JOSE Working Group
Internet-Draft
Intended status: Informational
Expires: November 7, 2013

R. Barnes
BBN

May 06, 2013

Proposed Refactoring of JOSE to Align Encryption and Key Wrapping
draft-barnes-jose-key-wrapping-01

Abstract

The discussions around key wrapping in the JOSE working group have raised new requirements for wrapped keys, namely: (1) Wrapping keys other than symmetric keys, (2) cryptographically binding attributes to keys, and (3) allowing the use of AEAD cryptographic algorithms for key wrapping (other than AES-KW). This document proposes a refactoring of the JOSE document set that provides a cleaner conceptual structure for JWS / JWE and transparent support for wrapped keys, all with a relatively minor impact on the compact form of JWS and JWE objects.

Table of Contents

1. Introduction

The goal of a JOSE object is to provide the recipient with the result of a cryptographic operation (encryption, MAC, or signature) and instructions for how to process that result. These instructutions are in two main parts: (1) A cryptographic algorithm to be applied, and (2) the key to be used with that algorithm (in wrapped form).

The current structure of the JWE and JWS headers scatters these two information elements across several different header parameters. JWE contains parameters for two different encryptions, the content encryption and the key encryption. It would be simpler if these two encryptions could be handled in the same way. Namely, the encrypted key within a JWE should become a JWE itself. Thus, we make the following proposal:

1. Define a simple, JWE-based format for wrapped keys. This provides a consolidated format for wrapped keys, within JWE and without.

2. Define the "recipients" parameter in JWE to be an array of JWE objects representeing wrapped keys.

3. Add a "kdf" parameter which is used to signal the use of a key derivation function such as ECDH or PBKDF2. This takes the place of the "alg" parameter for ECDH, and is the natural way to include PBKDF2.

4. Adjust the mapping between the compact and JSON serializations to account for the above changes.

These major changes will then drive several smaller, more editorial changes, including:

- Remove the "dir" value of the "alg" parameter. The use of direct encryption is signalled by the lack of a "recipients" parameter with wrapped keys.

- Removing the "ECDH-ES+A128KW" value of the "alg" parameter, since this is specified jointly, e.g., by setting "enc" to "A128KW" and "kdf" to "ECDH-ES".

- Combine "alg" and "enc" registries for JWE into a single registry, since the same set of algorithms can now be used for content encryption and key encryption.

- Remove the "alg" parameter from JWE. It is no longer necessary, since the algorithm used for key wrapping is signalled in the

"enc" parameter of the inner JWE.

This restructuring simplifies key wrapping, by using JWE for key wrapping. Since the encrypted CMK is just another encrypted object, and because the wrapped key is encapsulated in the "key" object, JOSE implementations need only a single "unwrap" operation, instead of separate "JWE unwrap" and "key unwrap" operations.

It also resolves the ambiguity that arises in the current JWE design as to what parameters should go in the overall header vs. per-recipient headers. Namely, the parameters in the overall header relate to encryption of content, and parameters in per-recipient content relate to encryption of keys.

```
CURRENT:
{
    "header": { "enc": "A128GCM" },
    "recipients:" [{
        "header": { "alg": "A128KW", "kid": "1" },
        "encrypted_key": "..."
    }],
    "initialization_vector": "...",
    "ciphertext": "...",
    "authentication_tag": "..."
}

PROPOSED:
{
    "header": { "enc": "A128GCM" },
    "recipients:" [{
        "header": { "enc": "A128KW", "kid": "1" },
        "ciphertext": "..."
    }],
    "initialization_vector": "...",
    "ciphertext": "...",
    "authentication_tag": "..."
}
```

Figure 1

This example shows how the change from the current syntax can be very minor. The only change here is that the "enc" parameter is used for both the outer JWE and the inner JWE; the fact that direct encryption is being done in the inner JWE is apparent from the lack of a "recipients" parameter in the inner JWE. The "encrypted_key" parameter is renamed "ciphertext" for compatibility with JWE.

