JOSE Working Group	M. Jones
Internet-Draft	Microsoft
Intended status: Standards Track	January 16, 2012
Expires: July 19, 2012	

TOC

JSON Web Algorithms (JWA)
draft-ietf-jose-json-web-algorithms-00

Abstract

The JSON Web Algorithms (JWA) specification enumerates cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS) and JSON Web Encryption (JWE) specifications.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119** [RFC2119].

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 July 19, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
2. Terminology
3. Cryptographic Algorithms for JWS
- **3.1. Creating a JWS with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512**
- **3.2. Creating a JWS with RSA SHA-256, RSA SHA-384, or RSA SHA-512**
- **3.3. Creating a JWS with ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512**
- **3.4. Additional Digital Signature/HMAC Algorithms**

4. Cryptographic Algorithms for JWE

4.1. Encrypting a JWE with TBD
4.2. Additional Encryption Algorithms
5. IANA Considerations
6. Security Considerations
7. Open Issues and Things To Be Done (TBD)
8. References
8.1. Normative References
8.2. Informative References
Appendix A. Digital Signature/HMAC Algorithm Identifier Cross-Reference
Appendix B. Encryption Algorithm Identifier Cross-Reference
Appendix C. Acknowledgements
Appendix D. Document History
§ Author's Address

1. Introduction

TOC

The JSON Web Algorithms (JWA) specification enumerates cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS) **[JWS]** and JSON Web Encryption (JWE) **[JWE]** specifications. Enumerating the algorithms and identifiers for them in this specification, rather than in the JWS and JWE specifications, is intended to allow them to remain unchanged in the face of changes in the set of required, recommended, optional, and deprecated algorithms over time. This specification also describes the semantics and operations that are specific to these algorithms and algorithm families.

2. Terminology

TOC

This specification uses the terminology defined by the JSON Web Signature (JWS) **[JWS]** and JSON Web Encryption (JWE) **[JWE]** specifications.

3. Cryptographic Algorithms for JWS

TOC

JWS uses cryptographic algorithms to sign the contents of the JWS Header and the JWS Payload. The use of the following algorithms for producing JWSs is defined in this section.

The table below **Table 1** is the set of `alg` (algorithm) header parameter values defined by this specification for use with JWS, each of which is explained in more detail in the following sections:

Alg Parameter Value	Algorithm
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm
RS256	RSA using SHA-256 hash algorithm
RS384	RSA using SHA-384 hash algorithm
RS512	RSA using SHA-512 hash algorithm
ES256	ECDSA using P-256 curve and SHA-256 hash algorithm
ES384	ECDSA using P-384 curve and SHA-384 hash algorithm
ES512	ECDSA using P-521 curve and SHA-512 hash algorithm

Table 1: JWS Defined "alg" Parameter Values

See **Appendix A** for a table cross-referencing the digital signature and HMAC `alg` (algorithm) values used in this specification with the equivalent identifiers used by other

standards and software packages.

Of these algorithms, only HMAC SHA-256 MUST be implemented by conforming JWS implementations. It is RECOMMENDED that implementations also support the RSA SHA-256 and ECDSA P-256 SHA-256 algorithms. Support for other algorithms and key sizes is OPTIONAL.

3.1. Creating a JWS with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512

TOC

Hash based Message Authentication Codes (HMACs) enable one to use a secret plus a cryptographic hash function to generate a Message Authentication Code (MAC). This can be used to demonstrate that the MAC matches the hashed content, in this case the JWS Secured Input, which therefore demonstrates that whoever generated the MAC was in possession of the secret. The means of exchanging the shared key is outside the scope of this specification.

The algorithm for implementing and validating HMACs is provided in **RFC 2104** [RFC2104]. This section defines the use of the HMAC SHA-256, HMAC SHA-384, and HMAC SHA-512 cryptographic hash functions as defined in **FIPS 180-3** [FIPS.180-3]. The `alg` (algorithm) header parameter values `HS256`, `HS384`, and `HS512` are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded HMAC value using the respective hash function.

