JOSE Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 28, 2015

M. Jones
Microsoft

May 27, 2015

Key Managed JSON Web Signature (KMJWS)
draft-jones-jose-key-managed-json-web-signature-01

Abstract

Key Managed JSON Web Signature (KMJWS) represents content that is integrity protected with a Message Authentication Code (MAC) in which key management is employed for the MAC key. This representation reuses key management functionality already present in the JSON Web Encryption (JWE) specification and MAC functionality already present in the JSON Web Signature (JWS) specification.

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 November 28, 2015.

Copyright Notice

Table of Contents

1. Introduction

Key Managed JSON Web Signature (KMJWS) represents content that is integrity protected with a Message Authentication Code (MAC) in which key management is employed for the MAC key. This representation reuses key management functionality already present in the "JSON Web Encryption (JWE)" [JWE] specification and MAC functionality already present in the "JSON Web Signature (JWS)" [JWS] specification.

A KMJWS is neither a JWS nor a JWE, but incorporates elements of both. Specifically, the Key Management algorithms registered in the JSON Web Signature and Encryption Algorithms Registry [IANA.JOSE.Algs] are used to provide MAC keys in the same way that they are used to provide content encryption keys in "JSON Web Encryption (JWE)" [JWE]. Likewise, the MAC algorithms registered in this registry are used to integrity protect the JWS Payload and JWS Protected Header in the same way that they are used to integrity protect the JWS Payload and JWS Protected Header in "JSON Web Signature (JWS)" [JWS].

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119]. The interpretation should only be applied when the terms appear in all capital letters.

UTF8(STRING) denotes the octets of the UTF-8 [RFC3629] representation of STRING, where STRING is a sequence of zero or more Unicode [UNICODE] characters.

ASCII(STRING) denotes the octets of the ASCII [RFC20] representation of STRING, where STRING is a sequence of zero or more ASCII characters.

The concatenation of two values A and B is denoted as A || B.

2. Terminology

This specification uses the same terminology as the "JSON Web Signature (JWS)" [JWS], "JSON Web Encryption (JWE)" [JWE], and "JSON Web Algorithms (JWA)" [JWA] specifications.

These terms are defined by this specification:

Key Managed JSON Web Signature (KMJWS)
: A data structure employing key management representing a MACed message.

MAC Key
: A symmetric key for the MAC algorithm used to integrity protect the JWS Payload and the JWS Protected Header.

KMJWS Encrypted Key
: Encrypted MAC Key. Note that for some algorithms, the KMJWS Encrypted Key value is specified as being the empty octet sequence.

3. Example KMJWS

This section provides an example of a KMJWS. Its computation is described in more detail in Appendix A, including specifying the key values used.

The following example JWS Protected Header declares that:

- The MAC Key is encrypted using the RSAES OAEP [RFC3447] algorithm to produce the KMJWS Encrypted Key.
- The JWS Protected Header and the JWS Payload are integrity protected using the HMAC SHA-256 [RFC2104] [SHS] algorithm.

```
{"alg":"RSA-OAEP","mac":"HS256"}
```

Encoding this JWS Protected Header as BASE64URL(UTF8(JWS Protected Header)) gives this value:

```
eyJhbGciOiJSU0EtT0FFUCIsIm1hYyI6IkhTMjU2In0
```

The payload in this example is the ASCII representation of the text "What I have written, I have written." The value BASE64URL(JWS Payload) is:

```
V2hhdCBJIGhhdmUgd3JpdHRlbiwgSSBoYXZlIHdyaXR0ZW4u
```

Computing the HMAC of the JWS Signing Input ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)) with the HMAC SHA-256 algorithm using the MAC Key specified in Appendix A.4 and base64url-encoding the result yields this BASE64URL(JWS Signature) value:

```
NjTOnXAAXtr7dA6RSxYkZcD6F-n5BOrLHRTxiTLptKM
```

The MAC Key is encrypted using the RSAES OAEP algorithm and the RSA key specified in Appendix A.5. The resulting BASE64URL(KMJWS Encrypted Key) value (with line breaks for display purposes only) is:

```
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg
```

Concatenating these values in the order Header.Payload.Signature.Encrypted_Key with period ('.') characters between the parts yields this complete KMJWS representation using the KMJWS Compact Serialization (with line breaks for display purposes only):

```
eyJhbGciOiJSU0EtT0FFUCIsIm1hYyI6IkhTMjU2In0
.
V2hhdCBJIGhhdmUgd3JpdHRlbiwgSSBoYXZlIHdyaXR0ZW4u
.
NjTOnXAAXtr7dA6RSxYkZcD6F-n5BOrLHRTxiTLptKM
.
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg
```

4. KMJWS Contents

A Key Managed JSON Web Signature (KMJWS) represents these logical values:

- JOSE Header
- JWS Payload
- JWS Signature
- KMJWS Encrypted Key

For a KMJWS, just as it is for a JWS, the JOSE Header members are the union of the members of these values:

- JWS Protected Header
- JWS Unprotected Header

The JWS Payload and JWS Signature are likewise the same for a KMJWS are they are for a JWS.

