JOSE Working Group M. Miller
Internet-Draft Cisco Systems, Inc.
Intended status: Informational December 04, 2013
Expires: June 7, 2014

Examples of Protecting Content using JavaScript Object Signing and Encryption (JOSE)
draft-ietf-jose-cookbook-00

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

Copyright Notice

Table of Contents

1. Introduction

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

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

1.1. Conventions Used in this Document

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

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

All of the examples in this document have whitespace added to improve formatting and readability. Except for plaintext or payload content, whitespace is not part of the cryptographic operations. Plaintext or payload content does include whitespace (unless otherwise noted), although line breaks (U+000A LINE FEED) have replaced spaces (U+0020 SPACE) is some cases 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, with line breaks (U+000A LINE FEED) replacing some " " (U+0020 SPACE) characters to improve formatting:

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

Figure 1: Payload content plaintext

The Payload - with line breaks (U+000A LINE FEED) replaced with " " (U+0020 SPACE) - encoded as [RFC4648] base64url:

```
SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91dCB5b3
VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZiB5b3UgZG9u
J3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2luZyB3aGVyZSB5b3
UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg
```


Figure 2: Payload content, base64url-encoded

3.1. RSA v1.5 Signature

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

3.1.1. Input Factors

The following are supplied before beginning the signing operation:

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

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

- "alg" parameter of "RS256".


```
{
  "kty": "RSA",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "n":   "n4EPtAOCc9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rH
          VTeT-O-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHz_AJmSCpMaJMRB
          SFKrKb2wqVwGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqk
          HHig3RBordaZ6Aj-oBHqFEHYpPe7Tpe-OfVfHd1E6cS6M1FZcD
          1NNLYD5lFHpPI9bTwJlsde3uhGqC0ZCuEHg8lhzwOHrtIQbS0F
          Vbb9k3-tVTU4fg_3L_vniUFAKwuCLqKnS2BYwdq_mzSnbLY7h_
          qixoR7jig3__kRhuaxwUkRz5iaiQkqgc5gHdrNP5zw",
  "e":   "AQAB",
  "d":   "bWUC9B-EFRIo8kpGfh0ZuyGPvMNKvYWNtB_ikiH9k20eT-O1q_
          I78eiZkpXxXQ0UTEs2LsNRS-8uJbvQ-A1irkwMSMkK1J3XTGgd
          rhCku9gRldY7sNA_AKZGh-Q661_42rINLRCe8W-nZ34ui_qOfk
          LnK9QWDDqpaIsA-bMwWWSDFu2MUBYwkHTMEzLYGqOe04noqeq1
          hExBTHBOBdkMXiuFhUq1BU6l-DqEiWxqg82sXt2h-LMnT3046A
          OYJoRioz75tSUQfGCshWTBnP5uDjd18kKhyv07lhfSJdrPdM5P
          lyl21hsFf4L_mHCuoFau7gdsPfHPxxjVOcOpBrQzwQ",
  "p":   "3Slxg_DwTXJcb6095RoXygQCAZ5RnAvZlno1yhHtnUex_fp7AZ
          _9nRaO7HX_-SFfGQeutao2TDjDAWU4Vupk8rw9JR0AzZ0N2fvu
          IAmr_WCsmGpeNqQnev1T7IyEsnh8UMt-n5CafhkikzhEsrmndH
          6LxOrvRJlsPp6Zv8bUq0k",
  "q":   "uKE2dh-cTf6ERF4k4e_jy78GfPYUIaUyoSSJuBzp3Cubk3OCqs
          6grT8bR_cu0Dm1MZwWmtdqDyI95HrUeq3MP15vMMON8lHTeZu2
```

```
             lmKvwqW7anV5UzhM1iZ7z4yMkuUwFWoBvyY898EXvRD-hdqRxH
             lSqAZ192zB3pVFJ0s7pFc",
     "dp":  "B8PVvXkvJrj2L-GYQ7v3y9r6Kw5g9SahXBwsWUzp19TVlgI-YV
             85q1NIb1rxQtD-IsXXR3-TanevuRPRt5OBOdiMGQp8pbt26glj
             YfKU_E9xn-RULHz0-ed9E9gXLKD4VGngpz-PfQ_q29pk5xWHoJ
             p009Qf1HvChixRX59ehik",
     "dq":  "CLDmDGduhylc9o7r84rEUVn7pzQ6PF83Y-iBZx5NT-TpnOZKF1
             pErAMVeKzFEl41DlHHqqBLSM0W1sOFbwTxYWZDm6sI6og5iTbw
             QGIC3gnJKbi_7k_vJgGHwHxgPaX2PnvP-zyEkDERuf-ry4c_Z1
             1Cq9AqC2yeL6kdKT1cYF8",
     "qi":  "3PiqvXQN0zwMeE-sBvZgi289XP9XCQF3VWqPzMKnIgQp7_Tugo
             6-NZBKCQsMf3HaEGBjTVJs_jcK8-TRXvaKe-7ZMaQj8VfBdYks
             sbu0NKDDhjJ-GtiseaDVWt7dcH0cfwxgFUHpQh7FoCrjFJ6h6Z
             EpMF6xmujs4qMpPz8aaI4"
   }
```


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

3.1.2. Signing Operation

The following are generated to complete the signing operation:

- 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


```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
```


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:

jYc0gEV3V-RogN63dfD39ubQDvRFpqT0pYN2zmDfhxzLEqvWNFmINxPHuBZyNb
8FDgfU7oFPgLMdbzWP8dzebwCAQH1j_MV98HMMoaQweDy8L_6XBy6JjcxGne_o
GDyMM-gBm6VyW_xqK03pLEvmUrUlFLAVWuMpkd675wX8lPtiiEmswOqph6aCtA
LnBDMTUOlFzPpOb6B6OXctf4AG1cTfzcbyLWIGhGjqnPdqmoHldn-57eRT-G-R
-UR_XcxxvQ1b7gYAhh5_367tNnlnhIvv0RNr2UaqtnSG50B3TUVdDuJ0eHmPxW
dD6kVwiEIYeHPT4uhaRe2XgbTSx2pTQg

Figure 6: Signature, base64url-encoded

3.1.3. Output Results

The following compose the resulting JWS object:

o Protected JWS header (Figure 4)

o Payload content (Figure 2)

o Signature (Figure 6)

The resulting JWS object using the Compact serialization:

eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91dCB5b3
VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZiB5b3UgZG9u
J3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2luZyB3aGVyZSB5b3
UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg
.
jYc0gEV3V-RogN63dfD39ubQDvRFpqT0pYN2zmDfhxzLEqvWNFmINxPHuBZyNb
8FDgfU7oFPgLMdbzWP8dzebwCAQH1j_MV98HMMoaQweDy8L_6XBy6JjcxGne_o
GDyMM-gBm6VyW_xqK03pLEvmUrUlFLAVWuMpkd675wX8lPtiiEmswOqph6aCtA
LnBDMTUOlFzPpOb6B6OXctf4AG1cTfzcbyLWIGhGjqnPdqmoHldn-57eRT-G-R
-UR_XcxxvQ1b7gYAhh5_367tNnlnhIvv0RNr2UaqtnSG50B3TUVdDuJ0eHmPxW
dD6kVwiEIYeHPT4uhaRe2XgbTSx2pTQg

Figure 7: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
```

```
         "protected":
           "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iY
            ml0b24uZXhhbXBsZSJ9",
         "signature":
           "jYc0gEV3V-RogN63dfD39ubQDvRFpqT0pYN2zmDfhxzLEqvWNFmIN
            xPHuBZyNb8FDgfU7oFPgLMdbzWP8dzebwCAQH1j_MV98HMMoaQweD
            y8L_6XBy6JjcxGne_oGDyMM-gBm6VyW_xqK03pLEvmUrUlFLAVWuM
            pkd675wX8lPtiiEmswOqph6aCtALnBDMTUOlFzPpOb6B6OXctf4AG
            1cTfzcbyLWIGhGjqnPdqmoHldn-57eRT-G-R-UR_XcxxvQ1b7gYAh
            h5_367tNnlnhIvv0RNr2UaqtnSG50B3TUVdDuJ0eHmPxWdD6kVwiE
            IYeHPT4uhaRe2XgbTSx2pTQg"
       }
     ]
   }
```


Figure 8: JSON Serialization

3.2. RSA-PSS Signature

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

3.2.1. Input Factors

The following are supplied before beginning the signing operation:

- o Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- o RSA private key; this example uses the key from Figure 3.
- o "alg" parameter of "RS256".

3.2.2. Signing Operation

The following are generated to complete the signing operation:

- o 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

eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
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:

kmV2DSGzAWL3qq4fZOpOfW1Jn-qFa0OyEOtaL-XiDt_JQVnhurpQlT698iBkiy
wXRzcvwyY-UgeTrCDT6kPAZHN3Tj6l_bsPwHt7B1AaphZObG94tYCdyQlwdrCy
lBBaDMwwjQuSvL9MP40KLAlV5BGmnps-2rAUK9VL_HmKRcjo2dQ_VRfbaCSmmI
-aohWvcdptMyI6kZhHL_zLCOWO5RoY5YWV42u46ZdW-e06QgBZkzdHEmS2Aimx
EIy6PamU6FKvRLR3s8tiagdmBEwqiXUoRq5i3VL-XRvGMtk6jUonloTOii-nsU
6jN1AwrFGwe7kd33X6AX9CaMtOJaUZVw

Figure 11: Signature, base64url-encoded

3.2.3. Output Results

The following compose the resulting JWS object:

- o Protected JWS header (Figure 10)

- o Payload content (Figure 2)

- o Signature (Figure 11)

The resulting JWS object using the Compact serialization:

eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91dCB5b3
VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZiB5b3UgZG9u
J3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2luZyB3aGVyZSB5b3
UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg
.
kmV2DSGzAWL3qq4fZOpOfW1Jn-qFa0OyEOtaL-XiDt_JQVnhurpQlT698iBkiy
wXRzcvwyY-UgeTrCDT6kPAZHN3Tj6l_bsPwHt7B1AaphZObG94tYCdyQlwdrCy
lBBaDMwwjQuSvL9MP40KLAlV5BGmnps-2rAUK9VL_HmKRcjo2dQ_VRfbaCSmmI
-aohWvcdptMyI6kZhHL_zLCOWO5RoY5YWV42u46ZdW-e06QgBZkzdHEmS2Aimx
EIy6PamU6FKvRLR3s8tiagdmBEwqiXUoRq5i3VL-XRvGMtk6jUonloTOii-nsU
6jN1AwrFGwe7kd33X6AX9CaMtOJaUZVw

Figure 12: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "protected":
        "eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iY
         ml0b24uZXhhbXBsZSJ9",
      "signature":
        "kmV2DSGzAWL3qq4fZOpOfW1Jn-qFa0OyEOtaL-XiDt_JQVnhurpQl
         T698iBkiywXRzcvwyY-UgeTrCDT6kPAZHN3Tj6l_bsPwHt7B1Aaph
         ZObG94tYCdyQlwdrCylBBaDMwwjQuSvL9MP40KLAlV5BGmnps-2rA
         UK9VL_HmKRcjo2dQ_VRfbaCSmmI-aohWvcdptMyI6kZhHL_zLCOWO
         5RoY5YWV42u46ZdW-e06QgBZkzdHEmS2AimxEIy6PamU6FKvRLR3s
         8tiagdmBEwqiXUoRq5i3VL-XRvGMtk6jUonloTOii-nsU6jN1AwrF
         Gwe7kd33X6AX9CaMtOJaUZVw"
    }
  ]
}
```


Figure 13: JSON Serialization

3.3. ECDSA Signature

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

3.3.1. Input Factors

The following are supplied before beginning the signing operation:

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

- EC private key on the curve P-521; this example uses the key from Figure 14.

- "alg" parameter of "ES512"


```
{
  "kty": "EC",
  "kid": "bilbo.baggins@hobbiton.example",
  "use": "sig",
  "crv": "P-521",
  "x":   "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sYl_oJX
```

```
             u9A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
     "y":   "AdymlHvOiLxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV5OhQHira
             VySsUdaQkAgDPrwQrJmbnX9cwlGfP-HqHZR1",
     "d":   "CFE43av1ypdfWGD5GgjpHW1fmnatQBh2akdmgLVc0znoq2xytfrN
             sqKlCsJb0IZkfdPi5umehMosNgn98Xf-sm0"
   }
```


Figure 14: Elliptic Curve P-521 Private Key

3.3.2. Signing Operation

The following are generated before beginning the signature process:

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


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


Figure 15: Protected JWS Header JSON


```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
```


Figure 16: Protected JWS Header, base64url-encoded

Performing the signature operation over the combined protected JWS header (Figure 16) and Payload content ({{jws-payload_b64u) produces the following signature:


```
GU4icJRWWqP0nDHX2HqiIZGueMWosZnx-RHjbNkkuJuVtW6ylbiHAHuOIuH9RD
dnildrg7VGvnjVK2Jv_47gyLQc8kweURgG5Zg6vauw6TyH7feCxMpfZ8BEqLSL
cLa_UUwYNLAFMB3FwQMIgSJJi7u510k1B6Nh-KcNJmViDeD2gA
```


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:


```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91dCB5b3V
yIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZiB5b3UgZG9uJ3
Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2luZyB3aGVyZSB5b3Ugb
WlnaHQgYmUgc3dlcHQgb2ZmIHRvLg
.
GU4icJRWWqP0nDHX2HqiIZGueMWosZnx-RHjbNkkuJuVtW6ylbiHAHuOIuH9RD
dnildrg7VGvnjVK2Jv_47gyLQc8kweURgG5Zg6vauw6TyH7feCxMpfZ8BEqLSL
cLa_UUwYNLAFMB3FwQMIgSJJi7u510k1B6Nh-KcNJmViDeD2gA
```


Figure 18: Compact Serialization

The resulting JWS object using the JSON serialization:


```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "protected":
        "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iY
         ml0b24uZXhhbXBsZSJ9",
      "signature":
        "GU4icJRWWqP0nDHX2HqiIZGueMWosZnx-RHjbNkkuJuVtW6ylbiHA
         HuOIuH9RDdnildrg7VGvnjVK2Jv_47gyLQc8kweURgG5Zg6vauw6T
         yH7feCxMpfZ8BEqLSLcLa_UUwYNLAFMB3FwQMIgSJJi7u510k1B6N
         h-KcNJmViDeD2gA"
    }
  ]
}
```


Figure 19: 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:

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

o AES symmetric key; this example uses the key from Figure 20.

o "alg" parameter of "HS256".


```
{
  "kty": "oct",
  "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037",
  "use": "sig",
  "k":   "hJtXIZ2uSN5kbQfbtTNWbpdmhkV8FJG-Onbc6mxCcYg"
}
```


Figure 20: AES 256-bit symmetric key

3.4.2. Signing Operation

The following are generated before completing the signing operation:

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


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


Figure 21: Protected JWS Header JSON


```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
```


Figure 22: Protected JWS Header, base64url-encoded

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


```
BC8xgQaFNKeZieRI0z7wDzbpRyG_ombR9gDU22IBJEM
```


Figure 23: Signature, base64url-encoded

3.4.3. Output Results

The following compose the resulting JWS object:

o Protected JWS header (Figure 22)

o Payload content (Figure 2)

o Signature (Figure 23)

The resulting JWS object using the Compact serialization:


```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
.
SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91dCB5b3
VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZiB5b3UgZG9u
J3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2luZyB3aGVyZSB5b3
UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg
.
BC8xgQaFNKeZieRI0z7wDzbpRyG_ombR9gDU22IBJEM
```


Figure 24: Compact Serialization

The resulting JWS object using the JSON serialization:


```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "protected":
        "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxY
         i1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature":
        "BC8xgQaFNKeZieRI0z7wDzbpRyG_ombR9gDU22IBJEM"
    }
  ]
}
```


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:

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

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

- o Signing algorithm; this example uses "RS256".

