JOSE Working Group
Internet-Draft
Intended status: Informational
Expires: January 23, 2015

M. Miller
Cisco Systems, Inc.

July 22, 2014

Examples of Protecting Content using JavaScript Object Signing and Encryption (JOSE)
draft-ietf-jose-cookbook-03

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 23, 2015.

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 encrypt and/or sign content using a variety of algorithms. 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

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

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 JWE plaintext or JWS payload content, whitespace is not part of the cryptographic operations. Unless otherwise noted, the JWE plaintext or JWS payload content does include " " (U+0020 SPACE) characters; line breaks (U+000A LINE FEED) are added to improve readability but are not present in the JWE plaintext or JWS payload.

2. Terminology

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

3. JSON Web Signature Examples

The following sections demonstrate how to generate various JWS objects.

All of the succeeding examples use the following payload plaintext, serialized as UTF-8; the sequence "\xe2\x80\x99" substituted for (U+2019 RIGHT SINGLE QUOTATION MARK), and line breaks (U+000A LINE FEED) replacing some " " (U+0020 SPACE) to improve readability:

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

Figure 1: Payload content plaintext

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

```
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
```

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_KuKPEHLd4rHVTeT
      -O-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHz_AJmSCpMaJMRBSFKrKb2wqV
      wGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqkHHig3RBordaZ6Aj-
      oBHqFEHYpPe7Tpe-OfVfHd1E6cS6M1FZcD1NNLYD5lFHpPI9bTwJlsde
      3uhGqC0ZCuEHg8lhzwOHrtIQbS0FVbb9k3-tVTU4fg_3L_vniUFAKwuC
      LqKnS2BYwdq_mzSnbLY7h_qixoR7jig3__kRhuaxwUkRz5iaiQkqgc5g
      HdrNP5zw",
  "e": "AQAB",
  "d": "bWUC9B-EFRIo8kpGfh0ZuyGPvMNKvYWNtB_ikiH9k20eT-O1q_I78e
      iZkpXxXQ0UTEs2LsNRS-8uJbvQ-A1irkwMSMkK1J3XTGgdrhCku9gRld
      Y7sNA_AKZGh-Q661_42rINLRCe8W-nZ34ui_qOfkLnK9QWDDqpaIsA-b
      MwWWSDFu2MUBYwkHTMEzLYGqOe04noqeq1hExBTHBOBdkMXiuFhUq1BU
      6l-DqEiWxqg82sXt2h-LMnT3046AOYJoRioz75tSUQfGCshWTBnP5uDj
      d18kKhyv07lhfSJdrPdM5Plyl21hsFf4L_mHCuoFau7gdsPfHPxxjVOc
      OpBrQzwQ",
  "p": "3Slxg_DwTXJcb6095RoXygQCAZ5RnAvZlno1yhHtnUex_fp7AZ_9nR
      aO7HX_-SFfGQeutao2TDjDAWU4Vupk8rw9JR0AzZ0N2fvuIAmr_WCsmG
      peNqQnev1T7IyEsnh8UMt-n5CafhkikzhEsrmndH6LxOrvRJlsPp6Zv8
      bUq0k",
  "q": "uKE2dh-cTf6ERF4k4e_jy78GfPYUIaUyoSSJuBzp3Cubk3OCqs6grT
      8bR_cu0Dm1MZwWmtdqDyI95HrUeq3MP15vMMON8lHTeZu2lmKvwqW7an
      V5UzhM1iZ7z4yMkuUwFWoBvyY898EXvRD-hdqRxHlSqAZ192zB3pVFJ0
      s7pFc",
  "dp": "B8PVvXkvJrj2L-GYQ7v3y9r6Kw5g9SahXBwsWUzp19TVlgI-YV85q
      1NIb1rxQtD-IsXXR3-TanevuRPRt5OBOdiMGQp8pbt26gljYfKU_E9xn
      -RULHz0-ed9E9gXLKD4VGngpz-PfQ_q29pk5xWHoJp009Qf1HvChixRX
      59ehik",
  "dq": "CLDmDGduhylc9o7r84rEUVn7pzQ6PF83Y-iBZx5NT-TpnOZKF1pEr
      AMVeKzFEl41DlHHqqBLSM0W1sOFbwTxYWZDm6sI6og5iTbwQGIC3gnJK
      bi_7k_vJgGHwHxgPaX2PnvP-zyEkDERuf-ry4c_Z11Cq9AqC2yeL6kdK
      T1cYF8",
  "qi": "3PiqvXQN0zwMeE-sBvZgi289XP9XCQF3VWqPzMKnIgQp7_Tugo6-N
      ZBKCQsMf3HaEGBjTVJs_jcK8-TRXvaKe-7ZMaQj8VfBdYkssbu0NKDDh
      jJ-GtiseaDVWt7dcH0cfwxgFUHpQh7FoCrjFJ6h6ZEpMF6xmujs4qMpP
      z8aaI4"
}
```


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

3.1.2. Signing Operation

The following are generated to complete the signing operation:

- 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:

```
MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmK
ZopdHM1O2U4mwzJdQx996ivp83xuglII7PNDi84wnB-BDkoBwA78185hX-Es4J
IwmDLJK3lfWRa-XtL0RnltuYv746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8w
W1Kt9eRo4QPocSadnHXFxnt8Is9UzpERV0ePPQdLuW3IS_de3xyIrDaLGdjluP
xUAhb6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_f
cIe8u9ipH84ogoree7vjbU5y18kDquDg
```

Figure 6: Signature, base64url-encoded

3.1.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the Compact serialization:

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.
MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHoxnW2e5CZ5NlKtainoFmK
ZopdHM1O2U4mwzJdQx996ivp83xuglII7PNDi84wnB-BDkoBwA78185hX-Es4J
IwmDLJK3lfWRa-XtL0RnltuYv746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8w
W1Kt9eRo4QPocSadnHXFxnt8Is9UzpERV0ePPQdLuW3IS_de3xyIrDaLGdjluP
xUAhb6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJushZ41Axf_f
cIe8u9ipH84ogoree7vjbU5y18kDquDg
```

Figure 7: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImJpbGJvLmJhZ2
          dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "MRjdkly7_-oTPTS3AXP41iQIGKa80A0ZmTuV5MEaHo
          xnW2e5CZ5NlKtainoFmKZopdHM1O2U4mwzJdQx996ivp83xuglII
          7PNDi84wnB-BDkoBwA78185hX-Es4JIwmDLJK3lfWRa-XtL0Rnlt
          uYv746iYTh_qHRD68BNt1uSNCrUCTJDt5aAE6x8wW1Kt9eRo4QPo
          cSadnHXFxnt8Is9UzpERV0ePPQdLuW3IS_de3xyIrDaLGdjluPxU
          Ahb6L2aXic1U12podGU0KLUQSE_oI-ZnmKJ3F4uOZDnd6QZWJush
          Z41Axf_fcIe8u9ipH84ogoree7vjbU5y18kDquDg"
    }
  ]
}
```

Figure 8: JSON Serialization

3.2. RSA-PSS Signature

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

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

3.2.1. Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the content from Figure 1, encoded using [RFC4648] base64url to produce Figure 2.
- RSA private key; this example uses the key from Figure 3.
- "alg" parameter of "PS384".

3.2.2. Signing Operation

The following are generated to complete the signing operation:

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

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

Figure 9: Protected JWS Header JSON

```
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:

```
cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
pN6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDItllVo6_1OLPpcbUrhiUSMxbbXU
vdvWXzg-UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRX
e8P_ijQ7p8Vdz0TTrxUeT3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT
0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQSUmFAAjVKPbByi7xho0uTOcbH510a
6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw
```

Figure 11: Signature, base64url-encoded

3.2.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the Compact serialization:

```
eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.
cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
pN6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDItllVo6_1OLPpcbUrhiUSMxbbXU
vdvWXzg-UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRX
e8P_ijQ7p8Vdz0TTrxUeT3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT
0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQSUmFAAjVKPbByi7xho0uTOcbH510a
6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw
```

Figure 12: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2
          dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy
          42miAh2qyBzk1xEsnk2IpN6-tPid6VrklHkqsGqDqHCdP6O8TTB5
          dDDItllVo6_1OLPpcbUrhiUSMxbbXUvdvWXzg-UD8biiReQFlfz2
          8zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRXe8P_ijQ7p8Vd
          z0TTrxUeT3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0q
          I0n6uiP1aCN_2_jLAeQTlqRHtfa64QQSUmFAAjVKPbByi7xho0uT
          OcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw"
    }
  ]
}
```


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.

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

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_oJXu9
      A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
  "y": "AdymlHvOiLxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV5OhQHiraVy
      SsUdaQkAgDPrwQrJmbnX9cwlGfP-HqHZR1",
  "d": "AAhRON2r9cqXX1hg-RoI6R1tX5p2rUAYdmpHZoC1XNM56KtscrX6zb
      KipQrCW9CGZH3T4ubpnoTKLDYJ_fF3_rJt"
}
```

Figure 14: Elliptic Curve P-521 Private Key

3.3.2. Signing Operation

The following are generated before beginning the signature process:

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

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

Figure 15: Protected JWS Header JSON

```
eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
```

Figure 16: Protected JWS Header, base64url-encoded

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

```
AE_R_YZCChjn4791jSQCrdPZCNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvb
u9Plon7KRTzoNEuT4Va2cmL1eJAQy3mtPBu_u_sDDyYjnAMDxXPn7XrT0lw-kv
AD890jl8e2puQens_IEKBpHABlsbEPX6sFY8OcGDqoRuBomu9xQ2
```

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
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.
AE_R_YZCChjn4791jSQCrdPZCNYqHXCTZH0-JZGYNlaAjP2kqaluUIIUnC9qvb
u9Plon7KRTzoNEuT4Va2cmL1eJAQy3mtPBu_u_sDDyYjnAMDxXPn7XrT0lw-kv
AD890jl8e2puQens_IEKBpHABlsbEPX6sFY8OcGDqoRuBomu9xQ2
```

Figure 18: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJFUzUxMiIsImtpZCI6ImJpbGJvLmJhZ2
          dpbnNAaG9iYml0b24uZXhhbXBsZSJ9",
      "signature": "AE_R_YZCChjn4791jSQCrdPZCNYqHXCTZH0-JZGYNl
          aAjP2kqaluUIIUnC9qvbu9Plon7KRTzoNEuT4Va2cmL1eJAQy3mt
          PBu_u_sDDyYjnAMDxXPn7XrT0lw-kvAD890jl8e2puQens_IEKBp
          HABlsbEPX6sFY8OcGDqoRuBomu9xQ2"
    }
  ]
}
```

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:

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

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

Figure 20: AES 256-bit symmetric key

3.4.2. Signing Operation

The following are generated before completing the signing operation:

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

```
{
  "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:

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 23: Signature, base64url-encoded

3.4.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the Compact serialization:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 24: Compact Serialization

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
          RkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
          0"
    }
  ]
}
```

Figure 25: JSON Serialization

3.5. Detached Signature

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

3.5.1. Input Factors

The following are supplied before beginning the signing operation:

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

3.5.2. Signing Operation

The following are generated before completing the signing operation:

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

The protected JWS header parameters:

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

Figure 26: Protected JWS Header JSON

```
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:

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 28: Signature, base64url-encoded

3.5.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the Compact serialization:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LW
VlZjMxNGJjNzAzNyJ9
.
.
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 29: JSON Serialization

The resulting JWS object using the JSON serialization:

```
{
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
          RkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
          0"
    }
  ]
}
```

Figure 30: JSON Serialization

3.6. Protecting Specific Header Fields

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

3.6.1. Input Factors

The following are supplied before beginning the signing operation:

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

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

- Signing algorithm; this example uses "HS256".

3.6.2. Signing Operation

The following are generated before completing the signing operation:

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

- 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:

```
bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20
```

Figure 34: Signature, base64url-encoded

3.6.3. Output Results

The following compose the resulting JWS object:

- Protected JWS header (Figure 32)

- Unprotected JWS header (Figure 33)

- Payload content (Figure 2)

- Signature (Figure 34)

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJIUzI1NiJ9",
      "header": {
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr2
          0"
    }
  ]
}
```

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:

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

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

- Signing algorithm; this example uses "HS256"

3.7.2. Signing Operation

The following are generated before completing the signing operation:

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

```
{
  "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:

```
xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvxhJxZuk
```

Figure 37: Signature, base64url-encoded

3.7.3. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "header": {
        "alg": "HS256",
        "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
      },
      "signature": "xuLifqLGiblpv9zBpuZczWhNj1gARaLV3UxvxhJxZu
          k"
    }
  ]
}
```

