmzmzNa-_q4F_e
vAPUmwlo-ZG45Mnq4uhMlfm_D9rBtWolqZSF3xGNNkpOMQKF1Cl8i8wjzRli7-
IXgyirlKQsbhhqRzkv8IcY6aHl24j03C-AR2lelr7URUArM79BY8soZU0lzwI
-sD5PZ3l4NDCCei9XkoIAfsXJWmySPoerb2Ni5UZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K767G9C9Azp73gKZD0DyUnlmn0WW5LmyX_yJ-3AR0q8p1WZBfG-ZyJ6l
95_JGG2m9Csg
.
WCCKNa-x4BeB9hIDIfFuhg
```

Figure 128: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYn
    JhbmR5YnVja0BidWNrbGFuZC5leGFtcGxlIiwizXBrIjp7Imt0eSI6Ik
    VDIiwia3J2IjojUC0yNTYiLCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqan
    FWclAxcldhXUXVfdndWT0hIdE5rZFlvQSIIsInkiOiI4QlFBc0ltR2VBuz
    Q2ZnlXdzVNaFlmRlRUMElqQnBGdzJTUzM0RHY0SXJzIn0sImVuYyI6Ik
    ExMjhDQkMtSFMMyNTYifQ",
  "iv": "yc9N8v5sYyv3iGQT926IUg",
  "ciphertext": "BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4O
    PKbWE1zSTEFjDfhU9IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEs
    DIqAYtskTTmzmzNa-_q4F_evAPUmwlo-ZG45Mnq4uhMlfm_D9rBtWolq
    ZSF3xGNNkpOMQKF1Cl8i8wjzRli7-IXgyirlKQsbhhqRzkv8IcY6aHl2
    4j03C-AR2lelr7URUArM79BY8soZU0lzwI-sD5PZ3l4NDCCei9XkoIA
    fsXJWmySPoerb2Ni5UZL4mYpvKDiwmyzGd65KqVw7MsFfI_K767G9C9A
    zp73gKZD0DyUnlmn0WW5LmyX_yJ-3AR0q8p1WZBfG-ZyJ6l95_JGG2m9
    Csg",
  "tag": "WCCKNa-x4BeB9hIDIfFuhg"
}
```

Figure 129: General JWE JSON Serialization

## 5.6. Direct Encryption Using AES-GCM

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

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

### 5.6.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

### 5.6.2. Generated Factors

The following is generated before encrypting:

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

refa467QzzKx6QAB

Figure 131: Initialization Vector, base64url-encoded

### 5.6.3. Encrypting the Content

The following is generated before encrypting the content:

- o 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

```
eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MT
diNWlONTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0
```

Figure 133: JWE Protected Header, base64url-encoded

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

- o CEK (Figure 130);
  - o Initialization Vector (Figure 131); and
  - o JWE Protected Header (Figure 133) as authenticated data
- produces the following:
- o Ciphertext from Figure 134.
  - o Authentication Tag from Figure 135.

```
JW_i_f52hww_ELQPGaYyeAB6HYGcr559l9TYnSovc23XJoBcW29rHP8yZOZG7Y
hLpT1bjFuvZPjQS-m0IFtVcXkZXdh_lr_FrdYt9HRUYkshtMmIUAYGmUnd9zM
DB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_
BIwCPTjb9o0SbdcdREEMJMyZBH8ySWMVilgPD9yxi-aQpGbSv_F9N4IZAxscj5
g-NJsUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRsIn
ZI-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios6lEKgyIQ4CWao8PFMj8TTnp
```

Figure 134: Ciphertext, base64url-encoded

```
vbb32Xvlllea2OtmHAdccRQ
```

Figure 135: Authentication Tag, base64url-encoded

#### 5.6.4. Output Results

The following compose the resulting JWE object:

- o JWE Protected Header (Figure 133)
- o Initialization Vector (Figure 131)
- o Ciphertext (Figure 134)
- o 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:

```
eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MT
diNWl0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0
.
.
refa467QzzKx6QAB
.
JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9TYnSovc23XJoBcW29rHP8yZOZG7Y
hLpT1bjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUYkshtrMmIUAYGmUnd9zM
DB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_
BIwCPTjb9o0SbdcdREEMJMyZBH8ySWMVilgPD9yxi-aQpGbSv_F9N4IZAxscj5
g-NJsUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRsIn
ZI-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios6lEKgyIQ4CWao8PFmj8TTnp
.
vbb32Xvllea2OtmHAdccRQ
```

Figure 136: JWE Compact Serialization



The resulting JWE object using the general JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MTdiNWl0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcr559l9TYnSovc23XJoBcW29rHP8yZOZG7YhLpTlbjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUYkshtrMmIUAYGmUnd9zMDB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_BIwCPTjb9o0SbdcdREEMJMyZBH8ySWMVilgPD9yxi-aQpGbSv_F9N4IZAxscj5g-NJsUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRSInZI-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios6lEKgyIQ4CWao8PFMj8TTnp",
  "tag": "vbb32Xvlllea2OtmHAdccRQ"
}
```

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:

- o Plaintext content; this example uses the content from Figure 72.
- o AES symmetric key; this example uses the key from Figure 138.
- o "alg" parameter of "A256GCMKW".
- o "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:

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

UWxARpat23nL9ReIj4WG3Dlee9I4r-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:

- o 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:

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

produces the following:

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

```
lJf3HbOApXMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNok
```

Figure 142: Encrypted Key, base64url-encoded

```
kfPduVQ3T3H6vnewt--ksw
```

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

#### 5.7.4. Encrypting the Content

The following is generated before encrypting the content:

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

```
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "kfPduVQ3T3H6vnewt--ksw",
  "iv": "KkYT0GX_2jHlfqN_",
  "enc": "A128CBC-HS256"
}
```

Figure 144: JWE Protected Header JSON

```
eyJhbGciOiJBbmJlU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRMlQzSDZ2bmV3dC0ta3N3
IiwiaXYiOiJLa1lUMEdyXzJqSGxmcU5fIiwiaZW5jIjoieTEyOENCQy1lUzIlNi
J9
```

Figure 145: JWE Protected Header, base64url-encoded

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

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

produces the following:

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

```
Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8EqoDZHyFKFBupS8iaE
eVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhWyWtZKX0gxKdy6HgLvqoGNbZCz
LjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQHLcqAHxy51449xkjZ7ewzZaGV3eFq
hpco8o4DijXaG5_7kp3h2cajRfDgymuxUbWgLaenQaJtvJmSMFuEOSAzW9Hde
b6yhdTynCRmu-kqtO5Dec4lT2OMZKpnxc_Fl_4yDJFcqb5CiDSmA-psB2k0Jtj
xAj4UPI6loONK7zzFIu4gBfjJCndsZfdvG7h8wGjV98QhrKErR7xKZ3KCr0_qR
lB-gxpNk3xWU
```

Figure 146: Ciphertext, base64url-encoded

