Examples of Protecting Content using JavaScript Object Signing and Encryption (JOSE)
draft-ietf-jose-cookbook-02

Abstract

A set of examples of using JavaScript Object Signing and Encryption (JOSE) to protect data. This document illustrates a representative sampling of various JSON Web Signature (JWS) and JSON Web Encryption (JWE) results given similar inputs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 23, 2014.

Copyright Notice

Copyright (c) 2014 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 ........................................... 4
   1.1. Conventions Used in this Document ................... 5
2. Terminology .............................................. 5
3. JSON Web Signature Examples .............................. 5
   3.1. RSA v1.5 Signature .................................. 6
      3.1.1. Input Factors .................................. 6
      3.1.2. Signing Operation ................................. 7
      3.1.3. Output Results .................................. 8
   3.2. RSA-PSS Signature .................................... 9
      3.2.1. Input Factors .................................. 10
      3.2.2. Signing Operation ................................. 10
      3.2.3. Output Results .................................. 10
   3.3. ECDSA Signature ..................................... 12
      3.3.1. Input Factors .................................. 12
      3.3.2. Signing Operation ................................. 12
      3.3.3. Output Results .................................. 13
   3.4. HMAC-SHA2 Integrity Protection ......................... 14
      3.4.1. Input Factors .................................. 14
      3.4.2. Signing Operation ................................. 15
      3.4.3. Output Results .................................. 15
   3.5. Detached Signature .................................. 16
      3.5.1. Input Factors .................................. 16
      3.5.2. Signing Operation ................................. 17
      3.5.3. Output Results .................................. 17
   3.6. Protecting Specific Header Fields ...................... 18
      3.6.1. Input Factors .................................. 18
      3.6.2. Signing Operation ................................. 18
      3.6.3. Output Results .................................. 19
   3.7. Protecting Content Only ............................... 20
      3.7.1. Input Factors .................................. 20
      3.7.2. Signing Operation ................................. 20
      3.7.3. Output Results .................................. 21
   3.8. Multiple Signatures .................................. 22
      3.8.1. Input Factors .................................. 22
      3.8.2. First Signing Operation ........................... 22
      3.8.3. Second Signing Operation ......................... 23
      3.8.4. Third Signing Operation ........................... 24
      3.8.5. Output Results .................................. 25
4. JSON Web Encryption Examples .............................. 26
   4.1. Key Encryption using RSA v1.5 and AES-HMAC-SHA2 ........ 27
      4.1.1. Input Factors .................................. 27
      4.1.2. Generated Factors ................................. 29
      4.1.3. Encrypting the Key ................................ 29
      4.1.4. Encrypting the Content ......................... 29
      4.1.5. Output Results ................................. 30
   4.2. Key Encryption using RSA-OAEP with A256GCM ........... 32
1. Introduction

The JavaScript Object Signing and Encryption (JOSE) technologies – JSON Web Key (JWK) [I-D.ietf-jose-json-web-key], JSON Web Signature (JWS) [I-D.ietf-jose-json-web-signature], JSON Web Encryption (JWE) [I-D.ietf-jose-json-web-encryption], and JSON Web Algorithms (JWA) [I-D.ietf-jose-json-web-algorithms] – collectively can be used to
protect content in a myriad of ways. The full set of permutations is extremely large, and might be daunting to some.

This document provides a number of examples of signing or encrypting content using JOSE. While not exhaustive, it does compile together a representative sample 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.

1.1. Conventions Used in this Document

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

Wherever possible, the examples include both the Compact and JSON serializations.

All of the examples in this document have whitespace added to improve formatting and readability. Except for plaintext or payload content, whitespace is not part of the cryptographic operations. Plaintext or payload content does include whitespace (unless otherwise noted), with line breaks (U+000A LINE FEED) added to improve readability.

2. Terminology

This document inherits terminology regarding JSON Web Key (JWK) technology from [I-D.ietf-jose-json-web-key], terminology regarding JSON Web Signature (JWS) technology from [I-D.ietf-jose-json-web-signature], terminology regarding JSON Web Encryption (JWE) technology from [I-D.ietf-jose-json-web-encryption], and terminology regarding algorithms from [I-D.ietf-jose-json-web-algorithms].

3. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

All of the succeeding examples use the following payload plaintext, serialized as UTF-8; the sequence "\xe2\x80\x99" substituted for (U+2019 RIGHT SINGLE QUOTATION MARK) and line breaks (U+000A LINE FEED) replacing some " " (U+0020 SPACE) to improve readability:
It’s a dangerous business, Frodo, going out your door. You step onto the road, and if you don’t keep your feet, there’s no knowing where you might be swept off to.

Figure 1: Payload content plaintext

The Payload - with the sequence "\xe2\x80\x99" replaced with (U+2019 RIGHT SINGLE QUOTATION MARK) and line breaks (U+000A LINE FEED) replaced with " " (U+0020 SPACE) - encoded as UTF-8 then as [RFC4648] base64url:

```
SXTigJ1zIGFzZGluZ2VuZmVjdCB0byB0aGUgZm9yZSB0aGUgZGV2ZSB0aGUgZGVuZ2VjIHkeyZSBB
```

Figure 2: Payload content, base64url-encoded

3.1. RSA v1.5 Signature

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

3.1.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- RSA private key; this example uses the key from Figure 3.
- "alg" parameter of "RS256".
"kty": "RSA",
"kid": "bilbo.baggins@hobbiton.example",
"use": "sig",
"n": "n4EPtACcc9AikeQHPrzHStgAbgs7bTZLwUBZdr8_KuKPEHld4rHVTet
-o-XV2JrojnhxJWJtVnd7nqQDVEIzQHzzA_jmScpMajMR3B5FKrKb2wqV
wGU_NaYqYL-QtiWn21bzcE6XC0dQapryDqlqHqKHj93RBordaZ6Aj
-oBHQhEHHy7pTe7Tpe-OvFVhd16c6S6M1ZcD1NNLYD51FhpP19bTj1sde
3uhGgcO0CuEHg81hwzOHrtIq8s0FVb99k3-tvTU4fg_3L_vnIuFAKwU
CQknS2BYwdq mzSnbLY7h_qixoR7jig3_kRhuaxwUkrz5ia1Qkqgc5g
HdRNP5zw",
"e": "AQAB",
"d": "bWUC9b-EFRIo8kpGfh02uyGPvMNkY+WnT_b_ikih9k20eT-01q_I78e
iZkpXxQ0UTEsZ0sNRS-0uJbVQ-AilrkwMsmK1J3XTGdzhThkq9gRld
Y7esNA_AKZGh-Q66i_42r_INLRCe8W-n23ui_qofkLn9QWDDqpaIa-a
MwWw5Df2UMBYwkHTMEzLYGqOe04nqeq1hExBTH0BDkXmYuFhUq1BU
61-DgEiwXq82sXt2h-LMnT3046AOYJorioz75tSUQFGCshW7BnPS0u
j18kKhyv071hsJdRpdM5Ply12lhsFf4L_mHCuOau7gdsPfHPxxjVOC
OpBrQzwQ",
"p": "3Slxg_DwTXJcb6095RoXygQC4Za5RnAvz1no1yhHnUex_fp7AZ_9nR
aO7Hx-_SFQGQeutoa2TDjDAW4vUpk8rw9J90AzZ0N2fuu1Amr WCsmG
peNqQnev1T7IyEnsh8UMt-n5cafhkikzhEsrmdnHLxOrvRJ1spP62v8
bUq0k",
"q": "nKE2zh-cTf6ERFk4k4e_jy78GfPYUIaUyoSSJuBzp3Cubk3OCqs6grT
8br_cuUDmIMWmntmqDyI55rUeq3MP15vMQON81hTeU21mKvw7an
V5UzhMliz7z4yNkUuFv1WfoBvyvY89EexvR-hdqRxH1sqaZ192B3pF0
J7jFc",
"dp": "B8PvVXkvrj2L-GlQ7v3y9r6Kw5g9SahXBwzWUpz19T7V9g-I-YV85q
1NIb1rXQD-IsXXXX-RPmR5OB0iMGQp8pbt26l1jYFKU_E9Xn-
RULHz0-9d9E9gZLKD4VngpZ-PF0_q29p5xWJoP0p9Qf1HvChrixR
59ehiki",
"dq": "CLDXMDGduy1c9797r84rEUVn7p5qF0F38Y-iBZX5N7-TopOZKf1pEr
AMVeKxFL141HqGgB5LMQWOSfBwtxYWZDM6si6c5i7bwQGIC3gnJk
bi_7k_vjGw6HxqfPaw2PnvP-zypEhDercy4c_ZL1Cq9AgC2yeL6ksK
TlcYF8",
"q1": "3PiqvXQ0zMeE-sBvZgI289X9CQF3VWqPzNKnIQp7_Tug06-N
ZBKQcaMf3ahaEGbjTVj_sjKC-KRB-XrvuKe-72MaQj8VфDvksbbonDKDh
j1-GtisaeDVW7tdH0cfwxgFUHqPq7FoCrjF6h62EpMF6xmujs4qMjr
z8aaI4"}

Figure 3: RSA 2048-bit Private Key, in JWK format

3.1.2. Signing Operation

The following are generated to complete the signing operation:

Miller          Expires October 23, 2014

o Protected JWS Header; this example uses the header from Figure 4, encoded using [RFC4648] base64url to produce Figure 5.

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

Figure 4: Protected JWS Header JSON

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSIsIiI
```

Figure 5: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 5) and Payload content (Figure 2) produces the following signature:

```
MRjdkly7-_-oTPTS3AXP4liQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtaineFmKZopdHM1O2U4mwzJdQx996iwp83xug1I17PNDi184wnB-BDkoBwa78185hX-Es4J
IwmDLJK31fWRa-XtL0RnltuYv746iyYTh_gHRD68BNt1uSNCrUCTJDt5aAE6x8wW1Kt9eRo4QPocsadnHXFxnt8Is9UzpERV0ePPQdLuW3IS_de3xyIrDaLGdjuP
xUAhb6L2aXic1Ui2podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6Q2WJushZ41Axf_f
cIe8u9ipH84ogoree?vjbU5y18kDquDg
```

Figure 6: Signature, base64url-encoded

### 3.1.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 4)
- Payload content (Figure 2)
- Signature (Figure 6)

The resulting JWS object using the Compact serialization:
The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vy+b3VzIGJ1c2luZXNzLCBgc3Rlc3Q9kbywgZ29pbmcgb3V0IgJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9J
  
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9J",
      "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5C5Z5Ni1ktaIoFmKZopdHMI02U4mz3DQx996iup83xugI17PNDi18wB-BDkoBwA78185hX-Es4J
        
        "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5C5Z5Ni1ktaIoFmKZopdHMI02U4mz3DQx996iup83xugI17PNDi18wB-BDkoBwA78185hX-Es4J
        
        "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5C5Z5Ni1ktaIoFmKZopdHMI02U4mz3DQx996iup83xugI17PNDi18wB-BDkoBwA78185hX-Es4J
        
        "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5C5Z5Ni1ktaIoFmKZopdHMI02U4mz3DQx996iup83xugI17PNDi18wB-BDkoBwA78185hX-Es4J
```

Figure 7: Compact Serialization

3.2. RSA-PSS Signature

This example illustrates signing content using the "PS256" (RSASSA-PSS with SHA-256) algorithm.

Miller Expires October 23, 2014 [Page 9]
3.2.1.  Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.

- RSA private key; this example uses the key from Figure 3.

- "alg" parameter of "PS384".