3.5.2. Signing Operation

The following are generated before completing the signing operation:

- o 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


```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
```


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:


```
ns-fxWMR0YjG5KJK5VAVdE9c9zEHL4SlnjJvw2yiRQw
```


Figure 28: Signature, base64url-encoded

3.5.3. Output Results

The following compose the resulting JWS object:

o Protected JWS header (Figure 27)

o Signature (Figure 28)

The resulting JWS object using the Compact serialization:


```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
.
.
ns-fxWMR0YjG5KJK5VAVdE9c9zEHL4SlnjJvw2yiRQw
```


Figure 29: JSON Serialization

The resulting JWS object using the JSON serialization:


```
{
  "signatures": [
    {
      "protected":
        "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid":
          "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature":
        "LHbXRdr8vWfAWIPsViW2RDE5edTLiChP_6fRTDbwHBM"
    }
  ]
}
```


Figure 30: JSON Serialization

3.6. Protecting Specific Header Fields

This example illustrates a signature where only certain header parameters are protected. Since this example contains both unprotected and protected header parameters, only the JSON serialization is possible.

3.6.1. Input Factors

The following are supplied before beginning the signing operation:

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

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

o Signing algorithm; this example uses "RS256".

3.6.2. Signing Operation

The following are generated before completing the signing operation:

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

o Unprotected JWS Header; this example uses the header from Figure 33.

The protected JWS header parameters:

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

Figure 31: Protected JWS Header JSON

eyJhbGciOiJIUzI1NiJ9

Figure 32: Protected JWS Header, base64url-encoded

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

Figure 33: Unprotected JWS Header JSON

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

LHbXRdr8vWfAWIPsViW2RDE5edTLiChP_6fRTDbwHBM

Figure 34: Signature, base64url-encoded

3.6.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JSON serialization:


```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "protected":
        "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid":
          "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature":
        "LHbXRdr8vWfAWIPsViW2RDE5edTLiChP_6fRTDbwHBM"
    }
  ]
}
```


Figure 35: 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:

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

o Signing algorithm; this example uses "RS256"

3.7.2. Signing Operation

The following are generated before completing the signing operation:

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


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


Figure 36: Unprotected JWS Header JSON

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:


```
RDrY7zngV8Mi0agUZpWOyS2WSIziPslf9tQllQYXC08
```


Figure 37: Signature, base64url-encoded

3.7.3. Output Results

The following compose the resulting JWS object:

o Unprotected JWS header (Figure 36)

o Payload content (Figure 2)

o Signature (Figure 37)

The resulting JWS object using the JSON serialization:


```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "header": {
        "alg":
          "HS256",
        "kid":
```

```
            "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
        },
        "signature":
          "RDrY7zngV8Mi0agUZpWOyS2WSIziPslf9tQllQYXC08"
      }
    ]
  }
```


JSON Serialization

3.8. Multiple Signatures

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

3.8.1. Input Factors

The following are supplied before beginning the signing operation:

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

- Signing keys; this example uses the following:

  * RSA private key from Figure 3 for the first signature

  * EC private key from Figure 14 for the second signature

  * AES symmetric key from Figure 20 for the third signature

- Signing algorithms; this example uses the following:

  * "RS256" for the first signature

  * "ES512" for the second signature

  * "HS256" for the third signature

3.8.2. First Signing Operation

The following are generated before completing the first signing operation:

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

- Unprotected JWS Header; this example uses the header from Figure 40.

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

Figure 38: Signature #1 Protected JWS Header JSON

eyJhbGciOiJSUzI1NiJ9

Figure 39: Signature #1 Protected JWS Header, base64url-encoded

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

Figure 40: Signature #1 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:

B4yWtHdhlWkVAB7hYEczTe4fNixKRbOV6XnTZ_LMIRabj3WLZe61BgWsaE_tXI
LGjS_hxIkY1YluKOOmC80vmGF-0j5T6mGKqcHxApoXbhTls9utFReQgg7OpXNB
r9F1-Dn4K1kTEiVWZMJqSEJljrGcznKj3bJTcEQOoZPf16YigOl39Vifani_qY
Qr0FLzSd0WTdO7M3b4WRCRYHGZQ9ssZXvFQ2A2C73zDARzKj3YBuUvgzKkTB_H
_aoCUH8tOhjE6XU5A6Uil508sldyYo-sYIe9waWWchM4snN_uWCAMecr4WmRIO
sb8rz7cRXK9MeH_6w8YntuDtgkCScdxQ

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":
    "B4yWtHdhlWkVAB7hYEczTe4fNixKRbOV6XnTZ_LMIRabj3WLZe61BgWsa
     E_tXILGjS_hxIkY1YluKOOmC80vmGF-0j5T6mGKqcHxApoXbhTls9utFR
     eQgg7OpXNBr9F1-Dn4K1kTEiVWZMJqSEJljrGcznKj3bJTcEQOoZPf16Y
     igOl39Vifani_qYQr0FLzSd0WTdO7M3b4WRCRYHGZQ9ssZXvFQ2A2C73z
     DARzKj3YBuUvgzKkTB_H_aoCUH8tOhjE6XU5A6Uil508sldyYo-sYIe9w
     aWWchM4snN_uWCAMecr4WmRIOsb8rz7cRXK9MeH_6w8YntuDtgkCScdxQ"
}
```


Figure 42: Signature #1 JSON

3.8.3. Second Signing Operation

The following are generated before completing the second signing operation:

o Unprotected JWS Header; this example uses the header from Figure 43.


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


Figure 43: Signature #2 JWS Header JSON

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:


```
GliCVJY7BmN6pRTLfpWIKBjczIXDJjFlXluppc24eYWPCJCNP8z1YRp9mBn7wq
UkU0xPaSzcq-GppxhQTUq27TsORK11Ab3i74DiNmsy_usLDyzlSh2UCW-jF6WA
H1jq0fCa32H4zxnTIJV_uwMDyLuuXdwgHLfDoA1hEyUoqeY50A
```


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":
    "GliCVJY7BmN6pRTLfpWIKBjczIXDJjFlXluppc24eYWPCJCNP8z1YRp9m
     Bn7wqUkU0xPaSzcq-GppxhQTUq27TsORK11Ab3i74DiNmsy_usLDyzlSh
     2UCW-jF6WAH1jq0fCa32H4zxnTIJV_uwMDyLuuXdwgHLfDoA1hEyUoqeY
     50A"
}
```


Figure 45: Signature #2 JSON

3.8.4. Third Signing Operation

The following are generated before completing the third signing operation:

o 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


```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
```


Figure 47: Signature #3 Protected JWS Header, base64url-encoded

Performing the third signature operation over the combined protected JWS header (Figure 47) and Payload content (Figure 2) produces the following signature:


```
RDrY7zngV8Mi0agUZpWOyS2WSIziPslf9tQllQYXC08
```


Figure 48: Signature #3, base64url-encoded

The following is the assembled third signature serialized as JSON:


```
{
  "protected":
   "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZm
    Q2LWVlZjMxNGJjNzAzNyJ9",
  "signature":
   "BC8xgQaFNKeZieRI0z7wDzbpRyG_ombR9gDU22IBJEM"
}
```


Figure 49: Signature #3 JSON

3.8.5. Output Results

The following compose the resulting JWS object:

- o Payload content (Figure 2)

- o Signature #1 JSON (Figure 42)

- o Signature #2 JSON (Figure 45)

- o Signature #3 JSON (Figure 49)

The resulting JWS object using the JSON serialization:


```
{
  "payload":
    "SXQncyBhIGRhbmdlcm91cyBidXNpbmVzcywgRnJvZG8sIGdvaW5nIG91d
     CB5b3VyIGRvb3IuIFlvdSBzdGVwIG9udG8gdGhlIHJvYWQsIGFuZCBpZi
     B5b3UgZG9uJ3Qga2VlcCB5b3VyIGZlZXQsIHRoZXJlJ3Mgbm8ga25vd2l
     uZyB3aGVyZSB5b3UgbWlnaHQgYmUgc3dlcHQgb2ZmIHRvLg",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiJ9",
      "header": {
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature":
        "B4yWtHdhlWkVAB7hYEczTe4fNixKRbOV6XnTZ_LMIRabj3WLZe61B
         gWsaE_tXILGjS_hxIkY1YluKOOmC80vmGF-0j5T6mGKqcHxApoXbh
         Tls9utFReQgg7OpXNBr9F1-Dn4K1kTEiVWZMJqSEJljrGcznKj3bJ
         TcEQOoZPf16YigOl39Vifani_qYQr0FLzSd0WTdO7M3b4WRCRYHGZ
         Q9ssZXvFQ2A2C73zDARzKj3YBuUvgzKkTB_H_aoCUH8tOhjE6XU5A
         6Uil508sldyYo-sYIe9waWWchM4snN_uWCAMecr4WmRIOsb8rz7cR
         XK9MeH_6w8YntuDtgkCScdxQ"
    },
    {
      "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature":
        "GliCVJY7BmN6pRTLfpWIKBjczIXDJjFlXluppc24eYWPCJCNP8z1Y
         Rp9mBn7wqUkU0xPaSzcq-GppxhQTUq27TsORK11Ab3i74DiNmsy_u
         sLDyzlSh2UCW-jF6WAH1jq0fCa32H4zxnTIJV_uwMDyLuuXdwgHLf
         DoA1hEyUoqeY50A"
    },
    {
      "protected":
```

```
          "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi
           1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
         "signature":
          "BC8xgQaFNKeZieRI0z7wDzbpRyG_ombR9gDU22IBJEM"
       }
     ]
   }
```


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 line breaks (U+000A LINE FEED) replacing some " " (U+0020 SPACE) characters to improve formatting:

```
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".


```
{
  "kty": "RSA",
```

```
     "kid": "frodo.baggins@hobbiton.example",
     "use": "enc",
     "n":   "maxhbsmBtdQ3CNrKvprUE6n9lYcregDMLYNeTAWcLj8NnPU9XIYe
             gTHVHQjxKDSHP2l-F5jS7sppG1wgdAqZyhnWvXhYNvcM7RfgKxqN
             x_xAHx6f3yy7s-M9PSNCwPC2lh6UAkR4I00EhV9lrypM9Pi4lBUo
             p9t5fS9W5UNwaAllhrd-osQGPjIeI1deHTwx-ZTHu3C60Pu_LJIl
             6hKn9wbwaUmA4cR5Bd2pgbaY7ASgsjCUbtYJaNIHSoHXprUdJZKU
             MAzV0WOKPfA6OPI4oypBadjvMZ4ZAj3BnXaSYsEZhaueTXvZB4eZ
             OAjIyh2e_VOIKVMsnDrJYAVotGlvMQ",
     "e":   "AQAB",
     "d":   "Kn9tgoHfiTVi8uPu5b9TnwyHwG5dK6RE0uFdlpCGnJN7ZEi963R7
             wybQ1PLAHmpIbNTztfrheoAniRV1NCIqXaW_qS461xiDTp4ntEPn
             qcKsyO5jMAji7-CL8vhpYYowNFvIesgMoVaPRYMYT9TW63hNM0aW
             s7USZ_hLg6Oe1mY0vHTI3FucjSM86Nff4oIENt43r2fspgEPGRrd
             E6fpLc9Oaq-qeP1GFULimrRdndm-P8q8kvN3KHlNAtEgrQAgTTgz
             80S-3VD0FgWfgnb1PNmiuPUxO8OpI9KDIfu_acc6fg14nsNaJqXe
             6RESvhGPH2afjHqSy_Fd2vpzj85bQQ",
     "p":    "2DwQmZ43FoTnQ8IkUj3BmKRf5Eh2mizZA5xEJ2MinUE3sdTYKSL
             taEoekX9vbBZuWxHdVhM6UnKCJ_2iNk8Z0ayLYHL0_G21aXf9-un
             ynEpUsH7HHTklLpYAzOOx1ZgVljoxAdWNn3hiEFrjZLZGS7lOH-a
             3QQlDDQoJOJ2VFmU",
     "q":   "te8LY4-W7IyaqH1ExujjMqkTAlTeRbv0VLQnfLY2xINnrWdwiQ93
             _VF099aP1ESeLja2nw-6iKIe-qT7mtCPozKfVtUYfz5HrJ_XY2kf
             exJINb9lhZHMv5p1skZpeIS-GPHCC6gRlKo1q-idn_qxyusfWv7W
             AxlSVfQfk8d6Et0",
     "dp":  "UfYKcL_or492vVc0PzwLSplbg4L3-Z5wL48mwiswbpzOyIgd2xHT
             HQmjJpFAIZ8q-zf9RmgJXkDrFs9rkdxPtAsL1WYdeCT5c125Fkdg
             317JVRDo1inX7x2Kdh8ERCreW8_4zXItuTl_KiXZNU5lvMQjWbIw
             2eTx1lpsflo0rYU",
     "dq":  "iEgcO-QfpepdH8FWd7mUFyrXdnOkXJBCogChY6YKuIHGc_p8Le9M
             bpFKESzEaLlN1Ehf3B6oGBl5Iz_ayUlZj2IoQZ82znoUrpa9fVYN
             ot87ACfzIG7q9Mv7RiPAderZi03tkVXAdaBau_9vs5rS-7HMtxkV
             rxSUvJY14TkXlHE",
     "qi":  "kC-lzZOqoFaZCr5l0tOVtREKoVqaAYhQiqIRGL-MzS4sCmRkxm5v
             ZlXYx6RtE1n_AagjqajlkjieGlxTTThHD8Iga6foGBMaAr5uR1hG
             QpSc7Gl7CF1DZkBJMTQN6EshYzZfxW08mIO8M6Rzuh0beL6fG9mk
             DcIyPrBXx2bQ_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:

o AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 53

o Initialization vector/nonce; this example uses the initialization vector from Figure 54

vQ6_Pof-pnIBBB_qhAxzuusbc25hFCB1pJuBIN7yMNU

Figure 53: Content Encryption Key, base64url-encoded

mR-7lneQlGq9vxe_udL4LA

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:

IDNYysyXa21oifTY_cy7sB7vAa9oHkE4RZZ78r88TdrGlKWbzltMJw4sJ7xpNo
vR8KZDHLeJUwiaQKIjWBFs2Dytdk4gHhVDc2rx9F2vHN2S1vQuC_TYslbSDLHx
nnZkH2_ymlJz2saY5RJAjh-9OHCMcTJI-j7hJpMEJmvWt_XrDp9tBby0xyjdwd
teAtwyJxD5nyzBUGTsfaCzfqZTF_3BJu2AKyuE10KEMbBo8EJVf1PP1JSS73qy
UqEt8oo0OHlYTicOwwwwhyiNshdrA4zQSeC2M0yxzDcQvXswQHQs1bXA8K-KJa
B-u6qkDMAwA1tJEch4R58z9WsYKyrhAw

Figure 55: Encrypted Key, base64url-encoded

4.1.4. Encrypting the Content

The following are generated before encrypting the plaintext:

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

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

Figure 56: Protected JWE Header JSON

eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0

Figure 57: Protected JWE Header, base64url-encoded

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

- o CEK (Figure 53);

- o Initialization vector/nonce (Figure 54); and

- o Protected JWE header (Figure 56) as authenticated data

produces the following:

- o Ciphertext from Figure 58.

- o Authentication tag from Figure 59.

LecEGK89HoOzWbbh97km04mExBgZp0kO8LMFTJgfTqNjHW5VSPh0QMww7zqSkl
a_8ZPoWIzm1Y6xGtKLA9enpRFTrHZxZxTH9eG9P9PjsIC20NsGVweYeYc_l7m2
vyC_E1BzTQ9jb3wS1DxrqSX6YRjJ5mqx8ZX3tJW-wWVZfW8-PSEXb4GlBi22iQ
goXfx8yHYfv-lXWlaQ2HjDWl21Mab41aW4ZYKt8maWZiglK4XckGv7-whchA42
VB4pNOQMY7e9BTyvm-DwVSS3Ul2bX3jz9kB--aTLxGtl9sR7z1ZgAyfRqoSs0S
op9J35heE89JveLIAjnuXH2ShsF0lW6T4HEYXFh9QsAF4TRdnpRs4

Figure 58: Ciphertext, base64url-encoded

3AIdtJkgAkWuhBdFo8iL8A

Figure 59: Authentication Tag, base64url-encoded

4.1.5. Output Results

The following compose the resulting JWE object:

- o Protected JWE header (Figure 57).

- o Encrypted Key (Figure 55).

- o Initialization vector/nonce (Figure 54).

- o Ciphertext (Figure 58).

- o Authentication Tag (Figure 59).

The resulting JWE object using the Compact serialization:

eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
.
IDNYysyXa21oifTY_cy7sB7vAa9oHkE4RZZ78r88TdrGlKWbzltMJw4sJ7xpNo

```
vR8KZDHLeJUwiaQKIjWBFs2Dytdk4gHhVDc2rx9F2vHN2S1vQuC_TYslbSDLHx
nnZkH2_ymlJz2saY5RJAjh-9OHCMcTJI-j7hJpMEJmvWt_XrDp9tBby0xyjdwd
teAtwyJxD5nyzBUGTsfaCzfqZTF_3BJu2AKyuE10KEMbBo8EJVf1PP1JSS73qy
UqEt8oo0OHlYTicOwwwwhyiNshdrA4zQSeC2M0yxzDcQvXswQHQs1bXA8K-KJa
B-u6qkDMAwA1tJEch4R58z9WsYKyrhAw
.
mR-7lneQlGq9vxe_udL4LA
.
LecEGK89HoOzWbbh97km04mExBgZp0kO8LMFTJgfTqNjHW5VSPh0QMww7zqSkl
a_8ZPoWIzm1Y6xGtKLA9enpRFTrHZxZxTH9eG9P9PjsIC20NsGVweYeYc_l7m2
vyC_E1BzTQ9jb3wS1DxrqSX6YRjJ5mqx8ZX3tJW-wWVZfW8-PSEXb4GlBi22iQ
goXfx8yHYfv-lXWlaQ2HjDWl21Mab41aW4ZYKt8maWZiglK4XckGv7-whchA42
VB4pNOQMY7e9BTyvm-DwVSS3Ul2bX3jz9kB--aTLxGtl9sR7z1ZgAyfRqoSs0S
op9J35heE89JveLIAjnuXH2ShsF0lW6T4HEYXFh9QsAF4TRdnpRs4
.
3AIdtJkgAkWuhBdFo8iL8A
```


Figure 60: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": {
    "encrypted_key":
      "IDNYysyXa21oifTY_cy7sB7vAa9oHkE4RZZ78r88TdrGlKWbzltMJw4
       sJ7xpNovR8KZDHLeJUwiaQKIjWBFs2Dytdk4gHhVDc2rx9F2vHN2S1v
       QuC_TYslbSDLHxnnZkH2_ymlJz2saY5RJAjh-9OHCMcTJI-j7hJpMEJ
       mvWt_XrDp9tBby0xyjdwdteAtwyJxD5nyzBUGTsfaCzfqZTF_3BJu2A
       KyuE10KEMbBo8EJVf1PP1JSS73qyUqEt8oo0OHlYTicOwwwwhyiNshd
       rA4zQSeC2M0yxzDcQvXswQHQs1bXA8K-KJaB-u6qkDMAwA1tJEch4R5
       8z9WsYKyrhAw"
  },
  "protected":
    "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpd
     G9uLmV4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "iv":
    "mR-7lneQlGq9vxe_udL4LA",
  "ciphertext":
    "LecEGK89HoOzWbbh97km04mExBgZp0kO8LMFTJgfTqNjHW5VSPh0QMww7
     zqSkla_8ZPoWIzm1Y6xGtKLA9enpRFTrHZxZxTH9eG9P9PjsIC20NsGVw
     eYeYc_l7m2vyC_E1BzTQ9jb3wS1DxrqSX6YRjJ5mqx8ZX3tJW-wWVZfW8
     -PSEXb4GlBi22iQgoXfx8yHYfv-lXWlaQ2HjDWl21Mab41aW4ZYKt8maW
     ZiglK4XckGv7-whchA42VB4pNOQMY7e9BTyvm-DwVSS3Ul2bX3jz9kB--
     aTLxGtl9sR7z1ZgAyfRqoSs0Sop9J35heE89JveLIAjnuXH2ShsF0lW6T
     4HEYXFh9QsAF4TRdnpRs4",
  "tag":
    "3AIdtJkgAkWuhBdFo8iL8A"
}
```

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:

- o Plaintext content; this example uses the plaintext from Figure 51.
- o RSA public key; this example uses the key from Figure 62.
- o "alg" parameter of "RSA-OAEP"
- o "enc" parameter of "A256GCM"


```
{
  "kty": "RSA",
  "kid": "samwise.gamgee@hobbiton.example",
  "use": "enc",
  "alg": "RSA-OAEP",
  "n":   "wbdxI55VaanZXPY29Lg5hdmv2XhvqAhoxUkanfzf2-5zVUxa6prH
          RrI4pP1AhoqJRlZfYtWWd5mmHRG2pAHIlh0ySJ9wi0BioZBl1XP2
          e-C-FyXJGcTy0HdKQWlrfhTm42EW7Vv04r4gfao6uxjLGwfpGrZL
          arohiWCPnkNrg71S2CuNZSQBIPGjXfkmIy2tl_VWgGnL22GplyXj
          5YlBLdxXp3XeStsqo571utNfoUTU8E4qdzJ3U1DItoVkPGsMwlmm
          nJiwA7sXRItBCivR4M5qnZtdw-7v4WuR4779ubDuJ5nalMv2S66-
          RPcnFAzWSKxtBDnFJJDGIUe7Tzizjg1nms0Xq_yPub_UOlWn0ec8
          5FCft1hACpWG8schrOBeNqHBODFskYpUc2LC5JA2TaPF2dA67dg1
          TTsC_FupfQ2kNGcE1LgprxKHcVWYQb86B-HozjHZcqtauBzFNV5t
          bTuB-TpkcvJfNcFLlH3b8mb-H_ox35FjqBSAjLKyoeqfKTpVjvXh
          d09knwgJf6VKq6UC418_TOljMVfFTWXUxlnfhOOnzW6HSSzD1c9W
          rCuVzsUMv54szidQ9wf1cYWf3g5qFDxDQKis99gcDaiCAwM3yEBI
          zuNeeCa5dartHDb1xEB_HcHSeYbghbMjGfasvKn0aZRsnTyC0xhW
          BlsolZE",
  "e":   "AQAB",
  "d":   "n7fzJc3_WG59VEOBTkayzuSMM780OJQuZjN_KbH8lOZG25ZoA7T4
          Bxcc0xQn5oZE5uSCIwg91oCt0JvxPcpmqzaJZg1nirjcWZ-oBtVk
          7gCAWq-B3qhfF3izlbkosrzjHajIcY33HBhsy4_WerrXg4MDNE4H
          Yojy68TcxT2LYQRxUOCf5TtJXvM8olexlSGtVnQnDRutxEUCwiew
          fmmrfveEogLx9EA-KMgAjTiISXxqIXQhWUQX1G7v_mV_Hr2YuImY
          cNcHkRvp9E7ook0876DhkO8v4UOZLwA1OlUX98mkoqwc58A_Y2lB
          YbVx1_s5lpPsEqbbH-nqIjh1fL0gdNfihLxnclWtW7pCztLnImZA
          yeCWAG7ZIfv-Rn9fLIv9jZ6r7r-MSH9sqbuziHN2grGjD_jfRluM
```

```
            Ha0l84fFKl6bcqN1JWxPVhzNZo01yDF-1LiQnqUYSepPf6X3a2SO
            dkqBRiquE6EvLuSYIDpJq3jDIsgoL8Mo1LoomgiJxUwL_GWEOGu2
            8gplyzm-9Q0U0nyhEf1uhSR8aJAQWAiFImWH5W_IQT9I7-yrindr
            _2fWQ_i1UgMsGzA7aOGzZfPljRy6z-tY_KuBG00-28S_aWvjyUc-
            Alp8AUyKjBZ-7CWH32fGWK48j1t-zomrwjL_mnhsPbGs0c9WsWgR
            zI-K8gE",
  "p":   "7_2v3OQZzlPFcHyYfLABQ3XP85Es4hCdwCkbDeltaUXgVy9l9etK
            ghvM4hRkOvbb01kYVuLFmxIkCDtpi-zLCYAdXKrAK3PtSbtzld_X
            Z9nlsYa_QZWpXB_IrtFjVfdKUdMz94pHUhFGFj7nr6NNxfpiHSHW
            FE1zD_AC3mY46J961Y2LRnreVwAGNw53p07Db8yD_92pDa97vqcZ
            OdgtybH9q6uma-RFNhO1AoiJhYZj69hjmMRXx-x56HO9cnXNbmzN
            SCFCKnQmn4GQLmRj9sfbZRqL94bbtE4_e0Zrpo8RNo8vxRLqQNwI
            y85fc6BRgBJomt8QdQvIgPgWCv5HoQ",
  "q":   "zqOHk1P6WN_rHuM7ZF1cXH0x6RuOHq67WuHiSknqQeefGBA9PWs6
            ZyKQCO-O6mKXtcgE8_Q_hA2kMRcKOcvHil1hqMCNSXlflM7WPRPZ
            u2qCDcqssd_uMbP-DqYthH_EzwL9KnYoH7JQFxxmcv5An8oXUtTw
            k4knKjkIYGRuUwfQTus0w1NfjFAyxOOiAQ37ussIcE6C6ZSsM3n4
            1UlbJ7TCqewzVJaPJN5cxjySPZPD3Vp01a9YgAD6a3IIaKJdIxJS
            1ImnfPevSJQBE79-EXe2kSwVgOzvt-gsmM29QQ8veHy4uAqca5dZ
            zMs7hkkHtw1z0jHV90epQJJlXXnH8Q",
  "dp":  "19oDkBh1AXelMIxQFm2zZTqUhAzCIr4xNIGEPNoDt1jK83_FJA-x
            nx5kA7-1erdHdms_Ef67HsONNv5A60JaR7w8LHnDiBGnjdaUmmuO
            8XAxQJ_ia5mxjxNjS6E2yD44USo2JmHvzeeNczq25elqbTPLhUpG
            o1IZuG72FZQ5gTjXoTXC2-xtCDEUZfaUNh4IeAipfLugbpe0JAFl
            FfrTDAMUFpC3iXjxqzbEanflwPvj6V9iDSgjj8SozSM0dLtxvu0L
            IeIQAeEgT_yXcrKGmpKdSO08kLBx8VUjkbv_3Pn20Gyu2YEuwpFl
            M_H1NikuxJNKFGmnAq9LcnwwT0jvoQ",
  "dq":  "S6p59KrlmzGzaQYQM3o0XfHCGvfqHLYjCO557HYQf72O9kLMCfd_
            1VBEqeD-1jjwELKDjck8kOBl5UvohK1oDfSP1DleAy-cnmL29DqW
            mhgwM1ip0CCNmkmsmDSlqkUXDi6sAaZuntyukyflI-qSQ3C_BafP
            yFaKrt1fgdyEwYa08pESKwwWisy7KnmoUvaJ3SaHmohFS78TJ25c
            fc10wZ9hQNOrIChZlkiOdFCtxDqdmCqNacnhgE3bZQjGp3n83ODS
            z9zwJcSUvODlXBPc2AycH6Ci5yjbxt4Ppox_5pjm6xnQkiPgj01G
            psUssMmBN7iHVsrE7N2iznBNCeOUIQ",
  "qi":  "FZhClBMywVVjnuUud-05qd5CYU0dK79akAgy9oX6RX6I3IIIPckC
            ciRrokxglZn-omAY5CnCe4KdrnjFOT5YUZE7G_Pg44XgCXaarLQf
            4hl80oPEf6-jJ5Iy6wPRx7G2e8qLxnh9cOdf-kRqgOS3F48Ucvw3
            ma5V6KGMwQqWFeV31XtZ8l5cVI-I3NzBS7qltpUVgz2Ju021eyc7
            IlqgzR98qKONl27DuEES0aK0WE97jnsyO27Yp88Wa2RiBrEocM89
            QZI1seJiGDizHRUP4UZxw9zsXww46wy0P6f9grnYp7t8LkyDDk8e
            oI4KX6SNMNVcyVS9IWjlq8EzqZEKIA"
}
```


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.