```
DKW7jrb4WaRSNfbXVPlT5g
```

Figure 147: Authentication Tag, base64url-encoded

### 5.7.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJBbmJlU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRMlQzSDZ2bmV3dC0ta3N3IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIiwizW5jIjoIQTEyOENCQy1lUzI1NiJ9
.
lJf3HbOApXMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNok
.
gz6NjyEFNm_vm8Gj6FwoFQ
.
Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8EqoDZHyFKFBupS8iaE
eVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhWyWtZKX0gxKdy6HgLvqoGNbZCz
LjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQHLcqAHxy51449xkjZ7ewzZaGV3eFq
hpc08o4Di jXaG5_7kp3h2ca jRfDgymuxUbWgLqaeNQaJtvJmSMFuEOSAzW9Hde
b6yhdTynCRmu-kqtO5Dec4lT2OMZKpnc_Fl_4yDJFcqb5CiDSmA-psB2k0Jtj
xAj4UPI6loONK7zzFIu4gBf jJCndsZfdvG7h8wGjV98QhrKEr7xKZ3KCr0_qR
lB-gxpNk3xWU
.
DKW7jrb4WaRSNfbXVPlT5g
```

Figure 148: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "lJf3HbOApXMEBkCMOoTnnABxs_CvTWUmZQ2ElL
        vYNok"
    }
  ],
  "protected": "eyJhbGciOiJBbmJlU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS
    1iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdV
    ZRMlQzSDZ2bmV3dC0ta3N3IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIi
    wiczW5jIjoiQTEyOENCQy1lUzI1NiJ9",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8E
    qoDZHyFKFBupS8iaEeVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhZWyW
    tZKX0gxKdy6HgLvqoGNbZCzLjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQ
    HLcqAHxy51449xkjZ7ewzZaGV3eFqhpc08o4Di jXaG5_7kp3h2cajRfD
    gymuxUbWgLqaeNqaJtvJmSMFuEOSAzW9Hdeb6yhdTynCRmu-kqtO5Dec
    4lT2OMZKpncx_Fl_4yDJFcqb5CiDSmA-psB2k0Jt jxAj4UPI6loONK7z
    zFIu4gBf jJCndsZfdvG7h8wGjV98QhrKErR7xKZ3KCr0_qRlB-gxpNk3
    xWU",
  "tag": "DKW7jrb4WaRSNfbXVPlT5g"
}
```

Figure 149: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJIbWJlbnR0NNS1ciLCJpdiiI6IktrWVQwRlhfMm
    pIbGZxTl8iLCJraWQiOiIxoGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYj
    RkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRMlQzSDZ2bmV3dC0ta3N3Ii
    wiczW5jiIjoIQTEyOENCQy1IUzI1NiJ9",
  "encrypted_key": "lJf3HbOApXMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNo
    k",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8E
    qoDZHyFKFBups8iaEeVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhZWyW
    tZKX0gxKdy6HgLvqoGNbZCzLjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQ
    HLCqAHxy51449xkjZ7ewzZaGV3eFqhpc08o4Di jXaG5_7kp3h2cajRfD
    gymuxUbWgLaenQaJtvJmSMFuEOSAzW9Hdeb6yhdTynCRmu-kqt05Dec
    4lT2OMZKpnxc_Fl_4yDJFcqb5CiDSmA-psB2k0JtjxAj4UPI6loONK7z
    zFIu4gBfjJCndsZfdvG7h8wGjV98QhrKEr7xKZ3KCr0_qRlB-gxpNk3
    xWU",
  "tag": "NvBveHr_vonkvflfnUrmBQ"
}
```

Figure 150: Flattened JWE JSON Serialization

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

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

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

### 5.8.1. Input Factors

The following are supplied before beginning the encryption process:

- o Plaintext content; this example uses the content from Figure 72.
- o AES symmetric key; this example uses the key from Figure 151.
- o "alg" parameter of "A128KW".
- o "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:

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

```
aY5_Ghmk9KxWPBLu_glx1w
```

Figure 152: Content Encryption Key, base64url-encoded

```
Qx0pmsDa8KnJc9Jo
```

Figure 153: Initialization Vector, base64url-encoded

#### 5.8.3. Encrypting the Key

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

```
CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx
```

Figure 154: Encrypted Key, base64url-encoded

#### 5.8.4. Encrypting the Content

The following is generated before encrypting the content:

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



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

Figure 155: JWE Protected Header JSON

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMzMzZkdktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0
```

Figure 156: JWE Protected Header, base64url-encoded

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

- o CEK (Figure 152);
- o Initialization Vector (Figure 153); and
- o JWE Protected Header (Figure 156) as authenticated data

produces the following:

- o Ciphertext from Figure 157.
- o Authentication Tag from Figure 158.

```
AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tPlbTdhtFJgJxeVmJkLD6
1AlhnWGetdg1lc9ADsnWgL56NyxwSYjU1ZEHcGkd3EkU0vjHi9gTlb90qSYFfe
F0LwkcTtjbYKCsINJQkcIplyeM03OmuiYSoyYJVSpf7ej6zaYcMv3WwdxDF18RE
wOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHLQeg7gDT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJfqFvETUxLAF-KjcBTS5dNy6egwkYtOt8EIHk-oEsKYtZRa
a8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kETHkaSqdYw0FkQZF
```

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:

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

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMzMzMyLTQzZDktYTQ2OC  
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0  
.  
CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx  
.  
Qx0pmsDa8KnJc9Jo  
.  
AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkVRkH0tP1bTdhtFJgJxeVmJkLD6  
1A1hnWGetdg1lc9ADsnWgL56NyxwSYjU1ZEHcGkd3EkU0vjHi9gTlb90qSYFfe  
F0LwkcTtjbYKCsINJQkcIplyeM03OmuiYSoyYJVSpf7ej6zaYcMv3WwdxDF18RE  
wOhNImk2Xld2JXq6BR53TSFkyT7PwVLUq-1GwtGHlQeg7gDT6xW0JqHDPn_H-p  
uQsmthc9Zg0ojmJfqFvETUxLAF-KjcBTS5dNy6egwkYtOt8EIHk-oEsKYtZR  
a8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kETHkaSqdYw0FkQZF  
.  
ER7MWJZ1FBI_NKvn7Zb1Lw
```