3.2.2.  Signing Operation

The following are generated to complete the signing operation:

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

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

Figure 9: Protected JWS Header JSON

```
eyJhbGciOiJQUzI1NiIsImtpZCI6ImltZyI7ImFjY291bnRzIjpbIl9AMqIyMkIiwiZXh0dXJlX2lkIjpbIl9AMjMwIiwiaGVzdCI6IjIiLCJwcm9ob3NpZ24iOjF9LCJpZCI6IiJ9
```

Figure 10: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 10) and Payload content (Figure 2) produces the following signature:

```
cu22eBqkYDKgI1IpzDXGvaFFz6WGoz7fUDcfT0kk0y42miAh2qyBzk1xEsnk2I
```

Figure 11: Signature, base64url-encoded

3.2.3.  Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 10)
- Payload content (Figure 2)
- Signature (Figure 11)

The resulting JWS object using the Compact serialization:

```
evJhbGciOiJQUzIzQ4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9j.
  SXTigJlzIGEgZGFuZ2Vyb3VzIGJlci2luZWNzLCBgc3N0eWxlci92MS4gZ2Fj
vdXl0eWxlci92MS4gZ2Fj
  cu22eBkgyDKgI14zDxGvaFfz6WGoz7fUDcft0kkOy42miAh2qyBzklxEnsk21
pN6-tPld6Vrk1HkqsGqDqHcDPo8TtB5ddIt11Vo6_1OLPcµ8huiUSmxbXU
  eyJhbGciOiJQUzIzQ4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9j.
  SXTigJlzIGEgZGFuZ2Vyb3VzIGJlci2luZWNzLCBgc3N0eWxlci92MS4gZ2Fj
vdXl0eWxlci92MS4gZ2Fj
  cu22eBkgyDKgI14zDxGvaFfz6WGoz7fUDcft0kkOy42miAh2qyBzklxEnsk21
pN6-tPld6Vrk1HkqsGqDqHcDPo8TtB5ddIt11Vo6_1OLPcµ8huiUSmxbXU
```

Figure 12: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJlci2luZWNzLCBgc3N0eWxlci92MS4gZ2Fj
    cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcft0kkOy42miAh2qyBzklxEnsk21
    eP_ijo7p8VdzOTTrxeUT3lm8d9shnr2lfJT8ImUjvAA2Xez2M1p8cBE5awDzT
    0q10n6uiPlaCN_2_jLaEnQTlqRhtfa64QQSUmFAajVPKbByi7xho0uOTc8bH510a
    6GYmJUAmWjzwZ60D4ifK08DYM-X72Eaw"
  "signatures": [
    {  "protected": "eyJhbGciOiJQUzIzQ4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9j",
      "signature": "cu22eBqkYDKgI14zDxGvaFfz6WGoz7fUDcft0kkOy42miAh2qyBzklxEnsk21
               eyJhbGciOiJQUzIzQ4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZS9j"
    }
  ]
}
```

Figure 13: JSON Serialization
3.3. ECDSA Signature

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

3.3.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- EC private key on the curve P-521; this example uses the key from Figure 14.
- "alg" parameter of "ES512"

```
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLL0sC0zz5cY97ewNUajB957y-C-U88c3v13nmG2x6sYl_oJXu9A5RkTKqjqvjyeKF-7ytDyRXygCF5cj0Kt",
  "y": "AdymlHvOiLxXkEhayXQmNXcVDX4h9htZaCJN34kfmC6pV5obQHiraVySsUdaQkAgDPrwQuJmbnX9cwrGFp-HqHZR1",
  "d": "AAhRON2r9cqxXX1hg-RoI6R1tX5p2rUAYdmpHZoClXNM56KtscrX6zbKipQrcW9CGZH3T4ubpnoTKLDYJ_fF3_rJt"
}
```

Figure 14: Elliptic Curve P-521 Private Key

3.3.2. Signing Operation

The following are generated before beginning the signature process:

- Protected JWS Header; this example uses the header from Figure 15, encoded using [RFC4648] base64url to produce Figure 16.

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

Figure 15: Protected JWS Header JSON
Performing the signature operation over the combined protected JWS header (Figure 16) and Payload content (Figure 2) produces the following signature:

AE_R_YZCChjn4791jSQCrP2CNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvbu9P1on7KRTzoNEuT4Va2cmL1eJAQy3mtPBu_u_sDDyYjnAMDxXPn7XrT01w-kvAD890j18e2puQens_IEKBpHAB1sbEFPX6sFY80cGDqoRuBomu9xQ2

Figure 17: Signature, base64url-encoded

3.3.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 16)
- Payload content (Figure 2)
- Signature (Figure 17)

The resulting JWS object using the Compact serialization:

Figure 18: Compact Serialization

The resulting JWS object using the JSON serialization:
3.4. HMAC-SHA2 Integrity Protection

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

3.4.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- HMAC symmetric key; this example uses the key from Figure 20.
- "alg" parameter of "HS256".

{ "kty": "oct", "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037", "use": "sig", "k": "hJtXIZ2uSN5kbQfbtTNWbpmhkV8FJG-Onbc6mxCcYg" }
3.4.2. Signing Operation

The following are generated before completing the signing operation:

- Protected JWS Header; this example uses the header from Figure 21, encoded using [RFC4648] base64url to produce Figure 22.

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

Figure 21: Protected JWS Header JSON

eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LTEyMDAzOSIsImh0dHA6Ly9ucHVsdml6cy5jb20iO

Figure 22: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 22) and Payload content (Figure 2) produces the following signature:

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 23: Signature, base64url-encoded

3.4.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 22)
- Payload content (Figure 2)
- Signature (Figure 23)

The resulting JWS object using the Compact serialization:
Figure 24: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{  
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgdGh1cmXiJlzIG5vIgtub3dpbmcmd2h1cmUgeW91IGp2h0IGJ1IHNN3XB0IG9mZiB0by4",
  "signatures": [
    {  
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWJtNDcxYiIiZmQ2IWMVZjJmNzJy99",
      "signature": "s0h6KThzkbBBkLspWh84VsJZFTsPPqMDA7g1Md7p"
    }
  ]
}
```

Figure 25: JSON Serialization

3.5. Detached Signature

This example illustrates a detached signature. This example is identical others, except the resulting JWS objects do not include the Payload content. Instead, the application is expected to locate it elsewhere. For example, the signature might be in a meta-data section, with the payload being the content.

3.5.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
3.5.2. Signing Operation

The following are generated before completing the signing operation:

- **Protected JWS Header**: this example uses the header from Figure 26, encoded using [RFC4648] base64url to produce Figure 27.

The protected JWS header parameters:

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

Figure 26: Protected JWS Header JSON

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWUlTRkOWItNDcxYiliZmQ2LW
V1ZjMxNGJjNzAzNyJ9
```

Figure 27: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 27) and Payload content (Figure 2) produces the following signature:

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 28: Signature, base64url-encoded

3.5.3. Output Results

The following compose the resulting JWS object:

- **Protected JWS header** (Figure 27)
- **Signature** (Figure 28)

The resulting JWS object using the Compact serialization:
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTlTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9.

s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0

Figure 29: JSON Serialization

The resulting JWS object using the JSON serialization:

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

Figure 30: JSON Serialization

3.6. 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 JSON serialization is possible.

3.6.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- Signing key; this example uses the AES symmetric key from Figure 20.
- Signing algorithm; this example uses "HS256".

3.6.2. Signing Operation

The following are generated before completing the signing operation:

- Protected JWS Header; this example uses the header from Figure 31, encoded using [RFC4648] base64url to produce Figure 32.
0 Unprotected JWS Header; this example uses the header from Figure 33.

The protected JWS header parameters:

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

Figure 31: Protected JWS Header JSON

eyJhbGciOiJIUzI1NiJ9

Figure 32: Protected JWS Header, base64url-encoded

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

Figure 33: Unprotected JWS Header JSON

Performing the signature operation over the combined protected JWS header (Figure 32) and Payload content (Figure 2) produces the following signature:

bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMW50r20

Figure 34: Signature, base64url-encoded

3.6.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 32)
- Unprotected JWS header (Figure 33)
- Payload content (Figure 2)
- Signature (Figure 34)

The resulting JWS object using the JSON serialization:
3.7. 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 JSON serialization is possible.

3.7.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.

- Signing key; this example uses the AES key from Figure 20.

- Signing algorithm; this example uses "HS256"

3.7.2. Signing Operation

The following are generated before completing the signing operation:

- Unprotected JWS Header; this example uses the header from Figure 36.

Figure 35: JSON Serialization
Performing the signature operation over the combined empty string (as there is no protected JWS header) and Payload content (Figure 2) produces the following signature:

```
xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvJxZuk
```

**Figure 37: Signature, base64url-encoded**

### 3.7.3. Output Results

The following compose the resulting JWS object:

- Unprotected JWS header (Figure 36)
- Payload content (Figure 2)
- Signature (Figure 37)

The resulting JWS object using the JSON serialization:

```json
{
   "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbm cmpb3VzIG9vci4gWW91IH1nnZXAgb250byB0aGUgc2NoZW4gd29ybSBob21biB0aGUgYW5kIGF!</n```

```json
}]
```

**JSON Serialization**
3.8. Multiple Signatures

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

3.8.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- Signing keys; this example uses the following:
  - RSA private key from Figure 3 for the first signature
  - EC private key from Figure 14 for the second signature
  - AES symmetric key from Figure 20 for the third signature
- Signing algorithms; this example uses the following:
  - "RS256" for the first signature
  - "ES512" for the second signature
  - "HS256" for the third signature

3.8.2. First Signing Operation

The following are generated before completing the first signing operation:

- Protected JWS Header; this example uses the header from Figure 38, encoded using [RFC4648] base64url to produce Figure 39.
- Unprotected JWS Header; this example uses the header from Figure 40.

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

Figure 38: Signature #1 Protected JWS Header JSON
Performing the first signature operation over the combined protected JWS header (Figure 39) and the Payload content (Figure 2) produces the following signature:

MIsjqtVlopa71KE-Mss8_Nq2Y4FGhiocsqrqgi5NvyG53uomic1tcMdSg-qqrtz27CG6Svw2Y13TDiqHzTurL_lr2zFcryNFiHkSw129EghGpwpkpxaTn_THJTCglNBAko1MZBCdwzJxwq2c-1RLpO2HiUYyXSwO97Bse0_evZKdjvvKSgsIqjytKSeAMbMBdMma622_BG5t4sdvuCTHpj9iJmkio47AIwqkZV1aIZsv33uPUqB0BCXbYojQj4t7mxPftHnN1GoOSMxr_3thmXTCm4US-xiN0yhbm8afKK64jU6_TptQHJeQx9G3T-Q-083B745_AfYOInC9w

Figure 41: Signature #1, base64url-encoded

The following is the assembled first signature serialized as JSON:

```
{  
  "protected": "eyJhbGciOiJSUzI1NiJ9",
  "header": {
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "MIsjqtVlOpa71KE-Mss8_Nq2Y4Fghiocsqrqgi5NvyG53uomic1tcMdSg-qqrtz27CG6Svw2Y13TDiqHzTurL_lr2zFcryNFiHkSw129EghGpwpkpxaTn_THJTCglNBAko1MZBCdwzJxwq2c-1RLpO2HiUYyXSwO97Bse0_evZKdjvvKSgsIqjytKSeAMbMBdMma622_BG5t4sdvuCTHpj9iJmkio47AIwqkZV1aIZsv33uPUqB0BCXbYojQj4t7mxPftHnN1GoOSMxr_3thmXTCm4US-xiN0yhbm8afKK64jU6_TptQHJeQx9G3T-Q-083B745_AfYOInC9w"
}
```