The HMAC SHA-256 MAC is generated as follows:

1. Apply the HMAC SHA-256 algorithm to the UTF-8 representation of the JWS Secured Input using the shared key to produce an HMAC value.
2. Base64url encode the resulting HMAC value.

The output is the Encoded JWS Signature for that JWS.

The HMAC SHA-256 MAC for a JWS is validated as follows:

1. Apply the HMAC SHA-256 algorithm to the UTF-8 representation of the JWS Secured Input of the JWS using the shared key.
2. Base64url encode the resulting HMAC value.
3. If the JWS Signature and the base64url encoded HMAC value exactly match, then one has confirmation that the shared key was used to generate the HMAC on the JWS and that the contents of the JWS have not be tampered with.
4. If the validation fails, the JWS MUST be rejected.

Securing content with the HMAC SHA-384 and HMAC SHA-512 algorithms is performed identically to the procedure for HMAC SHA-256 - just with correspondingly longer key and result values.

3.2. Creating a JWS with RSA SHA-256, RSA SHA-384, or RSA SHA-512

TOC

This section defines the use of the RSASSA-PKCS1-v1_5 digital signature algorithm as defined in **RFC 3447** [RFC3447], Section 8.2 (commonly known as PKCS#1), using SHA-256, SHA-384, or SHA-512 as the hash function. The RSASSA-PKCS1-v1_5 algorithm is described in **FIPS 186-3** [FIPS.186-3], Section 5.5, and the SHA-256, SHA-384, and SHA-512 cryptographic hash functions are defined in **FIPS 180-3** [FIPS.180-3]. The `alg` (algorithm) header parameter values `RS256`, `RS384`, and `RS512` are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded RSA digital signature using the respective hash function.

A 2048-bit or longer key length MUST be used with this algorithm.

The RSA SHA-256 digital signature is generated as follows:

1. Generate a digital signature of the UTF-8 representation of the JWS Secured Input using RSASSA-PKCS1-V1_5-SIGN and the SHA-256 hash function with the desired private key. The output will be a byte array.
2. Base64url encode the resulting byte array.

The output is the Encoded JWS Signature for that JWS.

The RSA SHA-256 digital signature for a JWS is validated as follows:

1. Take the Encoded JWS Signature and base64url decode it into a byte array. If decoding fails, the JWS MUST be rejected.
2. Submit the UTF-8 representation of the JWS Secured Input and the public key corresponding to the private key used by the signer to the RSASSA-PKCS1-V1_5-VERIFY algorithm using SHA-256 as the hash function.
3. If the validation fails, the JWS MUST be rejected.

Signing with the RSA SHA-384 and RSA SHA-512 algorithms is performed identically to the procedure for RSA SHA-256 - just with correspondingly longer key and result values.

3.3. Creating a JWS with ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512

TOC

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined by **FIPS 186-3** [FIPS.186-3]. ECDSA provides for the use of Elliptic Curve cryptography, which is able to provide equivalent security to RSA cryptography but using shorter key lengths and with greater processing speed. This means that ECDSA digital signatures will be substantially smaller in terms of length than equivalently strong RSA digital signatures.

This specification defines the use of ECDSA with the P-256 curve and the SHA-256 cryptographic hash function, ECDSA with the P-384 curve and the SHA-384 hash function, and ECDSA with the P-521 curve and the SHA-512 hash function. The P-256, P-384, and P-521 curves are also defined in FIPS 186-3. The `alg` (algorithm) header parameter values `ES256`, `ES384`, and `ES512` are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512 digital signature, respectively.

The ECDSA P-256 SHA-256 digital signature is generated as follows:

1. Generate a digital signature of the UTF-8 representation of the JWS Secured Input using ECDSA P-256 SHA-256 with the desired private key. The output will be the EC point (R, S), where R and S are unsigned integers.
2. Turn R and S into byte arrays in big endian order. Each array will be 32 bytes long.
3. Concatenate the two byte arrays in the order R and then S.
4. Base64url encode the resulting 64 byte array.