Figure 159: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvrRkH0tPlbTdhtFJgJxeVmJkLD6lAlhnWGetdg1lc9ADsnWgL56NyxwSYjU1ZEhcGkd3EkU0vjHi9gTlb90qSYFfeF0LwkcTtjbYKCSiNJQkcIplyeM03OmuiYSoYJVSpf7ej6zaYcMv3WwdxDF18REwOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHlQeg7gDT6xW0JqHDPn_H-puQsmthc9Zg0ojmJfqgFvETUxLAF-KjcBTS5dNy6egwkYtOt8EiHK-oEsKYtZRaa8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kETHkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI_NKvn7Zb1Lw"
}
```

Figure 160: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvrRkH0tPlbTdhtFJgJxeVmJkLD6lAlhnWGetdg1lc9ADsnWgL56NyxwSYjU1ZEhcGkd3EkU0vjHi9gTlb90qSYFfeF0LwkcTtjbYKCSiNJQkcIplyeM03OmuiYSoYJVSpf7ej6zaYcMv3WwdxDF18REwOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHlQeg7gDT6xW0JqHDPn_H-puQsmthc9Zg0ojmJfqgFvETUxLAF-KjcBTS5dNy6egwkYtOt8EiHK-oEsKYtZRaa8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kETHkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI_NKvn7Zb1Lw"
}
```

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:

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

### 5.9.2. Generated Factors

The following are generated before encrypting:

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

```
bY_BDcIwDEVX-QNU3QEOrIA4pqlDokYxchxVvbEDGzIJbioOSJwc-f____HPjBu
8KVFpVtAplVE1-wZo0YjNZo3C7R5v72pV5f5X382VWjYQpqZKAYjziZOr2B7kQ
PSy6oZIXUnDYbVKN4jNXi2u0yB7t1qSHTjmMODf9QgvrDzftTIQXnyQRuUya4zI
WG3vTOdir0v7BRHFYWq3k1k1A_gSDJqtcBF-GZxw8
```

Figure 162: Compressed Plaintext, base64url-encoded

```
hC-MpLZSuwWv8sexS6ydfw
```

Figure 163: Content Encryption Key, base64url-encoded

```
p9pUq6XHY0jfeZIl
```

Figure 164: Initialization Vector, base64url-encoded

### 5.9.3. Encrypting the Key

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

```
5vUT2WotQxKWcekM_IzVQwkGgzlFDwPi
```

Figure 165: Encrypted Key, base64url-encoded

### 5.9.4. Encrypting the Content

The following is generated before encrypting the content:

- o 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

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMzLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiaWVmlwIjoireVGIiwiaXN0
```

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:

- o CEK (Figure 163);
- o Initialization Vector (Figure 164); and
- o JWE Protected Header (Figure 167) as authenticated data

produces the following:

- o Ciphertext from Figure 168.
- o Authentication Tag from Figure 169.

```
HbDtOsdailoYziSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyez
SPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
```

Figure 168: Ciphertext, base64url-encoded

```
VILuUwuIxaLVmh5X-T7kmA
```

Figure 169: Authentication Tag, base64url-encoded

#### 5.9.5. Output Results

The following compose the resulting JWE object:

- o JWE Protected Header (Figure 167)
- o Encrypted Key (Figure 165)
- o Initialization Vector (Figure 164)
- o Ciphertext (Figure 168)
- o Authentication Tag (Figure 169)

The resulting JWE object using the JWE Compact Serialization:

```

eyJhbGciOiJBMTI1S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI1R0NNIiwiemlwIjoieREVVGIn0
.
5vUT2WotQxKWcekM_IzVQwkGgzlFDwPi
.
p9pUq6XHY0jfeZiI
.
HbDtOsdailoYziSx25KEeTxmwnh8L8jKMFNclK3zmMI6VB8hry57tDZ61jXyez
Spt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
.
VILuUwuIxaLVmh5X-T7kmA

```

Figure 170: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "5vUT2W0tQxKWcekM_IzVQwkGgzlFDwPi"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiaWVmlWJjoREVGVhbn0i",
  "iv": "p9pUq6XHY0jfeZiL",
  "ciphertext": "HbDtOsdailoYziSx25KEeTxmwnh8L8jKMFNcl3zmMI6VB8hry57tDZ61jXyezSPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMqOn9J--Xhhlyg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw",
  "tag": "VILuUwuIxaLVmh5X-T7kma"
}
```

Figure 171: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiaWmlwIjoiREVVGIn0",
  "encrypted_key": "5vUT2W0tQxKWcekM_IzVQwkGgzlFDwPi",
  "iv": "p9pUq6XHY0jfeZiI",
  "ciphertext": "HbDtOsdailoYziSx25KEeTxmwnh8L8jKMFNcl3zmMI6VB8hry57tDZ6ljXyezSpt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMQmOn9J--Xhhlyg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw",
  "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:

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

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

75mlALsYv10pZTKPWrsqdg

Figure 174: Content Encryption Key, base64url-encoded

veCx9ece2orS7c\_N

Figure 175: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsWlsidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxbImZuIix7fS  
widGV4dCIsIk1lcmlhZG9jIEJyYW5keWJlY2siXSxbIm4iLHt9LCJ0ZXh0Iixb  
IkJyYW5keWJlY2siLCJNZXJpYWRvYyIsIklyLiIsIiJdXSxbImJkYXkiLHt9LC  
J0ZXh0IiwieVEEGMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV4dCIsIk0iXVld

Figure 176: Additional Authenticated Data, base64url-encoded

### 5.10.3. Encrypting the Key

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

```
4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X
```

Figure 177: Encrypted Key, base64url-encoded

### 5.10.4. Encrypting the Content

The following is generated before encrypting the content:

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

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

Figure 178: JWE Protected Header JSON

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMzLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0
```

Figure 179: JWE Protected Header, base64url-encoded

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

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

produces the following:

- o Ciphertext from Figure 180.
- o Authentication Tag from Figure 181.

```
Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0Ui8p74SchQP8xygM1
oFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14T_4NFqF-p2Mx8zkbKxI7oPK
8KNarFbyxIDvICNqBLba-v3uzXBdB89fzOI-Lv4PjOFAQGHrgv1rjXAmKbgkft
9cB4WeyZw8MldbBhc-V_KWZslrsLNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4a
q3FXBxOxCys35PhCdaggy2kfUfl2OkwKnWUbgXVD1C6HxLilqHhCwXDG59weHr
RDQeHyMRoBljoV3X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV
```

Figure 180: Ciphertext, base64url-encoded

```
vOaH_RajnpY_3hOtqvZHRA
```

Figure 181: Authentication Tag, base64url-encoded

#### 5.10.5. Output Results

The following compose the resulting JWE object:

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

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

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv": "veCx9ece2orS7c_N",
  "aad": "WyJ2Y2FyZCIsWlsidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxbImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJlY2siXSxbIm4iLHt9LCJ0ZXh0IixbIkYyZW5keWJlY2siLCJNZXJpYWRvYyIsIklyLiIsIiJdXSxbImJkYXkiLHt9LCJ0ZXh0IiwieVEgMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV4dCIsIk0iXVld",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0Ui8p74SchQP8xygMloFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14T_4NFqF-p2Mx8zkbKxI7oPK8KNarFbyxIDvICNqBLba-v3uzXBdB89fzOI-Lv4PjOFAQGHrgv1rjXAmKbgkft9cB4WeyZw8MldbBhc-V_KWZslrsLNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4aq3FXBxOxCys35PhCdaggy2kfUfl2OkwKnWUbgXVD1C6HxLilqHhCwXDG59weHrRDQeHyMRoBljoV3X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQakFV",
  "tag": "vOaH_RajnpY_3hOtqvZHRA"
}
```