Figure 42: Signature #1 JSON

3.8.3. Second Signing Operation

The following are generated before completing the second signing operation:

- Unprotected JWS Header; this example uses the header from Figure 43.
Performing the second signature operation over the combined empty string (as there is no protected JWS header) and Payload content (Figure 2) produces the following signature:

ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwD01muhcdCoFZFfjISu0Cdkn9Ybd1m15h0o0x924DUz8sK7Zxkhc7AFM8ObLfTvNCrC13Jk12U5IX3utNh0DH6v7xyg1Qahsn0fryb4ZSAkje8bAWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD

Figure 44: Signature #2, base64url-encoded

The following is the assembled second signature serialized as JSON:

```json
{
  "header": {
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwD01muhcdCoFZFfjISu0Cdkn9Ybd1m15h0o0x924DUz8sK7Zxkhc7AFM8ObLfTvNCrC13Jk12U5IX3utNh0DH6v7xyg1Qahsn0fryb4ZSAkje8bAWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD"
}
```

Figure 45: Signature #2 JSON

### 3.8.4. Third Signing Operation

The following are generated before completing the third signing operation:

- Protected JWS Header; this example uses the header from Figure 46, encoded using [RFC4648] base64url to produce Figure 47.

```json
{
  "alg": "HS256",
  "kid": "018c0ae5-4d9b-471b-bfd6-eeef314bc7037"
}
```

Figure 46: Signature #3 Protected JWS Header JSON
Performing the third signature operation over the combined protected JWS header (Figure 47) and Payload content (Figure 2) produces the following signature:

\[ s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0 \]

Figure 48: Signature #3, base64url-encoded

The following is the assembled third signature serialized as JSON:

\[
\{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
\}
\]

Figure 49: Signature #3 JSON

### 3.8.5. Output Results

The following compose the resulting JWS object:

- Payload content (Figure 2)
- Signature #1 JSON (Figure 42)
- Signature #2 JSON (Figure 45)
- Signature #3 JSON (Figure 49)

The resulting JWS object using the JSON serialization:
4. JSON Web Encryption Examples

The following sections demonstrate how to generate various JWE objects.

All of the succeeding examples (unless otherwise noted) use the following plaintext content, serialized as UTF-8, with the sequence

Figure 50: JSON Serialization
You can trust us to stick with you through thick and thin— to the bitter end. And you can trust us to keep any secret of yours— closer 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 51: Plaintext content

4.1. 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.

4.1.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- RSA public key; this example uses the key from Figure 52.
- "alg" parameter of "RSA1_5".
- "enc" parameter of "A128CBC-HS256".


```json
{
    "kty": "RSA",
    "kid": "frodo.baggins@hobbiton.example",
    "use": "enc",
    "n": "maxhbsmBtdQ3CNrKVprUE6n91YcregDMLYNeTAWCj88NnP9XlYegTHVHqjxKDSHP2l+F5jS7sppGlgdAqZyhnWvXhYNvcM77fgKqnx_xAhx6f3yy7s-M9PSNCwF2c1h6AKR4t00Evh91rypM9p14tB0y9t5FS9j5W5NwahAllhrd-osQGPjIeIldeHTwx-ZTHu3C6OPu_-_JL16hK9wbsbUmA4cR58d2pgbaY7ASgsjCBUtYJaNIHSoHXprUdZKUMAZVW0KQFPa60Pi4oypBadvjvM242JxAsSy0e2haueTXvZB4eZOAJyih2e_VOIKVmsnDrJYA
VotGlVMQ",
    "e": "AQAB",
    "d": "Kn9tgoHfiTVi8Du5b9TnwyHwG5kd6RE0uFd1pCGnJN7Z41963R7wybQ1PLAHmpIBNTztrheoAniRV1NCiXawqS46lXiDTp4ntEPngcKsyO5jMAji7-CL8vhpYVowNFVesgMoVaPRMYT97W63hnNM0aWS7USZ_hLg60e1mYyHT3FucjSM86Nf6oIEnt43r2fspgEFGPrdE6fplC99a9q-geP1GFuLimRdndm-P8q8kvN3KH1NATegQAg7tgqZ80S-3VD0GfwgdnBNpiuPUx8OpI9KDIfu_acc6fg14nsAnaJqXe6RESvghP2afFqSy_Fd2v
pzj8bQQ",
    "p": "2DwQmZ4FoTnQ8IkJ3BmKRF5eh2miz5AxeJZ2nUE3sdTYKSLtaEOekXV9vbZwWvHdNM6UnKCJ_2IN80yLYH0L_G21aXf9-unyEnUSu7HHTK1LpYaO0x12gVljoAdWN3hIFrj2LZGS71OH-a3Q1IDDQ0JQ2VFMU",
    "q": "te8LY4-W7IyaqH1ExujMgkTaTErBvVFQnFLY2nxIrNdwQ93_VF039APlE5eLja2n6-6k1e-qT7mCFozKFVTUyZe5RCr_JYX2KfexJINJ9lhZHMv5pskZpe2S-HPHC6g7RLko1q-ipdn_qxysufW7WAX15SvFQx8d6Et0",
    "dp": "UfYkCfL_or492vVe0CpozLsp1bg4L3-25L48m1swbpzoyIgd2x2HTHQmJpFAI8q-zf9RmgJXkDrs9kdxPtAsLlWYdeCT5c125Fkdq317JVRD01nX7x2Rdhe8ERCww84xzItuTI_KixXZNU5lMqJwbIw2eTx1lpsfo0ryYU",
    "dq": "IeGqc-OJfepehH8Wfd7mUFyrXdnokXJBCogChY6YKuIHGC_p8Le9Mb
pFkEsEalLN1Hefb36ogB15iz_ayUI2j2IoQZ82zn0Urpa9fVYNot87ACf6iG7q9MV7rlFiAderZi03tkVXAdaBau_9vs5rS-7HMtxkVrxSUvYJY14
TkK1Hc",
    "qi": "Kc-1zZQQaFz2Cr5Ol0tOVtvREKoVqaAYhQiqIRGL-MzS4sCMRkxm5Zv1Xyx6B6Et1n_Aagjqjai2kjeGlXTTTHd8Iga6f0GMBaAr5uRl1hQqPSc7Gl7CF1D2kBMTQn6EshYzZfxW08mIO86Rzuh0beL6fG9mkDcIyPrBxX
2bQ_nM"
}
```

Figure 52: 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.)
4.1.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 53
- Initialization vector/nonce; this example uses the initialization vector from Figure 54

3qyTVhIWt5juq2UCpfRgpvauwB956MEJL2Rt-8qXKSo

Figure 53: Content Encryption Key, base64url-encoded

bbd5sTkYwhAIqfHsx8DayA

Figure 54: Initialization Vector, base64url-encoded

4.1.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 53) with the RSA key (Figure 52) results in the following encrypted key:

laLxI0j-nLH-_BgLOXMozKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePFvG2K-pvS1Wc9BRIazDzn50RcRai__3TDON395H3c62tIouJ4XaRvYHFjZTZ2G Xfz8YAImcc91Tfk0WXC2F5Xbb7lC1Q1DDH151t1pH77f2ff7xiSxz9oSeWYrcG TSLUeeCt36r1Kt30Sj7EyBQX0z1N7ixbyhMAfge1e7Mv1r0I0I1518NqeXXW8V1 znmoxaGMny3YnGr15Wf6Qt2nBq4DaPdnaAuJuGUGEeceleI01wx1BpyIfgvfjOh MBs9M8XL223Fg47x1GsMXdfuY-4jaqVw

Figure 55: Encrypted Key, base64url-encoded

4.1.4. Encrypting the Content

The following are generated before encrypting the plaintext:

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

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

Figure 56: Protected JWE Header JSON
Performing the content encryption operation on the Plaintext (Figure 51) using the following:

- CEK (Figure 53);
- Initialization vector/nonce (Figure 54); and
- Protected JWE header (Figure 56) as authenticated data produces the following:

- Ciphertext from Figure 58.
- Authentication tag from Figure 59.

The following compose the resulting JWE object:

- Protected JWE header (Figure 57).
- Encrypted Key (Figure 55).
- Initialization vector/nonce (Figure 54).
- Ciphertext (Figure 58).
- Authentication Tag (Figure 59).
The resulting JWE object using the Compact serialization:

eyJhbGciOiJSU0ExMi4iQ3V3d3QuY29tIjEyI24iLCJraWQiOiJyZXR0aWZzVGhlIiwiVHlwZyI6IkJLeVNvbnRlbnQiLCJ4cCI6IkJyZXR0aWZzVGhlIigiLCJjYXNvbWQiOiJ1c2VyIiwiYWJvdXQiOiIwIiwiVHlwZSI6IkJyZXR0aWZzVGhlIiwiZGVmYXVsdF9zZXJ2aWNlIjoiMSIsImlhdCI6MTI5MTQ5ODEzOSwiZGVmYXVsdF90b2tlbiI6IjEifQ.

Figure 60: Compact Serialization

The resulting JWE object using the JSON serialization:
"recipients": [
    {
        "encrypted_key": "laLxI0j-nLH-_BgLOXozKxmy9gffy2gTdqvzfTihJBUuzqg0V7yk1WC1nQePFvG2K-pvS1Wc9BR1azDrn50RcRai__3TDON395H3c62t1ouJ4XaRvYHFjZTZ2Gxfz8YAImcc91Tfk0WXC2F5Xbb71C1q1DDH151t1pH77f2ff7xiSxh9oSeWYrcGTSLUeeCt3r1Kt3OSj7EybQox21N7IxbyhMAfge77mv1rOTI5I8QeqeXXW8V1zNmoxaGMny3YNgir5Wf6Qt2nBq4qDaPdnaAuUGEe1101wx1BpyIfgvfjOhMBs9M8XL223Fg47xlGsmXMdfuY-4jaqVw"
    }
],
"protected": "eyJhbGciOiJSU0ExUzIiLCJraWiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uUmV4YW1wbGU1LCJ1bmMiOiJJBMTI4Q0JDLUhTMju2In0",
"iv": "bbd5sTkYwhAIqfHSx8DayA",
"ciphertext": "0fys_TY_na7f8dwSxfXliYdHaA2DxUjD67ieF7feVbIR62JhJvGZ4_FNVSiGc_raa0HnLQ6sP2sv3Xz1lpl1_o5wR_RsSr5S8z-wn13Jvo0mkpEEn1DmZvDu_k80WzJv7eZVEq1WKdyVzFpIyQU28GLopRc2VbVbK4dQKpdNjPPEmRqcaGreTV2VyeSUvf559yJZxRuSwWFf6KrNtmRdZ8R4mDOjHsrM_8suw1Fcq4r5GX8TKa10zT5CbL5QLw3Rc7u_hg0yKV0iRytEAes3v2kclfIpk6nbXdc_PkmDNS-ohP78T206_7uInMGheFeX4cW7G7Ve1HGeT93JfWDEEQi5_V9UN1rhXNryYo-0fVMkZAKX3VW17zA6BP430m",
"tag": "kvKuFBXHe5mQr41qqgobAUG"
}