The KMJWS Encrypted Key is the one value present in a KMJWS that is not present in a JWS. It enables key management for the MAC Key.

5. Header Parameters

A KMJWS uses these Header Parameter fields to convey the key management and MAC algorithms used:

alg
: The key management algorithm employed. This parameter has the same meaning, syntax, and processing rules as the "alg" Header Parameter defined in Section 4.1.1 of [JWE], except that the key being encrypted or determined is the MAC key, rather than the Content Encryption Key.

mac
: This parameter has the same meaning, syntax, and processing rules as the "alg" Header Parameter defined in Section 4.1.1 of [JWS], except that the algorithm MUST be a MAC algorithm and the MAC key is determined by the key management procedure employed.

The "dir" (direct) key management algorithm defined in Section 4.5 of [JWA] MUST NOT be used as the "alg" Header Parameter value, as doing so would unnecessarily create a second equivalent representation for content integrity-protected with a MAC with no key management. The normal JWS representation for this MUST be used instead.

The "jku", "jwk", "kid", "x5u", "x5c", "x5t", "x5t#S256", "typ", and "crit" Header Parameters defined in Section 4.1 of [JWE] are used identically, except that the key being encrypted or determined is the MAC key, rather than the Content Encryption Key. The "cty" Header Parameter defined in Section 4.1.10 of [JWS] is used identically, except that the JWS Payload is that of a KMJWS, rather than a JWS. The "enc" and "zip" Header Parameters defined in Section 4.1 of [JWE] MUST NOT be used.

6. Serializations

Like JWSs, KMJWSs can utilize one of two different serializations: the KMJWS Compact Serialization or the KMJWS JSON Serialization.

6.1. JWS Compact Serialization

Like the JWS Compact Serialization, the KMJWS Compact Serialization represents MACed content as a compact, URL-safe string. This string is:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature) || '.' ||
BASE64URL(KMJWS Encrypted Key)
```

Only one MAC is supported by the KMJWS Compact Serialization and it provides no syntax to represent a JWS Unprotected Header value.

6.2. JWS JSON Serialization

Like the JWS JSON Serialization, the KMJWS JSON Serialization represents MACed content as a JSON object RFC 7159 [RFC7159]. This representation is neither optimized for compactness nor URL-safe.

Also like the JWS JSON Serialization, two closely related syntaxes are defined for the KMJWS JSON Serialization: a fully general syntax, with which content can be secured with more than one MAC operation, and a flattened syntax, which is optimized for the single MAC case.

6.2.1. General KMJWS JSON Serialization Syntax

The following members are defined for use in top-level JSON objects used for the fully general KMJWS JSON Serialization syntax:

payload
: This is the same as the JWS "payload" member.

signatures
: This is the same as the JWS "signatures" member.

The following members are defined for use in the JSON objects that are elements of the "signatures" array:

protected
: This is the same as the JWS "protected" member.

header
: This is the same as the JWS "header" member.

signature
: This is the same as the JWS "signature" member.

encrypted_key
: The "encrypted_key" member MUST be present and contain the value BASE64URL(KMJWS Encrypted Key).

At least one of the "protected" and "header" members MUST be present for each MAC computation so that "alg" and "mac" Header Parameter values are conveyed.

Additional members can be present in both the JSON objects defined above; if not understood by implementations encountering them, they MUST be ignored.

The Header Parameter values used when creating or validating individual MAC values are the union of the two sets of Header Parameter values that may be present: (1) the JWS Protected Header represented in the "protected" member of the MAC's array element, and (2) the JWS Unprotected Header in the "header" member of the MAC's array element. The union of these sets of Header Parameters comprises the JOSE Header. The Header Parameter names in the two locations MUST be disjoint.

Each JWS Signature value is computed using the parameters of the corresponding JOSE Header value in the same manner as for the JWS Compact Serialization. This has the desirable property that each JWS Signature value represented in the "signatures" array is identical to the value that would have been computed for the same parameter in the KMJWS Compact Serialization, provided that the JWS Protected Header value for that MAC computation (which represents the integrity protected Header Parameter values) matches that used in the KMJWS Compact Serialization.

In summary, the syntax of a KMJWS using the general KMJWS JSON Serialization is as follows:

```
{
 "payload":"<payload contents>",
 "signatures":[
  {"protected":"<integrity-protected header 1 contents>",
   "header":<non-integrity-protected header 1 contents>,
   "signature":"<signature 1 contents>",
   "encrypted_key":"<encrypted key 1 contents>"},
  ...
  {"protected":"<integrity-protected header N contents>",
   "header":<non-integrity-protected header N contents>,
   "signature":"<signature N contents>",
   "encrypted_key":"<encrypted key N contents>"}]
}
```

6.2.2. Flattened KMJWS JSON Serialization Syntax

The flattened KMJWS JSON Serialization syntax is based upon the general syntax, but flattens it in the same way that the flattened JWS JSON Serialization syntax flattens its general syntax.

In summary, the syntax of a KMJWS using the flattened KMJWS JSON Serialization is as follows:

```
{
 "payload":"<payload contents>",
 "protected":"<integrity-protected header contents>",
 "header":<non-integrity-protected header contents>,
 "signature":"<signature contents>",
 "encrypted_key":"<encrypted key contents>"
}
```

7. Distinguishing between KMJWS, JWS, and JWE Objects

While KMJWSs have characteristics of both JWSs and JWEs, these methods can be used to distinguish KMJWSs from either of them. This section augments the information in Section 9 of [JWE].

- If the object is using a compact serialization, the number of base64url-encoded segments separated by period ('.') characters will differ. KMJWSs have four segments separated by three period ('.') characters and the others do not.

- If the object is using a JSON serialization, the members used will be different. KMJWSs have both a "payload" and an "encrypted_key"

member and the others do not.

- The JOSE Header for a KMJWS can also be distinguished from the JOSE Header for a JWS or JWE by determining whether a "mac" (MAC algorithm) member exists. If the "mac" member exists, it is a KMJWS; otherwise, it is not.

8. IANA Considerations

8.1. JWS and JWE Header Parameter Registration

This specification registers the "mac" (MAC algorithm) Header Parameter defined in Section 5 in the IANA JSON Web Signature and Encryption Header Parameters registry defined in [JWS].

8.1.1. Registry Contents

- Header Parameter Name: "mac"
- Header Parameter Description: MAC Algorithm
- Header Parameter Usage Location(s): KMJWS
- Change Controller: IETF
- Specification Document(s): Section 5 of [[this document]]

9. Security Considerations

The key management security considerations from [JWE] apply. The integrity protection security considerations from [JWS] apply. The algorithm security considerations from [JWA] apply.

10. References

10.1. Normative References

[IANA.JOSE.Algs]
Internet Assigned Numbers Authority (IANA), "JSON Web Signature and Encryption Algorithms Registry", <http://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms-types>.

[JWA] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, May 2015, <http://www.rfc-editor.org/info/rfc7518>.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC20] Cerf, V., "ASCII format for Network Interchange", STD 80, RFC 20, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

10.2. Informative References

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

Appendix A. Example KMJWS using RSAES OAEP and HMAC SHA-256

This example secures the payload using RSAES OAEP for key encryption and HMAC SHA-256 for integrity protection.

A.1. JOSE Header

The following example JWS Protected Header declares that:

- The MAC Key is encrypted using the RSAES OAEP [RFC3447] algorithm to produce the KMJWS Encrypted Key.

- The JWS Protected Header and the JWS Payload are integrity protected using the HMAC SHA-256 [RFC2104] [SHS] algorithm.

```
{"alg":"RSA-OAEP","mac":"HS256"}
```

Encoding this JWS Protected Header as BASE64URL(UTF8(JWS Protected Header)) gives this value:

```
eyJhbGciOiJSU0EtT0FFUCIsIm1hYyI6IkhTMjU2In0
```

A.2. Payload

The payload in this example is the ASCII representation of the text "What I have written, I have written." The representation of this payload (using JSON array notation) is:

```
[87, 104, 97, 116, 32, 73, 32, 104, 97, 118, 101, 32, 119, 114, 105,
116, 116, 101, 110, 44, 32, 73, 32, 104, 97, 118, 101, 32, 119, 114,
105, 116, 116, 101, 110, 46]
```

The value BASE64URL(JWS Payload) is:

```
V2hhdCBJIGhhdmUgd3JpdHRlbiwgSSBoYXZlIHdyaXR0ZW4u
```

A.3. JWS Signing Input

Combining these as BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload) gives this string (with line breaks for display purposes only):

