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
4.2.1. Input Factors ............................................. 32
4.2.2. Generated Factors ........................................ 34
4.2.3. Encrypting the Key ....................................... 35
4.2.4. Encrypting the Content .................................. 35
4.2.5. Output Results ........................................... 36
4.3. Key Wrap using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2 ... 38
  4.3.1. Input Factors ............................................. 38
  4.3.2. Generated Factors ........................................ 39
  4.3.3. Encrypting the Key ....................................... 40
  4.3.4. Encrypting the Content .................................. 40
  4.3.5. Output Results ........................................... 42
4.4. Key Agreement with Key Wrapping using ECDH-ES and AES-
       KeyWrap with AES-GCM ..................................... 43
  4.4.1. Input Factors ............................................. 43
  4.4.2. Generated Factors ........................................ 44
  4.4.3. Encrypting the Key ....................................... 44
  4.4.4. Encrypting the Content .................................. 45
  4.4.5. Output Results ........................................... 46
4.5. Key Agreement using ECDH-ES with AES-CBC-HMAC-SHA2 .......... 48
  4.5.1. Input Factors ............................................. 48
  4.5.2. Generated Factors ........................................ 49
  4.5.3. Key Agreement ............................................ 49
  4.5.4. Encrypting the Content .................................. 50
  4.5.5. Output Results ........................................... 51
4.6. Direct Encryption using AES-GCM ................................ 53
  4.6.1. Input Factors ............................................. 53
  4.6.2. Generated Factors ........................................ 53
  4.6.3. Encrypting the Content .................................. 53
  4.6.4. Output Results ........................................... 54
4.7. Key Wrap using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2 ...... 55
  4.7.1. Input Factors ............................................. 56
  4.7.2. Generated Factors ........................................ 56
  4.7.3. Encrypting the Key ....................................... 56
  4.7.4. Encrypting the Content .................................. 57
  4.7.5. Output Results ........................................... 58
4.8. Key Wrap using AES-KeyWrap with AES-GCM ...................... 60
  4.8.1. Input Factors ............................................. 60
  4.8.2. Generated Factors ........................................ 61
  4.8.3. Encrypting the Key ....................................... 61
  4.8.4. Encrypting the Content .................................. 61
  4.8.5. Output Results ........................................... 62
4.9. Compressed Content ........................................... 64
  4.9.1. Input Factors ............................................. 64
  4.9.2. Generated Factors ........................................ 64
  4.9.3. Encrypting the Key ....................................... 65
  4.9.4. Encrypting the Content .................................. 65
  4.9.5. Output Results ........................................... 66
4.10. Including Additional Authenticated Data ....................... 68
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.10.1</td>
<td>Input Factors</td>
<td>68</td>
</tr>
<tr>
<td>4.10.2</td>
<td>Generated Factors</td>
<td>68</td>
</tr>
<tr>
<td>4.10.3</td>
<td>Encrypting the Key</td>
<td>69</td>
</tr>
<tr>
<td>4.10.4</td>
<td>Encrypting the Content</td>
<td>69</td>
</tr>
<tr>
<td>4.10.5</td>
<td>Output Results</td>
<td>70</td>
</tr>
<tr>
<td>4.11</td>
<td>Protecting Specific Header Fields</td>
<td>71</td>
</tr>
<tr>
<td>4.11.1</td>
<td>Input Factors</td>
<td>71</td>
</tr>
<tr>
<td>4.11.2</td>
<td>Generated Factors</td>
<td>72</td>
</tr>
<tr>
<td>4.11.3</td>
<td>Encrypting the Key</td>
<td>72</td>
</tr>
<tr>
<td>4.11.4</td>
<td>Encrypting the Content</td>
<td>72</td>
</tr>
<tr>
<td>4.11.5</td>
<td>Output Results</td>
<td>73</td>
</tr>
<tr>
<td>4.12</td>
<td>Protecting Content Only</td>
<td>74</td>
</tr>
<tr>
<td>4.12.1</td>
<td>Input Factors</td>
<td>74</td>
</tr>
<tr>
<td>4.12.2</td>
<td>Generated Factors</td>
<td>75</td>
</tr>
<tr>
<td>4.12.3</td>
<td>Encrypting the Key</td>
<td>75</td>
</tr>
<tr>
<td>4.12.4</td>
<td>Encrypting the Content</td>
<td>75</td>
</tr>
<tr>
<td>4.12.5</td>
<td>Output Results</td>
<td>76</td>
</tr>
<tr>
<td>4.13</td>
<td>Encrypting to Multiple Recipients</td>
<td>77</td>
</tr>
<tr>
<td>4.13.1</td>
<td>Input Factors</td>
<td>77</td>
</tr>
<tr>
<td>4.13.2</td>
<td>Generated Factors</td>
<td>78</td>
</tr>
<tr>
<td>4.13.3</td>
<td>Encrypting the Key to the First Recipient</td>
<td>78</td>
</tr>
<tr>
<td>4.13.4</td>
<td>Encrypting the Key to the Second Recipient</td>
<td>79</td>
</tr>
<tr>
<td>4.13.5</td>
<td>Encrypting the Key to the Third Recipient</td>
<td>81</td>
</tr>
<tr>
<td>4.13.6</td>
<td>Encrypting the Content</td>
<td>82</td>
</tr>
<tr>
<td>4.13.7</td>
<td>Output Results</td>
<td>84</td>
</tr>
<tr>
<td>5.</td>
<td>Nesting Signatures and Encryption</td>
<td>85</td>
</tr>
<tr>
<td>5.1</td>
<td>Signing Input Factors</td>
<td>85</td>
</tr>
<tr>
<td>5.2</td>
<td>Signing Operation</td>
<td>87</td>
</tr>
<tr>
<td>5.3</td>
<td>Signing Output</td>
<td>88</td>
</tr>
<tr>
<td>5.4</td>
<td>Encryption Input Factors</td>
<td>89</td>
</tr>
<tr>
<td>5.5</td>
<td>Encryption Generated Factors</td>
<td>89</td>
</tr>
<tr>
<td>5.6</td>
<td>Encrypting the Key</td>
<td>89</td>
</tr>
<tr>
<td>5.7</td>
<td>Encrypting the Content</td>
<td>90</td>
</tr>
<tr>
<td>5.8</td>
<td>Encryption Output</td>
<td>91</td>
</tr>
<tr>
<td>6.</td>
<td>Security Considerations</td>
<td>93</td>
</tr>
<tr>
<td>7.</td>
<td>IANA Considerations</td>
<td>94</td>
</tr>
<tr>
<td>8.</td>
<td>Informative References</td>
<td>94</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Acknowledgements</td>
<td>94</td>
</tr>
<tr>
<td>Author’s Address</td>
<td>94</td>
<td></td>
</tr>
</tbody>
</table>
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\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 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:

SXTigJ1zIGFzZGQzVyb3VzIGVjIGJlZc2luZWNzLCBGcm9kdwpwZ29pbmcgb3Iy 
lvdXGf9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvb 
mcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0 
cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4g 
WW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIg 
ZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvb 
mcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0 
cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4g 
WW91IHN0cG9zdCBvbmcgYXlvdXIgZG9vci4gWW91IHN0cG9zdCBvbmcgYXlvdXIg 

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

3.1.2. Signing Operation

The following are generated to complete the signing operation:

```json
{
"kty": "RSA",
"kid": "bilbo.baggins@hobbiton.example",
"use": "sig",
"n": "n4EPTa0C3c9AlkeQHPzHStgAbgs7bTZWdBrdR8_KuKPEHld4rHVTdT
o-XV2jRojNhxJWTDvNd7nqQ0V8ElZQHz-A_JmSCpMaJMR3BSFKrKb2wqV
wGU_NaYOL-QtiW21bze8E6XO0aqPr5ydQLRqKHHg3Rborda26A-J
oBhqFEHYP7e7Tpe-0VFWhd16c8686SMIFZcD1NNLYD51FHpP19bTDisplays
3u6GgC0ZCuEH8g1hzwOhrtIQbS0FVb9k3-vTUT4fS_3L_vn1UIAFkWuC
LqKnS2BYwq_d_mZmLbLYT7h_qixoR7jig3_kRhauwUkRz5iaIQkq5c5g
HdSNP5zw",
"e": "AQAB",
"d": "bWUC9B-EFRIo8kpGfh02uyGPvMNKvYWntB_ikiH9k2oT-01q_i78e
i2kpXxQ0UTEs2LsNRS-8uJbbv-A1irkwMMSK1Kj3XTgdrhCku9gRld
Y7esNA_AKZGh-Q661-4r_INLRcE8W-n234ui_qofkLnK9QWDdqaIsa-b
MwWWDsFu2MUBYywHTEzLYGq0e40nqeq1hExBTHOBdMXiuFHuq1BU
61-DqEiwXqg82Sxt2h-LMnT3046AOYJoRizo75tsU0fGCshWTBnP6u5j
d18kKhyv071hfsJdPrMs5p1i21hsFf4L_mHCuoFau7gdsPfHPxxjVOC
OpBrQzw",
"p": "3S1xq_DwTXJc6097RoXygQCAZ5nAvZ1nolYhHtnUex_fp7AZ_9nR
a07fHx--SFfGQeuta02TdjAWU4Vpk8rw9jJR0AzZ0N2fvuIAmA_WCsm
peNqNqnev1T7IyEshb8UMt-n5Cafhikzhe3smnddH6LxOrvBJLspP62v8
bU0k",
"q": "uKE2dh-chT66REFk4k4e_yj78GfPYYUIaUyosSSJuBzp3Cubk3OCqs6gT
8bR_cu0DM1jZwWmtdqDYi545HrUq3MFp15VMGN081HTe2u21mKwvq7an
V5u9hMliz7y4YKnuuWnF0zwvY89EXvR-htdQxh1sqA512b3P3fV0J
s7Fc",
"dp": "B8P9vXkxvJr2L-GyQ7v3y9r6Kw5g9SahXBSwsWUpz197TVlqI-YV85q
1Ni1rXQaD-IsXXR3-TanevrUPTRT5OB0d1MGq8pbt26gljYFKU_E9xn
-RULHz-oed9E9gLkDV4GngpF-PF_q2985xWHoJp009QF1hvChrixR
59ehik",
"dq": "CLDmD0dVhyc1c097r84rEUVn7pq6F8F3Y-iBz5xN-TpnO2F1kEPerv
AVMvEzkF41LHqqB5LMOLW8sOFbwTXYWZdM6s16c517bwQGIC3gnkJ
bi_7k_vjGtGhwXgkPaX2PnP-wzyEDEUrf-ry4c_211Cq9AgC2yeL6kdK
T1cYF8",
"qi": "3PiqvXQNOzMe-E-SBvZg1289X9CQF3VWqPzMNkqQp7_Tug06-N
ZBKCQmM3aEBGjTVsjiK-KbTR-XvaKe-72MaQj8VfBdYkssbuONKDDh
Jj-GtisaeDVW7tdHC0fwxqFUPqHt7FocRjF6h62EPMF6xmujs4qMPP
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:
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

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSIsIiwia
```