Figure 182: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X",
  "aad": "WyJ2Y2FyZCIsWlsidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxbImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJlY2siXSxbIm4iLHt9LCJ0ZXh0IixbIkYyYW5keWJlY2siLCJNZXJpYWRvYyIsIklyLiIsIiJdXSxbImJkYXkiLHt9LCJ0ZXh0IiwiVEEgMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV4dCIsIk0iXVld",
  "iv": "veCx9ece2orS7c_N",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0Ui8p74SchQP8xygMloFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14T_4NFqF-p2Mx8zkbKxI7oPK8KNarFbyxIDvICNqBLba-v3uzXBdB89fzOI-Lv4PjOFAQGHrgv1rjXAmKbgkft9cB4WeyZw8MldbBhc-V_KWZslrsLNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4aq3FXBxOxCys35PhCdaggy2kfUfl2OkwKnWUbgXVD1C6HxLilqHhCwXDG59weHrRDQeHyMRoBljoV3X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV",
  "tag": "vOaH_RajnpY_3hOtqvZHRA"
}
```

Figure 183: Flattened JWE JSON Serialization

#### 5.11. Protecting Specific Header Fields

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

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

##### 5.11.1. Input Factors

The following are supplied before beginning the encryption process:

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

#### 5.11.2. 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 184.
- o Initialization Vector; this example uses the Initialization Vector from Figure 185.

```
WDgEptBmQs9ouUvArz6x6g
```

Figure 184: Content Encryption Key, base64url-encoded

```
WgEJsDS9bkoXQ3nR
```

Figure 185: Initialization Vector, base64url-encoded

#### 5.11.3. Encrypting the Key

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

```
jJIcM9J-hbx3wnqhf5FlkEYos0sHsF0H
```

Figure 186: Encrypted Key, base64url-encoded

#### 5.11.4. Encrypting the Content

The following is generated before encrypting the content:

- o 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:

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

produces the following:

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

```
lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2DM3swKkjOwQyZtWsFL
YMj5YeLht_StAn2ltHmQJuuNt64T8D4t6C7kC9OCCJ1IHAolUv4MyOt80MoPb8
fZYbNKqplzYJgIL58g8N2v46OgyG637d6uuKPwhAntGm_zWhqc_srOvgiLkzyF
XPq1lhBAURbc3-8BqeRb48iR1-_5g5UjWVD3lgiLCN_P7AW8mIiFvUNXBPJK3nO
WL4teUPS8yHLbWeL83olU4UAgL48x-8dDkH23JykibVSQju-f7e-1xreHWXzWL
HslNqBbre0dEwK3HX_xM0LjUz77Krppgegoutpf5qaKg3l-_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:

- o JWE Shared Unprotected Header (Figure 191)
- o JWE Protected Header (Figure 188)
- o Encrypted Key (Figure 186)
- o Initialization Vector (Figure 185)
- o Ciphertext (Figure 189)
- o 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-hbx3wnqhf5FlkEYos0sHsF0H"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "1IbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D
M3swKkjOwQyZtWsFLYMj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC9O
CCJ1IHAolUv4MyOt80MoPb8fZYbNKqplzYJgIL58g8N2v46OgyG637d6
uuKPwhAnTGm_zWhqc_srOvgiLkzyFXPqlhBAURbc3-8BqeRb48iR1-_5
g5UjWVD3lgiLCN_P7AW8mIiFvUNXBPJK3nOWL4teUPS8yHLbWeL83olU
4UAgL48x-8dDkH23JykibVSQju-f7e-1xreHWXzWLHs1NqBbre0dEwK3
HX_xM0LjUz77Krppgegoutpf5qaKg3l-_xMINmf",
  "tag": "fNYLqpUe84KD45lvDiaBAQ"
}
```

Figure 192: General JWE JSON Serialization



The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "encrypted_key": "jJicM9J-hbx3wnqhf5FlkEYos0sHsF0H",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D
M3swKkjOwQyZtWsFLYMj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC9O
CCJ1IHAolUv4MyOt80MoPb8fZYbNKqplzYJgIL58g8N2v46OgyG637d6
uuKPwhAntGm_zWhqc_srOvgiLkzyFXPqlhBAURbc3-8BqeRb48iR1-_5
g5UjWVD3lgiLCN_P7AW8mIiFvUNXBPJK3nOWL4teUPS8yHLbWeL83olU
4UAgL48x-8dDkH23JykibVSQju-f7e-1xreHWXzWLHs1NqBbre0dEwK3
HX_xM0LjUz77Krppgegoutpf5qaKg3l-_xMINmf",
  "tag": "fNYLqpUe84KD45lvDiaBAQ"
}
```

Figure 193: Flattened JWE JSON Serialization

## 5.12. Protecting Content Only

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

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

### 5.12.1. Input Factors

The following are supplied before beginning the encryption process:

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

#### 5.12.2. Generated Factors

The following are generated before encrypting:

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

KBooAF130QPV3vkcZlXnzQ

Figure 194: Content Encryption Key, base64url-encoded

YihBoVOGsR1l7jCD

Figure 195: Initialization Vector, base64url-encoded

#### 5.12.3. Encrypting the Key

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

244YHfO\_W7RMpQW81UjQrZcq5LSyqiPv

Figure 196: Encrypted Key, base64url-encoded

#### 5.12.4. Encrypting the Content

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

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

produces the following:

- o Ciphertext from Figure 197.
- o Authentication Tag from Figure 198.

```
qtPIMMaOBRgASL10dNqHOa7Gqrk7Eallvwh7R4TTluq-arsVCPaIeFwQfzrSS
6oEUWbBtxEase0vC6r7sphyVziMCVJEuRJyoAHFSP3eqQPb4Ic1SDSqyXjw_L3
svybhHYUGyQuTmUQEDjgjJfBOifwHIsDsRPeBz1NomqeifVPq5GTCWFO5k_MNI
QURR2Wj0AHC2k7JZfu2iWjUHLf8ExFZLZ4nlmsvJu_mvifMYiikfNfsZAudISO
a6O73yPZtL04k_1FI7WDfrb2w7OqKLWDXzlpcoxhPVOLQwpA3mFNRKdY-bQz4Z
4KX9lfz1cne31N4-8BKmojpW-OdQjKdLOGkC445Fb_K1t1DQXw2sBF
```