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:

```
MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53uoimic1tcMdSg-qpt
rzZc7CG6Svw2Y13TDIqHzTUrL_lR2ZFcryNFiHkSw129EghGpwkpxaTn_THJTC
glNbADko1MZBCdwzJxwqZc-1RlpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqjy
tKSeAMbhMBdMma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZV1aIZsv33uPUqB
BCXbYoQJwt7mxPftHmNlGoOSMxR_3thmXTCm4US-xiNOyhbm8afKK64jU6_TPt
QHiJeQJxz9G3Tx-083B745_AfYOnlC9w
```

Figure 41: Signature #1, base64url-encoded

The following is the assembled first signature serialized as JSON:

```
{
  "protected": "eyJhbGciOiJSUzI1NiJ9",
  "header": {
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrgi5NvyG53u
      oimic1tcMdSg-qptrzZc7CG6Svw2Y13TDIqHzTUrL_lR2ZFcryNFiHkS
      w129EghGpwkpxaTn_THJTCglNbADko1MZBCdwzJxwqZc-1RlpO2HibUY
      yXSwO97BSe0_evZKdjvvKSgsIqjytKSeAMbhMBdMma622_BG5t4sdbuC
      HtFjp9iJmkio47AIwqkZV1aIZsv33uPUqBBCXbYoQJwt7mxPftHmNlGo
      OSMxR_3thmXTCm4US-xiNOyhbm8afKK64jU6_TPtQHiJeQJxz9G3Tx-0
      83B745_AfYOnlC9w"
}
```

Figure 42: Signature #1 JSON

3.8.3. Second Signing Operation

The following are generated before completing the second signing operation:

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

```
{
  "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:

```
ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhcdCoFZFFjfISu0Cdkn9Yb
dlmi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrqcI3Jkl2U5IX3utNhODH6v7
xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD
```

Figure 44: Signature #2, base64url-encoded

The following is the assembled second signature serialized as JSON:

```
{
  "header": {
    "alg": "ES512",
    "kid": "bilbo.baggins@hobbiton.example"
  },
  "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhcdCoF
      ZFFjfISu0Cdkn9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFM8ObLfTvNCrq
      cI3Jkl2U5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8bAWz4vIfj5pCM
      Yxxm4fgV3q7ZYhm5eD"
}
```

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:

```
s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0
```

Figure 48: Signature #3, base64url-encoded

The following is the assembled third signature serialized as JSON:

```
{
  "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOW
      ItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
  "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p0"
}
```

Figure 49: Signature #3 JSON

3.8.5. Output Results

The following compose the resulting JWS object:

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

The resulting JWS object using the JSON serialization:

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiJ9",
      "header": {
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrgi5Nvy
          G53uoimic1tcMdSg-qptrzZc7CG6Svw2Y13TDIqHzTUrL_lR2ZFc
          ryNFiHkSw129EghGpwkpxaTn_THJTCglNbADko1MZBCdwzJxwqZc
          -1RlpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqjytKSeAMbhMBdM
          ma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZV1aIZsv33uPUqBBC
          XbYoQJwt7mxPftHmNlGoOSMxR_3thmXTCm4US-xiNOyhbm8afKK6
          4jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOnlC9w"
    },
    {
      "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhc
          dCoFZFFjfISu0Cdkn9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFM8Ob
          LfTvNCrqcI3Jkl2U5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8b
          AWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD"
    },
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
          RkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
          0"
    }
  ]
}
```


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; the sequence

"\xe2\x80\x93" replacing (U+2013 EN DASH), and 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\xe2\x80\x93to the bitter end. And you can trust us to
keep any secret of yours\xe2\x80\x93closer than you keep it
yourself. But you cannot trust us to let you face trouble
alone, and go off without a word. We are your friends, Frodo.
```

Figure 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": "maxhbsmBtdQ3CNrKvprUE6n9lYcregDMLYNeTAWcLj8NnPU9XIYegT
      HVHQjxKDSHP2l-F5jS7sppG1wgdAqZyhnWvXhYNvcM7RfgKxqNx_xAHx
      6f3yy7s-M9PSNCwPC2lh6UAkR4I00EhV9lrypM9Pi4lBUop9t5fS9W5U
      NwaAllhrd-osQGPjIeI1deHTwx-ZTHu3C60Pu_LJIl6hKn9wbwaUmA4c
      R5Bd2pgbaY7ASgsjCUbtYJaNIHSoHXprUdJZKUMAzV0WOKPfA6OPI4oy
      pBadjvMZ4ZAj3BnXaSYsEZhaueTXvZB4eZOAjIyh2e_VOIKVMsnDrJYA
      VotGlvMQ",
  "e": "AQAB",
  "d": "Kn9tgoHfiTVi8uPu5b9TnwyHwG5dK6RE0uFdlpCGnJN7ZEi963R7wy
      bQ1PLAHmpIbNTztfrheoAniRV1NCIqXaW_qS461xiDTp4ntEPnqcKsyO
      5jMAji7-CL8vhpYYowNFvIesgMoVaPRYMYT9TW63hNM0aWs7USZ_hLg6
      Oe1mY0vHTI3FucjSM86Nff4oIENt43r2fspgEPGRrdE6fpLc9Oaq-qeP
      1GFULimrRdndm-P8q8kvN3KHlNAtEgrQAgTTgz80S-3VD0FgWfgnb1PN
      miuPUxO8OpI9KDIfu_acc6fg14nsNaJqXe6RESvhGPH2afjHqSy_Fd2v
      pzj85bQQ",
  "p": "2DwQmZ43FoTnQ8IkUj3BmKRf5Eh2mizZA5xEJ2MinUE3sdTYKSLtaE
      oekX9vbBZuWxHdVhM6UnKCJ_2iNk8Z0ayLYHL0_G21aXf9-unynEpUsH
      7HHTklLpYAzOOx1ZgVljoxAdWNn3hiEFrjZLZGS7lOH-a3QQlDDQoJOJ
      2VFmU",
  "q": "te8LY4-W7IyaqH1ExujjMqkTAlTeRbv0VLQnfLY2xINnrWdwiQ93_V
      F099aP1ESeLja2nw-6iKIe-qT7mtCPozKfVtUYfz5HrJ_XY2kfexJINb
      9lhZHMv5p1skZpeIS-GPHCC6gRlKo1q-idn_qxyusfWv7WAxlSVfQfk8
      d6Et0",
  "dp": "UfYKcL_or492vVc0PzwLSplbg4L3-Z5wL48mwiswbpzOyIgd2xHTH
      QmjJpFAIZ8q-zf9RmgJXkDrFs9rkdxPtAsL1WYdeCT5c125Fkdg317JV
      RDo1inX7x2Kdh8ERCreW8_4zXItuTl_KiXZNU5lvMQjWbIw2eTx1lpsf
      lo0rYU",
  "dq": "iEgcO-QfpepdH8FWd7mUFyrXdnOkXJBCogChY6YKuIHGc_p8Le9Mb
      pFKESzEaLlN1Ehf3B6oGBl5Iz_ayUlZj2IoQZ82znoUrpa9fVYNot87A
      CfzIG7q9Mv7RiPAderZi03tkVXAdaBau_9vs5rS-7HMtxkVrxSUvJY14
      TkXlHE",
  "qi": "kC-lzZOqoFaZCr5l0tOVtREKoVqaAYhQiqIRGL-MzS4sCmRkxm5vZ
      lXYx6RtE1n_Aagjqajlkjie GlxTTThHD8Iga6foGBMaAr5uR1hGQpSc7
      Gl7CF1DZkBJMTQN6EshYzZfxW08mIO8M6Rzuh0beL6fG9mkDcIyPrBXx
      2bQ_mM"
}
```


Figure 52: RSA 2048-bit Key, in JWK format

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

4.1.2. Generated Factors

The following are generated before encrypting:

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

```
3qyTVhIWt5juqZUCpfRqpvauwB956MEJL2Rt-8qXKSo
```

Figure 53: Content Encryption Key, base64url-encoded

```
bbd5sTkYwhAIqfHsx8DayA
```

Figure 54: Initialization Vector, base64url-encoded

4.1.3. Encrypting the Key

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

```
laLxI0j-nLH-_BgLOXMozKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePF
vG2K-pvSlWc9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4XaRvYHFjZTZ2G
Xfz8YAImcc91Tfk0WXC2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcG
TSLUeeCt36r1Kt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8Vl
zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEecelIO1wx1BpyIfgvfjOh
MBs9M8XL223Fg47xlGsMXdfuY-4jaqVw
```

Figure 55: Encrypted Key, base64url-encoded

4.1.4. Encrypting the Content

The following are generated before encrypting the plaintext:

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

```
{
  "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:

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

produces the following:

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

```
0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_r
aa0HnLQ6s1P2sv3Xzl1p1l_o5wR_RsSzrS8Z-wnI3Jvo0mkpEEnlDmZvDu_k8O
WzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc2VbVbK4dQKPdNTjPPEmRqcaGeTWZV
yeSUvf5k59yJZxRuSvWFf6KrNtmRdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaI0
zT5CbL5Qlw3sRc7u_hg0yKVOiRytEAEs3vZkcfLkP6nbXdC_PkMdNS-ohP78T2
O6_7uInMGhFeX4ctHG7VelHGiT93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VW
i7lzA6BP430m
```

Figure 58: Ciphertext, base64url-encoded

```
kvKuFBXHe5mQr4lqgobAUg
```

Figure 59: Authentication Tag, base64url-encoded

4.1.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 57).
- Encrypted Key (Figure 55).
- Initialization vector/nonce (Figure 54).
- Ciphertext (Figure 58).
- Authentication Tag (Figure 59).

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW5zQGhvYmJpdG9uLm
V4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
.
laLxI0j-nLH-_BgLOXMozKxmy9gffy2gTdvqzfTihJBuuzxg0V7yk1WClnQePF
vG2K-pvSlWc9BRIazDrn50RcRai__3TDON395H3c62tIouJJ4XaRvYHFjZTZ2G
Xfz8YAImcc91Tfk0WXC2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcG
TSLUeeCt36r1Kt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8Vl
zNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEecelIO1wx1BpyIfgvfjOh
MBs9M8XL223Fg47xlGsMXdfuY-4jaqVw
.
bbd5sTkYwhAIqfHsx8DayA
.
0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62JhJvGZ4_FNVSiGc_r
aa0HnLQ6s1P2sv3Xzl1p1l_o5wR_RsSzrS8Z-wnI3Jvo0mkpEEnlDmZvDu_k8O
WzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc2VbVbK4dQKPdNTjPPEmRqcaGeTWZV
yeSUvf5k59yJZxRuSvWFf6KrNtmRdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaI0
zT5CbL5Qlw3sRc7u_hg0yKVOiRytEAEs3vZkcfLkP6nbXdC_PkMdNS-ohP78T2
O6_7uInMGhFeX4ctHG7VelHGiT93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VW
i7lzA6BP430m
.
kvKuFBXHe5mQr4lqgobAUg
```

Figure 60: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "laLxI0j-nLH-_BgLOXMozKxmy9gffy2gTdvqzf
          TihJBuuzxg0V7yk1WClnQePFvG2K-pvSlWc9BRIazDrn50RcRai_
          _3TDON395H3c62tIouJJ4XaRvYHFjZTZ2GXfz8YAImcc91Tfk0WX
          C2F5Xbb71ClQ1DDH151tlpH77f2ff7xiSxh9oSewYrcGTSLUeeCt
          36r1Kt3OSj7EyBQXoZlN7IxbyhMAfgIe7Mv1rOTOI5I8NQqeXXW8
          VlzNmoxaGMny3YnGir5Wf6Qt2nBq4qDaPdnaAuuGUGEecelIO1wx
          1BpyIfgvfjOhMBs9M8XL223Fg47xlGsMXdfuY-4jaqVw"
    }
  ],
  "protected": "eyJhbGciOiJSU0ExXzUiLCJraWQiOiJmcm9kby5iYWdnaW
      5zQGhvYmJpdG9uLmV4YW1wbGUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In
      0",
  "iv": "bbd5sTkYwhAIqfHsx8DayA",
  "ciphertext": "0fys_TY_na7f8dwSfXLiYdHaA2DxUjD67ieF7fcVbIR62
      JhJvGZ4_FNVSiGc_raa0HnLQ6s1P2sv3Xzl1p1l_o5wR_RsSzrS8Z-wn
      I3Jvo0mkpEEnlDmZvDu_k8OWzJv7eZVEqiWKdyVzFhPpiyQU28GLOpRc
      2VbVbK4dQKPdNTjPPEmRqcaGeTWZVyeSUvf5k59yJZxRuSvWFf6KrNtm
      RdZ8R4mDOjHSrM_s8uwIFcqt4r5GX8TKaI0zT5CbL5Qlw3sRc7u_hg0y
      KVOiRytEAEs3vZkcfLkP6nbXdC_PkMdNS-ohP78T2O6_7uInMGhFeX4c
      tHG7VelHGiT93JfWDEQi5_V9UN1rhXNrYu-0fVMkZAKX3VWi7lzA6BP4
      30m",
  "tag": "kvKuFBXHe5mQr4lqgobAUg"
}
```


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.

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

4.2.1. Input Factors

The following are supplied before beginning the encryption process:

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

o "enc" parameter of "A256GCM"