The output is the Encoded JWS Signature for the JWS.

The ECDSA P-256 SHA-256 digital signature for a JWS is validated as follows:

1. Take the Encoded JWS Signature and base64url decode it into a byte array. If decoding fails, the JWS MUST be rejected.
2. The output of the base64url decoding MUST be a 64 byte array.
3. Split the 64 byte array into two 32 byte arrays. The first array will be R and the second S. Remember that the byte arrays are in big endian byte order; please check the ECDSA validator in use to see what byte order it requires.
4. Submit the UTF-8 representation of the JWS Secured Input, R, S and the public key (x, y) to the ECDSA P-256 SHA-256 validator.
5. If the validation fails, the JWS MUST be rejected.

The ECDSA validator will then determine if the digital signature is valid, given the inputs. Note that ECDSA digital signature contains a value referred to as K, which is a random number generated for each digital signature instance. This means that two ECDSA digital signatures using exactly the same input parameters will output different signature values because their K values will be different. The consequence of this is that one must validate an ECDSA digital signature by submitting the previously specified inputs to an ECDSA validator.

Signing with the ECDSA P-384 SHA-384 and ECDSA P-521 SHA-512 algorithms is performed identically to the procedure for ECDSA P-256 SHA-256 - just with correspondingly longer key and result values.

3.4. Additional Digital Signature/HMAC Algorithms

TOC

Additional algorithms MAY be used to protect JWSs with corresponding `alg` (algorithm) header parameter values being defined to refer to them. New `alg` header parameter values SHOULD either be defined in the IANA JSON Web Signature Algorithms registry or be a URI that contains a collision resistant namespace. In particular, it is permissible to use the algorithm identifiers defined in **XML DSIG** [RFC3275] and related specifications as `alg` values.

4. Cryptographic Algorithms for JWE

TOC

JWE uses cryptographic algorithms to encrypt the Content Encryption Key (CEK) and the Plaintext. This section specifies a set of specific algorithms for these purposes.

The table below **Table 2** is the set of `alg` (algorithm) header parameter values that are defined by this specification for use with JWE. These algorithms are used to encrypt the CEK, which produces the JWE Encrypted Key.

alg Parameter Value	Encryption Algorithm
RSA1_5	RSA using RSA-PKCS1-1.5 padding, as defined in **RFC 3447** [RFC3447]
RSA-OAEP	RSA using Optimal Asymmetric Encryption Padding (OAEP), as defined in **RFC 3447** [RFC3447]
ECDH-ES	Elliptic Curve Diffie-Hellman Ephemeral Static, as defined in **RFC 6090** [RFC6090], and using the Concat KDF, as defined in **[NIST-800-56A]**, where the Digest Method is SHA-256
A128KW	Advanced Encryption Standard (AES) Key Wrap Algorithm using 128 bit keys, as defined in **RFC 3394** [RFC3394]
A256KW	Advanced Encryption Standard (AES) Key Wrap Algorithm using 256 bit keys, as defined in **RFC 3394** [RFC3394]
A128GCM	Advanced Encryption Standard (AES) using 128 bit keys in Galois/Counter Mode, as defined in **[FIPS-197]** and **[NIST-800-38D]**
A256GCM	Advanced Encryption Standard (AES) using 256 bit keys in Galois/Counter Mode, as defined in **[FIPS-197]** and **[NIST-800-38D]**

Table 2: JWE Defined "alg" Parameter Values

The table below **Table 3** is the set of `enc` (encryption method) header parameter values that are defined by this specification for use with JWE. These algorithms are used to encrypt the Plaintext, which produces the Ciphertext.

enc Parameter Value	Symmetric Encryption Algorithm
A128CBC	Advanced Encryption Standard (AES) using 128 bit keys in Cipher Block Chaining mode, as defined in **[FIPS-197]** and **[NIST-800-38A]**
A256CBC	Advanced Encryption Standard (AES) using 256 bit keys in Cipher Block Chaining mode, as defined in **[FIPS-197]** and **[NIST-800-38A]**
A128GCM	Advanced Encryption Standard (AES) using 128 bit keys in Galois/Counter Mode, as defined in **[FIPS-197]** and **[NIST-800-38D]**
A256GCM	Advanced Encryption Standard (AES) using 256 bit keys in Galois/Counter Mode, as defined in **[FIPS-197]** and **[NIST-800-38D]**

Table 3: JWE Defined "enc" Parameter Values

See **Appendix B** for a table cross-referencing the encryption `alg` (algorithm) and `alg` (encryption method) values used in this specification with the equivalent identifiers used by other standards and software packages.