Figure 61: JSON Serialization

4.2. Key Encryption using RSA-OAEP with A256GCM

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

4.2.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the plaintext from Figure 51.
- RSA public key; this example uses the key from Figure 62.
- "alg" parameter of "RSA-OAEP"
- "enc" parameter of "A256GCM"
"kty": "RSA",
"kid": "samwise.gamgee@hobbiton.example",
"use": "enc",
"n": "wbdxtI55VaanZXPY29Lg5hdmv2XhqvAhoxUkanzfz2-5zVUx6aprHRrI4pP1AhqJRL2fytWwD5mnHHRG2paH1l0ySj9W10BioZ2Bl1XP2eC-FyXJCtY0yHdKQWlrfHm42EW7Wv04r4gfa06uxjLGwpfGr2La6hociWCnkNrg71IS2zQz5oBIPFjXkfnYt2t1_WwGnL22Gp1yXj5Y1LdxXp3Xe7t
sq571tuNfoUT84Egqzj5U1D1toVkpGpsW1llnw9nJiwa7A7sXXRITCB1v04M
5qnt2dw-7v4wUr4779ubDuJnsa1nMvS66-ROPcnAZwSkxtBnFJDG1U
T7ezizj1vmc0XW_pubP0U1Wn0ec85FCfht1AcpeW8schrOBEnqBOD
F SDKyUpcULc5jA2TaPfW2dA67qfTTsc_Fupf2QkBGrEc1lgpxrKcHvWYQb
68B-HoHzJhcQtqatuBzFV5n5tbUW-UpkcvJFncFLLh3h8mb-H_ox35fjqB
SAJkLyqeqKtPfYjxhd09nxwgfJ6vKq6UC418_T01jMvFDTWUXUln1nhf
OnzW6hHSzslc9WrcVUz3UMv5sizqD9wfc9Wf3g5qFdx0QKls99qcDa
iCAwM3YEBizuNeeCa5d4tHdBi1EXEhcHSeYbhhMbMfGfvasVKn0aZRsnT
ycXohwB1s12E",
"e": "AQAB",
"alg": "RSA-OAEP",
"d": "n7fzJc3tQ59VEOBTKayzumuSMM7800JQuZjN_KbHB010ZG25zoA7T4Bx
cc0xQnS50zE5uSCiw9g10Ct0JvxPcpmqzaJ2glnirjczW-oBtvKq7CGaw
-B3ghfFP12lkbksorz3Haijcy33HBsysy4_WerrXg4MDNE4H0yjy68TcxW
ElYqRXUCf5r7jxWmo1elixSGCvqNDRutxECwiewfmmfrrveEogLx9E
A-KMGajTiISXqXgQWUX1G7v_mVr2H2uYmYcNcHkRvup9E70ok087
Dhx0Bv4U0ZLWZ010Hxmkcgw58A_Y21LYbVx1_s5lPsqebbH-nqIj
h1L0QndFhihLxncjWtpqCztLmInMzAYeCwAGW7Zf1v-Rn9fL1v9j3Zr7
-MSH9sabquiz1HNZugr7_jjR1u4Mha0184FK61bcqj1JWxPHvzNzo2o7rD
F-1L1qqYU9epF6x3a2S0dkqrBlenuE6EvLyUIpDqj3jIsqoL8mo1L
oomg1jXuLWQGW0Uqu28pglyzm-9QU00nyehFfluhrS88jaQAfWIFhMWH
_IQT917-yirndr_2fWQ_i1UgMsGzA7aoGzZfPljyr6z2-tY_ KubG00-28
S_aWvjyUC-A1p8A0YuYkJb7Z-CWwH32fGW48jt1-zomrwjln_mnshpsBs0c
9WsiwRzi-I8GE",
"p": "7_2v3OQ2zlPFCyfylABQ3XP85Es4hdCwkbDeltauXKgY91etKg
vM4hr0Vbb70VULxFmKcDtpi-3LCYADrrKAK3PtSbtzld_XZ9nlsY
a_C27wpXb_IRtfJiFDkUdm49p4uhGFgFj7nr6NNXpfiHSHWE1zCzC3m
Y46J96J12LrNweVfAcGwNw53p07Db8y9_d2p9a97vCc20dqybh9G6uma-
RFNHo1aoiJhYj69hj3mDRXX-x56H09cnXNbmzsNSCFCKnqm4QG1Mm9j
fBzGRq9bzb4tE4_e2z0Fp8N00v8XLRqQWny8v5c6BrGBj0m78qDQvIGP
qGWcVhQo",
"q": "zqOhk1PwWn_wHuM72F1cX0H6xR0Hg67WuHIKnqkQeeGBA9Ps62Y
KQCO-6mKXtceq8_E0hA2kJMrcR0cVH11hqMCN5xF1m7WPRPZu2qGdc
qssd_umBp-DqYthh_EzwL9KnYoH7JQFXXmcyv5An8coXUtTkw4knKj1YG
RuUwFtusw1LnfjFAXyooI1AQ37ussiCe6C6ZS3mN341UlbJ7TcqwzvVJ
aPJN5cxjySP2D3vp01a9yGa63a1IaKJdIxJSL1m1mFpev5QOBEY9-EX
e2kSwWqOvzt-gamM29Q8veHy4uAqca5d2zMs7hkkHtw1z0-jh9V90epQJ
JlXNNH8Q",
"dp": "190dKbh1AXelMIXQFm2zZTqUhrAzCIr4xNIEGPNoD1j7kxS_FJA-nx
xSKa7-1erDhms_EF67sOnNv5a60JAR7W8LHnDIBnJdaUmm08XaQ
J_ia5mjxNj5862EyD4Us02JmvHzeeNczq25elqTF1hUpG01IZu72F"
4.2.2. Generated Factors

The following are generated before encrypting:

o AES symmetric key as the Content Encryption CEK (CEK); this example uses the key from Figure 63.

o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 64.

mYMfsgggkTAm0Tbvt1Fh2hyoXnbEzJQjMxmgLN3d8xXA

Figure 63: Content Encryption Key, base64url-encoded

-nBoKLH0YkLZPSI9

Figure 64: Initialization Vector, base64url-encoded
4.2.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 63) with the RSA key (Figure 62) produces the following encrypted key:

rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCnUh7lCiud48LxeolRdtFF4nzQi beYO15S_PJJsAXz2SwXtDePz9hk-BbtsTBqC2UsPOdwjc9NhNupNNu9uHIVftDyu cvI6hvA1eZ60GnV4v1zxk2701D89mAzfw_-kT3kuorpDU-CpBENJHX1Q58 -Aad3FzMu03F3n9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8Bpx KdUV9ScfJQTcYm6eJEBe3saSwIaK4T3-dWpuBOhROQXBosJZs1asnuHtVMt2pK IIfux5BC6huIVm7Kv7W7aIUrpym_3H4zYvyMeq5pGqFmW2k8z0p878T51x7 pZfYDSXzS0CfKKkMoZt_q1cWZZsz4duYnt8hS429SglthXn9uDqdlw6ycMqgnQ f0Ts_lycTWMY-agQVSDKhjYNR0f3NiwrRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3 8UjQb0lvXn1SpyvY23WFC7W0JYa7a78DRN6MC6T-xDmMuxCG7S2rscw51QU 06MvTIIFOt0vufUKba03cxA_n1BIHLMjY2kOTxQmmpDTr6Cbo8aKaOnx6AS4E5 Jx9paBpnNMo0OKH35j_Q1rQhDWUN6A2Gg81FayJ69xDeHACGHRnx3woE12ozDR s

Figure 65: Encrypted Key, base64url-encoded

4.2.4. Encrypting the Content

The following are generated before encrypting the plaintext:

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

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

Figure 66: Protected JWE Header JSON

eyJhbGciOiJSU0EiTi0FFUICIsImtpZCI6InNhMdpc2U22FtZ2V1QGhvYmJpdG 9uLmV4YW1wbGUIC1Jl0mMiOjIBMjU2RDNIN0

Figure 67: Protected JWE Header, base64url-encoded

Performing the content encryption operation over the Plaintext (Figure 51) with the following:

- CEK (Figure 63);
- Initialization vector/nonce (Figure 64); and
- Protected JWE Header (Figure 67) as authenticated data
produces the following:

- Ciphertext from Figure 68.
- Authentication tag from Figure 69.

```
0k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXk1SggBdpACm6UJuJowOHCSytjqYgR L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw P7ua1DRfvaOjg2v6Ifa3brcAM64d8p51hhNcizPersuhw5f-pGYzseva-TUaL8 iWnctc-sSwy7SQmRkfhDjwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML 7Cc2GxqvI7zqWo0YIEc7aCf1LG1-8BboVWFdZKLK9vNoycYHumwzKluLWEbSV maPpOsl1Y2n525DxDFWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zV0w
```

Figure 68: Ciphertext, base64url-encoded

```
UCGiqJxhBI3IFVdPalHHvA
```

Figure 69: Authentication Tag, base64url-encoded

### 4.2.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 67)
- Encrypted key (Figure 65)
- Initialization vector/nonce (Figure 64)
- Ciphertext (Figure 68)
- Authentication tag (Figure 69)

The resulting JWE object using the Compact serialization:
Figure 70: Compact Serialization

The resulting JWE object using the JSON serialization:
4.3. 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" (A256-AES-CBC-HMAC-SHA-256) content encryption algorithm.

4.3.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the plaintext from Figure 72 (*NOTE* all whitespace added for readability)
o Password; this example uses the password from Figure 73

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": "GZy6sI26w19NJ0KB-jnmVQ"
        },
        {  
            "kty": "oct",
            "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
            "use": "enc",
            "alg": "A256GCMKW",
            "k": "qC571_u7m7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
        }
    ]
}
```

4.3.2. 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 74.

o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 75.
4.3.3. Encrypting the Key

The following are generated before encrypting the CEK:

- Salt; this example uses the salt from Figure 76.
- Iteration count; this example uses the interaction count 8192.

Performing the key encryption operation over the CEK (Figure 74) with the following:

- Password (Figure 73);
- Salt (Figure 76), encoded as an octet string; and
- Iteration count (8192)

produces the following encrypted key:

```
YKbKLsEoyw_JoNvhtuHo9aaeRNSEhhAW2OVHcuF_HLqS0n6hA_fgCA
```

Figure 77: Encrypted Key, base64url-encoded

4.3.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 78, encoded using [RFC4648] base64url to produce Figure 79.


```
{
  "alg": "PBES2-HS256+A128KW",
  "p2s": "8Q1SzinasR3xchYz6ZzCHA",
  "p2c": 8192,
  "cty": "jwk-set+json",
  "enc": "A128CBC-HS256"
}
```

Figure 78: Protected JWE Header JSON

```
eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4SlciLCJwMnMiOiI4UTFTemluYXNSM3hjaF16NlpaYoBIiwcDjjIjo4MTkyLCJjdHkiOiJqj2stc2V0K2pzb24iLCJ1bmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 79: Protected JWE Header, base64url-encoded

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

- CEK (Figure 74);
- Initialization vector/nonce (Figure 75); and
- Protected JWE header (Figure 79) as authenticated data produces the following:
  - Ciphertext from Figure 80.
  - Authentication tag from Figure 81.

```
23i-Tb1AV4n0WKVSSgcQrdg6GRq5UKxjruHXYSrTHAJLZ2nsnGIX86vMXqI6IRsfywCRFzLxEcZBnTvG3nhzPK0GDD7FMxUHpDjEYCNA_X0mzg8yZ9oyjo61T6s14qQF2EhzgFOCLQ_6h5EVg3vR75_hkBsnuoquM3dewjXBTiOodN84PegMb6asmash_p8sz7H10fc5i9Ix42xqvB1vYldF6exVmL933fOocOJbmK2GBQZL_cEg1lVv2cAODqBqRsaQ7Bq99tT80coH8ItBjgV08AtzXFXsx9qKvC982KLDpQMT1VJKqtV4Ru5LEVPB2XBNz2rTvISogy6Giuwsa-rCrDc_ePOGSuxvgtrokAKYPqmXUEoRdifJfWafkYEkiuDCV9vWGA11DH2xTafhJwcmwYIyzi4Bqrmdn_N-z15tUJYuvHkJxKv6ihbsV_k1hJGPAxJ6wUpmwc4PTQ2izEm0TuSE8oMKdTw8V3kobXZ77ulMwDs4p
```

Figure 80: Ciphertext, base64url-encoded

```
ALTKwvAefel-32NY7eTAQ
```

Figure 81: Authentication Tag, base64url-encoded
4.3.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 79)
- Encrypted key (Figure 77)
- Initialization vector/nonce (Figure 75)
- Ciphertext (Figure 80)
- Authentication tag (Figure 81)

The resulting JWE object using the Compact serialization:

 eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4SlciLCJwMnMiOiI4UTFTemluYXNSM3hjaF16NlpYoHIiwiCJjIjo4MTkyLCJjdHIoiJqd2stc2V0K2pz824iLCJlbmMiOiJlBMTI4Q0JDLUhTMjU2In0.LYKbKLsEoyw_JoNvhuHoa9aeRNSEhhAW2OVHuF_HLqS0n6hA_fgCA

 Figure 82: Compact Serialization

The resulting JWE object using the JSON serialization:

 Figure 82: JSON Serialization

The resulting JWE object using the JSON serialization:
4.4. 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.