```
{
  "kty": "RSA",
  "kid": "samwise.gamgee@hobbiton.example",
  "use": "enc",
  "n": "wbdxI55VaanZXPY29Lg5hdmv2XhvqAhoxUkanfzf2-5zVUxa6prHRr
      I4pP1AhoqJRlZfYtWWd5mmHRG2pAHIlh0ySJ9wi0BioZBl1XP2e-C-Fy
      XJGcTy0HdKQWlrfhTm42EW7Vv04r4gfao6uxjLGwfpGrZLarohiWCPnk
      Nrg71S2CuNZSQBIPGjXfkmIy2tl_VWgGnL22GplyXj5YlBLdxXp3XeSt
      sqo571utNfoUTU8E4qdzJ3U1DItoVkPGsMwlmmnJiwA7sXRItBCivR4M
      5qnZtdw-7v4WuR4779ubDuJ5nalMv2S66-RPcnFAzWSKxtBDnFJJDGIU
      e7Tzizjg1nms0Xq_yPub_UOlWn0ec85FCft1hACpWG8schrOBeNqHBOD
      FskYpUc2LC5JA2TaPF2dA67dg1TTsC_FupfQ2kNGcE1LgprxKHcVWYQb
      86B-HozjHZcqtauBzFNV5tbTuB-TpkcvJfNcFLlH3b8mb-H_ox35FjqB
      SAjLKyoeqfKTpVjvXhd09knwgJf6VKq6UC418_TOljMVfFTWXUxlnfhO
      OnzW6HSSzD1c9WrCuVzsUMv54szidQ9wf1cYWf3g5qFDxDQKis99gcDa
      iCAwM3yEBIzuNeeCa5dartHDb1xEB_HcHSeYbghbMjGfasvKn0aZRsnT
      yC0xhWBlsolZE",
  "e": "AQAB",
  "alg": "RSA-OAEP",
  "d": "n7fzJc3_WG59VEOBTkayzuSMM780OJQuZjN_KbH8lOZG25ZoA7T4Bx
      cc0xQn5oZE5uSCIwg91oCt0JvxPcpmqzaJZg1nirjcWZ-oBtVk7gCAWq
      -B3qhfF3izlbkosrzjHajIcY33HBhsy4_WerrXg4MDNE4HYojy68TcxT
      2LYQRxUOCf5TtJXvM8olexlSGtVnQnDRutxEUCwiewfmmrfveEogLx9E
      A-KMgAjTiISXxqIXQhWUQX1G7v_mV_Hr2YuImYcNcHkRvp9E7ook0876
      DhkO8v4UOZLwA1OlUX98mkoqwc58A_Y2lBYbVx1_s5lpPsEqbbH-nqIj
      h1fL0gdNfihLxnclWtW7pCztLnImZAyeCWAG7ZIfv-Rn9fLIv9jZ6r7r
      -MSH9sqbuziHN2grGjD_jfRluMHa0l84fFKl6bcqN1JWxPVhzNZo01yD
      F-1LiQnqUYSepPf6X3a2SOdkqBRiquE6EvLuSYIDpJq3jDIsgoL8Mo1L
      oomgiJxUwL_GWEOGu28gplyzm-9Q0U0nyhEf1uhSR8aJAQWAiFImWH5W
      _IQT9I7-yrindr_2fWQ_i1UgMsGzA7aOGzZfPljRy6z-tY_KuBG00-28
      S_aWvjyUc-Alp8AUyKjBZ-7CWH32fGWK48j1t-zomrwjL_mnhsPbGs0c
      9WsWgRzI-K8gE",
  "p": "7_2v3OQZzlPFcHyYfLABQ3XP85Es4hCdwCkbDeltaUXgVy9l9etKgh
      vM4hRkOvbb01kYVuLFmxIkCDtpi-zLCYAdXKrAK3PtSbtzld_XZ9nlsY
      a_QZWpXB_IrtFjVfdKUdMz94pHUhFGFj7nr6NNxfpiHSHWFE1zD_AC3m
      Y46J961Y2LRnreVwAGNw53p07Db8yD_92pDa97vqcZOdgtybH9q6uma-
      RFNhO1AoiJhYZj69hjmMRXx-x56HO9cnXNbmzNSCFCKnQmn4GQLmRj9s
      fbZRqL94bbtE4_e0Zrpo8RNo8vxRLqQNwIy85fc6BRgBJomt8QdQvIgP
      gWCv5HoQ",
  "q": "zqOHk1P6WN_rHuM7ZF1cXH0x6RuOHq67WuHiSknqQeefGBA9PWs6Zy
      KQCO-O6mKXtcgE8_Q_hA2kMRcKOcvHil1hqMCNSXlflM7WPRPZu2qCDc
      qssd_uMbP-DqYthH_EzwL9KnYoH7JQFxxmcv5An8oXUtTwk4knKjkIYG
      RuUwfQTus0w1NfjFAyxOOiAQ37ussIcE6C6ZSsM3n41UlbJ7TCqewzVJ
      aPJN5cxjySPZPD3Vp01a9YgAD6a3IIaKJdIxJS1ImnfPevSJQBE79-EX
      e2kSwVgOzvt-gsmM29QQ8veHy4uAqca5dZzMs7hkkHtw1z0jHV90epQJ
      JlXXnH8Q",
```

```
  "dp": "19oDkBh1AXelMIxQFm2zZTqUhAzCIr4xNIGEPNoDt1jK83_FJA-xn
      x5kA7-1erdHdms_Ef67HsONNv5A60JaR7w8LHnDiBGnjdaUmmuO8XAxQ
      J_ia5mxjxNjS6E2yD44USo2JmHvzeeNczq25elqbTPLhUpGo1IZuG72F
      ZQ5gTjXoTXC2-xtCDEUZfaUNh4IeAipfLugbpe0JAFlFfrTDAMUFpC3i
      XjxqzbEanflwPvj6V9iDSgjj8SozSM0dLtxvu0LIeIQAeEgT_yXcrKGm
      pKdSO08kLBx8VUjkbv_3Pn20Gyu2YEuwpFlM_H1NikuxJNKFGmnAq9Lc
      nwwT0jvoQ",
  "dq": "S6p59KrlmzGzaQYQM3o0XfHCGvfqHLYjCO557HYQf72O9kLMCfd_1
      VBEqeD-1jjwELKDjck8kOBl5UvohK1oDfSP1DleAy-cnmL29DqWmhgwM
      1ip0CCNmkmsmDSlqkUXDi6sAaZuntyukyflI-qSQ3C_BafPyFaKrt1fg
      dyEwYa08pESKwwWisy7KnmoUvaJ3SaHmohFS78TJ25cfc10wZ9hQNOrI
      ChZlkiOdFCtxDqdmCqNacnhgE3bZQjGp3n83ODSz9zwJcSUvODlXBPc2
      AycH6Ci5yjbxt4Ppox_5pjm6xnQkiPgj01GpsUssMmBN7iHVsrE7N2iz
      nBNCeOUIQ",
  "qi": "FZhClBMywVVjnuUud-05qd5CYU0dK79akAgy9oX6RX6I3IIIPckCc
      iRrokxglZn-omAY5CnCe4KdrnjFOT5YUZE7G_Pg44XgCXaarLQf4hl80
      oPEf6-jJ5Iy6wPRx7G2e8qLxnh9cOdf-kRqgOS3F48Ucvw3ma5V6KGMw
      QqWFeV31XtZ8l5cVI-I3NzBS7qltpUVgz2Ju021eyc7IlqgzR98qKONl
      27DuEES0aK0WE97jnsyO27Yp88Wa2RiBrEocM89QZI1seJiGDizHRUP4
      UZxw9zsXww46wy0P6f9grnYp7t8LkyDDk8eoI4KX6SNMNVcyVS9IWjlq
      8EzqZEKIA"
}
```


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.


```
mYMfsggkTAm0TbvtlFh2hyoXnbEzJQjMxmgLN3d8xXA
```


Figure 63: Content Encryption Key, base64url-encoded


```
-nBoKLH0YkLZPSI9
```


Figure 64: Initialization Vector, base64url-encoded

4.2.3. Encrypting the Key

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

```
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi
beYOl5S_PJsAXZwSXtDePz9hk-BbtsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyu
cvI6hvALeZ6OGnhNV4v1zx2k7O1D89mAzfw-_kT3tkuorpDU-CpBENfIHX1Q58
-Aad3FzMuo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8Bpx
KdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzS1asnuHtVMt2pK
IIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zpO878TRlZx7
pZfPYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
fOTs_lycTWmY-aqWVDKhjYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3
8UjQb0lvXn1SpyvYZ3WFc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8aKaOnx6ASE5
Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR
s
```

Figure 65: Encrypted Key, base64url-encoded

4.2.4. Encrypting the Content

The following are generated before encrypting the plaintext:

- Protected JWE Header; this example uses the 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.

```
o4k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXklSgqBdpACm6UJuJowOHC5ytjqYgR
L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw
P7ua1DRfvaOjgZv6Ifa3brcAM64d8p5lhhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sSwy7SQmRkfhDjwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML
7Cc2GxgvI7zqWo0YIEc7aCflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV
maPpOslY2n525DxDfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw
```

Figure 68: Ciphertext, base64url-encoded

```
UCGiqJxhBI3IFVdPalHHvA
```

Figure 69: Authentication Tag, base64url-encoded

4.2.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0
.
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi
beYOl5S_PJsAXZwSXtDePz9hk-BbtsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyu
cvI6hvALeZ6OGnhNV4v1zx2k7O1D89mAzfw-_kT3tkuorpDU-CpBENfIHX1Q58
-Aad3FzMuo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8Bpx
KdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzS1asnuHtVMt2pK
IIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zpO878TRlZx7
pZfPYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
fOTs_lycTWmY-aqWVDKhjYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3
8UjQb0lvXn1SpyvYZ3WFc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8aKaOnx6ASE5
Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR
s
.
-nBoKLH0YkLZPSI9
.
o4k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXklSgqBdpACm6UJuJowOHC5ytjqYgR
L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw
P7ua1DRfvaOjgZv6Ifa3brcAM64d8p5lhhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sSwy7SQmRkfhDjwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML
7Cc2GxgvI7zqWo0YIEc7aCflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV
maPpOslY2n525DxDfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw
.
UCGiqJxhBI3IFVdPalHHvA
```

Figure 70: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNu
          h7lCiud48LxeolRdtFF4nzQibeYOl5S_PJsAXZwSXtDePz9hk-Bb
          tsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyucvI6hvALeZ6OGnhNV4
          v1zx2k7O1D89mAzfw-_kT3tkuorpDU-CpBENfIHX1Q58-Aad3FzM
          uo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8B
          pxKdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzS1
          asnuHtVMt2pKIIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq
          5pGqFmW2k8zpO878TRlZx7pZfPYDSXZyS0CfKKkMozT_qiCwZTSz
          4duYnt8hS4Z9sGthXn9uDqd6wycMagnQfOTs_lycTWmY-aqWVDKh
          jYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe38UjQb0lvXn
          1SpyvYZ3WFc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
          06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8a
          KaOnx6ASE5Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xD
          EdHAVCGRzN3woEI2ozDRs"
    }
  ],
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2
      FtZ2VlQGhvYmJpdG9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0",
  "iv": "-nBoKLH0YkLZPSI9",
  "ciphertext": "o4k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXklSgqBdpACm6
      UJuJowOHC5ytjqYgRL-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYx
      rXfVzIIaRdhYtEMRBvBWbEwP7ua1DRfvaOjgZv6Ifa3brcAM64d8p5lh
      hNcizPersuhw5f-pGYzseva-TUaL8iWnctc-sSwy7SQmRkfhDjwbz0fz
      6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML7Cc2GxgvI7zqWo0YIEc7a
      CflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSVmaPpOslY2n525Dx
      DfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw",
  "tag": "UCGiqJxhBI3IFVdPalHHvA"
}
```


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:

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

- Password; this example uses the password from Figure 73

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

- "enc" parameter of "A128CBC-HS256"


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

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

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 75.