Of these algorithms, only RSA-PKCS1-1.5 with 2048 bit keys, AES-128-CBC, and AES-256-CBC MUST be implemented by conforming JWE implementations. It is RECOMMENDED that implementations also support ECDH-ES with 256 bit keys, AES-128-GCM, and AES-256-GCM. Support for other algorithms and key sizes is OPTIONAL.

4.1. Encrypting a JWE with TBD

TOC

TBD: Descriptions of the particulars of using each specified encryption algorithm go here.

4.2. Additional Encryption Algorithms

TOC

Additional algorithms MAY be used to protect JWEs with corresponding `alg` (algorithm) and `enc` (encryption method) header parameter values being defined to refer to them. New `alg` and `enc` header parameter values SHOULD either be defined in the IANA JSON Web Encryption Algorithms registry or be a URI that contains a collision resistant namespace. In particular, it is permissible to use the algorithm identifiers defined in **XML Encryption** [W3C.REC-xmlenc-core-20021210], **XML Encryption 1.1** [W3C.CR-xmlenc-core1-20110303], and related specifications as `alg` and `enc` values.

---

5. IANA Considerations

TOC

This specification calls for:

- A new IANA registry entitled "JSON Web Signature Algorithms" for values of the JWS `alg` (algorithm) header parameter is defined in **Section 3.4**. Inclusion in the registry is RFC Required in the **RFC 5226** [RFC5226] sense. The registry will just record the `alg` value and a pointer to the RFC that defines it. This specification defines inclusion of the algorithm values defined in **Table 1**.
- A new IANA registry entitled "JSON Web Encryption Algorithms" for values used with the JWE `alg` (algorithm) and `enc` (encryption method) header parameters is defined in **Section 4.2**. Inclusion in the registry is RFC Required in the **RFC 5226** [RFC5226] sense. The registry will record the `alg` or `enc` value and a pointer to the RFC that defines it. This specification defines inclusion of the algorithm values defined in **Table 2** and **Table 3**.

---

6. Security Considerations

TOC

TBD

---

7. Open Issues and Things To Be Done (TBD)

TOC

The following items remain to be done in this draft:

- Since RFC 3447 Section 8 explicitly calls for people NOT to adopt RSASSA-PKCS1 for new applications and instead requests that people transition to RSASSA-PSS, we probably need some Security Considerations text explaining why RSASSA-PKCS1 is being used (it's what's commonly implemented) and what the potential

consequences are.
- Consider having an algorithm that is a MAC using SHA-256 that provides content integrity but for which there is no associated secret. This would be like the JWT "alg":"none", in that no validation of the authenticity content is performed but a checksum is provided.
- Consider whether to define "alg":"none" here, rather than in the JWT spec.
- Should we define the use of RFC 5649 key wrapping functions, which allow arbitrary key sizes, in addition to the current use of RFC 3394 key wrapping functions, which require that keys be multiples of 64 bits? Is this needed in practice?
- Decide whether to move the JWK algorithm family definitions "EC" and "RSA" here. This would likely result in all the family-specific parameter definitions also moving here ("crv", "x", "y", "mod", "exp"), leaving very little normative text in the JWK spec itself. This seems like it would reduce spec readability and so was not done.
- It would be good to say somewhere, in normative language, that eventually the algorithms and/or key sizes currently specified will no longer be considered sufficiently secure and will be removed. Therefore, implementers MUST be prepared for this eventuality.
- Write the Security Considerations section.