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_-oTPTS3AXP4liQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5N1KtaintoFmKZopdHM1O2U4mzwJdQx9961vp83xug117PNDi84wnB-BDkoBwA78185hX-Es4J
```

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:
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGVtLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9
.SXTigJlzIGFzZ2Vyb3VzIGJ1c2luZXNzLCBgcmlkZ2VybmJvcmVzIG5ld2VyeVwIDAQAB
.MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ551KtaiFmKZopdHM1O2U4mwzJdQx996ivp83xug117PNDi84wB-BDKoBwA78185hX-Es4J
IwmDLJK3ifersw-maT0L0nltuYoV746iYTh_qHRD68BNt1uSNCrUCTJdt5aA6x8wW1Kt9eRo4QpocSadhHxFxt8Is9uzpERV0ePPdLuW3IS_de3xyIrDaLgjluPXU
xAh6L2aXic1lu1podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Ax_fci8u9ipH84ogooee7vjUb5y18kDquDg

Figure 7: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
"payload": "SXTigJlzIGFzZ2Vyb3VzIGJ1c2luZXNzLCBgcmlkZ2VybmJvcmVzIG5ld2VyeVwIDAQAB
.MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ551KtaiFmKZopdHM1O2U4mwzJdQx996ivp83xug117PNDi84wB-BDKoBwA78185hX-Es4J
IwmDLJK3ifersw-maT0L0nltuYoV746iYTh_qHRD68BNt1uSNCrUCTJdt5aA6x8wW1Kt9eRo4QpocSadhHxFxt8Is9uzpERV0ePPdLuW3IS_de3xyIrDaLgjluPXU
xAh6L2aXic1lu1podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Ax_fci8u9ipH84ogooee7vjUb5y18kDquDg"

"signatures": [ {
  "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGVtLmJhZ2dpbnNAaG9iYm10b24uZXhhbXBsZSJ9",
  "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ551KtaiFmKZopdHM1O2U4mwzJdQx996ivp83xug117PNDi84wB-BDKoBwA78185hX-Es4J
IwmDLJK3ifersw-maT0L0nltuYoV746iYTh_qHRD68BNt1uSNCrUCTJdt5aA6x8wW1Kt9eRo4QpocSadhHxFxt8Is9uzpERV0ePPdLuW3IS_de3xyIrDaLgjluPXU
xAh6L2aXic1lu1podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Ax_fci8u9ipH84ogooee7vjUb5y18kDquDg"
}
}
```

Figure 8: JSON Serialization

### 3.2. RSA-PSS Signature

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

```
eyJhbGciOiJQUzMiM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYm10b24uZX
hhbXBsZSJ9
```

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:

```
cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
pN6-tPid6Vrk1HkgsGqDqHCDp608TTB5dDDIt11Vo6_1OLPcbUrhSMxbbXU
vadvWXzg-UDBbiireQF1fz2s2zGWvdIswbnNAUf82nyPEgVFn4422dNqiVJRmBqrYRX
e8P_1qj7p8Vdz0TRrxUeT3lm8d9shnr2lfJ88mUjvAA2Xez2M1p8cBE5awDzT
0qI0n6uiP1aCN_2_jLaesQTlqRHTfa64QOSUQFAajVKPbByi7xho0uT0cbH510a
6GYmJUAfmWjwZ6oD4ifK08DYM-X72Eaw
```

Figure 11: Signature, base64url-encoded

3.2.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 10)
o Payload content (Figure 2)

o Signature (Figure 11)

The resulting JWS object using the Compact serialization:

```
eyJhbGciOiJQUzI1NiIsIn0.
```

```
SXTigJlzIGEgZGFua2Vybi3VzIGJjIjI2luZXNzLCBgc39kbywgd292ZmlsbG9yZWQg
ZC9yc2ggQ0FjZGVkLjBhZ2dpbnNAaG9iYml0b24uXGVhY2sgQ0luc3VsdGg6IjEi
IGNvbnRlZ3MgeW91dGxldCBhbmltLTI6MjA8IEZlZGx2IGJlIG19
```

Figure 12: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFua2Vybi3VzIGJjIjI2luZXNzLCBgc39kbywgd292ZmlsbG9yZWQg
ZC9yc2ggQ0FjZGVkLjBhZ2dpbnNAaG9iYml0b24uXGVhY2sgQ0luc3VsdGg6IjEi
IGNvbnRlZ3MgeW91dGxldCBhbmltLTI6MjA8IEZlZGx2IGJlIG19",
  "signatures": [
    {
      "protected": "eyJhbGciOiJQUzI1NiIsIn0",
      "signature": "cu22eBqkYDKgI1TzpZXGvaFFz6WGoz7fUDcft0kkOy42miAh2qyBzkxESnk2I
pN6-tPld6Vrk1HkqsGqDqHdCP608TTB5dDDIt11o6_1OLpcbuhrHiUSMbXbXU
vdvWxg-UD8iiReQ1fzf82zGWVsiNAUf8znyPegVFn442NdNqiVJRmBqrYRX

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"

```json
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x": "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmG2x6sY1_oJXu9A5RkTKqjqvjyekWF-7ytDyRXygCF5cj0Kt",
  "y": "AdymlHvOiLxXkEhayXQmNXcV4h9htZaCjN34kfmC6pV5ohQHiraVysSudaQkAgDPrwQrJmbnX9cw1Gfp-HqHZR1",
  "d": "AAhRON2r9ecqXX1hg-RoI6R1tX5p2rUAYdmpH2oC1XNM56KtscrX6zbKipQrCW9CGZH3T4ubpnoTKLDBGJj_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.

```json
{
  "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_YZCChjn4791jSQCrdfPZCNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvb
u9P1on7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDDyYjnAMDxXPN7XrT0lw-kv
AD890j18e2puQens_IKEBpHABlsbEPX6sFY80cGDqoRuBomu9xQ2

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:

eyJhbGciOiJFUzUzIzQxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hbXbsZSJ9.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gYW5kIG1lZGluc3V0IHN0cmluZyB0aGUgYmFzZTY0IG1pbmN0
b3IgZG9jZS9ib3N0IHN0eWxlLmNyb3NzNG90b2R1dGluZ3MgZGlzdG9yeSB5
b3JkcyByZXRlciB0byB0aGUgY29udGVudCBvdGFnZS90cmluZw.
AE_R_YZCChjn4791jSQCrdfPZCNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvb
u9P1on7KRTzoNEuT4Va2cmLleJAQy3mtPBu_u_sDDyYjnAMDxXPN7XrT0lw-kv
AD890j18e2puQens_IKEBpHABlsbEPX6sFY80cGDqoRuBomu9xQ2

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 "$HS256$".

```json
{
  "kty": "oct",
  "kid": "018ca0ae-5-4d9b-471b-bfd6-eef314bc7037",
  "use": "sig",
  "k": "hJtXI2uSN5kbQfRbtTNWbdhmhkV8FJG-Onbc6mxCcYg"
}
```

Figure 20: AES 256-bit symmetric key
### 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.

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

**Figure 21: Protected JWS Header JSON**

![Figure 22: Protected JWS Header, base64url-encoded](image)

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:
The resulting JWS object using the JSON serialization:

```json
{
  "payload": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkgfBBBkLspW1h84VsJZFTsPPqMDA7gL1Md7p"
    }
  ]
}
```

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

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWRlZjMxNGJjNzAxNyJ9
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:

```
s0h6KThzkfBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

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:
The resulting JWS object using the JSON serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LTWlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
    }
  ]
}
```

Figure 29: 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.
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:

```
bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20
```

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.
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:

xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvxhJxZuk

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": "SXTigJlzIGEgZGFuZ2Vybj3VzIGJ1c21uZXNzLCBGcm9kbywgZ29pbmcb3V0IHlvdXIgZG9vc2UgWW91c2VydmljZSByb29pZCBrZWVwIHlvdXIgZmVldCwgdGhlcm9sZSB0byB0byBwcm9kdWN0"
  "signatures": [
    {
      "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvxhJxZuk"
    }
  ]
}
```

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.