4.4.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51
- EC public key; this example uses the public key from Figure 84
- "alg" parameter of "ECDH-ES+A128KW"
- "enc" parameter of "A128GCM"
Figure 84: 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.)

4.4.2. Generated Factors

The following are generated before encrypting:

- Symmetric AES key as the Content Encryption Key (CEK); this example uses the key from Figure 85.
- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 86

Nou2ueKlP70ZXDbq9UrRwg

Figure 85: Content Encryption Key, base64url-encoded

mH-G2zVqgzTu6nW_

Figure 86: Initialization Vector, base64url-encoded

4.4.3. Encrypting the Key

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

- Ephemeral EC private key on the same curve as the EC public key; this example uses the private key from Figure 87.
Performing the key encryption operation over the CEK (Figure 85) with the following:

- The static Elliptic Curve public key (Figure 84); and
- The ephemeral Elliptic Curve private key (Figure 87);

produces the following JWE encrypted key:

0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2

Figure 88: Encrypted Key, base64url-encoded

4.4.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE header; this example uses the header from Figure 89, encoded to [RFC4648] base64url as Figure 90.

```json
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx510xowrd_oYzBmaz-GKFZu4xAFFkbYiWgutEK6iuEDsQ6wNdNg3",
    "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFDr5yPNVtW4Pgw2OyQTA-JdaY8tb7E0",
    "d": "D5H4Y_5PSKZvhfVFbcCYJOtccGZygRgfZkpsBr59Icmme9sW6nkZ8WfwhinUfWJg"
  },
  "enc": "A128GCM"
}
```

Figure 89: Protected JWE Header JSON
Performing the content encryption operation on the Plaintext (Figure 51) using the following:

- CEK (Figure 85);
- Initialization vector/nnonce (Figure 86); and
- Protected JWE header (Figure 90) as authenticated data

produces the following:

- Ciphertext from Figure 91.
- Authentication tag from Figure 92.

The following compose the resulting JWE object:

- Protected JWE header (Figure 90)
- Encrypted key (Figure 88)
- Initialization vector/nnonce (Figure 86)
- Ciphertext (Figure 91)
The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm92Z2guZXhhbXBsZSI6ImVwayI6eyJrdHkiOiJFQyIsImNydiI6IiAt
Mzg0IiwieCI6IiVCBzRrSFBlNTUng1bDB4b3dyZ9vWXPcCWF6LudLRlp1NH
hBRkk2ZrYl1pV2dlEDVLMnl1RURzUTZ3TmROZzMiLCJ5Ijoic3AzdcDTR2haV
MyZmFyW1JLUw51UyTW84S3BvWXJGRH1eVBOVnRXNFbnRXdaT3lRVEEtSmRhWT
h0YjdFMCJ9LCJlbmMiOiJdBWZzR1NiI0
```

Figure 93: Compact Serialization

The resulting JWE object using the JSON serialization:

```
```

Figure 93: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
    }
  ],
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmlVcm91Z2luY3VzLXVzYW5zIiwic3RydWN0IjoiMTQxMCIiLCJleHAiOjE2MDQ1NjYxODAsInN1YiI6IjIzYjViYmM4Y2ZmNjEwNzIiLCJpc3MiOiIyMDEwZDg5MzYxIiwic3RyaWV5IjoiZjJmNzViM2QwM2E5ZWRjZTAyNmZiMjUzYzViYjEiLCJpZCI6IjBhYmUwNzA1ZjUxZiIsImltbCI6IjBMb校长LcCm5vcm91Z2luY3VzIn0",
  "iv": "mH-G2zVqgzUtunK_",
  "ciphertext": "tkZuOO9h95OgHJmkkrlBisku8rgf6nzzVrhRM3sVOhXgz5NJ76oID71pAi_cPWJRLCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsxaEwdDxta9Mn5B7cCBoJKBO1gEnj_qfo1hi-uEkUpOZ8aLTZGfhp105jMwbKkTe2yK3mjF6SBAsgicQDVCckcY9BLluzx1RmC3ORXamOJHzB93YcdSDgpgpBWMrNU1ErkjeMcMoT_wtCex3w03XdLkJXiURzWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyc6QWcqAu07WNhjzJEPc4jVntrJ6K53NgP0S5p9913Z4080Uqj4ioYezbS6vTP1Q",
  "tag": "WuGzxmcreYjpHGJoal7EBg"
}
```

Figure 94: JSON Serialization

4.5. 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.

4.5.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- EC public key; this example uses the public key from Figure 95.
- "alg" parameter of "ECDH-ES"
- "enc" parameter of "A128CBC-HS256"

Miller Expires October 23, 2014 [Page 48]
Figure 95: 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.)

4.5.2. Generated Factors

The following are generated before encrypting:

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 96.

yc9N8v5sYv3jGQT926IUg

Figure 96: Initialization Vector, base64url-encoded

(*NOTE*: The Content Encryption Key (CEK) is not randomly generated; instead it is determined using key agreement.)

4.5.3. Key Agreement

The following are generated to agree on a CEK:

- Ephemeral private key; this example uses the private key from Figure 97.

{}
Performing the ECDH operation using the static EC public key (Figure 95) over the ephemeral private key Figure 97) produces the following CEK:

```
hzHdlfQIAEehb8Hrd_mFRhKsKLEzPfshfXs9l6areCc
```

Figure 98: Agreed-to Content Encryption Key, base64url-encoded

### 4.5.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 99, encoded to [RFC4648] as Figure 100.

```
{
    "alg": "ECDH-ES",
    "kid": "meriadoc.brandybuck@buckland.example",
    "epk": {
        "kty": "EC",
        "crv": "P-256",
        "x": "mPUKT_hAWGH1hg0TpjgVsqP1rXWQu_vwV0HTnkdYoA",
        "y": "8BQA8sImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4IrS"
    }
}
```

Figure 99: Protected JWE Header JSON

```
eyJhbGciOiJFQQRILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidWNRbGFuZC5leGFtcGxlIiwiZXSIjoiVW5leC5vdXIifQ
```

Figure 100: Protected JWE Header, base64url-encoded

Performing the content encryption operation on the Plaintext (Figure 51) using the following:

- CEK (Figure 98);
- Initialization vector/nonce (Figure 96); and
- Protected JWE header (Figure 100) as authenticated data

produces the following:
4.5.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 90)
- Initialization vector/nonce (Figure 86)
- Ciphertext (Figure 91)
- Authentication tag (Figure 92)

the resulting JWE object using the Compact serialization:
eyJhbGciOiJFQ0RILUVVTiwi2lkIjoibWVyaWFkb2MuYnJhbmr5YNvja0BidWNrbGFuZC5leGFtcGxlIiwibWViZXBrIjpi7mtoS16iKVDIiwi3Y3ZIzjoiUC0yNTYiLC4IjoibVBS1RFYkFXR0hJaGcwVH8qanFwClAxchUXVfndWT0hIdE5rZFlvQ5IsInkiOiI4QlFBc0l0Z2V8U2ZnLxZsVNAfImRiUWE1IQnBGdzJUzMoRHY0SXJZIn0sImVuYyI6IkJhDQkMtSFMyNTYiFQ.

BoDlwPnTnpYq-ivjmQVAJLb5Ql-F3LIQQomlz87yW4QPKbWE1zSEFjdFhU9IPiOSA9Bm147iDFwA-1ZXvhEksiDTw4R1xRGMesDiAYvtkTtmmzn-__q4_evAPUmlO-ZG45Mnq4uhM1fm_D9rBtWolqZF3xGNNkpOMQKF1i18wijeRii7-IXgYiRQsbhJrKzv8iC7i6a124i3C-AR2le1r7URUhrMr79BY8soZU0lzwi-sD5P2314NDCsei9XkoIafsXJWmySPoerB2Ni5UZL4YpKDiwnyzGd65KqVv7MsFfi_K7679C9Azp7gKZD0DyUn1mn0WW5lmYx_yJ-3ARQq8p1WZBfG-ZyJ6i95_JGG2m9Csg.

WCCKa-x4BeB9hIDIFuhg

Figure 103: Compact Serialization

the resulting JWE object using the JSON serialization:

```
{  
  "protected": "eyJhbGciOiJFQ0RILUVVTiwi2lkIjoibWVyaWFkb2MuYnJhbmr5YNvja0BidWNrbGFuZC5leGFtcGxlIiwibWViZXBrIjpi7mtoS16iKVDIiwi3Y3ZIzjoiUC0yNTYiLC4IjoibVBS1RFYkFXR0hJaGcwVH8qanFwClAxchUXVfndWT0hIdE5rZFlvQ5IsInkiOiI4QlFBc0l0Z2V8U2ZnLxZsVNAfImRiUWE1IQnBGdzJUzMoRHY0SXJZIn0sImVuYyI6IkJhDQkMtSFMyNTYiFQ",  
  "iv": "yc9N8v5sYyv3IQGT926lUg",  
  "ciphertext": "BoDlwPnTnpYq-ivjmQVAJLb5Ql-F3LIQQomlz87yW4QPKbWE1zSEFjdFhU9IPiOSA9Bm147iDFwA-1ZXvhEksiDTw4R1xRGMesDiAYvtkTtmmzn-__q4_evAPUmlO-ZG45Mnq4uhM1fm_D9rBtWolqZF3xGNNkpOMQKF1i18wijeRii7-IXgYiRQsbhJrKzv8iC7i6a124i3C-AR2le1r7URUhrMr79BY8soZU0lzwi-sD5P2314NDCsei9XkoIafsXJWmySPoerB2Ni5UZL4YpKDiwnyzGd65KqVv7MsFfi_K7679C9Azp7gKZD0DyUn1mn0WW5lmYx_yJ-3ARQq8p1WZBfG-ZyJ6i95_JGG2m9Csg",  
  "tag": "WCCKa-x4BeB9hIDIFuhg"  
}
```

Figure 104: JSON Serialization
4.6. Direct Encryption using AES-GCM

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

4.6.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 105.
- "alg" parameter of "dir"
- "enc" parameter of "A128GCM"

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

Figure 105: AES 128-bit key, in JWK format

4.6.2. Generated Factors

The following are generated before encrypting:

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 106.

```
refa467QzzKx6QAB
```

Figure 106: Initialization Vector, base64url-encoded

4.6.3. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 107, encoded as [RFC4648] base64url to produce Figure 108.
Figure 107: Protected JWE Header JSON

Encoded as [RFC4648] base64url:

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOj0zN2ZTExLTQ1Y2YtODY3Mi02MT
diW1ONTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0

Figure 108: Protected JWE Header, base64url-encoded

Performing the encryption operation on the Plaintext (Figure 51) using the following:

- CEK (Figure 105);
- Initialization vector/nonce (Figure 106); and
- Protected JWE header (Figure 108) as authenticated data

produces the following:

- Ciphertext from Figure 109.
- Authentication tag from Figure 110.