```
uwsjJXaBK407Qaf0_zpcpmr1Cs0CC50hIUEyGNEt3m0
```

Figure 74: Content Encryption Key, base64url-encoded

```
VBiCzVHNoLiR3F4V82uoTQ
```

Figure 75: Initialization Vector, base64url-encoded

4.3.3. Encrypting the Key

The following are generated before encrypting the CEK:

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

```
8Q1SzinasR3xchYz6ZZcHA
```

Figure 76: Salt, base64url-encoded

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

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

produces the following encrypted key:

```
pmKVzwf89K3dfkQqbERUTC0F2jGXD6tTQVmtnVpuKUK2J4Xx2RkkLw
```

Figure 77: Encrypted Key, base64url-encoded

4.3.4. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "alg": "PBES2-HS512+A256KW",
  "p2s": "8Q1SzinasR3xchYz6ZZcHA",
  "p2c": 8192,
  "cty": "jwk-set+json",
  "enc": "A128CBC-HS256"
}
```

Figure 78: Protected JWE Header JSON

```
eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3
hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 79: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

```
23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIX86vMXqIi6IR
sfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpDjEYCNA_XOmzg8yZR9oyjo6l
TF6si4q9FZ2EhzgFQCLO_6h5EVg3vR75_hkBsnuoqoM3dwejXBtIodN84PeqMb
6asmas_dpSsz7H10fC5ni9xIz424givB1YLldF6exVmL93R3fOoOJbmk2GBQZL
_SEGllv2cQsBgeprARsaQ7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKd
PQMTlVJKkqtV4Ru5LEVpBZXBnZrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrok
AKYPqmXUeRdjFJwafkYEkiuDCV9vWGAi1DH2xTafhJwcmywIyzi4BqRpmdn_N-
zl5tuJYyuvKhjKv6ihbsV_k1hJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V
3kobXZ77ulMwDs4p
```

Figure 80: Ciphertext, base64url-encoded

```
0HlwodAhOCILG5SQ2LQ9dg
```

Figure 81: Authentication Tag, base64url-encoded

4.3.5. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOiI4UTFTemluYXNSM3
hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOiJqd2stc2V0K2pzb24iLCJl
bmMiOiJBMTI4Q0JDLUhTMjU2In0
.
pmKVzwf89K3dfkQqbERUTC0F2jGXD6tTQVmtnVpuKUK2J4Xx2RkkLw
.
VBiCzVHNoLiR3F4V82uoTQ
.
23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2nsnGIX86vMXqIi6IR
sfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpDjEYCNA_XOmzg8yZR9oyjo6l
TF6si4q9FZ2EhzgFQCLO_6h5EVg3vR75_hkBsnuoqoM3dwejXBtIodN84PeqMb
6asmas_dpSsz7H10fC5ni9xIz424givB1YLldF6exVmL93R3fOoOJbmk2GBQZL
_SEGllv2cQsBgeprARsaQ7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKd
PQMTlVJKkqtV4Ru5LEVpBZXBnZrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrok
AKYPqmXUeRdjFJwafkYEkiuDCV9vWGAi1DH2xTafhJwcmywIyzi4BqRpmdn_N-
zl5tuJYyuvKhjKv6ihbsV_k1hJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdTw8V
3kobXZ77ulMwDs4p
.
0HlwodAhOCILG5SQ2LQ9dg
```

Figure 82: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "pmKVzwf89K3dfkQqbERUTC0F2jGXD6tTQVmtnV
          puKUK2J4Xx2RkkLw"
    }
  ],
  "protected": "eyJhbGciOiJQQkVTMi1IUzUxMitBMjU2S1ciLCJwMnMiOi
      I4UTFTemluYXNSM3hjaFl6NlpaY0hBIiwicDJjIjo4MTkyLCJjdHkiOi
      Jqd2stc2V0K2pzb24iLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  "iv": "VBiCzVHNoLiR3F4V82uoTQ",
  "ciphertext": "23i-Tb1AV4n0WKVSSgcQrdg6GRqsUKxjruHXYsTHAJLZ2
      nsnGIX86vMXqIi6IRsfywCRFzLxEcZBRnTvG3nhzPk0GDD7FMyXhUHpD
      jEYCNA_XOmzg8yZR9oyjo6lTF6si4q9FZ2EhzgFQCLO_6h5EVg3vR75_
      hkBsnuoqoM3dwejXBtIodN84PeqMb6asmas_dpSsz7H10fC5ni9xIz42
      4givB1YLldF6exVmL93R3fOoOJbmk2GBQZL_SEGllv2cQsBgeprARsaQ
      7Bq99tT80coH8ItBjgV08AtzXFFsx9qKvC982KLKdPQMTlVJKkqtV4Ru
      5LEVpBZXBnZrtViSOgyg6AiuwaS-rCrcD_ePOGSuxvgtrokAKYPqmXUe
      RdjFJwafkYEkiuDCV9vWGAi1DH2xTafhJwcmywIyzi4BqRpmdn_N-zl5
      tuJYyuvKhjKv6ihbsV_k1hJGPGAxJ6wUpmwC4PTQ2izEm0TuSE8oMKdT
      w8V3kobXZ77ulMwDs4p",
  "tag": "0HlwodAhOCILG5SQ2LQ9dg"
}
```


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_T5fsNIodcG8G5FWPrTPMyxpzsSOGaQL
      pe2FpxBmu2",
  "y": "A8-yxCHxkfBz3hKZfI1jUYMjUhsEveZ9THuwFjH2sCNdtksRJU7D5-
      SkgaFL1ETP",
  "d": "iTx2pk7wW-GqJkHcEkFQb2EFyYcO7RugmaW3mRrQVAOUiPommT0Idn
      YK2xDlZh-j"
}
```

Figure 84: Elliptic Curve P-384 Key, in JWK format

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

4.4.2. Generated Factors

The following are generated before encrypting:

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

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

```
Nou2ueKlP70ZXDbq9UrRwg
```

Figure 85: Content Encryption Key, base64url-encoded

```
mH-G2zVqgztUtnW_
```

Figure 86: Initialization Vector, base64url-encoded

4.4.3. Encrypting the Key

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

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

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

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

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

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

produces the following JWE encrypted key:

```
0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2
```

Figure 88: Encrypted Key, base64url-encoded

4.4.4. Encrypting the Content

The following are generated before encrypting the content:

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

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

Figure 89: Protected JWE Header JSON

```
eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InVCbzRrSFB3Nmtiang1bDB4b3dyZF9vWXpCbWF6LUdLRlp1NH
hBRkZrYllpV2d1dEVLNml1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkMy
ZmFYdW1JLWU5SlUyTW84S3BvWXJGRHI1eVBOVnRXNFBnRXdaT3lRVEEtSmRhWT
h0YjdFMCJ9LCJlbmMiOiJBMTI4R0NNIn0
```

Figure 90: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

```
tkZuOO9h95OgHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID7lpnAi_cP
WJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsXaEwDdXta9Mn5B7cCBoJKB0
IgEnj_qfo1hIi-uEkUpOZ8aLTZGHfpl05jMwbKkTe2yK3mjF6SBAsgicQDVCkc
Y9BLluzx1RmC3ORXaM0JaHPB93YcdSDGgpgBWMVrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXIuEr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu
07WNhjzJEPc4jVntRJ6K53NgPQ5p99l3Z408OUqj4ioYezbS6vTPlQ
```

Figure 91: Ciphertext, base64url-encoded

```
WuGzxmcreYjpHGJoa17EBg
```

Figure 92: Authentication Tag, base64url-encoded

4.4.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 90)
- Encrypted key (Figure 88)
- Initialization vector/nonce (Figure 86)
- Ciphertext (Figure 91)

o Authentication tag (Figure 92)

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcmVncmluLnRvb2tAdH
Vja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAt
Mzg0IiwieCI6InVCbzRrSFB3Nmtiang1bDB4b3dyZF9vWXpCbWF6LUdLRlp1NH
hBRkZrYllpV2d1dEVLNml1RURzUTZ3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkMy
ZmFYdW1JLWU5SlUyTW84S3BvWXJGRHI1eVBOVnRXNFBnRXdaT3lRVEEtSmRhWT
h0YjdFMCJ9LCJlbmMiOiJBMTI4R0NNIn0
.
0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2
.
mH-G2zVqgztUtnW_
.
tkZuOO9h95OgHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz5NJ76oID7lpnAi_cP
WJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzsXaEwDdXta9Mn5B7cCBoJKB0
IgEnj_qfo1hIi-uEkUpOZ8aLTZGHfpl05jMwbKkTe2yK3mjF6SBAsgicQDVCkc
Y9BLluzx1RmC3ORXaM0JaHPB93YcdSDGgpgBWMVrNU1ErkjcMqMoT_wtCex3w0
3XdLkjXIuEr2hWgeP-nkUZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu
07WNhjzJEPc4jVntRJ6K53NgPQ5p99l3Z408OUqj4ioYezbS6vTPlQ
.
WuGzxmcreYjpHGJoa17EBg
```

Figure 93: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "0DJjBXri_kBcC46IkU5_Jk9BqaQeHdv2"
    }
  ],
  "protected": "eyJhbGciOiJFQ0RILUVTK0ExMjhLVyIsImtpZCI6InBlcm
      VncmluLnRvb2tAdHVja2Jvcm91Z2guZXhhbXBsZSIsImVwayI6eyJrdH
      kiOiJFQyIsImNydiI6IlAtMzg0IiwieCI6InVCbzRrSFB3Nmtiang1bD
      B4b3dyZF9vWXpCbWF6LUdLRlp1NHhBRkZrYllpV2d1dEVLNml1RURzUT
      Z3TmROZzMiLCJ5Ijoic3AzcDVTR2haVkMyZmFYdW1JLWU5SlUyTW84S3
      BvWXJGRHI1eVBOVnRXNFBnRXdaT3lRVEEtSmRhWTh0YjdFMCJ9LCJlbm
      MiOiJBMTI4R0NNIn0",
  "iv": "mH-G2zVqgztUtnW_",
  "ciphertext": "tkZuOO9h95OgHJmkkrfLBisku8rGf6nzVxhRM3sVOhXgz
      5NJ76oID7lpnAi_cPWJRCjSpAaUZ5dOR3Spy7QuEkmKx8-3RCMhSYMzs
      XaEwDdXta9Mn5B7cCBoJKB0IgEnj_qfo1hIi-uEkUpOZ8aLTZGHfpl05
      jMwbKkTe2yK3mjF6SBAsgicQDVCkcY9BLluzx1RmC3ORXaM0JaHPB93Y
      cdSDGgpgBWMVrNU1ErkjcMqMoT_wtCex3w03XdLkjXIuEr2hWgeP-nkU
      ZTPU9EoGSPj6fAS-bSz87RCPrxZdj_iVyC6QWcqAu07WNhjzJEPc4jVn
      tRJ6K53NgPQ5p99l3Z408OUqj4ioYezbS6vTPlQ",
  "tag": "WuGzxmcreYjpHGJoa17EBg"
}
```


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": "Ze2loSV3wrroKUN_4zhwGhCqo3Xhu1td4QjeQ5wIVR0",
  "y": "HlLtdXARY_f55A3fnzQbPcm6hgr34Mp8p-nuzQCE0Zw",
  "d": "r_kHyZ-a06rmxM3yESK84r1otSg-aQcVStkRhA-iCM8"
}
```

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.

```
yc9N8v5sYyv3iGQT926IUg
```

Figure 96: Initialization Vector, base64url-encoded

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

4.5.3. Key Agreement

The following are generated to agree on a CEK:

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

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "mPUKT_bAWGHIhg0TpjjqVsP1rXWQu_vwVOHHtNkdYoA",
  "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs",
  "d": "AtH35vJsQ9SGjYfOsjUxYXQKrPH3FjZHmEtSKoSN8cM"
}
```

Figure 97: Ephemeral public key, in JWK format

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

```
hzHdlfQIAEehb8Hrd_mFRhKsKLEzPfshfXs9l6areCc
```

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

4.5.4. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "alg": "ECDH-ES",
  "kid": "meriadoc.brandybuck@buckland.example",
  "epk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "mPUKT_bAWGHIhg0TpjjqVsP1rXWQu_vwVOHHtNkdYoA",
    "y": "8BQAsImGeAS46fyWw5MhYfGTT0IjBpFw2SS34Dv4Irs"
  },
  "enc": "A128CBC-HS256"
}
```

Figure 99: Protected JWE Header JSON

```
eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYi
LCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqanFWc1AxclhXUXVfdndWT0hIdE5rZF
lvQSIsInkiOiI4QlFBc0ltR2VBUzQ2ZnlXdzVNaFlmR1RUMElqQnBGdzJTUzM0
RHY0SXJzIn0sImVuYyI6IkExMjhDQkMtSFMyNTYifQ
```

Figure 100: Protected JWE Header, base64url-encoded

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

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

produces the following:

- Ciphertext from Figure 101.

- Authentication tag from Figure 102.