This structure also provides a neat division of JWEs into "direct"

and "indirect" modes. If there is no "recipients" parameter, then the JWE is "direct"; no wrapped keys are provided. If there is a "recipients" parameter, then the JWE is "indirect"; the recipient knows that it needs to unwrap the CEK before decrypting content.

This proposal also allows the use of AEAD algorithms for key wrapping [[NIST-SP800-38F]], without the need for special code for key encryption vs. content encryption. For example, the following JWE object uses AES-GCM for key encryption:

```
{
    "header": { "enc": "A128CBC-HS256" },
    "recipients": [{
        "header": { "alg":"A128GCM", "kid":"1" }
        "initialization_vector": "..."
        "ciphertext": "...",
        "authentication_tag": "..."
    }],
    "initialization_vector": "...",
    "ciphertext": "...",
    "authentication_tag": "..."
}
```

Figure 2

In the remainder of this document, we provide text for the proposed changes, in cases where text is not immediately clear:

- Part 1: Wrapped Key format

- Part 3: "kdf" (Key Derivation Function) Header Parameter

- Part 4: Changes to the compact serialization

We also provide a set of detailed examples that show the proposed changes affect the representation of JWE objects in several important cases.

2. Wrapped Key Format [1]

[[To be included in JWK]]

A wrapped key is a JWE object with a key as its payload. If the key is a symmetric key with no attributes, it is express directly as an octet string; otherwise it is encoded in JSON as a JWK. The processing of wrapped keys is identical to normal JWE processing, with two additional rules:

1. If a wrapped key has no "cty" parameter, then its MUST be interpreted as the octets of a symmetric key. That is, the payload is equivalent to a JWK with "kty" equal to "oct", and "k" equal to the payload octets (base64url encoded).

2. If a wrapped key contains something other than a bare symmetric key, then a "cty" attribute MUST be present to specify the type of the wrapped key. For a JWK, the relevant "cty" value is "application/jwk+json".

3. "kdf" (Key Derivation Function) Parameter [3]

[[To be included in JWE]]

The "kdf" parameter indicates an algorithm that the recipient must apply to the the private key indicated in a JWE before it is used as an encryption key. For example, the value "ECDH-ES" for the "kdf" parameter indicates that the recipient must use ECDH key agreement and the Concat KDF to derive the encryption key. Likewise, a "PBKDF2" value for this parameter would indicate that the encryption key must be derived from the identified key by way of PBKDF2.

The "kdf" parameter MUST NOT be used in indirect mode. Use of this header parameter is OPTIONAL. This parameter MUST be understood by implementations.

4. Compact form [4]

[[To be included in JWE]]

The JWE compact form is a compact, URL-safe representation of a JWE JSON object. In order to be compatible with the compact form, all top level header parametrs in the JSON-formatted JWE object MUST be protected. Only the "protect" field is encoded; the "header" field is ignored. All parameters in the JWE objects in the "recipients" object MUST be unprotected. This section describes the transformation between the JSON and compact forms of a JWE object.

The translation from JSON form to compact form is as follows:

1. Set the Encoded JWE Header to the value of the "protect" field

2. If the "recipients" field is present, compute a Encoded JWE Key Header and Encoded JWE Encrypted Key for each recipient in the "recipients" field

   * If the "initialization_vector" or "authentication_tag" fields are present in the JWE, add them to the "header" field

   * Base64url encode the octets of the UTF-8 representation of the value of the "protect" field to create the Encoded JWE Key Header

   * Set the Encoded JWE Encrypted Key to the value of the "ciphertext" field

3. Set the Encoded JWE Key Headers value to the concatenation of the Encoded JWE Key Header values, separated by the tilde ('~') character, in the same order as the "recipients" array

4. Set the Encoded JWE Encrypted Keys value to the concatenation of the Encoded JWE Encrypted Key values, separated by the tilde ('~') character, in the same order as the "recipients" array