JW_i_f52hww_ELQPGaYyeAB6HYGcR55918TYnSovc23XJoBcW29rHP8yZOZG7Y
hLPt1bfUvz2PjQS-m0IFtVcXk2Xdh_1r_FrdYt9HRUYkshtrMmIUAYfGmUnd9zM
DB2n0crDIHAsFVeJUdxUwVAE7_YGRFdcqMy1BoCO-FbDe-Nceb4h3-FtBP-c_
B1wCPTjb9O0SbdcRDEJMYmZ8H8ysWVMvlgPD9yxi-aQpGbSv_F9N4iAZxcj5
g-NJsUPbjsk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZRSIn
Zl-wjsY0yu3cT4_aQ3ilo-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp

Figure 109: Ciphertext, base64url-encoded

vbb32Xvllea20tmHAHaccQ

Figure 110: Authentication Tag, base64url-encoded

4.6.4. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 108)
The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTezLTQ1Y2YtODY3Mi02MTdiNW1ONTI0M2EiLCJlbmMiOiJjbMTI4R0NNIn0.
```

![Figure 111: Compact Serialization](image)

The resulting JWE object using the JSON serialization:

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJjOC02ZTezLTQ1Y2YtODY3Mi02MTdiNW1ONTI0M2EiLCJlbmMiOiJjbMTI4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hw_ELQPGaYyeAB6HYGcrR5591TYnSovc23XJoBcW29rHP8yZOZ7YhLpTlbf4fuZpJS-m0IFtVcXkZxDH_lr_FrdYt9HRUYkshtMrMiUAyGmUnd9zMDB2n0cRIDHAzFVeJUDkUwVAE7_YGRPdcqMyBoCO-FbdE-Nce4h3-FtBP-c_BlwCPTjb900bdcdREEMJMyZBH8ySMV1lgPD9yxi-RqGbSv_F9N4IZAxcj5g-NJsUPbjk29-s7LJAbg15wEBiTPhxVCgyy53CoIKLHeJHXYx45Uz9aKZSRSInZI-wjsYyu3cT4-aQ3ilo-tIE-F81os61EKgyIQ4CWao8PFMj8TTnp",
  "tag": "vbb32Xv1lea20tmHAdccRQ"
}
```

![Figure 112: JSON Serialization](image)

### 4.7. 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.
4.7.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- AES symmetric key; this example uses the key from Figure 113.
- "alg" parameter of "A256GCMKW"
- "enc" parameter of "A128CBC-HS256"

```
    {   
      "kty": "oct",  
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",  
      "use": "enc",  
      "alg": "A256GCMKW",  
      "k": "qC571_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBdvQstiS8"  
    } 
```

Figure 113: AES 256-bit Key

4.7.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 114.
- Initialization vector/nonce for content encryption; this example uses the initialization vector/nonce from Figure 115.

```
UWxARpat23nL9ReIj4WG3D1ee9I4r-Mv5QLuFXdy_rE
```

Figure 114: Content Encryption Key, base64url-encoded

```
gz6NjyEFNm_vm8Gj6FwoFQ
```

Figure 115: Initialization Vector, base64url-encoded

4.7.3. Encrypting the Key

The following are generated before encrypting the CEK:

- Initialization vector/nonce for key wrapping; this example uses the initialization vector/nonce from Figure 116.
Performing the key encryption operation over the CEK (Figure 114) with the following:

- AES symmetric key (Figure 113);
- Key wrap initialization vector/nonce (Figure 116); and
- The empty string as authenticated data

produces the following:

- Encrypted Key from Figure 117.
- Key wrap authentication tag from Figure 118.

4.7.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 119, encoded to [RFC4648] base64url as Figure 120.

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

Figure 119: Protected JWE Header JSON
Performing the content encryption operation over the Plaintext (Figure 51) with the following:

- CEK (Figure 114);
- Initialization vector/nonce (Figure 115); and
- Protected JWE header (Figure 120) as authenticated data produces the following:
  - Ciphertext from Figure 121.
  - Authentication tag from Figure 122.

Figure 121: Ciphertext, base64url-encoded

DKW7jrb4WaRSNbXVP1T5g

Figure 122: Authentication Tag, base64url-encoded

4.7.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 120)
- encrypted key (Figure 117)
- Initialization vector/nonce (Figure 115)
- Ciphertext (Figure 121)
o Authentication tag (Figure 122)

The resulting JWE object using the Compact serialization:

eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYjIwNS0yYjRkJDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRM1QzSDZ2bXV3dC0ta3N3IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIiw2WjIjoiQTEyOENCQy1IUzI1NiJ9.

lJf3HbOApxMEBkCMoTnnABxs_CvTWUmZQ2ElLvYNok

gz6NjyEFNm_vm8Gj6FwoFQ

Figure 123: Compact Serialization

The resulting JWE object using the JSON serialization:

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

4.8.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- AES symmetric key; this example uses the key from Figure 125.
- "alg" parameter of "A128KW"
- "enc" parameter of "A128GCM"
4.8.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key; this example uses the key from Figure 126.

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 127.

Figure 125: AES 128-Bit Key

4.8.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 126) with the AES key (Figure 125) produces the following encrypted key:

Figure 126: Content Encryption Key, base64url-encoded

Figure 127: Initialization Vector, base64url-encoded

Figure 128: Encrypted Key, base64url-encoded

4.8.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 129, encoded to [RFC4648] base64url as Figure 130.
4.8.5. Output Results

The following compose the resulting JWE object:

o Protected JWE header (Figure 130)
o encrypted key (Figure 128)

o Initialization vector/nonce (Figure 127)

o Ciphertext (Figure 131)

o Authentication tag (Figure 132)

The resulting JWE object using the Compact serialization:

eyJhbGciOiJBMTI4S1ciLCJraWRqIjoiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0.

Figure 133: Compact Serialization

The resulting JWE object using the JSON serialization:

```json

Miller                  Expires October 23, 2014               [Page 63]
```
{'recipients': [{'encrypted_key': 'CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx'}, ...]}

Figure 134: JSON Serialization

4.9. Compressed Content

This example illustrates encrypting content that is first compressed. It reuses the AES key, key encryption algorithm, and content encryption algorithm from Section 4.8.

4.9.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.

- Recipient encryption key; this example uses the key from Figure 125.

- Key encryption algorithm; this example uses "A128KW".

- Content encryption algorithm; this example uses "A128GCM".

- "zip" parameter as "DEF".

4.9.2. Generated Factors

The following are generated before encrypting:
4.9.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 136) with the AES key (Figure 125) produces the following encrypted key:

5vUT2WOtQxKwcekM_IzVQwkGgz1FDwPi

Figure 138: Encrypted Key, base64url-encoded

4.9.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 139, encoded as [RFC4648] base64url as Figure 140.


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

Figure 139: Protected JWE Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTE2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiemlwIjoiREVGIn0

Figure 140: Protected JWE Header, base64url-encoded

Performing the content encryption operation over the compressed Plaintext (Figure 135, encoded as an octet string) with the following:

- CEK (Figure 136);
- Initialization vector/nonce (Figure 137); and
- Protected JWE header (Figure 140) as authenticated data

produces the following:

- Ciphertext from Figure 141.
- Authentication tag from Figure 142.

And authentication tag:

VILuUwuIXaLVmh5X-T7kmA

Figure 142: Authentication Tag, base64url-encoded

4.9.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 140)
o encrypted key (Figure 138)

o Initialization vector/nonce (Figure 137)

o Ciphertext (Figure 141)

o Authentication tag (Figure 142)

The resulting JWE object using the Compact serialization:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MFk0TFlYzgiLCJlbmMiOiJBMTI4R0NNIiwicmVwb3NlZ3J5IjoiREVGIn0.

5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi

p9Uq6XHYoifEZI1

HbDtOsdaIoyZiSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyesSPT0fdLVfe6Jf5y5-JaCap_JQbcb5opbmT60uWGm18ylimiMqMn9j--Xhhllyg0
m-BHqfDo5iTOWxPxFMJedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbryBK
hpYA7qi3AyijncJ7BP9rr3U8kExCpG3mK420Tjow

VIIlUwUIxaLVMh5X-T7kmA

Figure 143: Compact Serialization

The resulting JWE object using the JSON serialization:

{
  "recipients": [
    {
      "encrypted_key": "5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MFk0TFlYzgiLCJlbmMiOiJBMTI4R0NNIiwicmVwb3NlZ3J5IjoiREVGIn0",
  "iv": "p9Uq6XHYoifEZI1",
  "ciphertext": "HbDtOsdaIoyZiSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyesSPT0fdLVfe6Jf5y5-JaCap_JQbcb5opbmT60uWGm18ylimiMqMn9j--Xhhllyg0-BHqfDo5iTOWxPxFMJedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbryBKhpYA7qi3AyijncJ7BP9rr3U8kExCpG3mK420Tjow",
  "tag": "VIIlUwUIxaLVMh5X-T7kmA"
}

Figure 144: JSON Serialization
4.10. 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 Compact serialization, only the JSON serialization is possible.

4.10.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- Recipient encryption key; this example uses the key from Figure 125.
- Key encryption algorithm; this example uses "A128KW".
- Content encryption algorithm; this example uses "A128GCM".
- Additional authenticated data; this example uses a [RFC7095] vCard from Figure 145, serialized to UTF-8.

```json
[  
  "vcards",
  [
    [
      "version", {}, "text", "4.0" ],
    [
      "fn", {}, "text", "Meriadoc Brandybuck" ],
    [
      "n", {},
      "text", [
        "Brandybuck", "Meriadoc", "Mr.", ""
      ]
    ],
    [
      "bday", {}, "text", "TA 2982" ],
    [
      "gender", {}, "text", "M"
    ]
  ]
]
```

Figure 145: Additional Authenticated Data, in JSON format

*NOTE* whitespace between JSON values added for readability.

4.10.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 146.
4.10.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 146) with the AES key (Figure 125) produces the following encrypted key:

```
4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X
```

Figure 149: Encrypted Key, base64url-encoded

4.10.4. Encrypting the Content

The following are generated before encrypting the content:

- **Protected JWE Header; this example uses the header from Figure 150, encoded to [RFC4648] base64url as Figure 151.**

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

Figure 150: Protected JWE Header JSON
Performing the content encryption operation over the Plaintext with the following:

- CEK (Figure 146);
- Initialization vector/nonce (Figure 147); and
- Concatenation of the protected JWE header (Figure 151), ".", and the [RFC4648] base64url encoding of Figure 145 as authenticated data produces the following:

- Ciphertext from Figure 152.
- Authentication tag from Figure 153.