8. References

TOC

8.1. Normative References

TOC

[FIPS-197] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," FIPS PUB 197, November 2001.

[FIPS.180-3] National Institute of Standards and Technology, "**Secure Hash Standard (SHS)**," FIPS PUB 180-3, October 2008.

[FIPS.186-3] National Institute of Standards and Technology, "**Digital Signature Standard (DSS)**," FIPS PUB 186-3, June 2009.

[JWE] **Jones, M.**, **Rescorla, E.**, and **J. Hildebrand**, "**JSON Web Encryption (JWE)**," January 2012.

[JWS] **Jones, M.**, **Bradley, J.**, and **N. Sakimura**, "**JSON Web Signature (JWS)**," January 2012.

[NIST-800-38A] National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation," NIST PUB 800-38A, December 2001.

[NIST-800-38D] National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST PUB 800-38D, December 2001.

[NIST-800-56A] National Institute of Standards and Technology (NIST), "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)," NIST PUB 800-56A, March 2007.

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

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

[RFC3394] Schaad, J. and R. Housley, "**Advanced Encryption Standard (AES) Key Wrap Algorithm**," RFC 3394, September 2002 (**TXT**).

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

[RFC5226] Narten, T. and H. Alvestrand, "**Guidelines for Writing an IANA Considerations Section in RFCs**," BCP 26, RFC 5226, May 2008 (**TXT**).

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "**Fundamental Elliptic Curve Cryptography Algorithms**," RFC 6090, February 2011 (**TXT**).

8.2. Informative References

TOC

[CanvasApp] Facebook, "**Canvas Applications**," 2010.

[I-D.rescorla-jsms] Rescorla, E. and J. Hildebrand, "**JavaScript Message Security Format**," draft-rescorla-jsms-00 (work in progress), March 2011 (**TXT**).

[JCA] Oracle, "**Java Cryptography Architecture**," 2011.

[JSE] Bradley, J. and N. Sakimura (editor), "**JSON Simple Encryption**," September 2010.

[JSS] Bradley, J. and N. Sakimura (editor), "**JSON Simple Sign**," September 2010.

[MagicSignatures] Panzer (editor), J., Laurie, B., and D. Balfanz, "**Magic Signatures**," August 2010.

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "**(Extensible Markup Language) XML-Signature Syntax and Processing**," RFC 3275, March 2002 (**TXT**).

[W3C.CR-xmlenc-core1-20110303] Hirsch, F., Roessler, T., Reagle, J., and D. Eastlake, "**XML Encryption Syntax and Processing Version 1.1**," World Wide Web Consortium CR CR-xmlenc-core1-20110303, March 2011 (**HTML**).

[W3C.REC-xmlenc-core-20021210] Eastlake, D. and J. Reagle, "**XML Encryption Syntax and Processing**," World Wide Web Consortium Recommendation REC-xmlenc-core-20021210, December 2002 (**HTML**).

TOC

Appendix A. Digital Signature/HMAC Algorithm Identifier Cross-Reference

This appendix contains a table cross-referencing the digital signature and HMAC `alg` (algorithm) values used in this specification with the equivalent identifiers used by other standards and software packages. See **XML DSIG** [RFC3275] and **Java Cryptography Architecture** [JCA] for more information about the names defined by those documents.

Algorithm	JWS	XML DSIG	JCA	OID
HMAC using SHA-256 hash algorithm	HS256	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256	HmacSHA256	1.2.840.113549.2.9
HMAC using SHA-384 hash algorithm	HS384	http://www.w3.org/2001/04/xmldsig-more#hmac-sha384	HmacSHA384	1.2.840.113549.2.10
HMAC using SHA-512 hash algorithm	HS512	http://www.w3.org/2001/04/xmldsig-more#hmac-sha512	HmacSHA512	1.2.840.113549.2.11
RSA using SHA-256 hash algorithm	RS256	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256	SHA256withRSA	1.2.840.113549.1.1.11
RSA using SHA-384 hash algorithm	RS384	http://www.w3.org/2001/04/xmldsig-more#rsa-sha384	SHA384withRSA	1.2.840.113549.1.1.12
RSA using SHA-512 hash algorithm	RS512	http://www.w3.org/2001/04/xmldsig-more#rsa-sha512	SHA512withRSA	1.2.840.113549.1.1.13
ECDSA using P-256 curve and SHA-256 hash algorithm	ES256	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256	SHA256withECDSA	1.2.840.10045.4.3.2
ECDSA using P-384 curve and SHA-384 hash algorithm	ES384	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384	SHA384withECDSA	1.2.840.10045.4.3.3
ECDSA using P-521 curve and SHA-512 hash algorithm	ES512	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512	SHA512withECDSA	1.2.840.10045.4.3.4