```
09EnDWfdf6KCP09QbQQdhhoyFE5GoGAjdShgLGLol8k
```


Figure 63: Content Encryption Key, base64url-encoded


```
MKccO3TLKaMB67zj
```


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:


```
WaQnEArx6u7NRlv6o08qNgsMpo-UFVXs_ALqEbAhQJVOXMyNqacRhYoujIOFlt
i0u_ofz6Yh93Pu83iTZYIGk0fFD4C62-kSiX5Enl-UyORiMca925XxPItbOE6v
8sbzwzTBC-rzPXN9qrfPvKCdrlWbJv6oQAUbtreKpG8yD17YYyKw9qedx7BrSc
9TCvNH8ahrC93O1Qp_rZIPdkt92A8yhAje_cxMMaMHZ4YmlN_u05120iJhp6bg
5S_zPlpqmd5-BrpUIGmH0hwuBk2Z5Djbm47YlosNZUWIB69JBFazaDhGGRhsfM
eceRRhhGaG94gP2uZS42h1fZjYDH9zNxvXXHJ-4zqj8sM-LHLH16uDTVkvYaUw
ZPTUAj55fBKBs-o8o7rLCce8AWXEDBuqm-8rTnakLss5_Hc7l61RC8K4IJWnWB
zGLFlM1jRApbEwA5HZO61ku45WLGC64NLDdnyGahFxW9EikKXcE21Jc716oRfF
EeiOXaZGtJhYTjsuPr5IYJr8fWX_NC1y1xqYW1JnfLKklIemtznHTXvq7de6Vx
eSucwBklB24oafBFSeMEGBqKFQPhCBaAqCdJL1NkMhA9D6gqJkjluT1Aj0_Dq8
ikwSP2dG3grGlrl6EtTuhA-j_ArrqLzvSoJU51Nu_1k3usrvgIWLZ89i8Zt05S
A
```


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 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


```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0
```


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.


```
dLMYOKvvtY8Adfc8Tg4lSKElvvzBi2MJcSKDssll-jj0S_NIjI3P956z_qhZgJ
3bVLTVPr1p8JcThDcqTZtrA4ShYpyyTrBnEzOOyyzv4h6WWiuoCJTfI1Gxzm-J
PJNdPYifgOS0E5RKk5L64yDtotBP9AMTZZ4WjatyAlRcgD5hVhP2HrqIVG0j1S
b5g5tz4fi5vmSwIhGOeOxwgLvuryjn-8ECi_5LkdUiC3wQP53pZRtqmwmACyq_
uXvTpbUWSGna1LKaup-UC1PkOUGvHFa83WgkarkJbSkA0ZoVtebBJ_XvwNlfe1
VRDk8hxh0AmPvvsMiantgQ4oE6LCjOWKj6C4UNSWgla8zhousjSQ
```


Figure 68: Ciphertext, base64url-encoded


```
vCmVSWBtadRAKUhtizP5tw
```


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)

o Ciphertext (Figure 68)

o Authentication tag (Figure 69)

The resulting JWE object using the Compact serialization:


```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0
.
WaQnEArx6u7NRlv6o08qNgsMpo-UFVXs_ALqEbAhQJVOXMyNqacRhYoujIOFlt
i0u_ofz6Yh93Pu83iTZYIGk0fFD4C62-kSiX5Enl-UyORiMca925XxPItbOE6v
8sbzwzTBC-rzPXN9qrfPvKCdrlWbJv6oQAUbtreKpG8yD17YYyKw9qedx7BrSc
9TCvNH8ahrC93O1Qp_rZIPdkt92A8yhAje_cxMMaMHZ4YmlN_u05120iJhp6bg
5S_zPlpqmd5-BrpUIGmH0hwuBk2Z5Djbm47YlosNZUWIB69JBFazaDhGGRhsfM
eceRRhhGaG94gP2uZS42h1fZjYDH9zNxvXXHJ-4zqj8sM-LHLH16uDTVkvYaUw
ZPTUAj55fBKBs-o8o7rLCce8AWXEDBuqm-8rTnakLss5_Hc7l61RC8K4IJWnWB
zGLFlM1jRApbEwA5HZO61ku45WLGC64NLDdnyGahFxW9EikKXcE21Jc716oRfF
EeiOXaZGtJhYTjsuPr5IYJr8fWX_NC1y1xqYW1JnfLKklIemtznHTXvq7de6Vx
eSucwBklB24oafBFSeMEGBqKFQPhCBaAqCdJL1NkMhA9D6gqJkjluT1Aj0_Dq8
ikwSP2dG3grGlrl6EtTuhA-j_ArrqLzvSoJU51Nu_1k3usrvgIWLZ89i8Zt05S
A
.
MKccO3TLKaMB67zj
.
dLMYOKvvtY8Adfc8Tg4lSKElvvzBi2MJcSKDssll-jj0S_NIjI3P956z_qhZgJ
3bVLTVPr1p8JcThDcqTZtrA4ShYpyyTrBnEzOOyyzv4h6WWiuoCJTfI1Gxzm-J
PJNdPYifgOS0E5RKk5L64yDtotBP9AMTZZ4WjatyAlRcgD5hVhP2HrqIVG0j1S
b5g5tz4fi5vmSwIhGOeOxwgLvuryjn-8ECi_5LkdUiC3wQP53pZRtqmwmACyq_
uXvTpbUWSGna1LKaup-UC1PkOUGvHFa83WgkarkJbSkA0ZoVtebBJ_XvwNlfe1
VRDk8hxh0AmPvvsMiantgQ4oE6LCjOWKj6C4UNSWgla8zhousjSQ
.
vCmVSWBtadRAKUhtizP5tw
```


Figure 70: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
        "WaQnEArx6u7NRlv6o08qNgsMpo-UFVXs_ALqEbAhQJVOXMyNqacRh
         YoujIOFlti0u_ofz6Yh93Pu83iTZYIGk0fFD4C62-kSiX5Enl-UyO
         RiMca925XxPItbOE6v8sbzwzTBC-rzPXN9qrfPvKCdrlWbJv6oQAU
         btreKpG8yD17YYyKw9qedx7BrSc9TCvNH8ahrC93O1Qp_rZIPdkt9
         2A8yhAje_cxMMaMHZ4YmlN_u05120iJhp6bg5S_zPlpqmd5-BrpUI
         GmH0hwuBk2Z5Djbm47YlosNZUWIB69JBFazaDhGGRhsfMeceRRhhG
         aG94gP2uZS42h1fZjYDH9zNxvXXHJ-4zqj8sM-LHLH16uDTVkvYaU
```

```
            wZPTUAj55fBKBs-o8o7rLCce8AWXEDBuqm-8rTnakLss5_Hc7l61R
            C8K4IJWnWBzGLFlM1jRApbEwA5HZO61ku45WLGC64NLDdnyGahFxW
            9EikKXcE21Jc716oRfFEeiOXaZGtJhYTjsuPr5IYJr8fWX_NC1y1x
            qYW1JnfLKklIemtznHTXvq7de6VxeSucwBklB24oafBFSeMEGBqKF
            QPhCBaAqCdJL1NkMhA9D6gqJkjluT1Aj0_Dq8ikwSP2dG3grGlrl6
            EtTuhA-j_ArrqLzvSoJU51Nu_1k3usrvgIWLZ89i8Zt05SA"
       }
     ],
     "protected":
       "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2FtZ2VlQGhvY
        mJpdG9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0",
     "iv":
       "MKccO3TLKaMB67zj",
     "ciphertext":
       "dLMYOKvvtY8Adfc8Tg4lSKElvvzBi2MJcSKDssll-jj0S_NIjI3P956z_
        qhZgJ3bVLTVPr1p8JcThDcqTZtrA4ShYpyyTrBnEzOOyyzv4h6WWiuoCJ
        TfI1Gxzm-JPJNdPYifgOS0E5RKk5L64yDtotBP9AMTZZ4WjatyAlRcgD5
        hVhP2HrqIVG0j1Sb5g5tz4fi5vmSwIhGOeOxwgLvuryjn-8ECi_5LkdUi
        C3wQP53pZRtqmwmACyq_uXvTpbUWSGna1LKaup-UC1PkOUGvHFa83Wgka
        rkJbSkA0ZoVtebBJ_XvwNlfe1VRDk8hxh0AmPvvsMiantgQ4oE6LCjOWK
        j6C4UNSWgla8zhousjSQ",
     "tag":
       "vCmVSWBtadRAKUhtizP5tw"
   }
```


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

4.3.1. Input Factors

The following are supplied before beginning the encryption process:

- o 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":   "GZy6sIZ6wl9NJOKB-jnmVQ"
    },
    {
      "kty": "oct",
      "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
      "use": "enc",
      "alg": "A256GCMKW",
      "k":   "qC57l_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.

LbIgtUgy3OYW-gpNaXZro-2naGkhnyw9NnXDrijI4EI

Figure 74: Content Encryption Key, base64url-encoded

HnJqms6_fz9N6mpsML9NHA

Figure 75: Initialization Vector, base64url-encoded

4.3.3. Encrypting the Key

The following are generated before encrypting the CEK:

- o Salt; this example uses the salt from Figure 76.

- o Iteration count; this example uses the interaction count 8192.

8Q1SzinasR3xchYz6ZZcHA

Figure 76: Salt, base64url-encoded

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

- o Password (Figure 73;

- o Salt (Figure 76), encoded as an octet string; and

- o Iteration count (8192)

produces the following encrypted key:

WY1x1MsMrbQogWOeXDasyESSjYi-3iS4p8UjlWMwNJOS0j7_KFQE0w

Figure 77: Encrypted Key, base64url-encoded

4.3.4. Encrypting the Content

The following are generated before encrypting the content:

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


```
{
  "p2s": "8Q1SzinasR3xchYz6ZZcHA",
  "p2c": 8192,
  "alg": "PBES2-HS256+A128KW",
  "cty": "JWK-SET+JSON",
  "enc": "A128CBC-HS256"
}
```


Figure 78: Protected JWE Header JSON

eyJwMnMiOiI4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJhbG
ciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJjdHkiOiJKV0stU0VUK0pTT04iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0

Figure 79: Protected JWE Header, base64url-encoded

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

- o CEK (Figure 74);
- o Initialization vector/nonce (Figure 75); and
- o Protected JWE header (Figure 79) as authenticated data

produces the following:

- o Ciphertext from Figure 80.
- o Authentication tag from Figure 81.