4.10.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 151)
- encrypted key (Figure 149)
- Initialization vector/nonce (Figure 147)
- Additional authenticated data (Figure 148)
- Ciphertext (Figure 152)
The resulting JWE object using the JSON serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "4YiiQ_ZzH76TaIkJmYfrFg0VSMIpnx4X"
        }
    ],
    "protected": "eyJhbGciOiJBMTI4S1ciLCJaWQOiOii4MWeiMdS0N0MyLTQz
    "iv": "veC6x9ec2orS7c_N",
    "aad": "WyJ2Y2ZvZCIcS1sidmVyc2lvbiIse30sIiIeHQiLCIO
    "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGs
    "tag": "v0aH_Rajjpy_3h0tqyZHR"
}
```

Figure 154: JSON Serialization

4.11. Protecting Specific Header Fields

This example illustrates encrypting content where only certain JWE header parameters are protected. As this example includes unprotected JWE header parameters, only the JSON serialization is possible.

4.11.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- Recipient encryption key; this example uses the key from Figure 125.
- Key encryption algorithm; this example uses "A128KW".

Miller Expires October 23, 2014
4.11.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 155.
- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 156.

WDgEptBmQs9ouUvArz6x6g

Figure 155: Content Encryption Key, base64url-encoded

WgEJsDS9bkoXQ3nR

Figure 156: Initialization Vector, base64url-encoded

4.11.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 155) with the AES key (Figure 125) produces the following encrypted key:

jJIcM9J-hbx3wnqhf5FlkEYos0sHsF0H

Figure 157: Encrypted Key, base64url-encoded

4.11.4. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 158, encoded to [RFC4648] base64url as Figure 159.

```
{
  "enc": "A128GCM"
}
```

Figure 158: Protected JWE Header JSON

eyJlbmMiOiJBMTI4R0NNIn0

Figure 159: Protected JWE Header, base64url-encoded

Performing the content encryption operation over the Plaintext with the following:
4.11.5. Output Results

The following compose the resulting JWE object:

- Unprotected JWE header (Figure 162)
- Protected JWE header (Figure 159)
- Encrypted key (Figure 157)
- Initialization vector/nonce (Figure 156)
- Ciphertext (Figure 160)
- Authentication tag (Figure 161)

The following unprotected JWE header is generated before assembling the output results:

```json
{
  "alg": "none",
  "protected": ["type": "JWE", "algorithm": "A256GCM", "key_ops": ["encrypt", "decrypt"]]
}
```
The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "jJIcM9J-hbx3wnqhf5FlkEYos0sHsFOH"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJEsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3u9DPsqLL2DM3swKkjOwQyZtWsFLYMj5YeLht_StAn21tHmQjuuNt64T84t6C7kc90CCJ1IHAo1Uv8MyOt80MoPb8fZYbNKqplzyY3gIL58g8N2v46OgyG637d6uukPwhAnTgm_zWhqc_srOvgiLkzyFXPq1hBAURbc3-8BqeRb481R1-_5g5UjWVD31giLCN_F7Aw8mIiFvUNXBPJK3nOWL4teUPS8yHlWE1L83o1U4UAgl48x-8dDkH23JykibVSQju-f7e-1xreHWXzWLHs1NqBbre0dEwK3HX_xM0LjUz77Krpppegoutpf5qaKg3l-xMINmf",
  "tag": "fNYLqpe84KD451vDialBAQ"
}
```

Figure 163: JSON Serialization

4.12. Protecting Content Only

This example illustrates encrypting content where none of the JWE header parameters are protected. As this example includes only unprotected JWE header parameters, only the JSON serialization is possible.

4.12.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 51.
- Recipient encryption key; this example uses the key from Figure 125.
4.12.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key; this example the key from Figure 164.
- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 165.

KB0oAF130QPV3vkcZ1XnzQ

Figure 164: Content Encryption Key, base64url-encoded

YihBoVOGsR1l7jCD

Figure 165: Initialization Vector, base64url-encoded

4.12.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 164 with the AES key (Figure 125 produces the following encrypted key:

244YHfO_W7RMpQW81UjQrZcq5LSyqiPv

Figure 166: Encrypted Key, base64url-encoded

4.12.4. Encrypting the Content

Performing the content encryption operation over the Plaintext (Figure 51) using the following:

- CEK (Figure 164);
- Initialization vector/nonce (Figure 165); and
- Empty string as authenticated data

produces the following:

- Ciphertext from Figure 167.
- Authenticated data from Figure 168.
The following unprotected JWE header is generated before assembling the output results:

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

The following compose the resulting JWE object:

- Unprotected JWE header (Figure 169)
- Encrypted key (Figure 166)
- Initialization vector/nonce (Figure 165)
- Ciphertext (Figure 167)
- Authentication tag (Figure 168)

The resulting JWE object using the JSON serialization:
Figure 170: JSON Serialization

4.13. Encrypting to Multiple Recipients

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

4.13.1. Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the plaintext from Figure 51.
- Recipient keys; this example uses the following:
  - The RSA public key from Figure 52 for the first recipient.
  - The EC public key from Figure 84 for the second recipient.
  - The AES symmetric key from Figure 113 for the third recipient.
- Key encryption algorithms; this example uses the following:
  * "RSA1_5" for the first recipient.

```json
{
  "recipients": [
    {
      "encrypted_key": "244YHfo_W7RMpQW81UjQrZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoV0GsR1l7jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQhOa7Gqrk7Eal1vwht7R4T7uq-
  arsVCpAeFwFqzrS6oEUWbBtxEasE0vc6r7spyVzimCVJEuRJyoAHF
  SP3eqQpb4lScsyXjw_L3vybheYUGyQuTmQEDjgjJfBOfwHIsDs
  RPeBzlNomqefiFVq5GTcWo5K_MNIQRR2Wj0AHc2k7JZfu21jWjUHlF8
  ExFZL24nmsvJu_mviMyiikFf5ZsZAudISOa6073yPztL04k_1FI7Wdf
  rb2w70qKLDXz1pcxohPVOLQwpA3mFNRKdY-bQz424KX9fz1cne31N4
  8Kmojpw-0dQjKdLOGkC445Fb_K1t1QXw2sBF",
  "tag": "e2m0Vmv7JvkJ2VpCKXS-kyg"
}
```

* "ECDH-ES+A256KW" for the second recipient.

* "A256GCMKW" for the third recipient.

- Content encryption algorithm; this example uses "A128CBC-HS256"

### 4.13.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 171.

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 172.

zXayeJ4gm8NJr3UIInyokTUO-LbQNKKhe_zW1YbdpQ

Figure 171: Content Encryption Key, base64url-encoded

VgEIHY20EnzUtZFl2RpB1g

Figure 172: Initialization Vector, base64url-encoded

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

Performing the "RSA1_5" key encryption operation over the CEK (Figure 171 with the first recipient’s RSA key (Figure 52 produces the following encrypted key:

dYOD28kab0Vvf40DgxVAJXgHcSZICSoP8M5Izjwj4w6Y5G4XJQaNNIBiqyvUUA
OcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bRWrF7cHwEQJdXihi1AYWVajJ1a
RMXMrFRMV6iD1Rr076DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJD0Gb-w
qjw0-b-Sl1aSl1QVbuP78dQ7Fa0zAvzjHx-xvyM2wux_j_txr9c1N1n2MbeYS
rRicK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWP1nBq_V4n5strGeexp_-d
bHcGlihqg_wGgho9fLMK8J0ArYLCMDNQ

Figure 173: Recipient #1 Encrypted Key, base64url-encoded

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

- Recipient JWE header from Figure 174
The following is the assembled first recipient JSON:

```json
{
  "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcS2ICSOp8M51zjwj4w6Y5G4XJQsNNIBiqyvUUAoCpL7S7-cFe7Pi07gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXihidAYWvajJiaKMxMvFRMV6i1Dr076DFthg2_AV0_tSiV6xSEIFq1xnYPfm91tc5WJD0Gb-wqjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyMwxj_oxxr9cN1Lz2MbeYSrRicJK5xodWgkpIdkMHo4LvhdRRvzoKzlic89jFPW1nBq_V4n5strGuExtp_-_dbHcGlihqc_wGgho9fLMK80ArYlcoMDNQ",
  "header": {
    "alg": "RSA1_5",
    "kid": "frodo.baggins@hobbiton.example"
  }
}
```

Figure 175: Recipient #1 JSON

4.13.4. Encrypting the Key to the Second Recipient

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

- Ephemeral EC private key on the same curve as the EC public key; this example uses the private key from Figure 176.

```json
{
  "kty": "EC",
  "crv": "P-384",
  "x": "Uzdvk3pi5wKCRclizp5_r00jeqT-I68i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
  "y": "VDrRyFJh-Kwd1EjAqmj5Eo-CTHAZ53MC7PjppLioy3y1Eji1pOmbw9lfz84pbfm",
  "d": "1DKHfTv-PiIfVw2VBHM_ZiVcw0Mxx0yANS_1QHJcrDxVY3jhVCv2PwMxJKIE793C"
}
```

Figure 176: Ephemeral public key for Recipient #2, in JWK format

Performing the "ECDH-ES+A256KW" key encryption operation over the CEK (Figure 171 with the following: 

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

Figure 174: Recipient #1 JWE Header JSON
o Static Elliptic Curve public key (Figure 84).

o Ephemeral Elliptic Curve private key (Figure 176.

produces the following encrypted key:

ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw_elY4gSSIId_w

Figure 177: Recipient #2 Encrypted Key, base64url-encoded

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

o Recipient JWE Header from Figure 178.

```
{
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
        "kty": "EC",
        "crv": "P-384",
        "x": "Uzdvk3pi5wKCRc11zpi5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
        "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1Ej1pOMb
            w91fzZ84pbfm"
    }
}
```

Figure 178: Recipient #2 JWE Header JSON

The following is the assembled second recipient JSON:
4.13.5. Encrypting the Key to the Third Recipient

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

- Initialization vector/nonce for key wrapping; this example uses the initialization vector/nonce from Figure 180

$$AvpeoPZ9Ncn9mkBn$$

Figure 180

Performing the "A256GCMKW" key encryption operation over the CEK (Figure 171) with the following:

- AES symmetric key (Figure 113; and

- Initialization vector/nonce ((Figure 180 produces the following:

- Encrypted key from Figure 181.

- Key wrap authentication tag from Figure 182

$$a7Cc1Aejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E$$

Figure 181: Recipient #3 Encrypted Key, base64url-encoded
The following are generated after encrypting the CEK for the third recipient:

- Recipient JWE header; this example uses the header from Figure 183.

```json
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "59Nqh1LlYtVIfD3pgRGvw",
  "iv": "AvpeoPZ9Ncn9mkBn"
}
```

Figure 183: Recipient #3 JWE Header JSON

The following is the assembled third recipient JSON:

```json
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmt9-WyTps1E",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1LlYtVIfD3pgRGvw",
    "iv": "AvpeoPZ9Ncn9mkBn"
  }
}
```

Figure 184: Recipient #3 JSON

4.13.6. Encrypting the Content

The following are generated before encrypting the content:

- Protected JWE Header; this example uses the header from Figure 185, encoded to [RFC4648] base64url as Figure 186.

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

Figure 185: Protected JWE Header JSON
Performing the content encryption operation over the Plaintext (Figure 51) with the following:

- CEK (Figure 171),
- Initialization vector/nonce (Figure 172), and
- Protected JWE header (Figure 186) as the authenticated data produces the following:

- Ciphertext from Figure 187
- Authentication tag from Figure 188

The following is generated after encrypting the plaintext:

- Unprotected JWE header parameters; this example uses the header from Figure 189.