Figure 197: Ciphertext, base64url-encoded

```
e2m0Vm7JvjK2VpCKXS-kyg
```

Figure 198: Authentication Tag, base64url-encoded

#### 5.12.5. Output Results

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

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

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

Figure 199: JWE Shared Unprotected Header JSON

The following compose the resulting JWE object:

- o JWE Shared Unprotected Header (Figure 199)
- o Encrypted Key (Figure 196)
- o Initialization Vector (Figure 195)
- o Ciphertext (Figure 197)
- o Authentication Tag (Figure 198)

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoVOGsR1l7jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQhOa7Gqrk7Eallvwht7R4TTluq-
arsVCPaIeFwQfzrSS6oEUWbBtxEasE0vC6r7sphyVziMCVJEUrJyoAHF
SP3eqQPb4IclSDSqyXjw_L3svybhHYUGyQuTmUQEDjgjJfBOifwHIsDs
RPeBz1NomqeifVPq5GTCWfo5k_MNIQURR2Wj0AHC2k7JZfu2iWjUHLf8
ExFZLZ4nlmsvJu_mvifMYiikfNfsZAudISOa6O73yPZtL04k_1FI7Wdf
rb2w7OqKLWDXzlpcoxhPVOLQwpA3mFNRKdY-bQz4Z4KX9lfzlcne31N4
-8BKmojpw-OdQjKdLOGkC445Fb_K1tldQXw2sBF",
  "tag": "e2m0Vm7JvjK2VpCKXS-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": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv",
  "iv": "YihBoVOGsR1l7jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQhOa7Gqrk7Eallvwht7R4TTluq-
arsVCPaIeFwQfzrSS6oEUWbBtxEasE0vC6r7sphyVziMCVJEUrJyoAHF
SP3eqQPb4IclSDSqyXjw_L3svybhHYUGyQuTmUQEDjgjJfBOifwHIsDs
RPeBz1NomqeifVPq5GTCWfo5k_MNIQURR2Wj0AHC2k7JZfu2iWjUHLf8
ExFZLZ4nlmsvJu_mvifMYiikfNfsZAudISOa6O73yPZtL04k_1FI7Wdf
rb2w7OqKLWDXzlpcoxhPVOLQwpA3mFNRKdY-bQz4Z4KX9lfzlcne31N4
-8BKmojpw-OdQjKdLOGkC445Fb_K1tldQXw2sBF",
  "tag": "e2m0Vm7JvjK2VpCKXS-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:

- o Plaintext content; this example uses the Plaintext from Figure 72.
- o 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.
- o 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.
- o Content encryption algorithm; this example uses "A128CBC-HS256".

#### 5.13.2. 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 202.
- o Initialization Vector; this example uses the Initialization Vector from Figure 203.

```
zXayeJ4gvm8NJr3IUInyokTUO-LbQNKEhe_zWlYbdpQ
```

Figure 202: Content Encryption Key, base64url-encoded

```
VgEIH20EnzUtZFl2RpBlg
```

Figure 203: Initialization Vector, base64url-encoded

### 5.13.3. Encrypting the Key to the First Recipient

Performing the "RSA1\_5" key encryption operation over the CEK (Figure 202) with the first recipient's RSA key (Figure 73) produces the following Encrypted Key:

```
dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w6Y5G4XJQsNNIBiqyvUUA
OcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXihidAYWVajJIa
KMXMvFRMV6iDlRr076DFthg2_AV0_tSiV6xSEIFqtlxnYPpmP9ltc5WJDOb-w
qjw0-b-S1laS1lQVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9clNlLnZMbeYS
rRicJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-d
bHcGlihq_cWGgho9fLMK8JOArYLcMDNQ
```

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

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

- o Recipient JWE Unprotected Header from Figure 205.

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

Figure 205: Recipient #1 JWE Per-Recipient Unprotected Header JSON

The following is the assembled first recipient JSON:

```
{
  "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w
6Y5G4XJQsNNIBiqyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4b
WrBf7cHwEQJdXihidAYWVajJIaKMxMvFRMV6iDlRr076DFthg2_AV0_t
SiV6xSEIFqtlxnYPpmP91tc5WJDOGb-wqjw0-b-S1laS1lQVbuP78dQ7
Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbeYSrRicJK5xodvWgkpIdk
MHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-dbHcGlihqc_wG
gho9fLMK8JOArYLcMDNQ",
  "header": {
    "alg": "RSA1_5",
    "kid": "frodo.baggins@hobbiton.example"
  }
}
```

Figure 206: Recipient #1 JSON

#### 5.13.4. Encrypting the Key to the Second Recipient

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

- o 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-I68i8g2b8mva8diRhsE2xAn2Dt
MRb25Ma2CX",
  "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLi0y3ylEjI1pOMbw9
1fzZ84pbfm",
  "d": "lDKHfTv-PiifVw2VBHM_ZiVcwOMxkOyANS_lQHJcrDxVY3jhVCvZPw
MxJKIE793C"
}
```

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:

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

produces the following Encrypted Key:

```
ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw_elY4gSSId_w
```

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

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

- o Recipient JWE Unprotected Header from Figure 209.

```
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2
        DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLi0y3ylEjI1pOMb
        w91fzZ84pbfm"
  }
}
```

Figure 209: Recipient #2 JWE Per-Recipient Unprotected Header JSON



The following is the assembled second recipient JSON:

```
{
  "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw
    _eLY4gSSId_w",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I68i8g2b8mva8diRhsE2xA
        n2DtMRb25Ma2CX",
      "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEjI1pO
        MbW91fzZ84pbfm"
    }
  }
}
```

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:

- o 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:

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

produces the following:

- o Encrypted Key from Figure 212.
- o Authentication Tag from Figure 213.

```
a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E
```

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

```
59Nqh1LlYtVIhfD3pgRGvw
```

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

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

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

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

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

The following is the assembled third recipient JSON:

```
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1LlYtVIhfD3pgRGvw",
    "iv": "AvpeoPZ9Ncn9mkBn"
  }
}
```

Figure 215: Recipient #3 JSON

#### 5.13.6. Encrypting the Content

The following is generated before encrypting the content:

- o JWE Protected Header; this example uses the header from Figure 216, encoded to base64url [RFC4648] as Figure 217.