```
BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4OPKbWE1zSTEFjDfhU9
IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEsDIqAYtskTTmzmzNa-_q4F_e
vAPUmwlO-ZG45Mnq4uhM1fm_D9rBtWolqZSF3xGNNkpOMQKF1Cl8i8wjzRli7-
IXgyirlKQsbhhqRzkv8IcY6aHl24j03C-AR2le1r7URUhArM79BY8soZU0lzwI
-sD5PZ3l4NDCCei9XkoIAfsXJWmySPoeRb2Ni5UZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K767G9C9Azp73gKZD0DyUn1mn0WW5LmyX_yJ-3AROq8p1WZBfG-ZyJ61
95_JGG2m9Csg
```

Figure 101: Ciphertext, base64url-encoded

```
WCCkNa-x4BeB9hIDIfFuhg
```

Figure 102: Authentication Tag, base64url-encoded

4.5.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 90)

- Initialization vector/nonce (Figure 86)

- Ciphertext (Figure 91)

- Authentication tag (Figure 92)

the resulting JWE object using the Compact serialization:

```
eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYnJhbmR5YnVja0BidW
NrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYi
LCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqanFWc1AxclhXUXVfdndWT0hIdE5rZF
lvQSIsInkiOiI4QlFBc0ltR2VBUzQ2ZnlXdzVNaFlmR1RUMElqQnBGdzJTUzM0
RHY0SXJzIn0sImVuYyI6IkExMjhDQkMtSFMyNTYifQ
.
.
yc9N8v5sYyv3iGQT926IUg
.
BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4OPKbWE1zSTEFjDfhU9
IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEsDIqAYtskTTmzmzNa-_q4F_e
vAPUmwlO-ZG45Mnq4uhM1fm_D9rBtWolqZSF3xGNNkpOMQKF1Cl8i8wjzRli7-
IXgyirlKQsbhhqRzkv8IcY6aHl24j03C-AR2le1r7URUhArM79BY8soZU0lzwI
-sD5PZ3l4NDCCei9XkoIAfsXJWmySPoeRb2Ni5UZL4mYpvKDiwmyzGd65KqVw7
MsFfI_K767G9C9Azp73gKZD0DyUn1mn0WW5LmyX_yJ-3AROq8p1WZBfG-ZyJ61
95_JGG2m9Csg
.
WCCkNa-x4BeB9hIDIfFuhg
```

Figure 103: Compact Serialization

the resulting JWE object using the JSON serialization:

```
{
  "protected": "eyJhbGciOiJFQ0RILUVTIiwia2lkIjoibWVyaWFkb2MuYn
      JhbmR5YnVja0BidWNrbGFuZC5leGFtcGxlIiwiZXBrIjp7Imt0eSI6Ik
      VDIiwiY3J2IjoiUC0yNTYiLCJ4IjoibVBVS1RfYkFXR0hJaGcwVHBqan
      FWc1AxclhXUXVfdndWT0hIdE5rZFlvQSIsInkiOiI4QlFBc0ltR2VBUz
      Q2ZnlXdzVNaFlmR1RUMElqQnBGdzJTUzM0RHY0SXJzIn0sImVuYyI6Ik
      ExMjhDQkMtSFMyNTYifQ",
  "iv": "yc9N8v5sYyv3iGQT926IUg",
  "ciphertext": "BoDlwPnTypYq-ivjmQvAYJLb5Q6l-F3LIgQomlz87yW4O
      PKbWE1zSTEFjDfhU9IPIOSA9Bml4m7iDFwA-1ZXvHteLDtw4R1XRGMEs
      DIqAYtskTTmzmzNa-_q4F_evAPUmwlO-ZG45Mnq4uhM1fm_D9rBtWolq
      ZSF3xGNNkpOMQKF1Cl8i8wjzRli7-IXgyirlKQsbhhqRzkv8IcY6aHl2
      4j03C-AR2le1r7URUhArM79BY8soZU0lzwI-sD5PZ3l4NDCCei9XkoIA
      fsXJWmySPoeRb2Ni5UZL4mYpvKDiwmyzGd65KqVw7MsFfI_K767G9C9A
      zp73gKZD0DyUn1mn0WW5LmyX_yJ-3AROq8p1WZBfG-ZyJ6195_JGG2m9
      Csg",
  "tag": "WCCkNa-x4BeB9hIDIfFuhg"
}
```

Figure 104: JSON Serialization

4.6. Direct Encryption using AES-GCM

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

4.6.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

4.6.2. Generated Factors

The following are generated before encrypting:

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

```
refa467QzzKx6QAB
```

Figure 106: Initialization Vector, base64url-encoded

4.6.3. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "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.

```
JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9TYnSovc23XJoBcW29rHP8yZOZG7Y
hLpT1bjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUYkshtrMmIUAyGmUnd9zM
DB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_
BIwCPTjb9o0SbdcdREEMJMyZBH8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxscj5
g-NJsUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRSIn
ZI-wjsY0yu3cT4_aQ3i1o-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp
```

Figure 109: Ciphertext, base64url-encoded

```
vbb32Xvllea2OtmHAdccRQ
```

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
.
.
refa467QzzKx6QAB
.
JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9TYnSovc23XJoBcW29rHP8yZOZG7Y
hLpT1bjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUYkshtrMmIUAyGmUnd9zM
DB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb4h3-FtBP-c_
BIwCPTjb9o0SbdcdREEMJMyZBH8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxscj5
g-NJsUPbjk29-s7LJAGb15wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRSIn
ZI-wjsY0yu3cT4_aQ3i1o-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp
.
vbb32Xvllea2OtmHAdccRQ
```

Figure 111: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJiOC02ZTEzLT
      Q1Y2YtODY3Mi02MTdiNWI0NTI0M2EiLCJlbmMiOiJBMTI4R0NNIn0",
  "iv": "refa467QzzKx6QAB",
  "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9TYnSovc23XJ
      oBcW29rHP8yZOZG7YhLpT1bjFuvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9
      HRUYkshtrMmIUAyGmUnd9zMDB2n0cRDIHAzFVeJUDxkUwVAE7_YGRPdc
      qMyiBoCO-FBdE-Nceb4h3-FtBP-c_BIwCPTjb9o0SbdcdREEMJMyZBH8
      ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxscj5g-NJsUPbjk29-s7LJAGb1
      5wEBtXphVCgyy53CoIKLHHeJHXex45Uz9aKZSRSInZI-wjsY0yu3cT4_
      aQ3i1o-tiE-F8Ios61EKgyIQ4CWao8PFMj8TTnp",
  "tag": "vbb32Xvllea2OtmHAdccRQ"
}
```

Figure 112: JSON Serialization

4.7. Key Wrap using AES-GCM KeyWrap with AES-CBC-HMAC-SHA2

This example illustrates encrypting content using the "A256GCMKW" (AES-256-GCM-KeyWrap) key encryption algorithm with the "A128CBC-HS256" (AES-128-CBC-HMAC-SHA-256) content encryption algorithm.

4.7.1. Input Factors

The following are supplied before beginning the encryption process:

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

```
{
  "kty": "oct",
  "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
  "use": "enc",
  "alg": "A256GCMKW",
  "k": "qC57l_uxcm7Nm3K-ct4GFjx8tM1U8CZ0NLBvdQstiS8"
}
```

Figure 113: AES 256-bit Key

4.7.2. Generated Factors

The following are generated before encrypting:

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

```
UWxARpat23nL9ReIj4WG3D1ee9I4r-Mv5QLuFXdy_rE
```

Figure 114: Content Encryption Key, base64url-encoded

```
gz6NjyEFNm_vm8Gj6FwoFQ
```

Figure 115: Initialization Vector, base64url-encoded

4.7.3. Encrypting the Key

The following are generated before encrypting the CEK:

- Initialization vector/nonce for key wrapping; this example uses the initialization vector/nonce from Figure 116.

```
KkYT0GX_2jHlfqN_
```

Figure 116: Key Wrap Initialization Vector, base64url-encoded

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

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

produces the following:

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

```
lJf3HbOApxMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNok
```

Figure 117: Encrypted Key, base64url-encoded

```
kfPduVQ3T3H6vnewt--ksw
```

Figure 118: Key Wrap Authentication Tag, base64url-encoded

4.7.4. Encrypting the Content

The following are generated before encrypting the content:

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

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

Figure 119: Protected JWE Header JSON

```
eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRM1QzSDZ2bmV3dC0ta3N3
IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIiwiZW5jIjoiQTEyOENCQy1IUzI1Ni
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.

```
Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8EqoDZHyFKFBupS8iaE
eVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhzWyWtZKX0gxKdy6HgLvqoGNbZCz
LjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQHLcqAHxy51449xkjZ7ewzZaGV3eFq
hpco8o4DijXaG5_7kp3h2cajRfDgymuxUbWgLqaeNQaJtvJmSMFuEOSAzw9Hde
b6yhdTynCRmu-kqtO5Dec4lT2OMZKpnxc_F1_4yDJFcqb5CiDSmA-psB2k0Jtj
xAj4UPI61oONK7zzFIu4gBfjJCndsZfdvG7h8wGjV98QhrKEnR7xKZ3KCr0_qR
1B-gxpNk3xWU
```

Figure 121: Ciphertext, base64url-encoded

```
DKW7jrb4WaRSNfbXVPlT5g
```

Figure 122: Authentication Tag, base64url-encoded

4.7.5. Output Results

The following compose the resulting JWE object:

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

o Authentication tag (Figure 122)

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS1iZmE5LTRkOTUtYj
IwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdVZRM1QzSDZ2bmV3dC0ta3N3
IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIiwiZW5jIjoiQTEyOENCQy1IUzI1Ni
J9
.
lJf3HbOApxMEBkCMOoTnnABxs_CvTWUmZQ2ElLvYNok
.
gz6NjyEFNm_vm8Gj6FwoFQ
.
Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8EqoDZHyFKFBupS8iaE
eVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhzWyWtZKX0gxKdy6HgLvqoGNbZCz
LjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQHLcqAHxy51449xkjZ7ewzZaGV3eFq
hpco8o4DijXaG5_7kp3h2cajRfDgymuxUbWgLqaeNQaJtvJmSMFuEOSAzw9Hde
b6yhdTynCRmu-kqtO5Dec4lT2OMZKpnxc_F1_4yDJFcqb5CiDSmA-psB2k0Jtj
xAj4UPI61oONK7zzFIu4gBfjJCndsZfdvG7h8wGjV98QhrKEnR7xKZ3KCr0_qR
1B-gxpNk3xWU
.
DKW7jrb4WaRSNfbXVPlT5g
```

Figure 123: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "lJf3HbOApxMEBkCMOoTnnABxs_CvTWUmZQ2ElL
          vYNok"
    }
  ],
  "protected": "eyJhbGciOiJBMjU2R0NNS1ciLCJraWQiOiIxOGVjMDhlMS
      1iZmE5LTRkOTUtYjIwNS0yYjRkZDFkNDMyMWQiLCJ0YWciOiJrZlBkdV
      ZRM1QzSDZ2bmV3dC0ta3N3IiwiaXYiOiJLa1lUMEdYXzJqSGxmcU5fIi
      wiZW5jIjoiQTEyOENCQy1IUzI1NiJ9",
  "iv": "gz6NjyEFNm_vm8Gj6FwoFQ",
  "ciphertext": "Jf5p9-ZhJlJy_IQ_byKFmI0Ro7w7G1QiaZpI8OaiVgD8E
      qoDZHyFKFBupS8iaEeVIgMqWmsuJKuoVgzR3YfzoMd3GxEm3VxNhzWyW
      tZKX0gxKdy6HgLvqoGNbZCzLjqcpDiF8q2_62EVAbr2uSc2oaxFmFuIQ
      HLcqAHxy51449xkjZ7ewzZaGV3eFqhpco8o4DijXaG5_7kp3h2cajRfD
      gymuxUbWgLqaeNQaJtvJmSMFuEOSAzw9Hdeb6yhdTynCRmu-kqtO5Dec
      4lT2OMZKpnxc_F1_4yDJFcqb5CiDSmA-psB2k0JtjxAj4UPI61oONK7z
      zFIu4gBfjJCndsZfdvG7h8wGjV98QhrKEnR7xKZ3KCr0_qR1B-gxpNk3
      xWU",
  "tag": "DKW7jrb4WaRSNfbXVPlT5g"
}
```


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:

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

```
{
  "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:

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

```
aY5_Ghmk9KxWPBLu_glx1w
```

Figure 126: Content Encryption Key, base64url-encoded

```
Qx0pmsDa8KnJc9Jo
```

Figure 127: Initialization Vector, base64url-encoded

4.8.3. Encrypting the Key

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

```
CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx
```

Figure 128: Encrypted Key, base64url-encoded

4.8.4. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "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:

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

produces the following:

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

```
AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tP1bTdhtFJgJxeVmJkLD6
1A1hnWGetdg11c9ADsnWgL56NyxwSYjU1ZEHcGkd3EkU0vjHi9gTlb90qSYFfe
F0LwkcTtjbYKCsiNJQkcIp1yeM03OmuiYSoYJVSpf7ej6zaYcMv3WwdxDFl8RE
wOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHlQeg7gDT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJfqqFvETUxLAF-KjcBTS5dNy6egwkYtOt8EIHK-oEsKYtZRa
a8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kEThkaSqdYw0FkQZF
```

Figure 131: Ciphertext, base64url-encoded

And authentication tag:

```
ER7MWJZ1FBI_NKvn7Zb1Lw
```

Figure 132: Authentication Tag, base64url-encoded

4.8.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 130)

- encrypted key (Figure 128)
- Initialization vector/nonce (Figure 127)
- Ciphertext (Figure 131)
- Authentication tag (Figure 132)

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04MzMyLTQzZDktYTQ2OC
04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn0
.
CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx
.
Qx0pmsDa8KnJc9Jo
.
AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tP1bTdhtFJgJxeVmJkLD6
1A1hnWGetdg11c9ADsnWgL56NyxwSYjU1ZEHcGkd3EkU0vjHi9gTlb90qSYFfe
F0LwkcTtjbYKCsiNJQkcIp1yeM03OmuiYSoYJVSpf7ej6zaYcMv3WwdxDFl8RE
wOhNImk2Xld2JXq6BR53TSFkyT7PwVLuq-1GwtGHlQeg7gDT6xW0JqHDPn_H-p
uQsmthc9Zg0ojmJfqqFvETUxLAF-KjcBTS5dNy6egwkYtOt8EIHK-oEsKYtZRa
a8Z7MOZ7UGxGIMvEmxrGCPeJa14slv2-gaqK0kEThkaSqdYw0FkQZF
.
ER7MWJZ1FBI_NKvn7Zb1Lw
```

Figure 133: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "CBI6oDw8MydIx1IBntf_lQcw2MmJKIQx"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04Mz
      MyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn
      0",
  "iv": "Qx0pmsDa8KnJc9Jo",
  "ciphertext": "AwliP-KmWgsZ37BvzCefNen6VTbRK3QMA4TkvRkH0tP1b
      TdhtFJgJxeVmJkLD61A1hnWGetdg11c9ADsnWgL56NyxwSYjU1ZEHcGk
      d3EkU0vjHi9gTlb90qSYFfeF0LwkcTtjbYKCsiNJQkcIp1yeM03OmuiY
      SoYJVSpf7ej6zaYcMv3WwdxDFl8REwOhNImk2Xld2JXq6BR53TSFkyT7
      PwVLuq-1GwtGHlQeg7gDT6xW0JqHDPn_H-puQsmthc9Zg0ojmJfqqFvE
      TUxLAF-KjcBTS5dNy6egwkYtOt8EIHK-oEsKYtZRaa8Z7MOZ7UGxGIMv
      EmxrGCPeJa14slv2-gaqK0kEThkaSqdYw0FkQZF",
  "tag": "ER7MWJZ1FBI_NKvn7Zb1Lw"
}
```


Figure 134: JSON Serialization

4.9. Compressed Content

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

4.9.1. Input Factors

The following are supplied before beginning the encryption process:

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

4.9.2. Generated Factors

The following are generated before encrypting:

- Compressed plaintext from the original plaintext content; compressing Figure 51 using the DEFLATE [RFC1951] algorithm produces the compressed plaintext from Figure 135.

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

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

```
bY_BDcIwDEVX-QNU3QEOrIA4pqlDokYxchxVvbEDGzIJbioOSJwc-f___HPjBu
8KVFpVtAplVE1-wZo0YjNZo3C7R5v72pV5f5X382VWjYQpqZKAyjziZOr2B7kQ
PSy6oZIXUnDYbVKN4jNXi2u0yB7t1qSHTjmMODf9QgvrDzfTIQXnyQRuUya4zI
WG3vTOdir0v7BRHFYWq3k1k1A_gSDJqtcBF-GZxw8
```

Figure 135: Compressed Plaintext, base64url-encoded

```
hC-MpLZSuwWv8sexS6ydfw
```

Figure 136: Content Encryption Key, base64url-encoded

```
p9pUq6XHY0jfEZIl
```

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 (Figure 125) produces the following encrypted key:

```
5vUT2WOtQxKWcekM_IzVQwkGgzlFDwPi
```

Figure 138: Encrypted Key, base64url-encoded

4.9.4. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "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.

```
HbDtOsdai1oYziSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyez
SPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
```

Figure 141: Ciphertext, base64url-encoded

And authentication tag:

```
VILuUwuIxaLVmh5X-T7kmA
```

Figure 142: Authentication Tag, base64url-encoded

4.9.5. Output Results

The following compose the resulting JWE object:

- Protected JWE header (Figure 140)

- 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
.
5vUT2WOtQxKWcekM_IzVQwkGgzlFDwPi
.
p9pUq6XHY0jfEZIl
.
HbDtOsdai1oYziSx25KEeTxmwnh8L8jKMFNc1k3zmMI6VB8hry57tDZ61jXyez
SPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWGml8blyiMQmOn9J--XhhlYg0
m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDHj0aBMG6152PsM-w5E_o2B3jDbrYBK
hpYA7qi3AyijnCJ7BP9rr3U8kxExCpG3mK420TjOw
.
VILuUwuIxaLVmh5X-T7kmA
```

Figure 143: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "5vUT2WOtQxKWcekM_IzVQwkGgzlFDwPi"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04Mz
      MyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIi
      wiemlwIjoiREVGIn0",
  "iv": "p9pUq6XHY0jfEZIl",
  "ciphertext": "HbDtOsdai1oYziSx25KEeTxmwnh8L8jKMFNc1k3zmMI6V
      B8hry57tDZ61jXyezSPt0fdLVfe6Jf5y5-JaCap_JQBcb5opbmT60uWG
      ml8blyiMQmOn9J--XhhlYg0m-BHaqfDO5iTOWxPxFMUedx7WCy8mxgDH
      j0aBMG6152PsM-w5E_o2B3jDbrYBKhpYA7qi3AyijnCJ7BP9rr3U8kxE
      xCpG3mK420TjOw",
  "tag": "VILuUwuIxaLVmh5X-T7kmA"
}
```

Figure 144: JSON Serialization

4.10. Including Additional Authenticated Data

This example illustrates encrypting content that includes additional authenticated data. As this example includes an additional top-level property not present in the Compact serialization, only the JSON serialization is possible.

4.10.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

Figure 145: Additional Authenticated Data, in JSON format

NOTE whitespace between JSON values added for readability.

4.10.2. Generated Factors

The following are generated before encrypting:

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

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 147.

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

```
75m1ALsYv10pZTKPWrsqdg
```

Figure 146: Content Encryption Key, base64url-encoded

```
veCx9ece2orS7c_N
```

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:

```
4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X
```

Figure 149: Encrypted Key, base64url-encoded

4.10.4. Encrypting the Content

The following are generated before encrypting the content:

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

```
{
  "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.

Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0Ui8p74SchQP8xygM1
oFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14T_4NFqF-p2Mx8zkbKxI7oPK
8KNarFbyxIDvICNqBLba-v3uzXBdB89fzOI-Lv4PjOFAQGHrgv1rjXAmKbgkft
9cB4WeyZw8MldbBhc-V_KWZslrsLNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4a
q3FXBxOxCys35PhCdaggy2kfUfl2OkwKnWUbgXVD1C6HxLIlqHhCwXDG59weHr
RDQeHyMRoBljoV3X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV

Figure 152: Ciphertext, base64url-encoded

vOaH_Rajnpy_3hOtqvZHRA

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": "4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X"
    }
  ],
  "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04Mz
      MyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn
      0",
  "iv": "veCx9ece2orS7c_N",
  "aad": "WyJ2Y2FyZCIsW1sidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxb
      ImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4i
      LHt9LCJ0ZXh0IixbIkJyYW5keWJ1Y2siLCJNZXJpYWRvYyIsIk1yLiIs
      IiJdXSxbImJkYXkiLHt9LCJ0ZXh0IiwiVEEgMjk4MiJdLFsiZ2VuZGVy
      Iix7fSwidGV4dCIsIk0iXV1d",
  "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0
      Ui8p74SchQP8xygM1oFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14
      T_4NFqF-p2Mx8zkbKxI7oPK8KNarFbyxIDvICNqBLba-v3uzXBdB89fz
      OI-Lv4PjOFAQGHrgv1rjXAmKbgkft9cB4WeyZw8MldbBhc-V_KWZslrs
      LNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4aq3FXBxOxCys35PhCdaggy
      2kfUfl2OkwKnWUbgXVD1C6HxLIlqHhCwXDG59weHrRDQeHyMRoBljoV3
      X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV",
  "tag": "vOaH_Rajnpy_3hOtqvZHRA"
}
```


Figure 154: JSON Serialization

4.11. Protecting Specific Header Fields

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

4.11.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

4.11.2. Generated Factors

The following are generated before encrypting:

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

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

```
WDgEptBmQs9ouUvArz6x6g
```

Figure 155: Content Encryption Key, base64url-encoded

```
WgEJsDS9bkoXQ3nR
```

Figure 156: Initialization Vector, base64url-encoded

4.11.3. Encrypting the Key

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

```
jJIcM9J-hbx3wnqhf5FlkEYos0sHsF0H
```

Figure 157: Encrypted Key, base64url-encoded

4.11.4. Encrypting the Content

The following are generated before encrypting the content:

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

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

Figure 158: Protected JWE Header JSON

```
eyJlbmMiOiJBMTI4R0NNIn0
```

Figure 159: Protected JWE Header, base64url-encoded

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

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

produces the following:

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

```
lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2DM3swKkjOwQyZtWsFL
YMj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC9OCCJ1IHAolUv4MyOt80MoPb8
fZYbNKqplzYJgIL58g8N2v46OgyG637d6uuKPwhAnTGm_zWhqc_srOvgiLkzyF
XPq1hBAURbc3-8BqeRb48iR1-_5g5UjWVD3lgiLCN_P7AW8mIiFvUNXBPJK3nO
WL4teUPS8yHLbWeL83olU4UAgL48x-8dDkH23JykibVSQju-f7e-1xreHWXzWL
Hs1NqBbre0dEwK3HX_xM0LjUz77Krppgegoutpf5qaKg3l-_xMINmf
```

Figure 160: Ciphertext, base64url-encoded

```
fNYLqpUe84KD45lvDiaBAQ
```

Figure 161: Authentication Tag, base64url-encoded

4.11.5. Output Results

The following compose the resulting JWE object:

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

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

```
{
  "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": "jJIcM9J-hbx3wnqhf5FlkEYos0sHsF0H"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8"
  },
  "protected": "eyJlbmMiOiJBMTI4R0NNIn0",
  "iv": "WgEJsDS9bkoXQ3nR",
  "ciphertext": "lIbCyRmRJxnB2yLQOTqjCDKV3H30ossOw3uD9DPsqLL2D
      M3swKkjOwQyZtWsFLYMj5YeLht_StAn21tHmQJuuNt64T8D4t6C7kC9O
      CCJ1IHAolUv4MyOt80MoPb8fZYbNKqplzYJgIL58g8N2v46OgyG637d6
      uuKPwhAnTGm_zWhqc_srOvgiLkzyFXPq1hBAURbc3-8BqeRb48iR1-_5
      g5UjWVD3lgiLCN_P7AW8mIiFvUNXBPJK3nOWL4teUPS8yHLbWeL83olU
      4UAgL48x-8dDkH23JykibVSQju-f7e-1xreHWXzWLHs1NqBbre0dEwK3
      HX_xM0LjUz77Krppgegoutpf5qaKg3l-_xMINmf",
  "tag": "fNYLqpUe84KD45lvDiaBAQ"
}
```

Figure 163: JSON Serialization

4.12. Protecting Content Only

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

4.12.1. Input Factors

The following are supplied before beginning the encryption process:

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

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

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

4.12.2. Generated Factors

The following are generated before encrypting:

- AES symmetric key as the Content Encryption Key; this example the key from Figure 164.

- Initialization vector/nonce; this example uses the initialization vector/nonce from Figure 165.

```
KBooAFl30QPV3vkcZlXnzQ
```

Figure 164: Content Encryption Key, base64url-encoded

```
YihBoVOGsR1l7jCD
```

Figure 165: Initialization Vector, base64url-encoded

4.12.3. Encrypting the Key

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

```
244YHfO_W7RMpQW81UjQrZcq5LSyqiPv
```

Figure 166: Encrypted Key, base64url-encoded

4.12.4. Encrypting the Content

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

- CEK (Figure 164);

- Initialization vector/nonce (Figure 165); and

- Empty string as authenticated data

produces the following:

- Ciphertext from Figure 167.

- Authenticated data from Figure 168.