5. Set the Encoded JWE Initialization Vector to be the value of the "initialization_vector" field

6. Set the Encoded JWE Ciphertext to be the value of the "ciphertext" field

7. Set the Encoded JWE AUthentication Tag to be the value of the "authentication_tag" field

8. The compact form of the JWE is the concatenation of the following values, separated by five period ('.') characters.

   * Encoded JWE Header

   * Encoded JWE Key Headers

   * Encoded JWE Encrypted Keys

   * Encoded JWE Initialization Vector

   * Encoded JWE Ciphertext

   * Encoded JWE Authentication Tag

The translation from compact to JSON form is the reverse of this process. First the compact object is split on the period ('.') character to obtain the six parts listed above. If the Encoded JWE Key Headers and Encoded JWE Encrypted Keys segments are present, then they are split on the tilde ('~') character and used to reconstruct the "recipients" array. (If these two segments have different numbers of entries, then the implementation MUST reject the object as having invalid syntax.)

4.1. Sample JavaScript Conversion Routines

The following JavaScript function code samples illustrate the process of converting between the compact and JSON encodings (actually, between compact form and a JavaScript object equivalent to the JSON form). We make the following assumptions:

- PERIOD = '.'

- TILDE = '~'

- base64url_decode() and base64url_encode() perform their eponymous functions

```
function JWE_js2compact(jwe) {
  // Encode per-recipient information, if present
  var keyHeaders = [];
  var encryptedKeys = [];
  if (jwe.recipients) {
    for (var i=0; i < jwe.recipients.length; ++i) {
      if (jwe.recipients[i].initialization_vector) {
        jwe.recipients[i].header.initialization_vector
          = jwe.recipients[i].initialization_vector;
      }
      if (jwe.recipients[i].authentication_tag) {
        jwe.recipients[i].header.authentication_tag
          = jwe.recipients[i].authentication_tag;
      }

      keyHeaders.push( base64url_encode(
        JSON.stringify( jwe.recipients[i].header ) ) );
      encryptedKeys.push( jwe.recipients[i].ciphertext );

    }
  }

  return  jwe.protect
  + "." + keyHeaders.join(TILDE)
  + "." + encryptedKeys.join(TILDE)
  + "." + (jwe.initialization_vector || "")
  + "." + (jwe.ciphertext || "")
  + "." + (jwe.authentication_tag || "");
}
```

Figure 3

```
function JWE_compact2js(compact) {
  var parts = compact.split(PERIOD);
  if (parts.length != 6) {
    throw "Syntax error: compact form must have 6 components";
  }

  var jwe = {
    "protect": parts[0],
    "initialization_vector": parts[3],
    "ciphertext": parts[4],
    "authentication_tag": parts[5]
  };

  // Reconstruct per-recipient information, if present
  if (parts[1].length > 0 && parts[2].length > 0) {
    keyHeaders = parts[1].split(TILDE);
    encryptedKeys = parts[2].split(TILDE);
    if (keyHeaders.length != encryptedKeys.length) {
      throw "Syntax error: recipient arrays different lengths";
    }

    jwe.recipients = [];
    for (var i=0; i < keyHeaders.length; ++i) {
      var recipientJWE = {};
      var header = JSON.parse( base64url_decode(keyHeaders[i]) );
      if ("initialization_vector" in header) {
        recipientJWE.initialization_vector =
          header.initialization_vector;
        delete header["initialization_vector"];
      }
      if ("authentication_tag" in header) {
        recipientJWE.authentication_tag =
          header.authentication_tag;
        delete header["authentication_tag"];
      }
      recipientJWE.header = header;
      recipientJWE.ciphertext = encryptedKeys[i];
      jwe.recipients.push(recipientJWE);
    }
  }

  return jwe;
}
```

Figure 4

5. Detailed Examples

In this section, we present detailed examples for several important use cases. Note that binary values in this section are not generated using the indicated cryptographic algorithms; rather, they are randomly generated strings of appropriate length. Line breaks and white space are for readability only.