```json
{
  "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_Nq2YH4FGhiocsrqrgi5NvyG53uomic1tcMdSg-qptrrzC7CG6Sw2Y13TD1qHzTUrL_Lrz2FcryNFIhKSw129EghGpwkpxaTn_THJTCglnBAdk01MZBcdwzJxwq2c-1rLpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqytkSeAMbhMBdMma622_BG5t4sdbuChTfpj9iJmkio47AIwqkZV1aIZsv33uPUqBjBCXbYoQJw7mxPftHmNLgoOSMxR_3thmXTCm4US-xIN0yhbmb8afKK64jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOn1C9w

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_Nq2YH4FGhiocsrqrgi5NvyG53uomic1tcMdSg-qptrrzC7CG6Sw2Y13TD1qHzTUrL_Lrz2FcryNFIhKSw129EghGpwkpxaTn_THJTCglnBAdk01MZBcdwzJxwq2c-1rLpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqytkSeAMbhMBdMma622_BG5t4sdbuChTfpj9iJmkio47AIwqkZV1aIZsv33uPUqBjBCXbYoQJw7mxPftHmNLgoOSMxR_3thmXTCm4US-xIN0yhbmb8afKK64jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOn1C9w"
}

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-8t5BREVAuTY8n8YHjwDO1muhcdCoF2FFjfISu0CdKn9Yb
dlmi54ho0x924DUz8sK7Zxkhc7AFM8ObLfTvNCRqC13Jk12U5IX3utNh0DH6v7
xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxXm4fgV3q7ZYhm5eD

Figure 44: Signature #2, base64url-encoded

The following is the assembled second signature serialized as JSON:

```
{
    "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
    },
    "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhcdCoF2FFjfISu0CdKn9Ybdlmi54ho0x924DUz8sK7Zxkhc7AFM8ObLfTvNCRqC13Jk12U5IX3utNh0DH6v7xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxXm4fgV3q7ZYhm5eD"
}
```

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.

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

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:

```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:
Figure 50: 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
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".
Internet-Draft                JOSE Cookbook                   April 2014

{ "kty": "RSA",
 "kid": "frodo.baggins@hobbiton.example",
 "use": "enc",
 "n": "maxhbsmBtdQ3CNrKvprUE6n9lYcregDMLYNeTAWCj88NnPU9XYeqT
HvHqjxKDSHP21-W5nSsppGlwgdAq2yhnWvXhYNvcM7Tf9fgKxqKx
6f3yy7s-9McPSNZwFCzH6eAKr4U0hV9r1ypMP9l4IB09t5S9W5U
NwAHsIrd-o-OGPjElIIdcHtwx-2THu3C60Pu-_LJI6hK9wbaUm6A4c
R5Bd2pbgbaY7AsqjCUBtYJaaaNIHo5xHPxUrDZKumaZVW0OKPFa60P14oy
pBjadjvM24AZj3NxaSYsEZhaueTXvZB4eZOAjtIyh2e_VOIYKmsDrJYA
VotGlvMq",
 "e": "AQAB",
 "d": "Kn9tgoHFtIV8uPu5b9TnwyHwG5d6tEOuFdpLCGnJ72E19637wy
bQ1PLAhmpbNTxtrhocoAniVR1NCiQXaW_8s461lIDpyntEPngcKsyo
5jMAj17-CL8vhpYowNFVesgMoVaPRMYT9T63hNM0aWs7USZ_hL9g
0e1mYvHrH3FucjSM86Nnf4oIEN43r2fspqEPGRrdE6fpLc99aq-geP
1GFUlimRdndm-P8q8kVn3KEl1NATeEgrQAdTtgz80S-3VD0FgWfGbnP
miuPUSo8oPb9kDIFu觚cc6fgl4n3aJje66ESvGPH2afHqSy_Fd2v
p7y8bQQ",
 "p": "2DwQmZ4FoTnQ81Kej3BmKrFs5Eh2mizZA5eEJ2MnUE3sdtYKSLtaE
oeekx9vbB2wXhVn60UMKCI_2INK820ayLYHL0_G21aXf9-1unyEU
918HTk1LpYA0Ox12Gv1joxAdWN3hiEFrj2LZGS710H-a3Q1DDQoJOJ
2FpmU",
 "q": "te8LY4-W7tyaqHlEuxjMgktAlTeRb0VQLnFLy2INKrroWdi93_V
F099aP1ESeLjaZn-8ihZte-q7mCpoZKfVtUyfZ5HRJ_XY2kfeXJhb
91hZHMMv5pIskZpE-GPHCB6gRkl0q-1dn_qxysufW7WAX1sVFQfK8
d6Et0",
 "dp": "UyYkLpLr49vVc0PzwLSp1bg4L3-25lW48m1swbzp0YIgd2x2HTH
QmjJpFALQ2q-zf9RmgXJkDrFs9rkdxtPaStLWYdeCT5c125Fkgd317J
RDo1inX7x2Rdh8ERCReW8_4ZxItuT1_KizXN51YvMQjBw2wT1lpsf
lo0ryU",
 "dq": "1EgqO-QfpepH88Wd7mUFeYxdn0KXJBGcGhY6YKuIHGC_p8Le9Mb
pFKESzEalLN1Ehe3B6ObG15z_ayUl2j2ioQZ82znoUpa9fYVino87A
CfZlG7q9Mv7RiPaDzerI03tkVXAdaBau_9vs5rS-7HMtxkVrxSUvJY14
TkX1HE",
 "qi": "Kc-1zZQqxFa2Cr510tV0REkVqAYhqiIRGL-MxS4mCrmx5mVz
1LXy66tEl1_Aagjgaj1kkjxgLXTHHd81ga6f0GMBama5ruR1hGqSpC7
G17CFID2bBJMTQN6EshYzZfxW0d8mIO8M6Rzh0beL6fF9mkDcyPrBxx
2bQ_mM"
}

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-_BgL0XMoZKxmy9gfFy2gTdvqzfTihJBluzxg0V7yk1WC1nQePFvG2K-pvSlWc9BR1azDrn50RcRai_37D0N395H3c62tiouJJ4XaRvYHFjZTZZG XFz8YAImcc91Tfk0WXc2F5Xbb71C1Q1DDH151tIpH77f2ff7xi6zh9oSewYrcG TSLUeeCt36r1Kt30Sj7EyBQXoZ1IN7IxbhMAFgile7Mv1r0I0I18NQeXXW8V1 znmoxaGMny3YNGr5Wf6Q2nBq4DpRmAmuuGUGEeCe1I01wx1BpyfgvfvOh MBs9M8XL223Fg47x1GsmXdfuY-4jagVw`

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.

4.1.5. Output Results

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:

eyJhbGciOiJSU0ExLiJ9.eyJpc3MiOiJDb250ZW50IiwiaWQiOiJdb250ZW50cmVhdCIgY3Jvc3QgY29udGV4dCBwYXNmLiIsImZhbWVudF90aW1lIjoiMTA5MzQwNzgiLCJzdG9tIjoiUmVsdGFuZ2VycyIsInN1YiI6IjAyMjA5MjE4OTkifQ.GaS_dT6Qp29IcIu961gPvT9JkUv4cEFPpPa5bZ4GZV4

Figure 60: Compact Serialization

The resulting JWE object using the JSON serialization:


```json
{
  "recipients": [
    {
      "encrypted_key": "laLxI0j-nLH__BgLOXozKxmy9gffy2TdqvzfTihJBuzxqGV7yk1WClnQePFvG2K-pvSlWc9BRIazDrn50RcRai__3TDON3953c62rIoUJj4xaRYHIfjZTzZ2GXfz8YAImcc91TfkJ0WX
C2F5Xbb71C1Q1DDH151t1pH77f2ff7xiSxh9oSe6YrcGTSUUeeCt
36l1Kt30Sj7EyBQXo21N7IxbyhMAgFe77Mv1rOT01518Nq6exW8
V1zNmoxaGMny3YnGir5w6Qt2nBq4qDaPdnaAuUGEce1I01wx
1BpyIfgyvfjOhMBs9M8XL223Fg47x1GsMXduY-4jaqVw"
    }
  ],
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYMJdG9uLmV4YW1wbGU1CJ1bmMiOiJbMTI4Q0JDLUhTMju2In
0",
  "iv": "bbd5sTkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSfXLiyDHaA2DxUjD67ieF7fcVBIR62
JhJvGZ4_FNV5iGC_raaOnHnLQ6s1P2sv3Xz11l1p1l_05wR_RsSrSr8Z-wn
I3Jvo0mpkEEnLmZvDu_k80WzJv7eZVEqi1WkdYVzPlipyQU28GLOpRc
2VbhK4dQKpDNTjPPEmRqcaGtWzVyeSvF5k59yJZXsuSVwfwK68KtNtm
RdZ8R4mdOjHsMrM_8suw1Fcqt4r5GX8TKa1OzT5CBL51q3Sc7u_hogy
KVoiiRytEAEs3v2kcfLkp6nBxQc_PkmKsNo-ohP78T206_7uInMGhFeX4c
tHG7Ve1HG1T93JfWDEQ15_V9UN1rhXNyYu-0fVMkZAKX3Wl7zASBP4
30m",
  "tag": "kvKFbXHe5mQr4lqgobA Ug"
}
```

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": "wbdxI55VaanZXPY29Lg5hdmv2XhqvAhoxUkanzfz2-5zVUxa6prHRrI4pP1AhoqJR1ZfytWwd5mmnHRGp2Ah1l0ySj9w10Bioz2B1LX2Pe-C-FyXJCgy7Y9HdKQWrlrHm42Ew7v0v0r4gfa06uxjlGwfpGrZLarchiWCPnkNrg71S2CnUSZQBP1GJxFkmYt2t1WVgGnL22Gp1yXj5y1BLdxXp3Xe7t5sq571UtNfOuUTU8E4qdzJ3UIDtoVkpGsWmlmmnJiw7aXRTIC1vo4Rm5qnt2dw-7w4vuR4779ubDjuJna1mlVs666-RPcnAafWskxtBnFJJDGIU17Etsizj1vmns0X_pubP0U1Wn0ec85FCft1hACpW8g8crrBoNeqB0HFDsFyYpuCULC5aZTaPFw2ADi7g1TtsC_FupfQzKnGec1lpxrcKhCVWYQb6B-HozjHczqtauuFz95v5bTuB-TpkevJFncwFLH3b8mb-H-oxx35FqjBSA1jKyoeqKTPvhtxdO9nxwggJ6Vkv6UC418-To1jMVdFwTXUxln1hfnOzonW6hsSzD1c9wrcVuzsUmV5szidq9w1fcYWf3g5qFdxQk1s99gqDaicAwm3YEBizuNeeC5ardtHDb1xEB_HcHSeYbghbMjFgasvKn0aRsznTYx0chxWB1scl2E",
"e": "AQAB",
"alg": "RSA-OAEP",

d": "n7fzJc3XQ59V5E0BTKayzuSMM7800JQzuJn_KbH810Z2G520oA74Bxcc0xQn5zE5uSCIw9g910Ct0jxxPcpmqzaJZglnirjczW-oBtVkgc74wCaGwB3ghfFPziBbksorzJhajjcyY3HBBhys4_werrXg4MDNE4HYojy6LctxEL1YQxUOCF5t7jxVxwol1x15GcvQvNdRtxUECwiewfmmrfveEogLw9E6A-KMgajTiISSxqIXqHwUXQ17G7v_mV_Hr2YuImYcNhcKrvp9E7kook87Deh0V4U0ZLwa10lUxmcqow58A_y21BybVx1_s5lPsqsbbh-nqJh11L0gndFhf1LxnciWtpC7znLnmZyeCWAG7Zlfv-Rn9fliV9j2R7r-MSH9sbQuuziH1Nwgsrge-jjRl1uHMA014svFkl6bcq1j1wXVhZn2Nz0oRFD-L1lqQyUSpF6x3aK0dKqiBriqeE6EvLuS1YDapJq3jdIsqoL8mo11oog7tjxUW1qGWEOQwgu82plzym-90Q03hyEnfUhsR8aJQAWAFhW5MIW5Q_I QT917-yindr_2fWQ_i1UgMsGzA7aOGzzlPkJry6z-t_Y_Kubg00-Z8S_aWujyUc-Alp8AAYqyjK7B-7CW3H22FeWK48jl7-zomrwljr_mnhsPsB0c9WwsgIR-I-KgE", 
"p": "7_2v3OQzzIFCwHyfylABq3XP85Es4hCdKbD4taUXgV9y91etKghv4Mhrrk0vbv01kyUvlFMxKcdtpi-zLCLYadXKRkA3FptSbtZzQD_X2nlsY4a_aQ2wPbXb_RtfIvf1dKUmd49pHuhFGF7j7nSNXfpiHSHFWEzAzC3Mym46J96Y12lRnRweVwAGnWn53p07D8y7y_92pda97vqC2odtyb9Hg9uxa-RFN0h1ai0cHyjz69hjmMrXx-x56hO9cnXNbmzNCFCKnqGmQ4QjLRmRr9sb2ZbrQ9LhbbstE4_yE_0zrF2p08nROvJ14RqXNwYi85fc6BRgBJombt8QDdvGvQjPv4C5Wvhoq",

"q": "zqOHk1P6WN_rHuM7ZF1cXh0X6RuOgH67WuuHiSknoqeefGBa9Pws62YKQCO-6mKXtcqE8_q_Hak2MrKcOcvH11hqMCNCS1f7M7WPRPzU2qCdCqgssd_umBP-Dqytht_EzwL9KnYoH7JQFXxmv5aP8c8UlTtwk4knKj1YGRuUwFtUsw1nffjFxyO01AQ37ussIcE6C67sM3n41UibJ7TCqewzvJJpaJN5cxjySP2K3vP01aY9gAd6a3i1AkJd1XJS1ImnFpevS5JQBE79-EXe2KSwVoGozt-gosmM29Q8veHy4uAca5dZzmH7kkHtw1z0-jHv90epQJj1XNNH8Q",

dp": "19mBDhB1xelM1QFzmsZwZTqUahAzC14xNIGEPN0dt1jxK83_FJA-xx5kA7-1erdHdsm_EF67HisONNV5a60JJa78LHnDiBnJdaUmm0u8AXAQJlaM5jxNjS6E2yD44US02jMhVzeeNCzqz25elqBPlHUpGo1lZuG72F"
Figure 62: 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.)

4.2.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption CEK (CEK); this example uses the key from Figure 63.
- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 64.

mYMfsggkTAm0Tbvt1Fh2hyoXnbEzJQjMxmgLN3d8xA

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:

```
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi beYO155_PJsAX2w5XtDePz9hk-BbtsTBq2UsPOdwjC9NhNupNNu9uHIVftDyu cvI6hvALe260GnhNV4v1zx2k701D89mAzfw=_kT3tkuorpDU-CpBENfIHX1Q58 -Aad3FzMuo3F9nuEP2yXakLXYa15BUXQsupM4A1GD4_H4bD7V3u9h8Gkg8Bpx KdUV9ScfJQTCYm6eJEBz3aSwIaK4T3-dwWpuB0hRQXBosJzS1asnuHTVMt2pK IIIfux5BC6huIVmY7kv7V7aIUrpyM_3H4zYvyMeq5pGqFmW2k8z0p8078TRlZ7 pZfYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sgthXn9uDqd6wycMagnaQ f0Tsl_yctWmY-aqWVDKhrjYNRf03Niwr55BE-tOdFwCASQj3uuAgFGr02AWBe3 8UjQb01vXn1SpyvYZ3WFC7W0JYa7a8DRn6MC6T-xDMuxCOG7S2rscw51QU 06MV1TFot0VufKBa03cxA_nIB1hLmjY2kOTxQMmpDFTr6Cbo8aKaOnx6ASE5 Jx9paBpnNm00KH35j_Q1rQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzn3woE12ozDRs
```

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.

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

Figure 66: Protected JWE Header JSON

```
eyJhbGciOiJSU0EtT0FFUCIhSmtp2C16InNhvxDpc2U22Ft22V1QghvYmJpdG 9uLmV4Y1wbGULJC1bmMiOijBMU2RDNIn0
```

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.

Figure 68: Ciphertext, base64url-encoded
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:
eyJhbGciOiJUSSU0E[...]

Figure 70: Compact Serialization

The resulting JWE object using the JSON serialization:


```json
{
  "recipients": [
    {
      "encrypted_key": "rT99rwrBTbTI7IJM8fU3Eli17226HEB7IChCxNu
h71Ciud48lxeoiDrdtFF4nzQieYo1s5_PJsAXZwSxtDePz9hk-Bb
tsTBqC2UsPo9w1JNhupNNU9uHIVtDyucvI6hvALEZ6OOnhNV4
v1zxz2k701D93mAfzfw-_kT3tkuurpDu-CpBENIIH1Q58-Aad3FzMo
uo3Fr9buE2pYxakLXya15BUXQsupMA1GD4_H48d7Tv3u9hBGkftgB
pxKdU9V9cWjQTCy6mEJEB3aSwIAk4J3-dwWpuOhoR0QXBozsz1
asnuHtVM7z7ixw5BC6huivmY7kzV7W7aUIrpYm_3H4zYVymEq
5pGqFmW2k8zpoO87TR1z7p2fPYDSXZyS0CfKkkMoz7_qiCwZTSz
4Duy7n8hS429sGthX9n9DqD6wycMgnqOFyTSs_lycTwmY-agWVdK
hjYNRF03niwRtb5BE-t0dFwCASQjo3uuAgPrO2AWBe38UjQbo1lVx
1SpyvY23WFC7W0JYTa7A8DRn6MC6T-xDmMuxC0G7S2rscw51QQU
06MvZ1IPot0VufKba03cxA_nIB1h1Mjy2kOTxQMmpDPR6Cbo8a
KaOnx6A5E5Jx9paBnMoOKH35j_Q1rQhDWUN6A2GgFayJ69xK
EdHAVCGRzn3woeI2ozDRs"
    }
  ],
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2
FTzZ2VLQGhvYmJpdg9uLmV4YWlwbGUiLCJ1bmMiOiJBMjU2R0NNIn0",
  "iv": "-nBoKLHYKLPS39",
  "ciphertext": "o4k2cnGN8rSSw3IDolYuyskGwS_t2mL0k15ggBdpACm6
UUJw0oHC5ytqjgqRj-30P1wQVu4ugRWedwAGNw6vGW-xyM01TYx
rXVfIaDhYtETMRBvBWBwEF7uai1DRFva0jgZv6Ifa3brcAM64d8p51h
hNczPersuhn5F-pGyzseva-TUaL81wntct-cSwy75QmKheJwbD6fz
6kFovEgj64X1Is7E6GlpU5gfnBY1L1aQ14ML7Cc2GxvgJ7z9wQ0YIEc7a
Cf1L1G1-8BboWfzdKLLv50YocrYHumwzKLuIWEbSfnPpOsLY2n525Dx
DfWaVFUQkxMFE5vavnf4B8QmpWAbnypNimbM8zV0w",
  "tag": "UCGIqJxhBI31FVDpAlHHvA"
}
```

Figure 71: JSON Serialization

### 4.3. Key Wrap using PBES2-AES-KeyWrap with AES-CBC-HMAC-SHA2

The example illustrates encrypting content using the "PBES2-AES-KeyWrap" (PBES2 Password-based Encryption using HMAC-SHA-512 and AES-256) key encryption algorithm with the "A128CBC-HS256" (AES-256) 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": "GZy6sIz6w19NJ0KB-jnmVQ"
   },
   {
      "kty": "oct",
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
      "use": "enc",
      "alg": "A256GCMKW",
      "k": "qC571_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
   }
   ]
  }

Figure 72: Plaintext Content

entrap_o_peter_long_credit_tun

Figure 73: Password

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.
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.
### 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:

```
eyJhbGciOiJQQkVTMiiIUziIIIniTBMTI4S1cSIFJwMn5i5i4UTFTEmluUXNSM3
hjaF16N1paY0hBIiwcDJiyo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJi
bMMoiJJBMTI4Q0JDLUhTMju2In0.
```

![Figure 82: Compact Serialization](image)

The resulting JWE object using the JSON serialization:

```
{...
}
```

![Figure 82: JSON Serialization](image)

The resulting JWE object using the JSON serialization:
{  
  "recipients": [  
    {  
      "encrypted_key": "YKbKLsEoyw_JoNvhtuo9aaeRNSEhhAW2OVhcF_HLqS0n6hA_fgCA"  
    }  
  ],  
  "protected": "eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJwMnMiOiI4UTFTemluYXNSM3hjaFl6N1paY0hB1iwicDJjIjo4NTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJlbmMiOiJBMlBMTI4Q0JDLUhTMjU2In0",  
  "iv": "VBiCzVHN0LiR3F4V8uoTQ",  
  "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjrHXYsTHAHLZ2nGIX86vMXqi61RslywCRFzIXcZBRnTvG3nhhzFk0GDD7FMyXhUHpDjEYCNA_X0mzg8yZ9oyjo61TF61is149FZZZh2EhzgFQCLQ_6h5EVg3vR75_hkBsnuqcjtM3dwejXbTcOdN84PeqM6asmas_dpszz7H10f5ci911zg42 4givB1YLdf6exVmL93R3f0oOJbmk2GBQZL_SEG11v2cQsBgeprARsaQ7Bq99T80coH8ItBgV08AtxXFFsx9qKvC982KLKdpQM1VJKkqtVR4u5LEVpBZXBnZrtViS0qgg6Aiwsr-rCrCd_ePQSUxvgtrokAKYPqmXUeRdjFJwafkYEkiuDCV9vWGAi1DH2xTaffJwcmwywIyzi48qRpmdN_N-z15 tuJYuvKvKv6ihbsV_khJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdT w8V3kobXZ77ulMwDs4p",  
  "tag": "ALTKxvwAefEL-32NY7eTAq"  
}

Figure 83: 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"
{  
"kty": "EC",
"kid": "peregrin.took@tuckborough.example",
"use": "enc",
"crv": "P-384",
"x": "YU4rRUzdmVqmRtWOs2OpDE_T5fsNIodcG8G5FWPrTPMyxpzSOGaQLpe2FpxBmu2",
"y": "A8-yxChXkFb3hKZfI1jUYMjUhsEveZ9THuwFjH2sCNdtksRU7D5-SkgfL1Etp",
"d": "1Tx2pk7wW-GqJkHcEkFQb2EFyYcO7RugmaW3mRrQvAOUipommT0IdnYK2xD1Zh-j"
}

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-G2zVqgzUtznW_

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.

```
{
  "alg": "ECDH-ES+A128KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "uBo4kHPw6kbjx510xowrd_oYzBmaz-GKFZu4xAFkbyiWgutEK6iuE
    DsQ6wNdNg3",
    "y": "sp3p5SGhZVC2faXumI-e9JU2Mo8KpoYrFD5yPNVtW4PgEwZ0yQT-
    JdaY8tb7E0",
    "d": "D5H4Y_5PSKZvhfVFbcCYJOtG5ygRgfZkpBr591cmme9sw6nkZ8W
    fwhinUNfWJg"
  },
  "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/nonce (Figure 86); and
- Protected JWE header (Figure 90) as authenticated data

produces the following:

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

4.4.5. Output Results

The following compose the resulting JWE object:

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

eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVnclumLnRvb2tAdH
Vja2Jvcm9zL2gZxhcbXBsZSI6ImVwayI6eyJrdHkiOiJgYIsImNydiI6I1At
Mzg0IwieCI6InVcbzRrSFB3Nmtiang1bDB4b3dyZF9vWXpCbWF6LUdLRlp1NH
hBRkZry1lPV2d1dEVLNml1URZzUTZ3TmROZzMiLCJ5Ijoic3Azic3zcvTR2haV
MyZmFYdW1lWU51UyTW84S3BvWXJGRHI1eVBOVnRXNFbnRXdaT31RVEEt5SmRh
WT
h0YjdFMcj9LCJ1bmmOviJBMH4R0NNIn0
.
0DJjBXri_kBc46IkU5_Jk9BqaQeHdv2
.
mH-G2zVqgztUtnW_
.
WJRCcjSpaAUZ5dOR3Spy7QuEkMxkx8-3RCMhSYMzsaEwdctxa9Mn5B7cCBojK80
IgEnj_qfo1h1i-uEkUpOZ8aLTZHfpL05jMwbKkTe2yK3mjF6SBAsgicDVCCc
Y9BLuxz1Rmc3ORXaM0jaHPB93YccSDGpgBWVMrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXIuEr2hWgeP-nkU2TPU9EoGSPj6FAS-bSz87RCPxZdj_iVyc6QWcqAu
07WNhjzJEPc4jVntRJ6k53NgPQ5p99132408OUqj4ioYezbS6vTP1Q
.
WuGzxmc4YjyHGPoa17EBg

Figure 93: Compact Serialization

The resulting JWE object using the 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"
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.

yc9N8v5sYyv3iGQT926IUg

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.

{ "kty": "EC", "crv": "P-256", "x": "mPUKT_bAWGH1hg0TpjjqVsP1rXWQu_vwVOHHTnKdYoA", "y": "BBQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv41rs", "d": "AtH35vJsQ9SGjYfoSjUxYXQKrPH3Fj2HmEtSKoSN8cM" }

Figure 97: Ephemeral public key, in JWK format
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.

```json
{
  "alg": "ECDH-ES",
  "kid": "meriadoc.brandybuck@buckland.example",
  "epk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "mPUKT_hAWGH1hg0TpjggVsPlrXWQu_ywV0HTNkdYnA",
    "y": "8BQAisImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs"
  },
  "enc": "A128CBC-HS256"
}
```

Figure 99: Protected JWE Header JSON

eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidWNRbGFuZC5leGFtcGxlIiwidXN0b21lIjoiMTQ5MjAyMkI4NzU5NTY1Ljc4ODE2NDE5ZDc4MzgzNzktZDI2MDAtY2VlMjI1YS0gZDQ5MDY1ZjViMjQ0NGQ4NzQ4NjAyZjYiLCJleHAiOjE1NTEyMzk5NjQ1NTUzMjA2MjYxLCJpYi0iMzEiLCJzY3ZhIjoiMTg0MzUyNzE0NTc5ZDUzN2Y5MDcxNmQ5Mzc4MzI2YmZlNjE0ZDk0ZjIiLCJwIjoiMTY3NzY0OTUwMDYxODQ2MzQwMjUwNjI2NjAzMjE2MjUuNTIiLCJfIjoiZCIsInR5cCI6MX0=

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:
the resulting JWE object using the JSON serialization:

```json
{
  "protected": "eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmr5YnVja0BidWNrbGFuZC5leGFtcGxlIiwiZXJ2Ijoia2lwYW5nY2xlbWF0aXJvbGlkIiwiZXJ2IjoiUC0yNTYiLCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqanFWclAxc1hUXVFvmdnWT0hIdE5rZFlvQSIsInkiOiI4QlFBc0ltR2VBUzQ2ZnlXdzVNaflmRIUME1qQnBgzJTUzM0RHY0SXJzIn0sImVuYyI6IkExMjhDQkMtSFMyNTYifQ",
  "iv": "yc9N8v5sYyv3lQQT9261Ug",
  "ciphertext": "BoDlwPnTypYq-ivjmQvAYJLb5Q6L-F3LIGqomlz87yW4OPKbWE1zSTEFjDfhU9IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRMEmSDiAYtskTTmmzNa-_q4F_evAPUmwl0-ZG45Mnq4uhM1fm_D9rBtWolqZSF3xGNNkpOMQKR1C1I8wzjRli7-IxgyirlKQsbhqRzvkV8IcY6aH124j03C-AR2le1r7URUhrArM79BY8soZ0U1zW1-sD5PZ314NDCcei9XkoIAsXJWmySPErBi5UZL4mYpvdKDiwymyzGd65KqW7MsFfI_K767G9C9Azp7zKZD0DYUn1mnOWW5LmyX_yJ-3AROq8p1W2BF-GzyJ6195_JGG2m9Csg",
  "tag": "WCckNa-x4BeB9hIDIFhuhg"
}
```

Figure 103: Compact Serialization

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"

```json
{
  "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.