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

Figure 189: Unprotected JWE Header JSON
4.13.7. Output Results

The following compose the resulting JWE object:

- Recipient #1 JSON (Figure 175)
- Recipient #2 JSON (Figure 179)
- Recipient #3 JSON (Figure 184)
- Initialization vector/nonce (Figure 172)
- Ciphertext (Figure 187)
- Authentication tag (Figure 188)

The resulting JWE object using the JSON serialization:

```json
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf40DgxVAJXgHcSZICSOp8M51zj
wj4w6Y5G4XJQsNN1BiqvvUUA0cpL7S7-cFe7Pio7qV_Q06WmCSa-
vhW6me4bWrB77cHWaqJDhid7AYWvajJiaKMXmFRMv61d1Rr076
DFthg2_AV0_tSiV6xSEIFqtxnYPpmP91tc5WJDQGb-wqjw0-b-S
1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_oxtr9c1N1nZMbe
YSRicJK5xodwWgkpIkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n
5trGuExtp_-dbHcGlihq_c_wGho9fLMK8JGArYLCMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5t2lgoZXThD1zWKsHi
xJuw_elY4gSSId_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3pi5wKCRclizp5_r00jeqT-168i8g2b8mva8diRhs
E2xAn2DtMRb25Ma2CX",
          "y": "VDrFJh-Kwd1EjAgnj5Eo-CTHAZS3MC7PjypLloy3y1Ej
I1pOMbw91fzZ84pbfm"
        }
      }
    }
  ]
}
```
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.

5.1. Signing Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the JSON Web Token (JWT) content from Figure 191, encoded as [RFC4648] base64url to produce Figure 192.
- RSA private key; this example uses the key from Figure 193.
{  
"iss": "hobbiton.example",
"exp": 1300819380,
"http://example.com/is_root": true
}

Figure 191: Payload content, in JSON format

eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwibXhwIjoxMzAwODE5MzgwLCJodHRwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0

Figure 192: Payload content, base64url-encoded
"kty": "RSA",
"kid": "hobbiton.example",
"use": "sig",
"n": "knR1BcXMu6fcyv5i-QHQAQ-K8gsC3HJb7FYhYaw8hXbNJt-t8q0lD
KwL2qQXYV-ffWxXJlg12E4G521fMEeqDzYtTrQ3tepKFjMGg6I
y6fkk1l2Nsk2gEonlitSfzhA9GJwRTmtKpbk1s-hwx1I5um-AIe1NqBg
cFzv5E5w25_SGBOaROVdUxyQETDgmlZ5cKV42jJd2-1h4oVB07xkac6
LQdHqJUuySH_Er20DXx30Ky197PciXKTSt-QXNmm8ivyRCum22zoU
nd2BKC5o1G4wALhaL2Z2k8CsRdfy-7dg7z41Rp6d0eEvtaUp4bX4aK
ral4tFw",
"e": "AQAB",
"d": "ZLe_TiXpE9-W_n2VBA-HWvuYptfjxvWXClJFUnpsd8eag9Rmx34gE0
EtoYc2un3C3LtJi-mju5RAT8YSc76YJds3ZVw0U0i8mMB6eG6-inovg
obobNnxt575-xjTJ2UT2jor9k8z62KwdDq7HFyCdhUEycYHCIFc71L6
TibVhAhOFONWlq1JgEgwVYdorybNGKfdnpEbywHoMwY6HM1qvneEfpG7
I20YzHUT35x6jj4KvcAd72duFkhUaayseyW7mzX66f1jvdj1y9k1D1
fiz30X4cckoghKF5GONU6tNmMmNgAD6g1viye11Pr1xltBhCl14bRW
-zrpHgAQ",
"p": "yKWYoNIAqgMRq1gIBodT1INicbDNUUsr2h-pBaxd-nkweMt4Mg-0-B
2iSyvMr8shorh0v7vxQCagbAATGW-hAafUehWjxWSh-3KccRM8tol4
e0q7M-idRDOBXSoe722-CV2x_ZCY3R8qp642R13WgXqGDM4MbdUz2Sj
cY9-c",
"q": "uN4o15V30KDFs8fFJw589pV1vQv3NeilrinRUPHkkxaA2DZccGqr
WMmpXcGFI1NL35cCqIeU76-5IVQq0HwY1hVXQH7sgAGu-483A3D
ENC2L2fFonF45m7_2ooastJDe49MeLTTQKrSIB1_SkvqpfYvfPtcZ
kh9Kk",
"dp": "jmThEoq2qqa8ouaymjhJSCnsveUXnMqC2gAnQJRQkFqOu-zV2PKP
KnBPvKy1f5b2-L3tm3Ow2d2iNdyRUW1X7V510KwPTABSTOnTqAmYCh
Gi8kXXldhrcrSvxlBakC6asaxwT_TzqGY2MVCzZCnCvCXC4v4qjxoxF
p3HFU",
"dq": "R9FuvuB80VzEktKX13-5-WusE4DjHmndeZl1u3rifBdfLpq_P-iWP
BbGag9wzQ1c-J7szJqJkG1JVDv5yd2C7rnZ6kpzwBh_mL8zscAk1qsun
nt9CGAYz-sGWy1JGShFazfP52Th4r1CJ0yuEaQMrIzpY77_oLAhpm
DAOhLk",
"qi": "58tC7ZknW6p1TPjkwttQOPlVfmRfwRlFAViuDb8N9CrV_720Q
U2CqzmtHYAuumwGFB1m1WvRep7anLeWaJjXc_1b3fq_a14Q3pe-EK1Hg6
IMazuRtZLUR0cThrExDb5dYbscDnFwRULErZ41Be0bnxYpqxWkd9
QZwMo0"}

Figure 193: RSA 2048-bit Private Key, in JWK format

5.2. Signing Operation

The following are generated to complete the signing operation:

Miller Expires October 23, 2014 [Page 87]
5.3. Signing Output

The following compose the resulting JWS object:

- Protected JWS header (Figure 195))
- Payload content (Figure 192)
- Signature (Figure 196)

The resulting JWS object using the Compact Serialization (which is the plaintext input to the proceeding encryption operation):
5.4. Encryption Input Factors

The following are supplied before beginning the encryption process:

- Plaintext content; this example uses the content from Figure 197.
- RSA public key; this example uses the key from Figure 62.

5.5. Encryption Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption CEK (CEK); this example uses the key from Figure 198.
- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 199.

5.6. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 198) with the RSA key (Figure 62) produces the following encrypted key:
5.7. Encrypting the Content

The following are generated before encrypting the plaintext:

- Protected JWE Header; this example uses the header from Figure 201, encoded using [RFC4648] base64url to produce Figure 202.

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

Figure 201: Protected JWE Header JSON

Performing the content encryption operation over the Plaintext (Figure 197) with the following:

- CEK (Figure 198);
- Initialization vector/nonce (Figure 199); and
- Protected JWE Header (Figure 202) as authenticated data.

produces the following:

- Ciphertext from Figure 203.
5.8. Encryption Output

The following compose the resulting JWE object:

- **Protected JWE Header** (Figure 202)
- **Encrypted key** (Figure 200)
- **Initialization vector/nonce** (Figure 199)
- **Ciphertext** (Figure 203)
- **Authentication Tag** (Figure 204)

The resulting JWE object using the Compact serialization:
The resulting JWE object using the JSON serialization:

Figure 205: Compact Serialization

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6I1kxVCIIsImVuYyI6I1kxEmxhHQ00ifQ.
.a0JHRoITfpX4qRewImjIStn8m3CPxBVlueY1VhjurCyrBgl37YhCRYjphDOOS4E7rzbr2FnN6yQq-A-qg0FPkqNjVOGzG-bi13mwy7RoYjtkBEC67sMWMXXx4g
zMedpiJHQVeyI-zkZV7a9matqgevA1WrxZ0OUy1YTto2SN6gtUVt11aivjvb210Ou1u4XShVh-BjikyTeR_fuYJxHoKLQl9L44sKx2WGYb4zsB1IPF4ssl_e51
R7nany-25_UmC2urosNkoFz9cQ82MyP2P8gqbQJYPN-Fpp4Z-5o6yV64x6yzDUE
F_5JCID1-Qv6H5mdMVI7qleKpxcV11Wo_2FeEBqxxXy1jLeZivjNkzogCq3-I
apsJvMjBxjYLT8mauaowlyy1XXMunIpNc0Y3n4KkrXLrCceX85m4IIIHz
a38s1Hpr56fPsEma-J1tmt-a91EoDzhtxZ8AXY9tsCAV2XBWNG8c3KJusAa
mBKOYwfk7JhLhRDgOnjJjLJhn7TI4UxDp9dCmUX6Nz0v23W15qJXENJtqnblp
ymooeWAHC74e_Owbin1g0AePHUdA2i1LNNs9WTX_H_TXuPC8yDDh1smlxS_x
pkIHKiIHWDO1x03BpDTipvKkBWyqP2U2kcxqX2Fo_GnVrNw1K7Lgxw6FSQvdo
0.
GbX1i9kXz0sXXpMA

SZI4IVKHamwpazl_pJRXX3mHv1AAn0U4Wf9-utWYUCrBzN0eC20FMf66cSJ8k2Q
kxqQD3_R60MGE9ofomtwly3GFXMeGRjtpMt90AveLsAXB0_UTCGBvByg3C2fWLX
qzlffAAoJ1RUPRk-BimZY81zVBuIch7sQePcupu33SzMsFHjn41P_idrJz_g1Z
TNgKDr8ZdnUPauTKKDN0H1D4fuwYDfIAfGPyL5sVrbXpXgokEsZM-9C
hMpqW1ONhzuX_2u13bvrJwr7nUG2s4cUScY3n8E3AHCurlgls-A9mz1X38xEa
u1V1814F9g9tLejdAkU0q3PbqbeiHBQJe41WGD50deQ-Mtx4NhkhIWX-YKBB_Xoc
zi3Q_lsyjKU0s7yWW-HT_vqVFT0boj7Wjf2vB0TZ3dvsoGaTvPh2dyWwumUr
lX4gmPuzBdW06ubfYSDEUE5py0dOt3EmUScCYBKr-am7T4x2W6J0zlYjLf
hnm9zy-W19sOCZGuzFGfPahwXhpnvjt_t-O_ES96kogjJLXs1IM9Y5mnwZMyNc
9E1wnqscCg-hVuvzyFOSiruktM194_Sl1xgM170o3phcTMxtlm1zr88NKUWkB
s1XMCjy1Noue7MD-ShDp5dmM

KnIKEhN8U-3C9s4gtSpjSw
Figure 206: JSON Serialization

6. Security Considerations

This document introduces no new security considerations over those stated in [I-D.ietf-jose-json-web-algorithms], [I-D.ietf-jose-json-web-encryption], [I-D.ietf-jose-json-web-key], and [I-D.ietf-jose-json-web-signature].
7. IANA Considerations

This document has no actions for IANA.

8. Informative References

[I-D.ietf-jose-json-web-algorithms]

[I-D.ietf-jose-json-web-encryption]

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-25 (work in progress), March 2014.

[I-D.ietf-jose-json-web-signature]

[I-D.ietf-oauth-json-web-token]


Appendix A. Acknowledgements

All of the examples herein use quotes and character names found in the novels "The Hobbit"; "The Fellowship of the Ring"; "The Two Towers"; and "Return of the King", written by J. R. R. Tolkien.

Thanks to Richard Barnes and Jim Schaad for providing for their input on the outline for this document. Thanks to Brian Campbell for reviewing text and verifying most of the examples.
Author’s Address

Matthew Miller
Cisco Systems, Inc.

Email: mamille2@cisco.com