B39o2LfmeYhS_FiszP560P1VkHWNS6vukmQrUL2DdoQgzwz8debUcWgo1A9JXE
BUk4rr4ALHcn8wA1yRuzWOUlpk0LNBmBfrvdRpgItUQiknWa5U1KY_PqWIZKpJ
J-Gq0QTaBTsfnffUbk3BD7eillUdg3poI7EFHLsE7GN3nyuJKaCCdIkFngEekt
jM2WMUPPMuXracPftXsxJDPnUAwtCAEsShnHozPEUpMIIgWnnlM8dlofYaDewX
WySoYn321leWpLGCZVaJIEEgAttFH2iZpbb3MNV1UifDMgMCUS-Xbq4ohDcgu3
dv9xWg81PNib-GyXoFU93HN9HEblg8iZ6CfKVZ_KKvNS1oCVaoMKqPIf6Jgo-i
G4S_bblOma9esofjwIp-RU9h3fpx-taoMRvjb2pLEm1FQrYXkx5i3hfN0ESsHR
BW1WyCWnVK8M7mHJUHQqBL0FWZMKjpgWa00uZOnpZteZO4eyQKYSsBgyMRSuhF
6tceKFfxIWtclIno

Figure 80: Ciphertext, base64url-encoded

YLeY6UpSeM3dUNqg5lEu0Q

Figure 81: Authentication Tag, base64url-encoded

4.3.5. Output Results

The following compose the resulting JWE object:

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

o Authentication tag (Figure 81)

The resulting JWE object using the Compact serialization:


```
eyJwMnMiOiI4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJhbG
ciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJjdHkiOiJKV0stU0VUK0pTT04iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0
.
WY1x1MsMrbQogWOeXDasyESSjYi-3iS4p8UjlWMwNJOS0j7_KFQE0w
.
HnJqms6_fz9N6mpsML9NHA
.
B39o2LfmeYhS_FiszP560P1VkHWNS6vukmQrUL2DdoQgzwz8debUcWgo1A9JXE
BUk4rr4ALHcn8wA1yRuzWOUlpk0LNBmBfrvdRpgItUQiknWa5U1KY_PqWIZKpJ
J-Gq0QTaBTsfnffUbk3BD7eillUdg3poI7EFHLsE7GN3nyuJKaCCdIkFngEekt
jM2WMUPPMuXracPftXsxJDPnUAwtCAEsShnHozPEUpMIIgWnnlM8dlofYaDewX
WySoYn321leWpLGCZVaJIEEgAttFH2iZpbb3MNV1UifDMgMCUS-Xbq4ohDcgu3
dv9xWg81PNib-GyXoFU93HN9HEblg8iZ6CfKVZ_KKvNS1oCVaoMKqPIf6Jgo-i
G4S_bblOma9esofjwIp-RU9h3fpx-taoMRvjb2pLEm1FQrYXkx5i3hfN0ESsHR
BW1WyCWnVK8M7mHJUHQqBL0FWZMKjpgWa00uZOnpZteZO4eyQKYSsBgyMRSuhF
6tceKFfxIWtclIno
.
YLeY6UpSeM3dUNqg5lEu0Q
```


Figure 82: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
        "WY1x1MsMrbQogWOeXDasyESSjYi-3iS4p8UjlWMwNJOS0j7_KFQE0
         w"
    }
  ],
  "protected":
    "eyJwMnMiOiI4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyL
     CJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJjdHkiOiJKV0stU0VUK0
     pTT04iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "iv":
    "HnJqms6_fz9N6mpsML9NHA",
  "ciphertext":
    "B39o2LfmeYhS_FiszP560P1VkHWNS6vukmQrUL2DdoQgzwz8debUcWgo1
     A9JXEBUk4rr4ALHcn8wA1yRuzWOUlpk0LNBmBfrvdRpgItUQiknWa5U1K
     Y_PqWIZKpJJ-Gq0QTaBTsfnffUbk3BD7eillUdg3poI7EFHLsE7GN3nyu
     JKaCCdIkFngEektjM2WMUPPMuXracPftXsxJDPnUAwtCAEsShnHozPEUp
     MIIgWnnlM8dlofYaDewXWySoYn321leWpLGCZVaJIEEgAttFH2iZpbb3M
```

```
        NV1UifDMgMCUS-Xbq4ohDcgu3dv9xWg81PNib-GyXoFU93HN9HEblg8iZ
        6CfKVZ_KKvNS1oCVaoMKqPIf6Jgo-iG4S_bblOma9esofjwIp-RU9h3fp
        x-taoMRvjb2pLEm1FQrYXkx5i3hfN0ESsHRBW1WyCWnVK8M7mHJUHQqBL
        0FWZMKjpgWa00uZOnpZteZO4eyQKYSsBgyMRSuhF6tceKFfxIWtclIno",
     "tag":
       "YLeY6UpSeM3dUNqg5lEu0Q"
   }
```


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_T5fsNIodcG8G5FWPrTPMyxpzsSOGa
             QLpe2FpxBmu2",
     "y":   "A8-yxCHxkfBz3hKZfI1jUYMjUhsEveZ9THuwFjH2sCNdtksRJU7D
             5-SkgaFL1ETP",
     "d":   "iTx2pk7wW-GqJkHcEkFQb2EFyYcO7RugmaW3mRrQVAOUiPommT0I
             dnYK2xDlZh-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:

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

- o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 86

C3eS2iNXmSTA7W8tBpjs3w

Figure 85: Content Encryption Key, base64url-encoded

ubzqaTluloMJR8Ec

Figure 86: Initialization Vector, base64url-encoded

4.4.3. Encrypting the Key

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

- o Ephemeral EC private key on the same curve as the EC public key; this example uses the private key that matches the public key from Figure 87.


```
{
  "kty": "EC",
  "crv": "P-384",
  "x":   "qMz7Lgb3Bc1GNuVn4ZSxLDeDpihGWRwqA2fA1-2IJwDQtKMdpKY0
          XjNqBbjigcL-",
  "y":   "Ygt6Bc_o29f-DJ_5O3YCMoX2tXXz1ysj9MFRnucByIQoR0y3SVmq
          BBwQISq9grWe"
}
```


Figure 87: Ephemeral Elliptic Curve P-384 Key, in JWK format

Performing the key encryption operation over the CEK (Figure 85) with the following:

- o The static Elliptic Curve public key (Figure 84); and

- o The ephemeral Elliptic Curve private key (Figure 87);

produces the following JWE encrypted key:

zPCB2OMxJSGs6zA7zIYO2cUE4Yz5p7TY

Figure 88: Encrypted Key, base64url-encoded

4.4.4. Encrypting the Content

The following are generated before encrypting the content:

- o 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":   "qMz7Lgb3Bc1GNuVn4ZSxLDeDpihGWRwqA2fA1-2IJwDQtKMdpK
            Y0XjNqBbjigcL-",
    "y":   "Ygt6Bc_o29f-DJ_5O3YCMoX2tXXz1ysj9MFRnucByIQoR0y3SV
            mqBBwQISq9grWe"
  },
  "enc": "A128GCM"
}
```


Figure 89: Protected JWE Header JSON

eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InFNejdMZ2IzQmMxR051Vm40WlN4TERlRHBpaEdXUndxQTJmQT
EtMklKd0RRdEtNZHBLWTBYak5xQmJqaWdjTC0iLCJ5IjoiWWd0NkJjX28yOWYt
REpfNU8zWUNNb1gydFhYejF5c2o5TUZSbnVjQnlJUW9SMHkzU1ZtcUJCd1FJU3
E5Z3JXZSJ9LCJlbmMiOiJBMTI4R0NNIn0

Figure 90: Protected JWE Header, base64url-encoded

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

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

produces the following:

- o Ciphertext from Figure 91.

o Authentication tag from Figure 92.

zQVvyDdwBRvUVkPxQCHD0YtCihhKu462TdE4s4U8VDsCEvJ2t24YRChqKa-xC2
Ai-l1AvpPIYpwWYgwk3r9QBDTXsHbyn7FVhoVes0YAMthhmnLgbgf0_TQqG9PK
vFOki83X3aZ2PIHGcjSifIT6OQqxXE9YhdXwD0bXpkXUlq-JlnQ3pssAqQLpUV
_-4Ne6lZj4gFLunBEDGVcfhLiviyAF2BjlJG7mhToPq57d2Q99N10WfPmXVQ38
htg8thQ2qcenxi5Axd2PJXNjDsDroleU-ObLE3Bb8IJ1a04LzqB4Xmp_wgbwHC
VR-bqTKgth3h_NoDLqCxQ0QcG9E78i36iPJuLAzVgS0ChHzo5ULw

Figure 91: Ciphertext, base64url-encoded

5cJTRUT3kQRrw_UGwtMtDQ

Figure 92: Authentication Tag, base64url-encoded

4.4.5. Output Results

The following compose the resulting JWE object:

o Protected JWE header (Figure 90)

o Encrypted key (Figure 88)

o Initialization vector/nonce (Figure 86)

o Ciphertext (Figure 91)

o Authentication tag (Figure 92)

The resulting JWE object using the Compact serialization:

eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InFNejdMZ2IzQmMxR051Vm40WlN4TERlRHBpaEdXUndxQTJmQT
EtMklKd0RRdEtNZHBLWTBYak5xQmJqaWdjTC0iLCJ5IjoiWWd0NkJjX28yOWYt
REpfNU8zWUNNb1gydFhYejF5c2o5TUZSbnVjQnlJUW9SMHkzU1ZtcUJCd1FJU3
E5Z3JXZSJ9LCJlbmMiOiJBMTI4R0NNIn0
.
zPCB2OMxJSGs6zA7zIYO2cUE4Yz5p7TY
.
ubzqaTluloMJR8Ec
.
zQVvyDdwBRvUVkPxQCHD0YtCihhKu462TdE4s4U8VDsCEvJ2t24YRChqKa-xC2
Ai-l1AvpPIYpwWYgwk3r9QBDTXsHbyn7FVhoVes0YAMthhmnLgbgf0_TQqG9PK
vFOki83X3aZ2PIHGcjSifIT6OQqxXE9YhdXwD0bXpkXUlq-JlnQ3pssAqQLpUV
_-4Ne6lZj4gFLunBEDGVcfhLiviyAF2BjlJG7mhToPq57d2Q99N10WfPmXVQ38
htg8thQ2qcenxi5Axd2PJXNjDsDroleU-ObLE3Bb8IJ1a04LzqB4Xmp_wgbwHC
VR-bqTKgth3h_NoDLqCxQ0QcG9E78i36iPJuLAzVgS0ChHzo5ULw

```
.
5cJTRUT3kQRrw_UGwtMtDQ
```

Figure 93: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key":
        "zPCB2OMxJSGs6zA7zIYO2cUE4Yz5p7TY"
    }
  ],
  "protected":
    "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb
     2tAdHVja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsIm
     NydiI6IlAtMzg0IiwieCI6InFNejdMZ2IzQmMxR051Vm40WlN4TERlRHB
     paEdXUndxQTJmQTEtMklKd0RRdEtNZHBLWTBYak5xQmJqaWdjTC0iLCJ5
     IjoiWWd0NkJjX28yOWYtREpfNU8zWUNNb1gydFhYejF5c2o5TUZSbnVjQ
     nlJUW9SMHkzU1ZtcUJCd1FJU3E5Z3JXZSJ9LCJlbmMiOiJBMTI4R0NNIn
     0",
  "iv":
    "ubzqaTluloMJR8Ec",
  "ciphertext":
    "zQVvyDdwBRvUVkPxQCHD0YtCihhKu462TdE4s4U8VDsCEvJ2t24YRChqK
     a-xC2Ai-l1AvpPIYpwWYgwk3r9QBDTXsHbyn7FVhoVes0YAMthhmnLgbg
     f0_TQqG9PKvFOki83X3aZ2PIHGcjSifIT6OQqxXE9YhdXwD0bXpkXUlq-
     JlnQ3pssAqQLpUV_-4Ne6lZj4gFLunBEDGVcfhLiviyAF2BjlJG7mhToP
     q57d2Q99N10WfPmXVQ38htg8thQ2qcenxi5Axd2PJXNjDsDroleU-ObLE
     3Bb8IJ1a04LzqB4Xmp_wgbwHCVR-bqTKgth3h_NoDLqCxQ0QcG9E78i36
     iPJuLAzVgS0ChHzo5ULw",
  "tag":
    "5cJTRUT3kQRrw_UGwtMtDQ"
}
```

Figure 94: JSON Serialization

4.5. Key Agreement using ECDH-ES with AES-CBC-HMAC-SHA2

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

4.5.1. Input Factors

The following are supplied before beginning the encryption process:

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

- EC public key; this example uses the public key from Figure 95.

- "alg" parameter of "ECDH-ES"

- "enc" parameter of "A128CBC-HS256"


```
{
  "kty": "EC",
  "kid": "meriadoc.brandybuck@buckland.example",
  "use": "enc",
  "crv": "P-256",
  "x":   "XnXXKEsaUU4hPZza_zSHIbt02UA505B1rDWc7JNlcDE",
  "y":   "Md5NqzfiXCytoaMglA-9MstvgOBdMSroXA2Hb6vR6dQ",
  "d":   "44eY-VRWsn1zdz3VaWS6idEpOGt1ErydBARq7Iyh9pY"
}
```


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 examples uses the initialization vector/nonce from Figure 96.

BMbSNYW2uC7RX3xql1gbQw

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 that matches the public key from Figure 97.


```
{
  "kty": "EC",
  "crv": "P-256",
```

```
  "x":   "h_ImuH3OW5JxZNQZWIWCFTYAIigZYs1-QzsQR9tCEQ4",
  "y":   "4ZWJVVrTOWdEVbH266nb4Wy2QiwH_9XAcdpNh4S2oX0"
}
```


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:

W7j3XePj-Id6Zn71dv1b_QUQaNqJSMuxWhutlLqxLFE

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

4.5.4. Encrypting the Content

The following are generated before encrypting the content:

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


```
{
  "alg": "ECDH-ES",
  "kid": "meriadoc.brandybuck@buckland.example",
  "epk": {
    "kty": "EC",
    "crv": "P-256",
    "x":   "h_ImuH3OW5JxZNQZWIWCFTYAIigZYs1-QzsQR9tCEQ4",
    "y":   "4ZWJVVrTOWdEVbH266nb4Wy2QiwH_9XAcdpNh4S2oX0"
  },
  "enc": "A128CBC-HS256"
}
```


Figure 99: Protected JWE Header JSON


```
eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYi
LCJ4IjoiaF9JbXVIM09XNUp4Wk5RWldJV0NGVFlBSWlnWllzMS1RenNRUjl0Q0
VRNCIsInkiOiI0WldKVlZyVE9XZEVWYkgyNjZuYjRXeTJRaXdIXzlYQWNkcE5o
NFMyb1gwIn0sImVuYyI6IkExMjhDQkMtSFMyNTYifQ
```


Figure 100: Protected JWE Header, base64url-encoded

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

- o CEK (Figure 98);

o Initialization vector/nonce (Figure 96); and

o Protected JWE header (Figure 100) as authenticated data

produces the following:

o Ciphertext from Figure 101.

o Authentication tag from Figure 102.

mwSOHtsJDtD1R4Y4r0Ads9Bc8nTgk_Y4wVe_4pJsb7RERAgnfFRYRmlgjSaGPM
M7PytxfLss6clZI7YW366xh8DiqOWUavR7VFGLZIOHkrMsTPaehWlQZrQz77Ie
dSM20wSGVj-E4T0KRtX3CrZsEPjtXqNbm_EmDPgxVYTaTthGdWbyDnPMvp6eGL
T6gsMkctSLIHgaGvI2VWB0oNYdKnCRU-p2JFkLu5XQfOww4E5zKW9Xycx3mkh_
gA1dFU28Zs_boX-mm4UYseIJfaZAX_eqs7NDMpbrb29frJCFI-rYfahoVz6QhN
QXQMNmzL93pDo5QE_i9pIzR4KJu-uaItKTKNAdBKgSa9JZfc21dSw

Figure 101: Ciphertext, base64url-encoded

kqeubaGyskAjcj8mDymY6A

Figure 102: Authentication Tag, base64url-encoded

4.5.5. Output Results

The following compose the resulting JWE object:

o Protected JWE header (Figure 90)

o Initialization vector/nonce (Figure 86)

o Ciphertext (Figure 91)

o Authentication tag (Figure 92)

the resulting JWE object using the Compact serialization:

eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYi
LCJ4IjoiaF9JbXVIM09XNUp4Wk5RWldJV0NGVFlBSWlnWllzMS1RenNRUjl0Q0
VRNCIsInkiOiI0WldKVlZyVE9XZEVWYkgyNjZuYjRXeTJRaXdIXzlYQWNkcE5o
NFMyb1gwIn0sImVuYyI6IkExMjhDQkMtSFMyNTYifQ
.
.
BMbSNYW2uC7RX3xql1gbQw
.
mwSOHtsJDtD1R4Y4r0Ads9Bc8nTgk_Y4wVe_4pJsb7RERAgnfFRYRmlgjSaGPM
M7PytxfLss6clZI7YW366xh8DiqOWUavR7VFGLZIOHkrMsTPaehWlQZrQz77Ie