In the provided compact forms, we have assumed that all top-level header parameters are moved to the "protect" field from the "header" field.

5.1. RSA Direct

```
{
    "header": {
        "enc": "RSA-OAEP",
        "kid": "1"
    },
    "ciphertext": "badbLWEFcivAgcGXeKpm3YuCmldtqy
        t8Z6BkdaSR6PufFlzhTXINRthq9jwSVLWY1iGnMj8
        afGZ3AkF-mhl1f90Gt2ER-PON3eKL8pt19HkZnGUr
        koWeZ5b6mXSvOIg95zfnj4wgkY6CD-GahCMIjSBm8
        BE6HgL0liLrAy1A0uw"
}
```

Figure 5

The recipient can recognize that direct encryption is being done by the lack of a "recipient" parameter. The recipient then uses the private key corresponding to the "kid" parameter to decrypt the ciphertext using RSA-OAEP.

The compact form of this JWE is 214 octets long

```
 eyJlbmMiOiJSU0EtT0FFUCIsImtpZCI6IjEifQ
.
.
.
.badbLWEFcivAgcGXeKpm3YuCmldtqyt8Z6BkdaSR6PufFlzh
 TXINRthq9jwSVLWY1iGnMj8afGZ3AkF-mhl1f90Gt2ER-PON
 3eKL8pt19HkZnGUrkoWeZ5b6mXSvOIg95zfnj4wgkY6CD-Ga
 hCMIjSBm8BE6HgL0liLrAy1A0uw
.
```

Figure 6

5.2. Direct Encryption with a Symmetric Algorithm

```
{
    "header": {
        "enc": "A128GCM",
        "kid": "1"
    },
    "initialization_vector": "dRdeCTjt3255QKjrpvZVsA",
    "ciphertext": "uRgFnOogcs7MnOnzPpzsojlrMXuZb96D",
    "authentication_tag": "66Z3o-ArcjmCMtbGhDVEaA"
}
```

Figure 7

The recipient can recognize that direct encryption is being done by the lack of a "recipient" parameter. The recipient then uses the symmetric key identified by the "kid" parameter to decrypt the ciphertext using AES-GCM.

The compact form of this JWE is 117 octets long

```
 eyJlbmMiOiJBMTI4R0NNIiwia2lkIjoiMSJ9
.
.
.dRdeCTjt3255QKjrpvZVsA
.uRgFnOogcs7MnOnzPpzsojlrMXuZb96D
.66Z3o-ArcjmCMtbGhDVEaA
```

Figure 8

5.3. Direct Symmetric Encryption with an ECDH-derived key

```
{
    "header": {
        "enc": "A128GCM",
        "kdf": "ECDH",
        "kid": "1",
        "epk": {
            "kty":"EC",
            "crv":"P-256",
            "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
            "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM"
        },
        "apu": "VaBLqlyRDFQdstcg8ZcJ0ea9hUC_LH2K4YCezuVj3qx_rM_
                DurEUHC_PqFUA7HcXmaxtOax4uO6-p6rmNFIvSA"
    },
    "initialization_vector": "dRdeCTjt3255QKjrpvZVsA",
    "ciphertext": "uRgFnOogcs7MnOnzPpzsojlrMXuZb96D",
    "authentication_tag": "66Z3o-ArcjmCMtbGhDVEaA"
}
```

Figure 9

The recipient can recognize that direct encryption is being done by the lack of a "recipient" parameter. The presence of the "kdf" parameter indicates the need to derive a CEK from the shared secret identified by the "kid" value (in this case, an ECDH private key). The recipient combines the identified private key with the ephemeral ECDH public key in the "epk" parameter to obtain a shared secret ZZ. The shared secret ZZ is then combined with the information in the "apu" field and used with the Concat KDF to derive the encryption key. Finally, the recipient uses the derived encryption key to decrypt the content using AES-GCM.