```
qtPIMMaOBRgASL10dNQhOa7Gqrk7Eal1vwht7R4TT1uq-arsVCPaIeFwQfzrSS
6oEUWbBtxEasE0vC6r7sphyVziMCVJEuRJyoAHFSP3eqQPb4Ic1SDSqyXjw_L3
svybhHYUGyQuTmUQEDjgjJfBOifwHIsDsRPeBz1NomqeifVPq5GTCWFo5k_MNI
QURR2Wj0AHC2k7JZfu2iWjUHLF8ExFZLZ4nlmsvJu_mvifMYiikfNfsZAudISO
a6O73yPZtL04k_1FI7WDfrb2w7OqKLWDXzlpcxohPVOLQwpA3mFNRKdY-bQz4Z
4KX9lfz1cne31N4-8BKmojpw-OdQjKdLOGkC445Fb_K1tlDQXw2sBF
```

Figure 167: Ciphertext, base64url-encoded

```
e2m0Vm7JvjK2VpCKXS-kyg
```

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:

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

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "244YHfO_W7RMpQW81UjQrZcq5LSyqiPv"
    }
  ],
  "unprotected": {
    "alg": "A128KW",
    "kid": "81b20965-8332-43d9-a468-82160ad91ac8",
    "enc": "A128GCM"
  },
  "iv": "YihBoVOGsR1l7jCD",
  "ciphertext": "qtPIMMaOBRgASL10dNQhOa7Gqrk7Eal1vwht7R4TT1uq-
      arsVCPaIeFwQfzrSS6oEUWbBtxEasE0vC6r7sphyVziMCVJEuRJyoAHF
      SP3eqQPb4Ic1SDSqyXjw_L3svybhHYUGyQuTmUQEDjgjJfBOifwHIsDs
      RPeBz1NomqeifVPq5GTCWFo5k_MNIQURR2Wj0AHC2k7JZfu2iWjUHLF8
      ExFZLZ4nlmsvJu_mvifMYiikfNfsZAudISOa6O73yPZtL04k_1FI7WDf
      rb2w7OqKLWDXzlpcxohPVOLQwpA3mFNRKdY-bQz4Z4KX9lfz1cne31N4
      -8BKmojpw-OdQjKdLOGkC445Fb_K1tlDQXw2sBF",
  "tag": "e2m0Vm7JvjK2VpCKXS-kyg"
}
```


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.

```
zXayeJ4gvm8NJr3IUInyokTUO-LbQNKEhe_zWlYbdpQ
```

Figure 171: Content Encryption Key, base64url-encoded

```
VgEIHY20EnzUtZFl2RpB1g
```

Figure 172: Initialization Vector, base64url-encoded

4.13.3. Encrypting the Key to the First Recipient

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

```
dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w6Y5G4XJQsNNIBiqyvUUA
OcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4bWrBf7cHwEQJdXihidAYWVajJIa
KMXMvFRMV6iDlRr076DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-w
qjw0-b-S1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbeYS
rRicJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-d
bHcGlihqc_wGgho9fLMK8JOArYLcMDNQ
```

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": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zjwj4w
      6Y5G4XJQsNNIBiqyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-vhW6me4b
      WrBf7cHwEQJdXihidAYWVajJIaKMXMvFRMV6iDlRr076DFthg2_AV0_t
      SiV6xSEIFqt1xnYPpmP91tc5WJDOGb-wqjw0-b-S1laS11QVbuP78dQ7
      Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbeYSrRicJK5xodvWgkpIdk
      MHo4LvdhRRvzoKzlic89jFWPlnBq_V4n5trGuExtp_-dbHcGlihqc_wG
      gho9fLMK8JOArYLcMDNQ",
  "header": {
    "alg": "RSA1_5",
    "kid": "frodo.baggins@hobbiton.example"
  }
}
```

Figure 175: Recipient #1 JSON

4.13.4. Encrypting the Key to the Second Recipient

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

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

```
{
  "kty": "EC",
  "crv": "P-384",
  "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2Dt
      MRb25Ma2CX",
  "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEjI1pOMbw9
      1fzZ84pbfm",
  "d": "1DKHfTv-PiifVw2VBHM_ZiVcwOMxkOyANS_lQHJcrDxVY3jhVCvZPw
      MxJKIE793C"
}
```

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:

- Static Elliptic Curve public key (Figure 84).

- Ephemeral Elliptic Curve private key (Figure 176.

produces the following encrypted key:

```
ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw_elY4gSSId_w
```

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

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

- Recipient JWE Header from Figure 178.

```
{
  "alg": "ECDH-ES+A256KW",
  "kid": "peregrin.took@tuckborough.example",
  "epk": {
    "kty": "EC",
    "crv": "P-384",
    "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xAn2
        DtMRb25Ma2CX",
    "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEjI1pOMb
        w91fzZ84pbfm"
  }
}
```

Figure 178: Recipient #2 JWE Header JSON

The following is the assembled second recipient JSON:

```
{
  "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHixJuw
      _elY4gSSId_w",
  "header": {
    "alg": "ECDH-ES+A256KW",
    "kid": "peregrin.took@tuckborough.example",
    "epk": {
      "kty": "EC",
      "crv": "P-384",
      "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhsE2xA
          n2DtMRb25Ma2CX",
      "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEjI1pO
          Mbw91fzZ84pbfm"
    }
  }
}
```

Figure 179: Recipient #2 JSON

4.13.5. Encrypting the Key to the Third Recipient

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

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

```
AvpeoPZ9Ncn9mkBn
```

Figure 180: Recipient #2 Initialization Vector, base64url-encoded

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

- AES symmetric key (Figure 113; and
- Initialization vector/nonce ((Figure 180

produces the following:

- Encrypted key from Figure 181.
- Key wrap authentication tag from Figure 182

```
a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1E
```

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

```
59Nqh1LlYtVIhfD3pgRGvw
```

Figure 182: Recipient #3 Authentication Tag, base64url-encoded

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

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

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

Figure 183: Recipient #3 JWE Header JSON

The following is the assembled third recipient JSON:

```
{
  "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-WyTpS1
      E",
  "header": {
    "alg": "A256GCMKW",
    "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
    "tag": "59Nqh1LlYtVIhfD3pgRGvw",
    "iv": "AvpeoPZ9Ncn9mkBn"
  }
}
```

Figure 184: Recipient #3 JSON

4.13.6. Encrypting the Content

The following are generated before encrypting the content:

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

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

Figure 185: Protected JWE Header JSON

```
eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Figure 186: Protected JWE Header, base64url-encoded

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

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

produces the following:

- Ciphertext from Figure 187
- Authentication tag from Figure 188

```
ajm2Q-OpPXCr7-MHXicknb1lsxLdXxK_yLds0KuhJzfWK04SjdxQeSw2L9mu3a
_k1C55kCQ_3xlkcVKC5yr__Is48VOoK0k63_QRM9tBURMFqLByJ8vOYQX0oJW4
VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSaP6ga7-siYxStR7_G07Thd1jh-zGT0
wxM5g-VRORtq0K6AXpLlwEqRp7pkt2zRM0ZAXqSpe1O6FJ7FHLDyEFnD-zDIZu
kLpCbzhzMDLLw2-8I14FQrgi-iEuzHgIJFIJn2wh9Tj0cg_kOZy9BqMRZbmYXM
Y9YQjorZ_P_JYG3ARAIF3OjDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVH2
ofDQdLChtazE
```

Figure 187: Ciphertext, base64url-encoded

```
BESYyFN7T09KY7i8zKs5_g
```

Figure 188: Authentication Tag, base64url-encoded

The following is generated after encrypting the plaintext:

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

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

Figure 189: Unprotected JWE Header JSON

4.13.7. Output Results

The following compose the resulting JWE object:

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

The resulting JWE object using the JSON serialization:


```
{
  "recipients": [
    {
      "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zj
          wj4w6Y5G4XJQsNNIBiqyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-
          vhW6me4bWrBf7cHwEQJdXihidAYWVajJIaKMXMvFRMV6iDlRr076
          DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-wqjw0-b-S
          1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbe
          YSrRicJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n
          5trGuExtp_-dbHcGlihqc_wGgho9fLMK8JOArYLcMDNQ",
      "header": {
        "alg": "RSA1_5",
        "kid": "frodo.baggins@hobbiton.example"
      }
    },
    {
      "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHi
          xJuw_elY4gSSId_w",
      "header": {
        "alg": "ECDH-ES+A256KW",
        "kid": "peregrin.took@tuckborough.example",
        "epk": {
          "kty": "EC",
          "crv": "P-384",
          "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhs
              E2xAn2DtMRb25Ma2CX",
          "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEj
              I1pOMbw91fzZ84pbfm"
        }
      }
```

```
      },
      {
        "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-Wy
            TpS1E",
        "header": {
          "alg": "A256GCMKW",
          "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
          "tag": "59Nqh1LlYtVIhfD3pgRGvw",
          "iv": "AvpeoPZ9Ncn9mkBn"
        }
      }
    ],
    "unprotected": {
      "cty": "text/plain"
    },
    "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
    "iv": "VgEIHY20EnzUtZFl2RpB1g",
    "ciphertext": "ajm2Q-OpPXCr7-MHXicknb1lsxLdXxK_yLds0KuhJzfWK
        04SjdxQeSw2L9mu3a_k1C55kCQ_3xlkcVKC5yr__Is48VOoK0k63_QRM
        9tBURMFqLByJ8vOYQX0oJW4VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSa
        P6ga7-siYxStR7_G07Thd1jh-zGT0wxM5g-VRORtq0K6AXpLlwEqRp7p
        kt2zRM0ZAXqSpe1O6FJ7FHLDyEFnD-zDIZukLpCbzhzMDLLw2-8I14FQ
        rgi-iEuzHgIJFIJn2wh9Tj0cg_kOZy9BqMRZbmYXMY9YQjorZ_P_JYG3
        ARAIF3OjDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVH2ofDQdLCht
        azE",
    "tag": "BESYyFN7T09KY7i8zKs5_g"
  }
```

Figure 190: JSON Serialization

5. Nesting Signatures and Encryption

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

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

5.1. Signing Input Factors

The following are supplied before beginning the signing operation:

- Payload content; this example uses the JSON Web Token (JWT) [I-D.ietf-oauth-json-web-token] content from Figure 191, encoded as [RFC4648] base64url to produce Figure 192.

- RSA private key; this example uses the key from Figure 193

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

Figure 191: Payload content, in JSON format

```
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiZXhwIjoxMzAwODE5MzgwLCJodH
RwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0
```

Figure 192: Payload content, base64url-encoded

```
{
  "kty": "RSA",
  "kid": "hobbiton.example",
  "use": "sig",
  "n": "kNrPIBDXMU6fcyv5i-QHQAQ-K8gsC3HJb7FYhYaw8hXbNJa-t8q0lD
      KwLZgQXYV-ffWxXJv5GGrlZE4GU52lfMEegTDzYTrRQ3tepgKFjMGg6I
      y6fkl1ZNsx2gEonsnlShfzA9GJwRTmtKPbk1s-hwx1IU5AT-AIelNqBg
      cF2vE5W25_SGGBoaROVdUYxqETDggM1z5cKV4ZjDZ8-lh4oVB07bkac6
      LQdHpJUUySH_Er20DXx30Kyi97PciXKTS-QKXnmm8ivyRCmux22ZoPUi
      nd2BKC5OiG4MwALhaL2Z2k8CsRdfy-7dg7z41Rp6D0ZeEvtaUp4bX4aK
      raL4rTfw",
  "e": "AQAB",
  "d": "ZLe_TIxpE9-W_n2VBa-HWvuYPtjvxwVXClJFOpJsdea8g9RMx34qEO
      EtnoYc2un3CZ3LtJi-mju5RAT8YSc76YJds3ZVw0UiO8mMBeG6-iOnvg
      obobNx7K57-xjTJZU72EjOr9kB7z6ZKwDDq7HFyCDhUEcYcHFVc7iL_6
      TibVhAhOFONWlqlJgEgwVYd0rybNGKifdnpEbwyHoMwY6HM1qvnEFgP7
      iZ0YzHUT535x6jj4VKcdA7ZduFkhUauysySEW7mxZM6fj1vdjJIy9LD1
      fIz30Xv4ckoqhKF5GONU6tNmMmNgAD6gIViyEle1PrIxl1tBhCI14bRW
      -zrpHgAQ",
  "p": "yKWYoNIAqwMRQlgIBOdT1NIcbDNUUs2Rh-pBaxD_mIkweMt4Mg-0-B
      2iSYvMrs8horhonV7vxCQagcBAATGW-hAafUehWjxWSH-3KccRM8toL4
      e0q7M-idRDOBXSoe7Z2-CV2x_ZCY3RP8qp642R13WgXqGDIM4MbUkZSj
      cY9-c",
  "q": "uND4o15V30KDzf8vFJw589p1vlQVQ3NEilrinRUPHkkxaAzDzccGgr
      WMWpGxGFFnNL3w5CqPLeU76-5IVYQq0HwYVl0hVXQHr7sgaGu-483Ad3
      ENcL23FrOnF45m7_2ooAstJDe49MeLTTQKrSIBl_SKvqpYvfSPTczPcZ
      kh9Kk",
  "dp": "jmTnEoq2qqa8ouaymjhJSCnsveUXnMQC2gAneQJRQkFqQu-zV2PKP
      KNbPvKVyiF5b2-L3tM3OW2d2iNDyRUWXlT7V5l0KwPTABSTOnTqAmYCh
      Gi8kXXdlhcrtSvXldBakC6saxwI_TzGGY2MVXzc2ZnCvCXHV4qjSxOrf
      P3pHFU",
  "dq": "R9FUvU88OVzEkTkXl3-5-WusE4DjHmndeZIlu3rifBdfLpq_P-iWP
      BbGaq9wzQ1c-J7SzCdJqkEJDv5yd2C7rnZ6kpzwBh_nmL8zscAk1qsun
      nt9CJGAYz7-sGWy1JGShFazfP52ThB4rlCJ0YuEaQMrIzpY77_oLAhpm
      DA0hLk",
  "qi": "S8tC7ZknW6hPITkjcwttQOPLVmRfwirRlFAViuDb8NW9CrV_7F2Oq
      UZCqmzHTYAumwGFHI1WVRep7anleWaJjxC_1b3fq_al4qH3Pe-EKiHg6
      IMazuRtZLUROcThrExDbF5dYbsciDnfRUWLErZ4N1Be0bnxYuPqxwKd9
      QZwMo0"
}
```


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

5.2. Signing Operation

The following are generated to complete the signing operation:

- Protected JWS Header; this example uses header from Figure 194, encoded using [RFC4648] base64url to produce Figure 195.