```
dSM20wSGVj-E4T0KRtX3CrZsEPjtXqNbm_EmDPgxVYTaTthGdWbyDnPMvp6eGL
T6gsMkctSLIHgaGvI2VWB0oNYdKnCRU-p2JFkLu5XQfOww4E5zKW9Xycx3mkh_
gA1dFU28Zs_boX-mm4UYseIJfaZAX_eqs7NDMpbrb29frJCFI-rYfahoVz6QhN
QXQMNmzL93pDo5QE_i9pIzR4KJu-uaItKTKNAdBKgSa9JZfc21dSw
.
kqeubaGyskAjcj8mDymY6A
```

Figure 103: Compact Serialization

the resulting JWE object using the JSON serialization:

```
{
  "protected":
    "eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja
     0BidWNrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2Ij
     oiUC0yNTYiLCJ4IjoiaF9JbXVIM09XNUp4Wk5RWldJV0NGVFlBSWlnWll
     zMS1RenNRUjl0Q0VRNCIsInkiOiI0WldKVlZyVE9XZEVWYkgyNjZuYjRX
     eTJRaXdIXzlYQWNkcE5oNFMyb1gwIn0sImVuYyI6IkExMjhDQkMtSFMyN
     TYifQ",
  "iv":
    "BMbSNYW2uC7RX3xql1gbQw",
  "ciphertext":
    "mwSOHtsJDtD1R4Y4r0Ads9Bc8nTgk_Y4wVe_4pJsb7RERAgnfFRYRmlgj
    SaGPMM7PytxfLss6clZI7YW366xh8DiqOWUavR7VFGLZIOHkrMsTPaehWl
    QZrQz77IedSM20wSGVj-E4T0KRtX3CrZsEPjtXqNbm_EmDPgxVYTaTthGd
    WbyDnPMvp6eGLT6gsMkctSLIHgaGvI2VWB0oNYdKnCRU-p2JFkLu5XQfOw
    w4E5zKW9Xycx3mkh_gA1dFU28Zs_boX-mm4UYseIJfaZAX_eqs7NDMpbrb
    29frJCFI-rYfahoVz6QhNQXQMNmzL93pDo5QE_i9pIzR4KJu-uaItKTKNA
    dBKgSa9JZfc21dSw",
  "tag":
    "kqeubaGyskAjcj8mDymY6A"
}
```

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.

o AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 105.

o "alg" parameter of "dir"

o "enc" parameter of "A128GCM"


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


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

4.6.2. Generated Factors

The following are generated before encrypting:

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


```
OI-fESJKbHHk1-rA
```


Figure 106: Initialization Vector, base64url-encoded

4.6.3. Encrypting the Content

The following are generated before encrypting the content:

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


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


Figure 107: Protected JWE Header JSON

Encoded as [RFC4648] base64url:

eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MT
diNWI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0

Figure 108: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

18KNUnRDhesDLn7Ec4ui6q0aptYFNkbx6Vf64wWItX7hMQe2XgbNTt-GVG_3Dz
-5mscM9bKe0TkgEecWAovlTFkuwhL-TZhbcnYdMXtaNtqYe2TEZ5fFlRiEr9is
8gBeJ7YOwazxwtE8O6FwwqeAOnR-PI8M30ODcq9B8UVzEISWu3Pf4yugvVMpLR
DxJEbyVDVr5MjiXsXpOkEdc7uUisJ0H0ygoP_mjSjUHROjh2_QVqpTUwzx2qto
3KVDj-MZehUb2FivjT7FecB3Yz-m-KhYXvXR5l5XnoqCT0ioaFzeW9zbiAMj_o
1gvWgPLv8HRD9OxMERTCwbJt4O3baG9Roz-5We10hx-sb2EKtN0g

Figure 109: Ciphertext, base64url-encoded

mBCmmmn0W0j4BS7ln3nxVA

Figure 110: Authentication Tag, base64url-encoded

4.6.4. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 108)
- Initialization vector/nonce (Figure 106)
- Ciphertext (Figure 109)
- Authentication tag (Figure 110)

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3Mi02MT
diNWI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0
.
.
OI-fESJKbHHk1-rA
.
18KNUnRDhesDLn7Ec4ui6q0aptYFNkbx6Vf64wWItX7hMQe2XgbNTt-GVG_3Dz
-5mscM9bKe0TkgEecWAovlTFkuwhL-TZhbcnYdMXtaNtqYe2TEZ5fFlRiEr9is
8gBeJ7YOwazxwtE8O6FwwqeAOnR-PI8M30ODcq9B8UVzEISWu3Pf4yugvVMpLR
DxJEbyVDVr5MjiXsXpOkEdc7uUisJ0H0ygoP_mjSjUHROjh2_QVqpTUwzx2qto
3KVDj-MZehUb2FivjT7FecB3Yz-m-KhYXvXR5l5XnoqCT0ioaFzeW9zbiAMj_o
1gvWgPLv8HRD9OxMERTCwbJt4O3baG9Roz-5We10hx-sb2EKtN0g
.
mBCmmmn0W0j4BS7ln3nxVA
```


Figure 111: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "protected":
    "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLTQ1Y2YtODY3M
     i02MTdiNWI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv":
    "OI-fESJKbHHk1-rA",
  "ciphertext":
    "18KNUnRDhesDLn7Ec4ui6q0aptYFNkbx6Vf64wWItX7hMQe2XgbNTt-GV
     G_3Dz-5mscM9bKe0TkgEecWAovlTFkuwhL-TZhbcnYdMXtaNtqYe2TEZ5
     fFlRiEr9is8gBeJ7YOwazxwtE8O6FwwqeAOnR-PI8M30ODcq9B8UVzEIS
     Wu3Pf4yugvVMpLRDxJEbyVDVr5MjiXsXpOkEdc7uUisJ0H0ygoP_mjSjU
     HROjh2_QVqpTUwzx2qto3KVDj-MZehUb2FivjT7FecB3Yz-m-KhYXvXR5
     l5XnoqCT0ioaFzeW9zbiAMj_o1gvWgPLv8HRD9OxMERTCwbJt4O3baG9R
     oz-5We10hx-sb2EKtN0g",
  "tag":
    "mBCmmmn0W0j4BS7ln3nxVA"
}
```


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 initilization vector/nonce from Figure 115.

a2gN8ASDdVKI86lMJC8rKI8RV8U8OltRlVzygIo48NA

Figure 114: Content Encryption Key, base64url-encoded

Z3wPFyzW8czy88sUmzcnlg

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.

3llIgu3y7Vu5dZW7

Figure 116: Key Wrap Initialization Vector, base64url-encoded

Performing the key encryption operation over the CEK (Figure 114) with the following:

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

produces the following:

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

cfBkmK29hCy31FM6VhHHgqbGa2DQvXZgnqSSl8zcOsE

Figure 117: Encrypted Key, base64url-encoded

7qiY1gOLorD7ro67FZqYRw

Figure 118: Key Wrap Authentication Tag, base64url-encoded

4.7.4. Encrypting the Content

The following are generated before encrypting the content:

- o 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": "7qiY1gOLorD7ro67FZqYRw",
  "iv":  "3llIgu3y7Vu5dZW7",
  "enc": "A128CBC-HS256"
}
```


Figure 119: Protected JWE Header JSON

eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiI3cWlZMWdPTG9yRDdybzY3RlpxWVJ3
IiwiaXYiOiIzbGxJZ3UzeTdWdTVkWlc3IiwiZW5jIjoiQTEyOENCQy1IUzI1Ni
J9

Figure 120: Protected JWE Header, base64url-encoded

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.


```
YSoJLPEGGMUoFM7zbKAwZivdakcAZWsyoQycpRG-4haDdLdXXGtLCev_HEs-Tu
5xRlK-4FFIQJ8l6bfSTR9glEa2FaVS8tgkZO1X9BbPAY9_4SCuLLO4n5LFK0mI
TQ8WOgpa0FTfG_1ml76MWGVtgADHGzvqSib9xoW39YsIdOu3Evj2GmvvPIm1WZ
K3HjQhQkvfKbpSFLFRkH3xsHyYYkKiH2PEOCZOzHNzc8PRMavtkBO64zmpWTfy
tMshzm0sgbroEBFU-vCHWzt5fVx_A9oUn5szL7RlkXU12fOCc7VJ2X5TtYPPr_
bM4z6KB5FBLS3hVVfHZee83e9IDrk0k7AIcf3KpfzapJmZ3kdZgOg
```


Figure 121: Ciphertext, base64url-encoded


```
d7dahIDc06hrpWqDiQzaXQ
```


Figure 122: Authentication Tag, base64url-encoded

4.7.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the Compact serialization:


```
eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiI3cWlZMWdPTG9yRDdybzY3RlpxWVJ3
IiwiaXYiOiIzbGxJZ3UzeTdWdTVkWlc3IiwiZW5jIjoiQTEyOENCQy1IUzI1Ni
J9
```

```
.
cfBkmK29hCy31FM6VhHHgqbGa2DQvXZgnqSSl8zcOsE
.
Z3wPFyzW8czy88sUmzcnlg
.
YSoJLPEGGMUoFM7zbKAwZivdakcAZWsyoQycpRG-4haDdLdXXGtLCev_HEs-Tu
5xRlK-4FFIQJ8l6bfSTR9glEa2FaVS8tgkZO1X9BbPAY9_4SCuLLO4n5LFK0mI
TQ8WOgpa0FTfG_1ml76MWGVtgADHGzvqSib9xoW39YsIdOu3Evj2GmvvPIm1WZ
K3HjQhQkvfKbpSFLFRkH3xsHyYYkKiH2PEOCZOzHNzc8PRMavtkBO64zmpWTfy
tMshzm0sgbroEBFU-vCHWzt5fVx_A9oUn5szL7RlkXU12fOCc7VJ2X5TtYPPr_
bM4z6KB5FBLS3hVVfHZee83e9IDrk0k7AIcf3KpfzapJmZ3kdZgOg
.
W7cYYn27uUYttxShJ2yYhQ
```

Figure 123: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key":
        "cfBkmK29hCy31FM6VhHHgqbGa2DQvXZgnqSSl8zcOsE"
    }
  ],
  "protected":
    "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkO
     TUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiI3cWlZMWdPTG9yRDdybz
     Y3RlpxWVJ3IiwiaXYiOiIzbGxJZ3UzeTdWdTVkWlc3IiwiZW5jIjoiQTE
     yOENCQy1IUzI1NiJ9",
  "iv":
    "Z3wPFyzW8czy88sUmzcnlg",
  "ciphertext":
    "YSoJLPEGGMUoFM7zbKAwZivdakcAZWsyoQycpRG-4haDdLdXXGtLCev_H
     Es-Tu5xRlK-4FFIQJ8l6bfSTR9glEa2FaVS8tgkZO1X9BbPAY9_4SCuLL
     O4n5LFK0mITQ8WOgpa0FTfG_1ml76MWGVtgADHGzvqSib9xoW39YsIdOu
     3Evj2GmvvPIm1WZK3HjQhQkvfKbpSFLFRkH3xsHyYYkKiH2PEOCZOzHNz
     c8PRMavtkBO64zmpWTfytMshzm0sgbroEBFU-vCHWzt5fVx_A9oUn5szL
     7RlkXU12fOCc7VJ2X5TtYPPr_bM4z6KB5FBLS3hVVfHZee83e9IDrk0k7
     AIcf3KpfzapJmZ3kdZgOg",
  "tag":
    "W7cYYn27uUYttxShJ2yYhQ"
}
```

Figure 124: JSON Serialization

4.8. Key Wrap using AES-KeyWrap with AES-GCM

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

4.8.1. Input Factors

The following are supplied before beginning the encryption process:

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


```
{
  "kty": "oct",
  "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
  "use": "enc",
  "alg": "A128KW",
  "k":   "GZy6sIZ6wl9NJOKB-jnmVQ"
}
```


Figure 125: AES 128-Bit Key

4.8.2. Generated Factors

The following are generated before encrypting:

- o AES symmetric key as the Content Encryption Key; this example uses the key from Figure 126.
- o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 127.

'Hv3Kmjt7vR2in57dLm-Pfw

Figure 126: Content Encryption Key, base64url-encoded

wxdDTG0-QnGvBZew

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:

RMMWwegPo5GY-5DeqC51gevcIOQpc4CH

Figure 128: Encrypted Key, base64url-encoded

4.8.4. Encrypting the Content

The following are generated before encrypting the content:

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


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


Figure 129: Protected JWE Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

Figure 130: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

DoM1vi13RWus_t3EsvGWk4gDH3F8TGRnBo4p3uImtmboRrT1pniDLDQTipfOin
86hMl343jhxcRObGyiKgIyPI-tG8M9E92VkHeOE8O77-s6wRj9XxxEs8zw9YPX
baILJYTbR5aWyRLpTwlEhAf5_DVL2b5vnvTNctEp5JaojvqXF5F3jkZAaJwa4u
IjhqGd7gJvf7zKbwF7Is_GbSm9rf9Z0dacH5LQQn2P_VYEb8ptUWmgz4Gg1YFF
tGg16H5JAutG9a6GqFUdkSZ-mKSothgDEHv9gnAqYnWKLaE3E2hzhxcgtNwNKf
lLSfmV247xbRYZhR8NeJ_GoKCjrH7isFvUM0Uzx43cPpPDypyiGA

Figure 131: Ciphertext, base64url-encoded

And authentication tag:

N7CDBxgruPcQozgqPeihlw

Figure 132: Authentication Tag, base64url-encoded

4.8.5. Output Results

The following compose the resulting JWE object:

- o Protected JWE header (Figure 130)
- o encrypted key (Figure 128)
- o Initialization vector/nonce (Figure 127)
- o Ciphertext (Figure 131)
- o Authentication tag (Figure 132)

The resulting JWE object using the Compact serialization:


```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0
.
RMMWwegPo5GY-5DeqC51gevcIOQpc4CH
.
wxdDTG0-QnGvBZew
.
DoM1vi13RWus_t3EsvGWk4gDH3F8TGRnBo4p3uImtmboRrT1pniDLDQTipfOin
86hMl343jhxcRObGyiKgIyPI-tG8M9E92VkHeOE8O77-s6wRj9XxxEs8zw9YPX
baILJYTbR5aWyRLpTwlEhAf5_DVL2b5vnvTNctEp5JaojvqXF5F3jkZAaJwa4u
IjhqGd7gJvf7zKbwF7Is_GbSm9rf9Z0dacH5LQQn2P_VYEb8ptUWmgz4Gg1YFF
tGg16H5JAutG9a6GqFUdkSZ-mKSothgDEHv9gnAqYnWKLaE3E2hzhxcgtNwNKf
lLSfmV247xbRYZhR8NeJ_GoKCjrH7isFvUM0Uzx43cPpPDypyiGA
.
N7CDBxgruPcQozgqPeihlw
```


Figure 133: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
        "RMMWwegPo5GY-5DeqC51gevcIOQpc4CH"
    }
```

```
    ],
    "protected":
      "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktY
       TQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
    "iv":
      "wxdDTG0-QnGvBZew",
    "ciphertext":
      "DoM1vi13RWus_t3EsvGWk4gDH3F8TGRnBo4p3uImtmboRrT1pniDLDQTi
       pfOin86hMl343jhxcRObGyiKgIyPI-tG8M9E92VkHeOE8O77-s6wRj9Xx
       xEs8zw9YPXbaILJYTbR5aWyRLpTwlEhAf5_DVL2b5vnvTNctEp5Jaojvq
       XF5F3jkZAaJwa4uIjhqGd7gJvf7zKbwF7Is_GbSm9rf9Z0dacH5LQQn2P
       _VYEb8ptUWmgz4Gg1YFFtGg16H5JAutG9a6GqFUdkSZ-mKSothgDEHv9g
       nAqYnWKLaE3E2hzhxcgtNwNKflLSfmV247xbRYZhR8NeJ_GoKCjrH7isF
       vUM0Uzx43cPpPDypyiGA",
    "tag":
      "N7CDBxgruPcQozgqPeihlw"
  }
```


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.

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

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


```
eJxtj0EOwyAMBL-yD0jyh_bQL1Q9EmICCsKVMYry-zpEPVTqycizu15e3OBdgU
qrilahjKrJb9iTRhyGNQq3Ndo8164s56uMoyk1EuakSgIqy4SbweNP4kb0NueB
Sl5IweGUSR1Hn7maW6M5TmdXJr0w5TDh3vSbWVh_YjNdKDhPBrjNmeAyFxp6z5
XtUug_YUtx2Fms5dNEQv0EgiRrXgc8hBeePt1uYQ0
```


Figure 135: Compressed Plaintext, base64url-encoded


```
03x4Y5d1Lk0K2VbMmePmMw
```


Figure 136: Content Encryption Key, base64url-encoded


```
9UnGd0z8-Yx3BFV3
```


Figure 137: Initialization Vector, base64url-encoded

4.9.3. Encrypting the Key

Performing the key encryption operation over the CEK (Figure 136) with the AES key ({{jwe-aeskw-key) produces the following encrypted key:


```
fUGYywsd6dWWr5JpNP0EMMN2XkXk8_h5
```


Figure 138: Encrypted Key, base64url-encoded

4.9.4. Encrypting the Content

The following are generated before encrypting the content:

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


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


Figure 139: Protected JWE Header JSON

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiemlwIjoiREVGIn0

Figure 140: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

b962BmHXeA9iYY8u9GnpxtXnme1MNm7vhhBqcxJHof08hGn1ltC7Mpf0dnlB0y
ZqAlqBWDJrGs3eVseTlFEFm0pDHDlIven74xwZPdJdEylDKPTeZLaCf6TjK46C
UfWJBajPZ2wiupjQJb5FYz_1KsWYCXAE4k6xt9v5wkwm_FHpYevNXuE7hokcim
LRiCi1R_xjnG7sHCTWKb56L0ZsqacWn_52jkO9B3Q

Figure 141: Ciphertext, base64url-encoded

And authentication tag:

Im5q_DU2ZafibIuj5GNI5Q

Figure 142: Authentication Tag, base64url-encoded

4.9.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 140)
- encrypted key (Figure 138)
- Initialization vector/nonce (Figure 137)
- Ciphertext (Figure 141)
- Authentication tag (Figure 142)

The resulting JWE object using the Compact serialization:


```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiemlwIjoiREVGIn0
.
fUGYywsd6dWWr5JpNP0EMMN2XkXk8_h5
.
9UnGd0z8-Yx3BFV3
.
b962BmHXeA9iYY8u9GnpxtXnme1MNm7vhhBqcxJHof08hGn1ltC7Mpf0dnlB0y
ZqAlqBWDJrGs3eVseTlFEFm0pDHDlIven74xwZPdJdEylDKPTeZLaCf6TjK46C
UfWJBajPZ2wiupjQJb5FYz_1KsWYCXAE4k6xt9v5wkwm_FHpYevNXuE7hokcim
LRiCi1R_xjnG7sHCTWKb56L0ZsqacWn_52jkO9B3Q
.
Im5q_DU2ZafibIuj5GNI5Q
```


Figure 143: Compact Serialization

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
        "fUGYywsd6dWWr5JpNP0EMMN2XkXk8_h5"
    }
  ],
  "protected":
    "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktY
     TQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIiwiemlwIjoiRE
     VGIn0",
  "iv":
    "9UnGd0z8-Yx3BFV3",
  "ciphertext":
    "b962BmHXeA9iYY8u9GnpxtXnme1MNm7vhhBqcxJHof08hGn1ltC7Mpf0d
     nlB0yZqAlqBWDJrGs3eVseTlFEFm0pDHDlIven74xwZPdJdEylDKPTeZL
     aCf6TjK46CUfWJBajPZ2wiupjQJb5FYz_1KsWYCXAE4k6xt9v5wkwm_FH
     pYevNXuE7hokcimLRiCi1R_xjnG7sHCTWKb56L0ZsqacWn_52jkO9B3Q",
  "tag":
    "Im5q_DU2ZafibIuj5GNI5Q"
}
```


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:

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

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

Figure 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:

- o 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.

uGL3QU7R3HMR3ik-oTW82w

Figure 146: Content Encryption Key, base64url-encoded

HorZstLCLfNNC7TN

Figure 147: Initialization Vector, base64url-encoded

WyJ2Y2FyZCIsW1sidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxbImZuIix7fS
widGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4iLHt9LCJ0ZXh0Iixb
IkJyYW5keWJ1Y2siLCJNZXJpYWRvYyIsIk1yLiIsIiJdXSxbImJkYXkiLHt9LC
J0ZXh0IiwiVEEgMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV4dCIsIk0iXV1d

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:

MJjYoJ6DKa__0KTJP5PT8pR0T_tybLRc

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

eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0

Figure 151: Protected JWE Header, base64url-encoded

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.

36qblaXJa6XlM7EHkAWVcrAvUA-w0zUsaSiK9ajj1CsPp-oHpElk7bktsA2u9p
b_T0yeXpjeaGKc0tWO6VKMIpIEJed-reIzaHva_JrHKt63tKWRmGDtQ9EHDCgw
Vv_0EwUoVW_RzfugR-71IsoTSYeziVi2XL_nsHpcVGFQOgD2C-nvwqo4_8f9pZ
_bmK_kj0eAc54qp2laNG7odWGOSpOvW4Vr2ujW8QnHQlaKUNUqh0ODvCu0hFWN
pzxEgja4X6UlSkY6uTQR-mBBpwlA4rAnjP-pn0zuq0T13vkCplokt2GKhRLysE
6UqLjnyfexHGjC349nzsBHoCXk2tKJwrqPpssCnsqPaffU

Figure 152: Ciphertext, base64url-encoded

tp_Idm6BMHn3iJQ86T4sRA

Figure 153: Authentication Tag, base64url-encoded

4.10.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 151)
- encrypted key (Figure 149)
- Initialization vector/nonce (Figure 147)
- Additional authenticated data (Figure 148)
- Ciphertext (Figure 152)
- Authentication tag (Figure 153)

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key":
        "Aa2ArPkcYIHxdlA3lsGWtcC9sBkqTYHr"
    }
  ],
  "protected":
    "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktY
     TQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv":
    "HorZstLCLfNNC7TN",
  "aad":
    "WyJ2Y2FyZCIsW1sidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxbImZuI
    ix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4iLHt9LCJ
    0ZXh0IixbIkJyYW5keWJ1Y2siLCJNZXJpYWRvYyIsIk1yLiIsIiJdXSxbI
    mJkYXkiLHt9LCJ0ZXh0IiwiVEEgMjk4MiJdLFsiZ2VuZGVyIix7fSwidGV
    4dCIsIk0iXV1d",
  "ciphertext":
    "36qblaXJa6XlM7EHkAWVcrAvUA-w0zUsaSiK9ajj1CsPp-oHpElk7bkts
     A2u9pb_T0yeXpjeaGKc0tWO6VKMIpIEJed-reIzaHva_JrHKt63tKWRmG
     DtQ9EHDCgwVv_0EwUoVW_RzfugR-71IsoTSYeziVi2XL_nsHpcVGFQOgD
     2C-nvwqo4_8f9pZ_bmK_kj0eAc54qp2laNG7odWGOSpOvW4Vr2ujW8QnH
     QlaKUNUqh0ODvCu0hFWNpzxEgja4X6UlSkY6uTQR-mBBpwlA4rAnjP-pn
     0zuq0T13vkCplokt2GKhRLysE6UqLjnyfexHGjC349nzsBHoCXk2tKJwr
     qPpssCnsqPaffU",
  "tag":
    "tp_Idm6BMHn3iJQ86T4sRA"
}
```


Figure 154: JSON Serialization

4.11. Protecting Specific Header Fields

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

4.11.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

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

4.11.2. Generated Factors

The following are generated before encrypting:

o AES symmetric key as the Content Encryption Key (CEK); this example uses the key from Figure 155.

o Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 156.

uGL3QU7R3HMR3ik-oTW82w

Figure 155: Content Encryption Key, base64url-encoded

HorZstLCLfNNC7TN

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:

MJjYoJ6DKa__0KTJP5PT8pR0T_tybLRc

Figure 157: Encrypted Key, base64url-encoded

4.11.4. Encrypting the Content

The following are generated before encrypting the content:

o 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:

- o CEK (Figure 155);
- o Initialization vector/nonce (Figure 156); and
- o Protected JWE header (Figure 159) as authenticated data

produces the following:

- o Ciphertext from Figure 160.
- o Authentication tag from Figure 161.


```
XR98Or5bnT-qBoQ0-K8WbR6hphUsSoJZdE6W0c3CYJ2kSk-6NuycqF4ZrKy6YC
-Gs3jfCwkCmW955kmDgTIlc-fSQ-w__kwrM8wepy1h61OeY2HCM8-vJpK3yHcW
HrbJhuqJExRWlnR6l9y9kcAzc3F1YWBJ5B5uY0PvbbbiQnRL5d9VFxKZFjF8qT
a6T1OsXR22bKkb-oG8JWSiefhxZlGQCpqRUfmYDRMZhfakIa1hXVDaLZIapkka
gpw510A5yt0E5W8qkcCrezJZtCSFeHTJFqspCcnTvdfDqkGVQwo1cxKr0Wu-2K
wb3dP8TlZJ2dMSOxXyMQav1nqZcfKn2qg4xI87D5mhfCrW
```


Figure 160: Ciphertext, base64url-encoded


```
Hjccc2tFAQi12LH6FF-jFA
```


Figure 161: Authentication Tag, base64url-encoded

4.11.5. Output Results

The following compose the resulting JWE object:

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

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

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


Figure 162: Unprotected JWE Header JSON

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
        "MJjYoJ6DKa__0KTJP5PT8pR0T_tybLRc"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected":
    "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv":
    "HorZstLCLfNNC7TN",
  "ciphertext":
    "XR98Or5bnT-qBoQ0-K8WbR6hphUsSoJZdE6W0c3CYJ2kSk-6NuycqF4Zr
     Ky6YC-Gs3jfCwkCmW955kmDgTIlc-fSQ-w__kwrM8wepy1h61OeY2HCM8
     -vJpK3yHcWHrbJhuqJExRWlnR6l9y9kcAzc3F1YWBJ5B5uY0PvbbbiQnR
     L5d9VFxKZFjF8qTa6T1OsXR22bKkb-oG8JWSiefhxZlGQCpqRUfmYDRMZ
     hfakIa1hXVDaLZIapkkagpw510A5yt0E5W8qkcCrezJZtCSFeHTJFqspC
     cnTvdfDqkGVQwo1cxKr0Wu-2Kwb3dP8TlZJ2dMSOxXyMQav1nqZcfKn2q
     g4xI87D5mhfCrW",
  "tag":
    "Hjccc2tFAQi12LH6FF-jFA"
}
```


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:

- o Plaintext content; this example uses the content from Figure 51.
- o Recipient encryption key; this example uses the key from Figure 125.
- o Key encryption algorithm; this example uses "A128KW".
- o Content encryption algorithm; this example uses "A128GCM".

4.12.2. Generated Factors

The following are generated before encrypting:

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

5UVirgqilMhbWpSnM3alUQ

Figure 164: Content Encryption Key, base64url-encoded

zdbIl4BrrziYK55_

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:

yyuirCy7Hd_nY0gL5Jfq6sJ7RXRR0DtF

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:

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

produces the following:

o Ciphertext from Figure 167.

o Authenticated data from Figure 168.

3MtsMr7GhYafTv6KNiWMEg5vl4tE2FHfmvfxhTJnioynNBD7G6LEEI6uLDHK-p
A2vINROgEAEiN9srAPN2qxl1kxJs4FBBin21pErXalJF_yqotv5OX-sXpyMSd2
X4peV29PRKVI2gaeVH8QjhuV5ar1UYaFW9qTqxwsN_NrbN8x709Exvhl3LoX6H
5XH9KFAc0nEk_AXvAvtYbq3GpWu3OONrXQuq6Oy7LCvBwCjlSKUEMR094sPim5
GVB7p_CX_xDuWGkPiaCTru0qJOfPjIbzzAjnf5m4Nw9kB1bMmYl4k_nvBSbUa1
-ybdYyGcK1ldGbWzYsCYZFII4DmK8rXHYDHRA1jR8StGEo

Figure 167: Ciphertext, base64url-encoded

0qbCArWBoY_iqVMwfjNC4Q

Figure 168: Authentication Tag, base64url-encoded

4.12.5. Output Results

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:

o Unprotected JWE header (Figure 169)

o encrypted key (Figure 166)

o Initialization vector/nonce (Figure 165)

o Ciphertext (Figure 167)

o Authentication tag (Figure 168)

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key":
```

```
          "yyuirCy7Hd_nY0gL5Jfq6sJ7RXRR0DtF"
        }
      ],
      "unprotected": {
        "alg": "A128KW",
        "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
        "enc": "A128GCM"
      },
      "iv":
        "zdbIl4BrrziYK55_",
      "ciphertext":
        "3MtsMr7GhYafTv6KNiWMEg5vl4tE2FHfmvfxhTJnioynNBD7G6LEEI6uL
         DHK-pA2vINROgEAEiN9srAPN2qxl1kxJs4FBBin21pErXalJF_yqotv5O
         X-sXpyMSd2X4peV29PRKVI2gaeVH8QjhuV5ar1UYaFW9qTqxwsN_NrbN8
         x709Exvhl3LoX6H5XH9KFAc0nEk_AXvAvtYbq3GpWu3OONrXQuq6Oy7LC
         vBwCjlSKUEMR094sPim5GVB7p_CX_xDuWGkPiaCTru0qJOfPjIbzzAjnf
         5m4Nw9kB1bMmYl4k_nvBSbUa1-ybdYyGcK1ldGbWzYsCYZFII4DmK8rXH
         YDHRA1jR8StGEo",
      "tag":
        "0qbCArWBoY_iqVMwfjNC4Q"
    }
```


Figure 170: JSON Serialization

4.13. Encrypting to Multiple Recipients

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

4.13.1. Input Factors

The following are supplied before beginning the encryption process:

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

- Recipient keys; this example uses the following:

  * The RSA public key from Figure 52 for the first recipient.

  * The EC public key from Figure 84 for the second recipient.

  * The AES symmetric key from Figure 113 for the third recipient.

- Key encryption algorithms; this example uses the following:

  * "RSA1_5" for the first recipient.

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

   * "A256GCMKW" for the third recipient.

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

4.13.2. Generated Factors

The following are generated before encrypting:

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

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

OYs79m0f3LEuMZzmWBCywRn4u8B09BVidJb9j0ojDsY

Figure 171: Content Encryption Key, base64url-encoded

nY-xFgdef1LrsU7u

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:

EBbDunXtz-j0Gn0q4c9vtueHlb0E-oBkSMno9PUg8eR7Y5T71aU9t0JkvxtKiO
xibNkeeUSYPLPGvxslWAYgmqYW--uP_R64hQFp2fcB5MnyQ69GBkMU6Poie-Ct
Q2y9Z3Mv9-NMbT7LO99A_2EUEXuxzGnHYSftk7KKjyw38LBuvSOVdokkHWMP4p
VLeUJB1ovbT4M1j3pxUzyM2426sD6LfjorQhY8vsChyDaFST0Oe8uBvcRyA5ma
bEyRYlUet8PRH_CjINMipv7LCDRsKVnr3oHwZEfCJFGNC-w_-Qn4xcmkmxyaz1
-kZEpS_t2kWJsFqx3mg7QAXJBxdGmy6A

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

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

o Recipient JWE header from Figure 174

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

Figure 174: Recipient #1 JWE Header JSON

The following is the assembled first recipient JSON:


```
{
  "encrypted_key":
    "EBbDunXtz-j0Gn0q4c9vtueHlb0E-oBkSMno9PUg8eR7Y5T71aU9t0Jkv
     xtKiOxibNkeeUSYPLPGvxslWAYgmqYW--uP_R64hQFp2fcB5MnyQ69GBk
     MU6Poie-CtQ2y9Z3Mv9-NMbT7LO99A_2EUEXuxzGnHYSftk7KKjyw38LB
     uvSOVdokkHWMP4pVLeUJB1ovbT4M1j3pxUzyM2426sD6LfjorQhY8vsCh
     yDaFST0Oe8uBvcRyA5mabEyRYlUet8PRH_CjINMipv7LCDRsKVnr3oHwZ
     EfCJFGNC-w_-Qn4xcmkmxyaz1-kZEpS_t2kWJsFqx3mg7QAXJBxdGmy6A",
  "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:

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


```
{
  "kty": "EC",
  "crv": "P-384",
  "x":   "-fcIsKguXqJTTsWdHbJ4iRY_xARz9O_JdAxVEJnrxo7sJRbkwH
          mwlMi4AxCVzG_I",
  "y":   "JcuN_3pD0dbTjry07BoFoJ-_W-SumUHH9XIAvXkKeFkJV4s5GM
          CwXwxtgkNeZ32T"
}
```


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:

Dd1kQYNhhSIlKEAyE9UYhjtUBGahteiYVnRUg_tWf8S9VJZKL_8YOw

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

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

o Recipient JWE Header from Figure 178.


```
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x":   "-fcIsKguXqJTTsWdHbJ4iRY_xARz9O_JdAxVEJnrxo7sJRbkwH
            mwlMi4AxCVzG_I",
    "y":   "JcuN_3pD0dbTjry07BoFoJ-_W-SumUHH9XIAvXkKeFkJV4s5GM
            CwXwxtgkNeZ32T"
  }
}
```


Figure 178: Recipient #2 JWE Header JSON

The following is the assembled second recipient JSON:


```
{
  "encrypted_key":
    "Dd1kQYNhhSIlKEAyE9UYhjtUBGahteiYVnRUg_tWf8S9VJZKL_8YOw",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x":   "-fcIsKguXqJTTsWdHbJ4iRY_xARz9O_JdAxVEJnrxo7sJRbk
              wHmwlMi4AxCVzG_I",
      "y":   "JcuN_3pD0dbTjry07BoFoJ-_W-SumUHH9XIAvXkKeFkJV4s5
              GMCwXwxtgkNeZ32T"
    }
  }
}
```


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:

o Initialization vector/nonce for key wrapping; this example uses the initialization vector/nonce from {{jwe-multi-kwiv_3}

kZtitxRDXfzCS6ZK

Figure 180

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

o AES symmetric key (Figure 113; and

o Initialization vector/nonce ((Figure 180

produces the following:

o Encrypted key from Figure 181.

o Key wrap authentication tag from Figure 182

iiVL4XCDCnsWCSZCTysGxl41vdnJqIThbumNa9wSQBo

Figure 181: Recipient #3 Encrypted Key, base64url-encoded

DOVpODvbotRWOHEqTRcXkg

Figure 182: Recipient #3 Encrypted Key, base64url-encoded

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

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

```
{
  "alg": "A256GCMKW",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "tag": "DOVpODvbotRWOHEqTRcXkg",
  "iv":  "kZtitxRDXfzCS6ZK"
}
```

Figure 183: Recipient #3 JWE Header JSON

The following is the assembled third recipient JSON:

```
{
  "encrypted_key":
    "DOVpODvbotRWOHEqTRcXkg",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "DOVpODvbotRWOHEqTRcXkg",
    "iv":  "kZtitxRDXfzCS6ZK"
  }
```


Figure 184: Recipient #3 JSON

4.13.6. Encrypting the Content

The following are generated before encrypting the content:

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


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


Figure 185: Protected JWE Header JSON

eyJlbmMiOiJBMTI4R0NNIn0

Figure 186: Protected JWE Header, base64url-encoded

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

o CEK (Figure 171),

o Initialization vector/nonce (Figure 172), and

o Protected JWE header (Figure 186) as the authenticated data

produces the following:

o Ciphertext from Figure 187

o Authentication tag from Figure 188


```
aG6vvrUIPIE5AunujYfPvgO1ypah6leCfYeW721swK9Nr8ERrKJn-HFkEkcx2r
HnLgp33hKX6jPBWlSwilwGl2e2xg3SxQiA9OYncXBkpUcUK4KoIg7qCvtTsVFp
sVRJYTBDqpGuecYdYOeZPWUuB1vX4jrCFIpHh3BIraAE6iTxdmxhHP-OXGZQpU
N4Y2qcromUQP2jSreVGp2Gn9b4bWELfLny4WqRVmB_bySnyUxdglzGAQEse7s_
```

o1s_6i1fOZnB5WzcoNo2aTZIKWLjJ347XL95KcF9aYwMAZSi7N4n41Zs2Yaa8-
u07LpV9fQ7ubDQj1fQ4clpxPv_IDbHJ3tgdlH2lWSHwZADwgpIOA

Figure 187: Ciphertext, base64url-encoded

ESZx8edqbU4osp8P8H0a5Q

Figure 188: Authentication Tag, base64url-encoded

The following is generated after encrypting the plaintext:

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

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

Figure 189: Unprotected JWE Header JSON

4.13.7. Output Results

The following compose the resulting JWE object:

o Recipient #1 JSON (Figure 175)

o Recipient #2 JSON (Figure 179)

o Recipient #3 JSON (Figure 184)

o Initialization vector/nonce (Figure 172)

o Ciphertext (Figure 187)

o Authentication tag (Figure 188)

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key":
        "EBbDunXtz-j0Gn0q4c9vtueHlb0E-oBkSMno9PUg8eR7Y5T71aU9t
         0JkvxtKiOxibNkeeUSYPLPGvxslWAYgmqYW--uP_R64hQFp2fcB5M
         nyQ69GBkMU6Poie-CtQ2y9Z3Mv9-NMbT7LO99A_2EUEXuxzGnHYSf
         tk7KKjyw38LBuvSOVdokkHWMP4pVLeUJB1ovbT4M1j3pxUzyM2426
         sD6LfjorQhY8vsChyDaFST0Oe8uBvcRyA5mabEyRYlUet8PRH_CjI
         NMipv7LCDRsKVnr3oHwZEfCJFGNC-w_-Qn4xcmkmxyaz1-kZEpS_t
```

```
            2kWJsFqx3mg7QAXJBxdGmy6A",
          "header": {
            "alg": "RSA1_5",
            "kid": "frodo.baggins@hobbiton.example"
          }
        },
        {
          "encrypted_key":
            "Dd1kQYNhhSIlKEAyE9UYhjtUBGahteiYVnRUg_tWf8S9VJZKL_8YO
             w",
          "header": {
            "alg": "ECDH-ES+A256KW",
            "kid": "peregrin.took@tuckborough.example",
            "epk": {
              "kty": "EC",
              "crv": "P-384",
              "x":
                "-fcIsKguXqJTTsWdHbJ4iRY_xARz9O_JdAxVEJnrxo7sJRbkw
                 HmwlMi4AxCVzG_I",
              "y":
                "JcuN_3pD0dbTjry07BoFoJ-_W-SumUHH9XIAvXkKeFkJV4s5G
                 MCwXwxtgkNeZ32T"
            }
          }
        },
        {
          "encrypted_key":
            "iiVL4XCDCnsWCSZCTysGxl41vdnJqIThbumNa9wSQBo",
          "header": {
            "alg": "A256GCMKW",
            "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
            "tag": "DOVpODvbotRWOHEqTRcXkg",
            "iv":  "kZtitxRDXfzCS6ZK"
          }
        }
      ],
      "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
      "unprotected": {
        "cty": "text/plain"
      },
      "iv":
        "nY-xFgdef1LrsU7u",
      "ciphertext":
        "aG6vvrUIPIE5AunujYfPvgO1ypah6leCfYeW721swK9Nr8ERrKJn-HFkE
         kcx2rHnLgp33hKX6jPBWlSwilwGl2e2xg3SxQiA9OYncXBkpUcUK4KoIg
         7qCvtTsVFpsVRJYTBDqpGuecYdYOeZPWUuB1vX4jrCFIpHh3BIraAE6iT
         xdmxhHP-OXGZQpUN4Y2qcromUQP2jSreVGp2Gn9b4bWELfLny4WqRVmB_
         bySnyUxdglzGAQEse7s_o1s_6i1fOZnB5WzcoNo2aTZIKWLjJ347XL95K
```

```
        cF9aYwMAZSi7N4n41Zs2Yaa8-u07LpV9fQ7ubDQj1fQ4clpxPv_IDbHJ3
        tgdlH2lWSHwZADwgpIOA",
     "tag":
       "ESZx8edqbU4osp8P8H0a5Q"
   }
```


Figure 190: JSON Serialization

5. 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].

6. IANA Considerations

This document has no actions for IANA.

7. Informative References

[I-D.ietf-jcardcal-jcard]
Kewisch, P., "jCard: The JSON format for vCard", draft-ietf-jcardcal-jcard-07 (work in progress), October 2013.

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-18 (work in progress), November 2013.

[I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-18 (work in progress), November 2013.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-18 (work in progress), November 2013.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-18 (work in progress), November 2013.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

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.

Author's Address

Matthew Miller
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

Phone: +1-303-308-3204
Email: mamille2@cisco.com