```
{
  "enc": "A128CBC-HS256"
}
```

Figure 216: JWE Protected Header JSON

```
eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 217: JWE Protected Header, base64url-encoded

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

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

```
ajm2Q-OpPXC7-MHXicknbllsxLdXxK_yLds0KuhJzfWK04SjdxQeSw2L9mu3a
_k1C55kCQ_3xlkcVKC5yr__Is48VOoK0k63_QRM9tBURMFqLByJ8vOYQX0oJW4
VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSaP6ga7-siYxStR7_G07Thdljh-zGT0
wxM5g-VRORtq0K6AXpLlwEqRp7pkt2zRM0ZAXqSpe106FJ7FHLdYEFnD-zDIZu
kLpCbzhzMDLLw2-8I14FQrgi-iEuzHgIJFIJn2wh9Tj0cg_kOZy9BqMRZbmYXM
Y9YQjorZ_P_JYG3ARAI30jDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVH2
ofDQdLChataE
```

Figure 218: Ciphertext, base64url-encoded

```
BESYyFN7T09KY7i8zKs5_g
```

Figure 219: Authentication Tag, base64url-encoded

The following is generated after encrypting the Plaintext:

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

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

Figure 220: JWE Shared Unprotected Header JSON

#### 5.13.7. Output Results

The following compose the resulting JWE object:

- o Recipient #1 JSON (Figure 206)
- o Recipient #2 JSON (Figure 210)
- o Recipient #3 JSON (Figure 215)
- o Initialization Vector (Figure 203)
- o Ciphertext (Figure 218)
- o 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:

```
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zj
        wj4w6Y5G4XJQsNNIBiqyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-
        vhW6me4bWrBf7cHwEQJdXihidAYWVajJIaKMXMvFRMV6iDlRr076
        DFthg2_AV0_tSiV6xSEIFqtlxnYPpmP91tc5WJDOGb-wqjw0-b-S
        1laS1lQVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbe
        YSrRicJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n
        5trGuExtP_-dbHcGlihqc_wGgho9fLMK8JOArYLcMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKSHi
        xJuw_ely4gSSId_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3pi5wKCRclizp5_r0OjeqT-I68i8g2b8mva8diRhs
            E2xAn2DtMRb25Ma2CX",
          "y": "VDRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEj
            IlpOMbw91fzZ84pbfm"
        }
      }
    },
    {
      "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-Wy
        TpSlE",
      "header": {
        "alg": "A256GCMKW",
        "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
        "tag": "59Nqh1LlYtVIhfD3pgRGvw",
        "iv": "AvpeoPZ9Ncn9mkBn"
      }
    }
  ],
  "unprotected": {
    "cty": "text/plain"
  },
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
}
```

```

"iv": "VgEIHY20EnzUtZF12RpB1g",
"ciphertext": "ajm2Q-OpPXC7-MHXicknb1lsxLdXxK_yLds0KuhJzfWK
04SjdxQeSw2L9mu3a_k1C55kCQ_3xlkcVKC5yr__Is48VOoK0k63_QRM
9tBURMFqLByJ8vOYQX0oJW4VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSa
P6ga7-siYxStR7_G07Thd1jh-zGT0wxM5g-VR0Rtq0K6AXpLlwEqRp7p
kt2zRM0ZAXqSpel06FJ7FHLdYEFnD-zDIZukLpCbzhzMDLLw2-8I14FQ
rgi-iEuzHgIJFIJn2wh9Tj0cg_kOzy9BqMRZbmYXMY9YQjorZ_P_JYG3
ARAI30jDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVH2ofDQdLcht
azE",
"tag": "BESYyFN7T09KY7i8zKs5_g"
}

```

Figure 221: General JWE JSON Serialization

## 6. Nesting Signatures and Encryption

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

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

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

### 6.1. Signing Input Factors

The following are supplied before beginning the signing operation:

- o Payload content; this example uses the JSON Web Token [JWT] content from Figure 222, encoded as base64url [RFC4648] to produce Figure 223.
- o RSA private key; this example uses the key from Figure 224.
- o "alg" parameter of "PS256".

```

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

```

Figure 222: Payload Content, in JSON Format

```
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiaXhwIjoxMzAwODE5MzgWLCJodH
RwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0
```

Figure 223: Payload Content, base64url-encoded

```
{
  "kty": "RSA",
  "kid": "hobbiton.example",
  "use": "sig",
  "n": "kNrPIBDXMU6fcyv5i-QHQAQ-K8gsC3HJb7FYhYaw8hXbNJa-t8q0lD
    KwLZgQXYV-ffWxXJv5GGrlZE4GU52lfMEegTDzYTrRQ3tepgKFjMGg6I
    y6fkl1ZNsx2gEonsnlShfzA9GJwRTmtKPbkls-hwx1IU5AT-AIelNqBg
    cF2vE5W25_SGGBoaROVdUYxqETDggM1z5cKV4ZjDZ8-lh4oVB07bkac6
    LQdHpJUUYSH_Er20DXx30KyI97PciXKTS-QKXnm8ivvRCmux22ZoPui
    nd2BKC5OiG4MwAlhaL2Z2k8CsRdfy-7dg7z41Rp6D0ZeEvtaUp4bX4aK
    raL4rTfw",
  "e": "AQAB",
  "d": "ZLe_TIxpE9-W_n2VBa-HWvuYPtjvxwVXC1JFOPjsdea8g9RMx34qEO
    EtnoYc2un3CZ3LtJi-mju5RAT8YSc76YJds3ZVw0UiO8mMBeG6-iOnvg
    obobNx7K57-xjTJZU72EjOr9kB7z6ZKwDDq7HFyCDhUEcYcHFVc7iL_6
    TibVhAhOFONWlqlJgEgwVYd0rybNGKifdnpebwyHoMwY6HM1qvnEFgP7
    iZ0YzHUT535x6jj4VKcdA7ZduFkhUauysySEW7mxZM6fj1vdjJIy9LD1
    fIz30Xv4ckoqKF5GONU6tNmMmNgAD6gIViyElelPrIxlltBhCI14bRW
    -zrpHgAQ",
  "p": "yKWYonIAqWMRQlgIBOdT1NIcbDNUUs2Rh-pBaxD_mIkweMt4Mg-0-B
    2iSYvMrs8horhonV7vxCQagcBAATGW-hAafUehWjxWSH-3KccRM8toL4
    e0q7M-idRDOBXSoe7Z2-CV2x_ZCY3RP8qp642R13WgXqGDIM4MbUkZSj
    cY9-c",
  "q": "uND4o15V30KDzf8vFJw589p1vlQVQ3NEilrinRUPHkxxaAzDzccGgr
    WMWpGxGFFnNL3w5CqPLeU76-5IVYQq0HwYVl0hVXQHr7sgaGu-483Ad3
    ENcL23FrOnF45m7_2ooAstJDe49MeLTTQKrSIBl_SKvqpYvfSPTczPcZ
    kh9Kk",
  "dp": "jmTnEoq2qqa8ouaymjhJSCnsveUXnMQC2gAneQJRQkFqQu-zV2PKP
    KNbPvKVyiF5b2-L3tM3OW2d2iNDyRUWXLt7V5l0KwPTABSTOnTqAmYCh
    Gi8kXXdlhcrtSvXldBakC6saxwI_TzGGY2MVXzc2ZnCvCXHV4qjSxOrf
    P3pHFU",
  "dq": "R9FUvU88OVzEkTkXl3-5-Wuse4DjHmndeZilu3rifBdfLpq_P-iWP
    BbGa9wzQ1c-J7SzCdJqkEJDv5yd2C7rnZ6kpzwBh_nmL8zscAk1qsun
    nt9CJGAYz7-sGWylJGShFazfP52ThB4rlCJ0YuEaQMrIzpY77_oLAhpm
    DA0hLk",
  "qi": "S8tC7ZknW6hPITkjcwtQOPLVmRfwirRlFAViudb8NW9CrV_7F2Oq
    UZCqmzHTYAumwGFHilWVRep7anleWaJjxC_1b3fq_al4qH3Pe-EKiHg6
    IMazuRtZLUROcThrExDbF5dYbsciDnfrUWLErZ4N1Be0bnxYuPqxwKd9
    QZwMo0"
}
```