The compact form of this JWE is 439 octets long

```
 eyJlbmMiOiJBMTI4R0NNIiwia2RmIjoiRUNESCIsImtpZCI6
 IjEiLCJlcGsiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIs
 IngiOiJNS0JDVE5JY0tVU0RpaTExeVNzMzUyNmlEWjhBaVRv
 N1R1NktQQXF2N0Q0IiwieSI6IjRFdGw2U1JXMllpTFVyTjV2
 ZnZWSHVocDd4OFB4bHRtV1dsYmJNNElGeU0ifSwiYXB1Ijoi
 VmFCTHFseVJERlFkc3RjZzhaY0owZWE5aFVDX0xIMks0WUNl
 enVWajNxeF9yTV9EdXJFVUhDX1BxRlVBN0hjWG1heHRPYXg0
 dU82LXA2cm1ORkl2U0EifQ
.
.
.dRdeCTjt3255QKjrpvZVsA
.uRgFnOogcs7MnOnzPpzsojlrMXuZb96D
.66Z3o-ArcjmCMtbGhDVEaA
```

Figure 10

5.4. Indirect Encryption with an AES-wrapped Key

```
{
    "header": {
        "enc": "A128GCM",
    },
    "recipients": [{
        "header": {
            "enc": "A128KW",
            "kid": "1"
        },
        "ciphertext": "A8nMMWXKxQCw8UYQsNCH6_mdAKOUDqJI"
    }],
    "initialization_vector": "dRdeCTjt3255QKjrpvZVsA",
    "ciphertext": "uRgFnOogcs7MnOnzPpzsojlrMXuZb96D",
    "authentication_tag": "66Z3o-ArcjmCMtbGhDVEaA"
}
```

Figure 11

The recipient can recognize by the presence of the "recipients" header that indirect encryption is being used, and thus that it needs to decrypt the CEK from one of the wrapped keys in the "recipients" array. Assuming the recipient has access to the identified symmetric key, it decrypts the encrypted key and uses it to decrypt the content using AES-GCM.

The compact form of this JWE is 171 octets long

```
eyJlbmMiOiJBMTI4R0NNIn0
.eyJlbmMiOiJBMTI4S1ciLCJraWQiOiIxIn0
.A8nMMWXKxQCw8UYQsNCH6_mdAKOUDqJI
.dRdeCTjt3255QKjrpvZVsA
.uRgFnOogcs7MnOnzPpzsojlrMXuZb96D
.66Z3o-ArcjmCMtbGhDVEaA
```

Figure 12

5.5. Indirect Encryption with RSA- and ECDH-Wrapped Keys

```
{
    "header": {
        "enc": "A128GCM"
    },
    "recipients": [{
        "header": {
            "enc": "RSA-OAEP",
            "kid": "1"
        },
        "ciphertext": "badbLWEFcivAgcGXeKpm3YuCmldtqy
            t8Z6BkdaSR6PufFlzhTXINRthq9jwSVLWY1iGnMj8
            afGZ3AkF-mhl1f90Gt2ER-PON3eKL8pt19HkZnGUr
            koWeZ5b6mXSvOIg95zfnj4wgkY6CD-GahCMIjSBm8
            BE6HgL0liLrAy1A0uw"
    },{
        "header": {
            "enc": "A128KW",
            "kdf": "ECDH",
            "kid": "1",
            "epk": {
                "kty":"EC",
                "crv":"P-256",
                "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
                "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM"
            },
            "apu": "VaBLqlyRDFQdstcg8ZcJ0ea9hUC_LH2K4YCezuVj3qx_rM_
                    DurEUHC_PqFUA7HcXmaxtOax4uO6-p6rmNFIvSA"
        },
        "ciphertext": "A8nMMWXKxQCw8UYQsNCH6_mdAKOUDqJI"
    }],
    "initialization_vector": "dRdeCTjt3255QKjrpvZVsA",
    "ciphertext": "uRgFnOogcs7MnOnzPpzsojlrMXuZb96D",
    "authentication_tag": "66Z3o-ArcjmCMtbGhDVEaA"
}
```

Figure 13

The recipient can recognize by the presence of the "recipients" header that indirect encryption is being used, and thus that it needs to decrypt the CEK from one of the wrapped keys in the "recipients" array. Assuming the recipient has the private key corresponding to the identified RSA or ECDH keys, it processes the corresponding wrapped key object as a direct JWE to decrypt the CEK. The recipient then uses the CEK to decrypt the content using AES-GCM.