```
eyJhbGciOiJSU0EtT0FFUCIsIm1hYyI6IkhTMjU2In0
.
V2hhdCBJIGhhdmUgd3JpdHRlbiwgSSBoYXZlIHdyaXR0ZW4u
```

The resulting JWS Signing Input value, which is the ASCII representation of above string, is the following octet sequence (using JSON array notation):

```
[101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 83, 85, 48, 69,
116, 84, 48, 70, 70, 85, 67, 73, 115, 73, 109, 49, 104, 89, 121, 73,
54, 73, 107, 104, 84, 77, 106, 85, 50, 73, 110, 48, 46, 86, 50, 104,
104, 100, 67, 66, 74, 73, 71, 104, 104, 100, 109, 85, 103, 100, 51,
74, 112, 100, 72, 82, 108, 98, 105, 119, 103, 83, 83, 66, 111, 89,
88, 90, 108, 73, 72, 100, 121, 97, 88, 82, 48, 90, 87, 52, 117]
```

A.4. Integrity Protection

Compute the HMAC of the JWS Signing Input with the HMAC SHA-256 algorithm. This example uses the MAC Key below:

[177, 161, 244, 128, 84, 143, 225, 115, 63, 180, 3, 255, 107, 154, 212, 246, 138, 7, 110, 91, 112, 46, 34, 105, 47, 130, 203, 46, 122, 234, 64, 252]

The resulting JWS Signature value is:

[54, 52, 206, 157, 112, 0, 94, 218, 251, 116, 14, 145, 75, 22, 36, 101, 192, 250, 23, 233, 249, 4, 234, 203, 29, 20, 241, 137, 50, 233, 180, 163]

Base64url-encoding the result yields this BASE64URL(JWS Signature) value:

NjTOnXAAXtr7dA6RSxYkZcD6F-n5BOrLHRTxiTLptKM

A.5. Key Encryption

Encrypt the MAC Key with the recipient's public key using the RSAES OAEP algorithm to produce the KMJWS Encrypted Key. This example uses the RSA key represented in JWK [JWK] format below (with line breaks within values for display purposes only):

```
    {"kty":"RSA",
     "n":"oahUIoWw0K0usKNuOR6H4wkf4oBUXHTxRvgb48E-BVvxkeDNjbC4he8rUW
          cJoZmds2h7M70imEVhRU5djINXtqllXI4DFqcI1DgjT9LewND8MW2Krf3S
          psk_ZkoFnilakGygTwpZ3uesH-PFABNIUYpOiN15dsQRkgr0vEhxN92i2a
          sbOenSZeyaxziK72UwxrrKoExv6kc5twXTq4h-QChLOln0_mtUZwfsRaMS
          tPs6mS6XrgxnxbWhojf663tuEQueGC-FCMfra36C9knDFGzKsNa7LZK2dj
          YgyD3JR_MB_4NUJW_TqOQtwHYbxevoJArm-L5StowjzGy-_bq6Gw",
     "e":"AQAB",
     "d":"kLdtIj6GbDks_ApCSTYQtelcNttlKiOyPzMrXHeI-yk1F7-kpDxY4-WY5N
          WV5KntaEeXS1j82E375xxhWMHXyvjYecPT9fpwR_M9gV8n9Hrh2anTpTD9
          3Dt62ypW3yDsJzBnTnrYu1iwWRgBKrEYY46qAZIrA2xAwnm2X7uGR1hghk
          qDp0Vqj3kbSCz1XyfCs6_LehBwtxHIyh8Ripy40p24moOAbgxVw3rxT_vl
          t3UVe4WO3JkJOzlpUf-KTVI2Ptgm-dARxTEtE-id-4OJr0h-K-VFs3VSnd
          VTIznSxfyrj8ILL6MG_Uv8YAu7VILSB3lOW085-4qE3DzgrTjgyQ",
     "p":"1r52Xk46c-LsfB5P442p7atdPUrxQSy4mti_tZI3Mgf2EuFVbUoDBvaRQ-
          SWxkbkmoEzL7JXroSBjSrK3YIQgYdMgyAEPTPjXv_hI2_1eTSPVZfzL0lf
          fNn03IXqWF5MDFuoUYE0hzb2vhrlN_rKrbfDIwUbTrjjgieRbwC6Cl0",
     "q":"wLb35x7hmQWZsWJmB_vle87ihgZ19S8lBEROLIsZG4ayZVe9Hi9gDVCOBm
          UDdaDYVTSNx_8Fyw1YYa9XGrGnDew00J28cRUoeBB_jKI1oma0Orv1T9aX
          IWxKwd4gvxFImOWr3QRL9KEBRzk2RatUBnmDZJTIAfwTs0g68UZHvtc",
     "dp":"ZK-YwE7diUh0qR1tR7w8WHtolDx3MZ_OTowiFvgfeQ3SiresXjm9gZ5KL
          hMXvo-uz-KUJWDxS5pFQ_M0evdo1dKiRTjVw_x4NyqyXPM5nULPkcpU827
          rnpZzAJKpdhWAgqrXGKAECQH0Xt4taznjnd_zVpAmZZq60WPMBMfKcuE",
     "dq":"Dq0gfgJ1DdFGXiLvQEZnuKEN0UUmsJBxkjydc3j4ZYdBiMRAy86x0vHCj
          ywcMlYYg4yoC4YZa9hNVcsjqA3FeiL19rk8g6Qn29Tt0cj8qqyFpz9vNDB
          UfCAiJVeESOjJDZPYHdHY8v1b-o-Z2X5tvLx-TCekf7oxyeKDUqKWjis",
     "qi":"VIMpMYbPf47dT1w_zDUXfPimsSegnMOA1zTaX7aGk_8urY6R8-ZW1FxU7
          AlWAyLWybqq6t16VFd7hQd0y6flUK4SlOydB61gwanOsXGOAOv82cHq0E3
          eL4HrtZkUuKvnPrMnsUUFlfUdybVzxyjz9JF_XyaY14ardLSjf4L_FNY"
    }
```


The resulting KMJWS Encrypted Key value is:


```
[56, 163, 154, 192, 58, 53, 222, 4, 105, 218, 136, 218, 29, 94, 203,
22, 150, 92, 129, 94, 211, 232, 53, 89, 41, 60, 138, 56, 196, 216,
82, 98, 168, 76, 37, 73, 70, 7, 36, 8, 191, 100, 136, 196, 244, 220,
145, 158, 138, 155, 4, 117, 141, 230, 199, 247, 173, 45, 182, 214,
74, 177, 107, 211, 153, 11, 205, 196, 171, 226, 162, 128, 171, 182,
13, 237, 239, 99, 193, 4, 91, 219, 121, 223, 107, 167, 61, 119, 228,
173, 156, 137, 134, 200, 80, 219, 74, 253, 56, 185, 91, 177, 34, 158,
89, 154, 205, 96, 55, 18, 138, 43, 96, 218, 215, 128, 124, 75, 138,
243, 85, 25, 109, 117, 140, 26, 155, 249, 67, 167, 149, 231, 100, 6,
41, 65, 214, 251, 232, 87, 72, 40, 182, 149, 154, 168, 31, 193, 126,
215, 89, 28, 111, 219, 125, 182, 139, 235, 195, 197, 23, 234, 55, 58,
63, 180, 68, 202, 206, 149, 75, 205, 248, 176, 67, 39, 178, 60, 98,
193, 32, 238, 122, 96, 158, 222, 57, 183, 111, 210, 55, 188, 215,
206, 180, 166, 150, 166, 106, 250, 55, 229, 72, 40, 69, 214, 216,
104, 23, 40, 135, 212, 28, 127, 41, 80, 175, 174, 168, 115, 171, 197,
```

89, 116, 92, 103, 246, 83, 216, 182, 176, 84, 37, 147, 35, 45, 219,
172, 99, 226, 233, 73, 37, 124, 42, 72, 49, 242, 35, 127, 184, 134,
117, 114, 135, 206]

Encoding this KMJWS Encrypted Key as BASE64URL(KMJWS Encrypted Key) gives this value (with line breaks for display purposes only):

```
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg
```

A.6. Complete Representation

Assemble the final representation: The Compact Serialization of this result is the string BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload) || '.' || BASE64URL(JWS Signature) || '.' || BASE64URL(KMJWS Encrypted Key).

The final result in this example (with line breaks for display purposes only) is:

```
eyJhbGciOiJSU0EtT0FFUCIsIm1hYyI6IkhTMjU2In0
.
V2hhdCBJIGhhdmUgd3JpdHRlbiwgSSBoYXZlIHdyaXR0ZW4u
.
NjTOnXAAXtr7dA6RSxYkZcD6F-n5BOrLHRTxiTLptKM
.
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg
```

Appendix B. Acknowledgements

Richard Barnes and Jim Schaad contributed to discussions on the possibility of introducing key management as an option for JWSs using MACs.

Appendix C. Document History

[[to be removed by the RFC editor before publication as an RFC]]

-01

- Prohibited using "dir" as the "alg" Header Parameter value.

-00

- Created draft-jones-jose-key-managed-json-web-signature.

Author's Address

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/