Figure 224: RSA 2048-Bit Private Key, in JWK Format

## 6.2. Signing Operation

The following is generated to complete the signing operation:

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

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

Figure 225: JWS Protected Header JSON

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

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:

dPpMqWRZxFYilUfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ\_aZqciPrz0OWaIA  
kIvnlqskChirjKvY9ESZNUCP4JjvfyPS-nqjJxYoA5ztWOyFk2cZNIPXjcJXSQ  
wXPO9tEe-v4VSqgD0aKHqPxYog4N6Cz1lKph1UlsYDSI67\_bLL7elg\_vkjfMp5  
\_W5l5LuUYGMeh6hxQIaIUxf9EwV2JmvTMuZ-vBOWy0SniylEFo72CRTvmtrIf5  
AR0o5MNliY3KtUxeP-SOmD-LEYwW9SlkohYzMVAZDDOrVbv7KVRHpeYNak75KE  
QqdCEEkS\_rskZS-Qtt\_nlegTWhlmEYaA

Figure 227: JWS Signature, base64url-encoded

## 6.3. Signing Output

The following compose the resulting JWS object:

- o JWS Protected Header (Figure 226)
- o Payload content (Figure 223)
- o Signature (Figure 227)



The resulting JWS object using the JWS Compact Serialization (which is the plaintext input to the following encryption operation):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiaXhwIjoxMzAwODE5MzgWLCJodH
RwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0
.
dPpMqWRZxFYilUfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIA
kIvnlqskChirjKvY9ESZNUCP4JjvfyPS-nqjJxYoA5ztW0yFk2cZNIPXjcJXSQ
wXPO9tEe-v4VSqgD0aKHqPxYog4N6Cz1lKphlUlsYDSI67_bLL7elg_vkjfMp5
_W5l5LuUYGMeh6hxQIaIUxf9EwV2JmvTMuZ-vBOWy0SniylEFo72CRTvmtrIf5
AROO5MNliY3KtUxeP-SOmD-LEYwW9SlkohYzMVAZDDOrVbv7KVRHpeYNak75KE
QqdCEEkS_rskZS-Qtt_nlegTWhlmEYaA
```

Figure 228: JWS Compact Serialization

#### 6.4. Encryption Input Factors

The following are supplied before beginning the encryption process:

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

#### 6.5. Encryption 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 229.
- o Initialization Vector; this example uses the Initialization Vector from Figure 230.

```
0RHSNYwN-6-2QBGsYTZLSQ
```

Figure 229: Content Encryption Key, base64url-encoded

```
GbXli9kXz0sxXPmA
```

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:

```
a0JHROITfpX4qRwImjlStn8m3CPxBVlueYlVhjurCyrBg3I7YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bil3mwy7RoYhjTkBEC6P7sMYMXXx4g
zMedpiJHQVeyI-zkZV7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb2l
00ul4YxSHV-ByKlkyeetRp_fuYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e5I
R7nany-25_UmC2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dMVIY7qleKpXcVllWO_2FefEBqXxXvIjLeZivjNkzogCq3-I
apSjVFnMjBxjpYLT8muaawolyylXXMuinIpNcOY3n4KKrXLRcctex85m4IIHMZ
a38slHpr56fPPseMA-Jltmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBK0Ywfk7JhLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNjTqnbIp
ymooeWAHCT4e_Owbimlg0AEpTHUdA2iiLNs9WTX_H_TXuPC8yDDhilsmxS_X_x
pkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcqxX2Fo_GnVrNwLk7LgXw6FSQvDO
0
```

Figure 231: Encrypted Key, base64url-encoded

## 6.7. Encrypting the Content

The following is generated before encrypting the Plaintext:

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

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

Figure 232: JWE Protected Header JSON

```
eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ
```

Figure 233: JWE Protected Header, base64url-encoded

Performing the content encryption operation over the Plaintext (Figure 228) with the following:

- o CEK (Figure 229);
- o Initialization Vector (Figure 230); and
- o JWE Protected Header (Figure 233) as authenticated data

produces the following:

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

```
SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBNgCe2OFMf66cSJ8k2Q
kxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt90AvVLsAXB0_UTCBGyBg3C2bWLX
qZlfJAAoJRUPRk-BimYZY81zVBuIhc7HsQePCpu33SzMsFHjn4lP_idrJz_glZ
TNgKDt8zdnUPauKTKDNOH1DD4fuzvDYfDIAfqGPyL5sVRwbiXpXdGokEszM-9C
hMPqWlQNhzuX_Zul3bvrJwr7nuGZs4cUScY3n8yE3AHCLurgls-A9mz1X38xEa
ulV18l4Fg9tLejdkAuQZjPbqeqHQBJe4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2
zI3Q_lsYjKUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUr
lx4gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-aM7tXg26qJo2lgYjLf
hn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5XmnwZMyNc
9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgMl7o03phcTMxtlMizR88NKU1WkB
siXMCjylNoue7MD-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:

- o JWE Protected Header (Figure 233)
- o Encrypted Key (Figure 231)
- o Initialization Vector (Figure 230)
- o Ciphertext (Figure 234)
- o Authentication Tag (Figure 235)