```text
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.
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.

The following compose the resulting JWE object:

- Protected JWE header (Figure 108)

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:

```
eyJhbGciOiJkaXIiLCJraWRQOkIiN2Z0ZTJiOC02ZzBTZzLTQ1Y2YtODY3Mi02MTEiNWI0NTI0M2IEJllbMiO1jBMTI4R0NNIn0
```

refa467QzzKx6QAB

```
JW_i_f52hww_ELQPGaYyeAB6HYGcR55919TYnSovc23XJoBcW29rHP8yZ0Z7Y
hLpTibfjFuvZkJOS-m0IFtvXkXZH_lFr_dYt9HRUYkshrtrMmIUAYGmUnd9zM
DB2n0cRDIAZrFeJUDxUwVA7E-YGRPdcqMiyboCO-FbDE-Ncebh43-FtBP-c_
BiWCPtb9o0sbdcdREEMJMyZB8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxscj5
J-NJsUpbk29-s7LJAGb15wEBxtPhVCGyy53coIKLHHeJHXex45U9aKZSRSIn
ZI-wjS0yu3cT4_aQ3ilo-tIE-F8Io61EKgyIQ4CWao8PFMj8TTnp
```

Figure 111: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWRQOkIiN2Z0ZTJiOC02ZzBTZzLTQ1Y2YtODY3Mi02MTEiNWI0NTI0M2IEJllbMiO1jBMTI4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcR55919TYnSovc23XJoBcW29rHP8yZ0Z7Y
hLpTibfjFuvZkJOS-m0IFtvXkXZH_lFr_dYt9HRUYkshrtrMmIUAYGmUnd9zM
DB2n0cRDIAZrFeJUDxUwVA7E-YGRPdcqMiyboCO-FbDE-Ncebh43-FtBP-c_
BiWCPtb9o0sbdcdREEMJMyZB8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxscj5
J-NJsUpbk29-s7LJAGb15wEBxtPhVCGyy53coIKLHHeJHXex45U9aKZSRSIn
ZI-wjS0yu3cT4_aQ3ilo-tIE-F8Io61EKgyIQ4CWao8PFMj8TTnp",
  "tag": "vvb32Xv1le2atmHAdccRQ"
}
```

Figure 112: JSON Serialization

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": "qC57l_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
}
```

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.

```bash
UWxARpat23nL9ReIj4WG3D1ee9I4r-Mv5QLuFXdy_rE
```

Figure 114: Content Encryption Key, base64url-encoded

```bash
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.

```
{
   "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.

The following compose the resulting JWE object:

- Protected JWE header (Figure 120)
- encrypted key (Figure 117)
- Initialization vector/nonce (Figure 115)
- Ciphertext (Figure 121)
The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZGFnMDMyMWMiLCJraWQiOiJrZlBkdVZRM1QzSD22bmV3dC0ta3N3
liwiaXYiOiJLa1lUMEdyXzJqSGxmcU5fliiwI2W5jIjoiQTEyOENCQy1IUzI1Ni
J9.
```

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/nnonce; this example uses the initialization vector/nnonce from Figure 127.

aY5_Ghmk9KxWPBlu_glx1w

Figure 126: Content Encryption Key, base64url-encoded

Qx0pmsDa8KnJc9Jo

Figure 127: Initialization Vector, base64url-encoded

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:

CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx

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

- CEK (Figure 126);
- Initialization vector/nonce (Figure 127); and
- Protected JWE header (Figure 130) as authenticated data

produces the following:

- Ciphertext from Figure 131.
- Authentication tag from Figure 132.

And authentication tag:

ER7MWJ21FBI_NKvn7ZblLw

Figure 132: Authentication Tag, base64url-encoded

4.8.5. Output Results

The following compose the resulting JWE object:

- 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:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0
.
CBI6oDw8MydIx1IBntf_lQcw2MmJIKQx
.
Qx0pmsDa8KnJc9Jo
.
AwII-P-KmWgsZ37ByzCefNen6VTbRK3QMA4TkvRkH0tP1bTdhFJgJxeVmJkLD6
1AIhnWGetdg11c9ADsnWgL56NywxSYjU1ZEHCgkd3EkU0vjHi9gT1b90qSYFfe
F0LwkcTtjbYKcsiNJQkcIp1yeM03OmuiYS0JYVSpf7ej6zaYcMv3WwdxDF18RE
wOhNImk2X1d2JXq6BR53TSFkT7PwVLuq-1GwtGHLQeg79DT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJfqqFvETUxFrAYF-LkjcBTS5dNy6egwkyTbOt8EIHK-oEsKytZRa
a8Z7MOZ7UGxGIMvEmxrGCFeJa14slv2-gaqK0kETHkaSqdYw0FKQ2F
.
ER7MWJZ1FBI_NKvn7Zb1Lw

Figure 133: Compact Serialization

The resulting JWE object using the JSON serialization:
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:
Compressed plaintext from the original plaintext content; compressing Figure 51 using the DEFLATE [RFC1951] algorithm produces the compressed plaintext from Figure 135.

AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 136.

Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 137.

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

```
5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi
```

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.

```
bY_BDcIwDEVX-QNU3QEOrIA4pq15okYxchxVvbEDGziJbicoSjwc-f___HPjBu
8KVFPvUa1VE1-wZo0YjNz3C7R5v72pV5f5X82VWvjYQpqZKjyjiZ0r2B7kQ
PSy6oZIXUnDYbVKN4jNXi2u0yB7t1qSHTjnmOMDf9qvrDzfTIQnyQRuUya4zI
WG3vTODir0v7BRHFYWyq3k1k1A_gSDJqtcBF-GZxw8
```

Figure 135: Compressed Plaintext, base64url-encoded

```
hC-MpLZSuwWv8sexS6ydfw
```

Figure 136: Content Encryption Key, base64url-encoded

```
p9pUq6XHY0jfEZI1
```

Figure 137: Initialization Vector, 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:

VILuUwuIxaLvhm5X-T7kmA

4.9.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 140)
The resulting JWE object using the Compact serialization:

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwImlwIjoiREVGIn0.

5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi

p9pUq6XHYojfEZI1

HbDtOsdailoYziSx25KEEtXwRh8l8jKMFNc1k3zmM16VB8hry57tD261jXyexSPt0fdLVfe6Jf5y5-JaCap_JQbc5opbmT60uWg18lyiMq0n9j--Xhh1Yg0
m-BHagfdO5iTOWxPxFMJedv7Wy8mgDHj0aBMB6152PsM-w5E_o2B3jDbryBk
hpyA7qi3AyijnC7BP9rr3U8xxExCpG3mK420Tjow

VILuUwuIxaLVmh5X-T7kmA

Figure 143: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "5vUT2WOtQxKWcekM_IzVQwkGgz1FDwPi"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwImlwIjoiREVGIn0",
  "iv": "p9pUq6XHYojfEZI1",
  "ciphertext": "HbDtOsdailoYziSx25KEEtXwRh8l8jKMFNc1k3zmM16VB8hry57tD261jXyexSPt0fdLVfe6Jf5y5-JaCap_JQbc5opbmT60uWg18lyiMq0n9j--Xhh1Yg0m-BHagfdO5iTOWxPxFMJedv7Wy8mgDHj0aBMB6152PsM-w5E_o2B3jDbryBk
hpyA7qi3AyijnC7BP9rr3U8xxExCpG3mK420Tjow",
  "tag": "VILuUwuIxaLVmh5X-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
{
  "vcard",
  {
    ["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.
o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 147.

o Encoded additional authenticated data (AAD); this example uses the additional authenticated data from Figure 145, encoded to [RFC4648] base64url as Figure 148.

75m1ALsYv10pZTKPWrsqd

Figure 146: Content Encryption Key, base64url-encoded

veCx9ece2orS7c_N

Figure 147: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc21vbilse30sInRleHQiLCIOLEiXaXsbImZuIix7fS
widGV4dCIiK1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4iLHt9LCJ0ZnJ0Iixb
IkJyYW5keWJ1Y2siLCJNZXJpYWRvYyIsIklyLiIsIjJdXSxbImJkXYksiLHt9LC
J0ZnJ0IiwVEEgMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV4dCIiK0iXVld

Figure 148: Additional Authenticated Data, base64url-encoded

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:

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

{
  "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.

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:

```

```
4.11.2. Generated Factors

The following are generated before encrypting:

- Content encryption algorithm; this example uses "A128GCM".
- 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-hbx3wnqh5FlkEYos0sHsF0H

    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:

\[ \text{Unprotected JWE header (Figure 162)} \]
The resulting JWE object using the JSON serialization:

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

Figure 162: Unprotected JWE Header JSON

The resulting JWE object using the JSON serialization:

```json
{
    "recipients": [
        {
            "encrypted_key": "jJIcM9J-hbx3wnqhf5FlkEYos0sHsFOH"
        }
    ],
    "unprotected": {
        "alg": "A128KW",
        "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
    },
    "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
    "iv": "WgEJs9bcoX3nR",
    "ciphertext": "lIbCyRmRJxnB2yLQQTqjCDKV3H30ossGw3u9DpsqLL2D
M3swKkjQwQy2tWsdFLYmj5YeLht_StAn21tHmJuuNt64T8D4t6C7kcC90
CCJ1HAl0Uv80MyO8fZybiWgpmJl58g8N2v460gyG637d6
uuKPhXgTgm_zWhqSc_sroVqIkJyjFXPq1hBAURbc3-8BqeRb48iR1-_-5
g5UjWVD3lgiLCN_P7AW8nMIfvUNXBPJK3nOWL4teUPS8yHLbWEL83o1U
4UAgL48y-8dDk23JykibVSOj-f7-e-1xeHWxWILHs1QnBBre0dEwK3
HX_xM0Ljuz77Krppqegoutpf5qaKg3l-_xMINmf",
    "tag": "fNYLqpu84KD451vDiaBAAQ"
}
```

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.

KB0oAFl30QPv3vcZlXnzQ

Figure 164: Content Encryption Key, base64url-encoded

YihBoV0GsR1l7jCD

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_W7RMpQW81UjZ3q5LSyqiPv

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"
}
```

Figure 169: Unprotected JWE Header JSON

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:
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.
* "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.

zXayeJ4gv8NJr3IUInyokTUO-LbQNKEhe_zWlYbdpQ

**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:

dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w6Y5G4XJQsNNIBiqyvUUA
OcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXiihidAYWVajJIA
KMXMvFRMv6iD1Rr076Dfhg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDQGbo-w
qjw0-b-Sl1aS1lQVbu78dQ7Fa0zAVzVjHxf-xyvM2wxj_otxr9c1N1In2MbeYS
rRicJ5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWP1nBq_V4n5trGuExtp-d
bHcGlhhqc_wGgho9fLMK8JQArYlCmDNQ

**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": "dYOD28kab0Vvf4ODgxVAJXgHcS2ICSOp8M51zjwj4w6Y5G4XJQsN\nNIIbqyvUUAocpL7S7-cFe7PiogV_Q06WmCSa-vhW6me4b\nWrbF7chWEQJdXiiidAYWvajJiakMxMvFRMV6iD1Rr076DFthg2_AV0_t\nSiV6xSEIFqt1xnyFpmP91tc5WJDOGb-wqjw0-b-S1laS1lQvbuP78dQ7\nFa0zAVzjHX-xvyM2wxj_otxr9cN1Lzn2MbeYSrRicJK5xodvWgkpIdk\nMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-_dbHcGlihqc_wG\ngho9fLMK8JoArYLcMDNQ",
  "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": "Uzdvk3pi5wKCRc1izp5_r00jeqT-I68i8g2b8mva8diRsE2xAn2DtM\nRb25Ma2CX",
  "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1EjI1pOMbw9\n1fzZ84pbfm",
  "d": "1DKHfTv-PiffVw2VBHm_ZiVcwOMXk0yANS_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:
o Static Elliptic Curve public key (Figure 84).

o Ephemeral Elliptic Curve private key (Figure 176.

produces the following encrypted key:

ExInT0io9BqBMYF6-maw5tZ1goZXThDlzWKSHiXJuw_elY4gSSId_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.

```json
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRc1izp5_r0Ojeqt-168i8g2b8mva8diRhsE2xAn2DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3y1Ej1pOMbW91fZ84pbfm"
  }
}
```

Figure 178: Recipient #2 JWE Header JSON

The following is the assembled second recipient JSON:
"encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuweIY4gSSId_w",
"header": {
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRC1izp5_rO0jeqT-168i8g2b8mva8d1RhsE2xA
n2DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgm5Eo-CTHAZ53MC7PjIoy3y1EjIlp0
Mbw91fz284pbfm"
  }
}

Figure 179: Recipient #2 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

a7CclAejo_7JSuPB8zeagxXRam8dwCfmt9-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": "59Nqh1LlYtVIhfD3pgRGvw",
  "iv": "AvpeoPZ9Ncn9mkBn"
}
```

Figure 183: Recipient #3 JWE Header JSON

The following is the assembled third recipient JSON:

```json
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpSlE",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1LlYtVIhfD3pgRGvw",
    "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
Figure 186: Protected JWE Header, base64url-encoded

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

ajm2Q-OpPXCr7-MHxicknb1lxLdXXK_yLds0KuhJzfWK04SjdxQeSw2L9mu3a
_kNC55kCQ_3xlkcVKC5yr__Is48V0oK0k63_QRM9tBURMFqLByJ8vOYQX0oJW4
VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSaP6ga7-s1yxStR7_G07Thd1jh-zG0
wxM5g-VROQtq0K6AXpLlwEqrP7ptkt2zRMO2AXqSpe106FJ7FHLDeEFnD-zDIZu
kLpCbzhzMDLLw2-8I14FQrgi-iEuzHgIJF1Jn2wh9Tj0cg_kO2y9BqMR2bwfYM
Y9YqorZ_P_JYG3ARAIF3OjDNqpdYe-K_5Q5crGJSNyi_ygE1ItR5jssQVH2
ofDQdLChtaZ

Figure 187: Ciphertext, base64url-encoded

BESyFN709K718zKs5_g

Figure 188: Authentication Tag, base64url-encoded

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:

```
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf40DgxVAJXgHcSZICSOp8M51zj
wj4w6Y5G4XQsNsN1BiqvvUUA0cPl7S7-cFe7Pi07qV_Q06WmCSA-
vhW6me4bWrBf7chWwEQJdXihIdAYWVajJIaRMXwFRMV6iDr0r76
DFthg2_AVO_tSiV6xSEIFqt1nxypmP91tc5WJDOGb-wqjw0-b-S
llaS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9c1N1nZMbe
YSRicJK5xodvwgkpIdkMHo4LvdhRRvzozKz1c89jFWP1nBq_V4n
5trGuExtp_-dbHcGlhlqc_wGgho9lfMK8JGArYLcMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5t2lgo2XThD1zWKSHi
xJuw_elY4gSSId_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3pi5wKCRclizp5_r00jeqT-I68i8g2b8mva8diRhs
E2xAn2DtMRb25Ma2CX",
          "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLloy3ylEj
I1pOMbw9lfzS84pbfm"
        }
      }
    }
  ]
}
```
Figure 190: JSON Serialization

5. 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.

5.1. Signing Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the JSON Web Token (JWT) [I-D.ietf-oauth-json-web-token] 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

eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiw1XhwIjoxMzAwODE5MzgwLCJodHRwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0

Figure 192: Payload content, base64url-encoded
"kty": "RSA",
"kid": "hobbiton.example",
"use": "sig",
"n": "kNrP1BDXM6fvcy5i-QHQAQ-K8gsC3HJb7FYhYaw8hXbNJa-t8q01D
KwLzQXYV-ffWxXJGr1ZEG4U52lfMEegTdzYTrQR3tepgKFjMGg6I
y6fkll2Nxs2gEoonsl1ShfzA9JGwRTmKFbkls-hwx1lU5AT-AIe1NqB
Cf2vE5wZ5-SGGBoaROVdUYxqEDTgqMl5cKv42jJd2-1h4oVB07bkac6
LQdHpJUuYSH_Er20DXX30Ky197PciXkTTS-QKXmmn8ivRCmxu220Pu
ndZBKC50iG4MwALhaL2Z2k8CsRdfy-7dg7z41Rp6D0zeEvtaUp4bX4aK
ral4Tfw",
"e": "AQAB",
"d": "ZLe_TIxpE9-W_n2VBA-HWvuYPTjxvwVXClJFGpJsdea8g9RMx34gEO
EtnoYc2un3C3LtJi-mjju5RAT8YSc76YJds3ZVw0Uio8MBEg6-inoVg
oboNx7K57-xjTJ2U72ejor9kb7z62KwDq7HFyCdUhEcYHCFlvC71L_6
TibVhAhOFONWlqlJgEgwVYdorybNGKfddenpEbyhoMwY6HM1qvrnFgP7
iZ0YzHUT35x6jjyvKcvHd2uFuayseySEW7mxZ6fj1vdj1y9Dl1
fIz3Xv4ckqoqKF5GONU6tnMmMnGADg6IGiyEly1Prx1ltBhC114bR
-wzrpHgAQ",
"p": "yKWYoNIaQwMrQ0gLbDtdTN1ICbDNUUsrh-pBaxd_mIkwemM4g-0-B
2iSyvm8hrs8horhBy7vxcQaqcBAATGw-haAfUhewjwXWSH-3KccRM80tL4
e0q7M-ldRDOBXSoe72Z-CV2x Zy3RP8qp642R13WgXqDJM4MbUZSj
cy9-c",
"q": "uND4o15V30Kdf8fJw5891pl1QV3NEilrinRUPHkkxaAzDzccGqr
WMpGxGFFhNL35cQqLEe7U-51IVQqHoWV10hvXVHr7sgaGu-483Ad3
ECN26f36OnF45m7_2ooastJDe49MeLTTQKsrSIB_1SkvqyvYfPFCzE
kh9Kk",
"dp": "jmThEoq2qqa8uaymjhJSCnsveUXnMqC2gAneQRQkFqOu-zV2PKP
KNBpvRvYiyk5b2-L3tM0W2d21NdYRUW1T7V510KwPTABSTOnTqAmYCh
5i8kXXd1hcrtSvXldBakC6saxwI_TzGYY2MVVzc22CvCXHV4qjSx0rF
P3phFU",
"dq": "R9Fuv880VZeKtx13-5-WusE4DjHmnd2Z1u3rifBdfLPq-P-iWP
BbGaq9wZq1c-J7SzQjgkdJEVd5yd2C7rrn26kpzwBh_nmL8scaK1qnsu
nt9CGJAYz7-sGWy1JGSFazF52Th8r1C0YuEaQMrIzpy77_oLAhpm
DA0hlk",
"qi": "58tC7ZkwH6pF1TkjwttQOPLVMrfswrLFAvdb8N90Cv_V7F2Oq
UZCqmzHTyAmumGFH11WVRep7anleWajjXc_1b3fOq_a1qH3Pe-EK1H6
IMazuRtZLUR0cThrExDbf5dYbscDnFWURLErZ41Ne0bnyPuqXwKd9
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]
o Protected JWS Header; this example uses header from Figure 194, encoded using [RFC4648] base64url to produce Figure 195.