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

Figure 194: Protected JWS Header JSON

```
eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9
```

Figure 195: Protected JWS Header, base64url-encoded

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

```
dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIA
kIvn1qskChirjKvY9ESZNUCP4JjvfyPS-nqjJxYoA5ztWOyFk2cZNIPXjcJXSQ
wXPO9tEe-v4VSqgD0aKHqPxYog4N6Cz1lKph1U1sYDSI67_bLL7elg_vkjfMp5
_W5l5LuUYGMeh6hxQIaIUXf9EwV2JmvTMuZ-vBOWy0Sniy1EFo72CRTvmtrIf5
AROo5MNliY3KtUxeP-SOmD-LEYwW9SlkohYzMVAZDDOrVbv7KVRHpeYNaK75KE
QqdCEEkS_rskZS-Qtt_nlegTWh1mEYaA
```

Figure 196: Signature, base64url-encoded

5.3. Signing Output

The following compose the resulting JWS object:

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

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

```
eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9
.
eyJpc3MiOiJob2JiaXRvbi5leGFtcGxlIiwiZXhwIjoxMzAwODE5MzgwLCJodH
RwOi8vZXhhbXBsZS5jb20vaXNfcm9vdCI6dHJ1ZX0
.
dPpMqwRZxFYi1UfcDAaf8M99o7kwUWtiXZ-ByvVuJih4MhJ_aZqciprz0OWaIA
kIvn1qskChirjKvY9ESZNUCP4JjvfyPS-nqjJxYoA5ztWOyFk2cZNIPXjcJXSQ
wXPO9tEe-v4VSqgD0aKHqPxYog4N6Cz1lKph1U1sYDSI67_bLL7elg_vkjfMp5
_W5l5LuUYGMeh6hxQIaIUXf9EwV2JmvTMuZ-vBOWy0Sniy1EFo72CRTvmtrIf5
AROo5MNliY3KtUxeP-SOmD-LEYwW9SlkohYzMVAZDDOrVbv7KVRHpeYNaK75KE
QqdCEEkS_rskZS-Qtt_nlegTWh1mEYaA
```

Figure 197: Compact Serialization

5.4. Encryption Input Factors

The following are supplied before beginning the encryption process:

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

5.5. Encryption Generated Factors

The following are generated before encrypting:

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

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

Figure 198: Content Encryption Key, base64url-encoded

```
GbX1i9kXz0sxXPmA
```

Figure 199: Initialization vector, base64url-encoded

5.6. Encrypting the Key

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

```
a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueYlVhjurCyrBg3I7YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bi13mwy7RoYhjTkBEC6P7sMYMXXx4g
zMedpiJHQVeyI-zkZV7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb21
O0ul4YxSHV-ByK1kyeetRp_fuYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e5I
R7nany-25_UmC2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dMVIY7q1eKpXcV1lWO_2FefEBqXxXvIjLeZivjNkzogCq3-I
apSjVFnMjBxjpYLT8muaawo1yy1XXMuinIpNcOY3n4KKrXLrCcteX85m4IIHMZ
a38s1Hpr56fPPseMA-Jltmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBKOYwfk7JhLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNJtqnblp
ymooeWAHCT4e_Owbim1g0AEpTHUdA2iiLNs9WTX_H_TXuPC8yDDhi1smxS_X_x
pkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcxqX2Fo_GnVrNwlK7Lgxw6FSQvDO
0
```

Figure 200: Encrypted Key, base64url-encoded

5.7. Encrypting the Content

The following are generated before encrypting the plaintext:

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

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

Figure 201: Protected JWE Header JSON

```
eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ
```

Figure 202: Protected JWE Header, base64url-encoded

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

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

produces the following:

- Ciphertext from Figure 203.

o Authentication tag from Figure 204.

```
SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBNgCe2OFMf66cSJ8k2Q
kxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt9OAvVLsAXB0_UTCBGyBg3C2bWLX
qZlfJAAoJRUPRk-BimYZY81zVBuIhc7HsQePCpu33SzMsFHjn4lP_idrJz_glZ
TNgKDt8zdnUPauKTKDNOH1DD4fuzvDYfDIAfqGPyL5sVRwbiXpXdGokEszM-9C
hMPqW1QNhzuX_Zul3bvrJwr7nuGZs4cUScY3n8yE3AHCLurgls-A9mz1X38xEa
ulV18l4Fg9tLejdkAuQZjPbqeHQBJe4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2
zI3Q_1sYjKUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUr
lx4gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-aM7tXg26qJo21gYjLf
hn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5XmnwZMyNc
9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgMl7o03phcTMxtlMizR88NKU1WkB
siXMCjy1Noue7MD-ShDp5dmM
```

Figure 203: Ciphertext, base64url-encoded

```
KnIKEhN8U-3C9s4gtSpjSw
```

Figure 204: Authentication tag, base64url-encoded

5.8. Encryption Output

The following compose the resulting JWE object:

o Protected JWE Header (Figure 202)

o Encrypted key (Figure 200)

o Initialization vector/nonce (Figure 199)

o Ciphertext (Figure 203)

o Authentication Tag (Figure 204)

The resulting JWE object using the Compact serialization:

```
eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ
.
a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueYlVhjurCyrBg3I7YhCRYjphDOOS4
E7rXbr2Fn6NyQq-A-gqT0FXqNjVOGrG-bi13mwy7RoYhjTkBEC6P7sMYMXXx4g
zMedpiJHQVeyI-zkZV7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb21
O0ul4YxSHV-ByK1kyeetRp_fuYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e5I
R7nany-25_UmC2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5o6yV64x6yzDU
F_5JCIdl-Qv6H5dMVIY7q1eKpXcV1lWO_2FefEBqXxXvIjLeZivjNkzogCq3-I
apSjVFnMjBxjpYLT8muaawo1yy1XXMuinIpNcOY3n4KKrXLrCcteX85m4IIHMZ
a38s1Hpr56fPPseMA-Jltmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAa
mBKOYwfk7JhLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNJtqnblp
ymooeWAHCT4e_Owbim1g0AEpTHUdA2iiLNs9WTX_H_TXuPC8yDDhi1smxS_X_x
pkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcxqX2Fo_GnVrNwlK7Lgxw6FSQvDO
0
.
GbX1i9kXz0sxXPmA
.
SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBNgCe2OFMf66cSJ8k2Q
kxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt9OAvVLsAXB0_UTCBGyBg3C2bWLX
qZlfJAAoJRUPRk-BimYZY81zVBuIhc7HsQePCpu33SzMsFHjn4lP_idrJz_glZ
TNgKDt8zdnUPauKTKDNOH1DD4fuzvDYfDIAfqGPyL5sVRwbiXpXdGokEszM-9C
hMPqW1QNhzuX_Zul3bvrJwr7nuGZs4cUScY3n8yE3AHCLurgls-A9mz1X38xEa
ulV18l4Fg9tLejdkAuQZjPbqeHQBJe4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2
zI3Q_1sYjKUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUr
lx4gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-aM7tXg26qJo21gYjLf
hn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5XmnwZMyNc
9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgMl7o03phcTMxtlMizR88NKU1WkB
siXMCjy1Noue7MD-ShDp5dmM
.
KnIKEhN8U-3C9s4gtSpjSw
```

Figure 205: Compact Serialization

The resulting JWE object using the JSON serialization:

```
{
  "recipients": [
    {
      "encrypted_key": "a0JHRoITfpX4qRewImjlStn8m3CPxBV1ueYlVh
          jurCyrBg3I7YhCRYjphDOOS4E7rXbr2Fn6NyQq-A-gqT0FXqNjVO
          GrG-bi13mwy7RoYhjTkBEC6P7sMYMXXx4gzMedpiJHQVeyI-zkZV
          7A9matpgevAJWrXzOUysYGTtwoSN6gtUVtlLaivjvb21O0ul4YxS
          HV-ByK1kyeetRp_fuYJxHoKLQL9P424sKx2WGYb4zsBIPF4ssl_e
          5IR7nany-25_UmC2urosNkoFz9cQ82MypZP8gqbQJyPN-Fpp4Z-5
          o6yV64x6yzDUF_5JCIdl-Qv6H5dMVIY7q1eKpXcV1lWO_2FefEBq
          XxXvIjLeZivjNkzogCq3-IapSjVFnMjBxjpYLT8muaawo1yy1XXM
          uinIpNcOY3n4KKrXLrCcteX85m4IIHMZa38s1Hpr56fPPseMA-Jl
          tmt-a9iEDtOzhtxz8AXy9tsCAZV2XBWNG8c3kJusAamBKOYwfk7J
          hLRDgOnJjlJLhn7TI4UxDp9dCmUXEN6z0v23W15qJIEXNJtqnblp
          ymooeWAHCT4e_Owbim1g0AEpTHUdA2iiLNs9WTX_H_TXuPC8yDDh
          i1smxS_X_xpkIHkiIHWDOLx03BpqDTivpKkBYwqP2UZkcxqX2Fo_
          GnVrNwlK7Lgxw6FSQvDO0"
    }
  ],
  "protected": "eyJhbGciOiJSU0EtT0FFUCIsImN0eSI6IkpXVCIsImVuYy
      I6IkExMjhHQ00ifQ",
  "iv": "GbX1i9kXz0sxXPmA",
  "ciphertext": "SZI4IvKHmwpazl_pJQXX3mHv1ANnOU4Wf9-utWYUcKrBN
      gCe2OFMf66cSJ8k2QkxaQD3_R60MGE9ofomwtky3GFxMeGRjtpMt9OAv
      VLsAXB0_UTCBGyBg3C2bWLXqZlfJAAoJRUPRk-BimYZY81zVBuIhc7Hs
      QePCpu33SzMsFHjn4lP_idrJz_glZTNgKDt8zdnUPauKTKDNOH1DD4fu
      zvDYfDIAfqGPyL5sVRwbiXpXdGokEszM-9ChMPqW1QNhzuX_Zul3bvrJ
      wr7nuGZs4cUScY3n8yE3AHCLurgls-A9mz1X38xEaulV18l4Fg9tLejd
      kAuQZjPbqeHQBJe4IwGD5Ee0dQ-Mtz4NnhkIWx-YKBb_Xo2zI3Q_1sYj
      KUuis7yWW-HTr_vqvFt0bj7WJf2vzB0TZ3dvsoGaTvPH2dyWwumUrlx4
      gmPUzBdwTO6ubfYSDUEEz5py0d_OtWeUSYcCYBKD-aM7tXg26qJo21gY
      jLfhn9zy-W19sOCZGuzgFjPhawXHpvnj_t-0_ES96kogjJLxS1IMU9Y5
      XmnwZMyNc9EIwnogsCg-hVuvzyP0sIruktmI94_SL1xgMl7o03phcTMx
      tlMizR88NKU1WkBsiXMCjy1Noue7MD-ShDp5dmM",
  "tag": "KnIKEhN8U-3C9s4gtSpjSw"
}
```


Figure 206: JSON Serialiation

6. Security Considerations

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

7. IANA Considerations

This document has no actions for IANA.

8. Informative References

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-31 (work in progress), July 2014.

[I-D.ietf-jose-json-web-encryption]
Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-31 (work in progress), July 2014.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-31 (work in progress), July 2014.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-31 (work in progress), July 2014.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-25 (work in progress), July 2014.

[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.

[RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, January 2014.

Appendix A. Acknowledgements

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

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

Author's Address

Matthew Miller
Cisco Systems, Inc.

Email: mamille2@cisco.com