The resulting JWE object using the JWE Compact Serialization:

```
eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkJExMjhHQ00ifQ
.
a0JHRoITfpX4qRewImj1Stn8m3CPxBVlueYlVhjurCyrBg3I7YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bil3mwy7RoYhjTkBEC6P7sMYMXXx4g
zMedpiJHQVeyI-zkZV7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb2l
O0ul4YxSHV-ByKlkyeetRp_fuYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e5I
R7nany-25_UmC2urosnkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dMVIY7qleKpXcV1lWO_2FefEBqXxXvIjLeZivjNkzogCq3-I
apSjVFnMjBxjpYLT8muaawolyylXXMuinIpNcOY3n4KKrXLRcctex85m4IIHMZ
a38slHpr56fPPseMA-Jltmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBKOYwfk7JhLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNJtqnb1p
ymooeWAHCT4e_Owbimlg0AEpTHUDa2iilNs9WTX_H_TXuPC8yDDhilsmxS_X_x
pkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcxcqX2Fo_GnVrNwlK7Lgxw6FSQvDO
0
.
Gbx1i9kXz0sxXPmA
.
SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBNgCe2OFMf66cSJ8k2Q
kxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt9OAvVLsAXB0_UTCBGyBg3C2bWLX
qZlfJAAoJRUPRk-BimYZY81zVBuIhc7HsQePCpu33SzMsFHjn4lP_idrJz_g1Z
TNgKDt8zdnUPauKTKDNOH1DD4fuzvDYfDIAfqGPyl5sVRwbiXpXdGokEszM-9C
hMPqWlQNhzux_Zul3bvrJwr7nuGZs4cUScy3n8yE3AHCLurgls-A9mzlX38xEa
ulV18l4Fg9tLejdkAuQZjPbqehQBje4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2
zI3Q_1sYjKUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUr
lx4gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-aM7tXg26qJo21gYjLf
hn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5XmnwZMyNc
9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgm17o03phcTMxtlMizR88NKU1WkB
siXMCjylNoue7MD-ShDp5dmM
.
KnIEhN8U-3C9s4gtSpjSw
```

Figure 236: JWE Compact Serialization

The resulting JWE object using the general JWE JSON Serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBVlueYlVh
        jurCyrBg3I7YhCRYjphDOOS4E7rXbr2Fn6NyQq-A-gqT0FXqNjVO
        GrG-bil3mwy7RoYhjTkBEC6P7sMYMXXx4gzMedpiJHQVeyI-zkZV
        7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb2100ul4YxS
        HV-ByKlkyeetRp_fuYJxHoKQLQL9P424sKx2WGYb4zsBIPF4ssl_e
        5IR7nany-25_UmC2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5
        o6yV64x6yzDUF_5JCIdl-Qv6H5dMVIY7qleKpXcV1lWO_2FefEBq
        XxXvIjLeZivjNkzogCq3-IapSjVFnmjBxjpYLT8muaawolyy1XXM
        uinIpNcOY3n4KKrXLRcteX85m4IIHMZa38s1Hpr56fPPseMA-Jl
        tmt-a9iEdtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAamBK0Ywfk7J
        hLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNJtqnbIp
        ymooeWAHCT4e_Owbimlg0AEpTHUdA2iilNs9WTX_H_TXuPC8yDDh
        ilsmxS_X_xpkIHkiIHWDOlx03BpqDTivpKkBYwqP2UZkcxqX2Fo_
        GnVrNwlK7Lgxw6FSQvDO0"
    }
  ],
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6I6I6IExmJmhhQ00ifQ",
  "iv": "Gbxli9kXz0sxXPmA",
  "ciphertext": "SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBN
    gCe2OFMf66cSJ8k2QkxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt90Av
    VLsAXB0_UTCBGyBg3C2bWLXqZlfJAAoJRUPRk-BimYZY81zVBuIhc7Hs
    QePCpu33SzMsFHjn4lP_idrJz_glZTNgKDt8zdnUPauKTKDNOH1DD4fu
    zvDYfDIAfqGPyl5sVRwbiXpXdGokEszM-9ChMPqWlQNhzuX_Zul3bvrJ
    wr7nuGZs4cUScy3n8yE3AHCLurgls-A9mz1X38xEaulV1814Fg9tLejd
    kAuQZjPbqehQBje4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2zI3Q_1sYj
    KUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUrlx4
    gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-am7tXg26qJo21gY
    jLfhn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5
    XmnwZMyNc9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgM17o03phcTMx
    tlmizR88NKU1WkBsIXMCjy1Noue7MD-ShDp5dmM",
  "tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

Figure 237: General JWE JSON Serialization

The resulting JWE object using the flattened JWE JSON Serialization:

```
{
  "encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBVlueYlVhjurC
    yrBg3I7YhCRYjphDOOS4E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bi13
    mwy7RoYhjTkBEC6P7sMYMXXx4gzMedpiJHQVeyI-zkZV7A9matpgevAJ
    WrXzOUysYGTtwoSN6gtUVtlLaivjvb2l00ul4YxSHV-ByKlkyeetRp_f
    uYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e5IR7nany-25_UmC2uros
    NkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDUF_5JCIdl-Qv6
    H5dMVIY7qleKpXcV1lWO_2FefEBqXxXvIjLeZivjNkzogCq3-IapSjVF
    nMjBxjpyLT8muaawolyylXXMuinIpNcOY3n4KKrXLrCcteX85m4IIHMZ
    a38slHpr56fPPseMA-Jltmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3
    kJusAamBKOYwfk7JhLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15q
    JIEXNjTqbnlpymoeeWAHCT4e_Owbimlg0AEpTHUdA2iiLNs9WTX_H_TX
    uPC8yDDhilsmxS_X_xpkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcqxQ
    2Fo_GnVrNwlK7Lgxw6FSQvDO0",
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYy
    I6IkExMjhHQ00ifQ",
  "iv": "GbXli9kXz0sxXPmA",
  "ciphertext": "SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBN
    gCe2OFMf66cSJ8k2QkxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt90Av
    VLsAXB0_UTCBGyBg3C2bWLXqZlfJAAoJRUPRk-BimYZY81zVBuIhc7Hs
    QePCpu33SzMsFHjn4lP_idrJz_glZTNgKDt8zdnUPauKTKDNOHlDD4fu
    zvDYfDIAfqGPyl5sVRwbiXpXdGokEszM-9ChMPqWlQNhzuX_Zul3bvrJ
    wr7nuGZs4cUScy3n8yE3AHCLurgls-A9mz1X38xEaulV1814Fg9tLejd
    kAuQZjPbqehQBje4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2zI3Q_1sYj
    KUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUrlx4
    gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-am7tXg26qJo21gY
    jLfhn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5
    XmnwZMyNc9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgM17o03phcTMx
    t1MizR88NKU1WkBsIXMCjy1Noue7MD-ShDp5dmM",
  "tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```

Figure 238: Flattened JWE JSON Serialization

## 7. Security Considerations

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

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

- [JWA] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<http://www.rfc-editor.org/info/rfc7518>>.
- [JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.
- [JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.
- [JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<http://www.rfc-editor.org/info/rfc7515>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.

## 8.2. Informative References

- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.
- [LOTR-FELLOWSHIP] Tolkien, J., "The Fellowship of the Ring", HarperCollins Publishers, ePub Edition, ISBN 9780061952838, March 2009.
- [RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <<http://www.rfc-editor.org/info/rfc1951>>.
- [RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <<http://www.rfc-editor.org/info/rfc7095>>.

## 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](mailto:mamille2@cisco.com)