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

Figure 194: Protected JWS Header JSON

eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9

Figure 195: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 195) and Payload content (Figure 191) produces the following signature:

dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprzo0WaIA
KiVnlqskChirjKvY9ES2NUCP4jyfyoPS-nqjJxYoA5ztW0yFkKc2cZNIyXjcXSSQ
wXP09tEe-v4VSmqD0aKKhQpxYog4N6CzlKph1U1sYDSI67_bLL7elg_vkjfMp5
_W51SdUYGMe6hxQIaUxf9EwV2JmvTMuZ-vBOWyOSn1y1EFo72CRTvmtrIf5
AR0o5MNliY3KtUxeP-SOmd-LEYwW9S1kohYzMVA22DoVb7KVRHpeYNak75KE
QqdCxEEKs_rskZS-Qtt_nleqIWh1mEYaA

Figure 196: Signature, base64url-encoded

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:

Figure 197: Compact Serialization
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

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCJ9ImVuYyI6ImIiLCJhbGciOiJSU0EtT0FQUCIsImVuYyI6IkpXVCJ9
a0JHRoITfpX4qRewImjIStn8m3CPxBVlueY1vhjurCyrBg3I7YhCRYjphDOOS4E7rzBr2FnnNy6Qz-A-ggQFPxQnJvOGzG-dl13mwy7RoYyTkBTC67sMYYXN4g
zMedp1JHQVeYI-zkZ2V79matpgevAJSrXzOuY1GttooSN6gtUVT1laijvnb21O0u14YSn7-Hy-BX1kyyeEp_fuYJxHoKLQL9P424sKx2WGYb4zs8BPFF4ssl_e51R7nany-25_Umc2urosNkoFz9cQ82MypZP8gbbQJYPN-Fpp4Z-5o6yV64x6yzDUF5JCIld1-Qv6H5mdMVI7QyekxVcV11U0-2FefEBqxXxV1jLeZiujNkJzogCq3-IapSjVFmMjBxjPYLT8muawaloyly1XMMuinNpOCy3n4KkrXlrCctex85m4lIHMZa38s1Hypr56fPpseMA-Jltmt-a9iED022kttxz8AYXy9tsCAZV2XBWNG8c3KJusAA
mBKOywfk7jJhRLDgOnjjiJLhn7TI4UxOdp9dCmUXEN6z0v23W15qJIEXNJtqnb1pymooeWAHCT4e-Owbtimg0AEpTHUdA2liLNn9WTX-H_TXuPC8yDDhilsmxS_X_xpkIHkiIHWDOLx03BpDTivpKkBYYwp2UZkcxqX2Fo_GnVrNwlK7Lgxw6FSQvDO0
GbX1i9kXz0sxXpmA
SZI41vKHmpwaziL-pJQXX3mHv1aAnn0U4Wf9-utWYUncKrBNGCe20Fm66cSJ8k2QkxaQD3_R60ME9ofomtawy3GFXMeGRjtpMt9oAvtLsAXB0_UTCGBryBg3C2WLXqZlfJAAoJRUPRk-BimYZY81zVBuithc7HsQePcpu33SzMsFhjn41P_idrJz_glZTNgKDt8zdUnPauKTKDN0H1D04fuzvYDFdIAfGPyL5sVrwbixPxdGokEszM-9C
hMpqW1ONhn2X7u13bvrjw7nGZsQ4cUScY3n8E3AHCLJrgls-A9zmz138x8Ea
ulV1814Fg9tLejkdAucQzjPbqeqHQBJe41WGD5E0dQ-Mtz4NnhkIWX-YKBb_Xcozi3JQ_1sYjKUus7YrW-HTr-sqvPt0bj7Wjf2vzBTZ3dvsoGaTvPh2dyWwumWrlx4gmPuzBdwoT06ubyF5DUDEz3p0y0d_OtWeUSyQCyBkK--aM7tXg26qJi2TjyJf
hn9zy-W19sOCZGuzGJFhawXhpPvNj-t-O_ES96kogjjLlXsI1MU9Y5xmnwZMYNc9E1wnogscG-c-hVuvzyF0sIrultmI94_SI1xgM17o03phcTMxti1M1zR88NKU1WkB
siXMCjy1Noue7MD-SDp5dMm
KnIKEhN8U-3C9s4gtSpjSw

The resulting JWE object using the JSON serialization:

% Figure 205: Compact Serialization

eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCJ9ImVuYyI6ImIiLCJhbGciOiJSU0EtT0FQUCIsImVuYyI6IkpXVCJ9
a0JHRoITfpX4qRewImjIStn8m3CPxBVlueY1vhjurCyrBg3I7YhCRYjphDOOS4E7rzBr2FnnNy6Qz-A-ggQFPxQnJvOGzG-dl13mwy7RoYyTkBTC67sMYYXN4g
zMedp1JHQVeYI-zkZ2V79matpgevAJSrXzOuY1GttooSN6gtUVT1laijvnb21O0u14YSn7-Hy-BX1kyyeEp_fuYJxHoKLQL9P424sKx2WGYb4zs8BPFF4ssl_e51R7nany-25_Umc2urosNkoFz9cQ82MypZP8gbbQJYPN-Fpp4Z-5o6yV64x6yzDUF5JCIld1-Qv6H5mdMVI7QyekxVcV11U0-2FefEBqxXxV1jLeZiujNkJzogCq3-IapSjVFmMjBxjPYLT8muawaloyly1XMMuinNpOCy3n4KkrXlrCctex85m4lIHMZa38s1Hypr56fPpseMA-Jltmt-a9iED022kttxz8AYXy9tsCAZV2XBWNG8c3KJusAA
mBKOywfk7jJhRLDgOnjjiJLhn7TI4UxOdp9dCmUXEN6z0v23W15qJIEXNJtqnb1pymooeWAHCT4e-Owbtimg0AEpTHUdA2liLNn9WTX-H_TXuPC8yDDhilsmxS_X_xpkIHkiIHWDOLx03BpDTivpKkBYYwp2UZkcxqX2Fo_GnVrNwlK7Lgxw6FSQvDO0
GbX1i9kXz0sxXpmA
SZI41vKHmpwaziL-pJQXX3mHv1aAnn0U4Wf9-utWYUncKrBNGCe20Fm66cSJ8k2QkxaQD3_R60ME9ofomtawy3GFXMeGRjtpMt9oAvtLsAXB0_UTCGBryBg3C2WLXqZlfJAAoJRUPRk-BimYZY81zVBuithc7HsQePcpu33SzMsFhjn41P_idrJz_glZTNgKDt8zdUnPauKTKDN0H1D04fuzvYDFdIAfGPyL5sVrwbixPxdGokEszM-9C
hMpqW1ONhn2X7u13bvrjw7nGZsQ4cUScY3n8E3AHCLJrgls-A9zmz138x8Ea
ulV1814Fg9tLejkdAucQzjPbqeqHQBJe41WGD5E0dQ-Mtz4NnhkIWX-YKBb_Xcozi3JQ_1sYjKUus7YrW-HTr-sqvPt0bj7Wjf2vzBTZ3dvsoGaTvPh2dyWwumWrlx4gmPuzBdwoT06ubyF5DUDEz3p0y0d_OtWeUSyQCyBkK--aM7tXg26qJi2TjyJf
hn9zy-W19sOCZGuzGJFhawXhpPvNj-t-O_ES96kogjjLlXsI1MU9Y5xmnwZMYNc9E1wnogscG-c-hVuvzyF0sIrultmI94_SI1xgM17o03phcTMxti1M1zR88NKU1WkB
siXMCjy1Noue7MD-SDp5dMm
KnIKEhN8U-3C9s4gtSpjSw
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