The compact form of this JWE is 703 octets long

```
eyJlbmMiOiJBMTI4R0NNIn0
.eyJlbmMiOiJSU0EtT0FFUCIsImtpZCI6IjEifQ
~eyJlbmMiOiJBMTI4S1ciLCJrZGYiOiJFQ0RIIiwia2lkIjoi
 MSIsImVwayI6eyJrdHkiOiJFQyIsImNydiI6IlAtMjU2Iiwi
 eCI6Ik1LQkNUTkljS1VTRGlpMTF5U3MzNTI2aURaOEFpVG83
 VHU2S1BBcXY3RDQiLCJ5IjoiNEV0bDZTUlcyWWlMVXJONXZm
 dlZIdWhwN3g4UHhsdG1XV2xiYk00SUZ5TSJ9LCJhcHUiOiJW
 YUJMcWx5UkRGUWRzdGNnOFpjSjBlYTloVUNfTEgySzRZQ2V6
 dVZqM3F4X3JNX0R1ckVVSENfUHFGVUE3SGNYbWF4dE9heDR1
 TzYtcDZybU5GSXZTQSJ9
.badbLWEFcivAgcGXeKpm3YuCmldtqyt8Z6BkdaSR6PufFlzh
 TXINRthq9jwSVLWY1iGnMj8afGZ3AkF-mhl1f90Gt2ER-PON
 3eKL8pt19HkZnGUrkoWeZ5b6mXSvOIg95zfnj4wgkY6CD-Ga
 hCMIjSBm8BE6HgL0liLrAy1A0uw
~A8nMMWXKxQCw8UYQsNCH6_mdAKOUDqJI
.dRdeCTjt3255QKjrpvZVsA
.uRgFnOogcs7MnOnzPpzsojlrMXuZb96D
.66Z3o-ArcjmCMtbGhDVEaA
```

Figure 14

6. Security Considerations

The refactoring proposed in this document has several security benefits.

First, by using the JWE format for wrapped keys in JWE, JWE can benefit from general AEAD algorithms for key wrapping, for example, AES-GCM as opposed to AES key wrap. These other AEAD algorithms are more widely available than AES key wrap, and offer better security properties in some situations. This benefit is available to the compact serialization as well as the revised JSON format.

Second, by using the same format for key encryption and content encryption, code for processing objects in the proposed format will only have to have support one way of decrypting objects. This simplification will reduce the chance for bugs in implementations.

Third, the use of consolidated algorithm and key objects allows for simpler validation rules on JOSE objects, again reducing the chance that an improperly-constructed JOSE object will be able to trigger implementation bugs.

There are two areas where this proposal might require increased caution by applications. The first is in the choice of JWE mode, direct vs. indirect. The revised JWE specification should emphasize that indirect-mode JWE is much more secure, because attackers have to break two levels of encryption to access long-lived keys, instead of just one.

The second area of concern is the selection of algorithms. The security considerations in the revised JWE specification should note that applications that use indirect JWEs should choose different algorithms for key encryption and content encryption. That way, an attacker must compromise two different algorithms in order to mis-use a JWE object.

7. IANA Considerations

This memo makes no request of IANA. However, changes to the JOSE specs resulting from this proposal might require adjustments to some IANA registrations.

8. Normative References

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-08 (work in progress), December 2012.

[I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-08 (work in progress), December 2012.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-08 (work in progress), December 2012.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-08 (work in progress), December 2012.

Author's Address

Richard Barnes
BBN

Email: rlb@ipv.sx