Table 4: Digital Signature/HMAC Algorithm Identifier Cross-Reference

TOC

Appendix B. Encryption Algorithm Identifier Cross-Reference

This appendix contains a table cross-referencing the `alg` (algorithm) and `enc` (encryption method) values used in this specification with the equivalent identifiers used by other standards and software packages. See **XML Encryption** [W3C.REC-xmlenc-core-20021210], **XML Encryption 1.1** [W3C.CR-xmlenc-core1-20110303], and **Java Cryptography Architecture** [JCA] for more information about the names defined by those documents.

Algorithm	JWE	XML ENC	JCA
RSA using RSA-PKCS1-1.5 padding	RSA1_5	http://www.w3.org/2001/04/xmlenc#rsa-1_5	RSA/ECB/PKCS1Padding
RSA using Optimal Asymmetric Encryption Padding (OAEP)	RSA-OAEP	http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p	RSA/ECB/OAEPWithSHA-1AndMGF1Padding
Elliptic Curve Diffie-Hellman Ephemeral Static	ECDH-ES	http://www.w3.org/2009/xmlenc11#ECDH-ES	TBD
Advanced Encryption Standard (AES) Key Wrap Algorithm **RFC 3394** [RFC3394] using 128 bit keys	A128KW	http://www.w3.org/2001/04/xmlenc#kw-aes128	TBD
Advanced Encryption Standard (AES) Key Wrap Algorithm **RFC 3394** [RFC3394] using 256 bit keys	A256KW	http://www.w3.org/2001/04/xmlenc#kw-aes256	TBD
Advanced Encryption Standard (AES) using 128 bit keys in Cipher Block Chaining mode	A128CBC	http://www.w3.org/2001/04/xmlenc#aes128-cbc	AES/CBC/PKCS5Padding
Advanced Encryption Standard (AES) using 256 bit keys in Cipher Block Chaining mode	A256CBC	http://www.w3.org/2001/04/xmlenc#aes256-cbc	AES/CBC/PKCS5Padding
Advanced Encryption Standard (AES) using 128 bit keys in Galois/Counter Mode	A128GCM	http://www.w3.org/2009/xmlenc11#aes128-gcm	AES/GCM/NoPadding
Advanced Encryption Standard (AES) using 256 bit keys in Galois/Counter Mode	A256GCM	http://www.w3.org/2009/xmlenc11#aes256-gcm	AES/GCM/NoPadding

Table 5: Encryption Algorithm Identifier Cross-Reference

Appendix C. Acknowledgements

TOC

Solutions for signing and encrypting JSON content were previously explored by **Magic Signatures** [MagicSignatures], **JSON Simple Sign** [JSS], **Canvas Applications** [CanvasApp], **JSON Simple Encryption** [JSE], and **JavaScript Message Security Format** [I-D.rescorla-jsms], all of which influenced this draft. Dirk Balfanz, John Bradley, Yaron Y. Goland, John Panzer, Nat Sakimura, and Paul Tarjan all made significant

contributions to the design of this specification and its related specifications.

Appendix D. Document History

TOC

-00

- Created the initial IETF draft based upon draft-jones-json-web-signature-04 and draft-jones-json-web-encryption-02 with no normative changes.
- Changed terminology to no longer call both digital signatures and HMACs "signatures".

Author's Address

TOC

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: http://self-issued.info/