Proposed Documents for JOSE:

JSON Web Signature (JWS)
JSON Web Encryption (JWE)
JSON Web Key (JWK)

Mike Jones

Standards Architect – Microsoft

IETF 82 – November 14, 2011

Motivation

- Clear need for industry-standard JSON-based:
  - Security Token format
  - Signature format
  - Encryption format
  - Public Key format
- Specs written and in use filling these needs:
  - JSON Web Token (JWT)
  - JSON Web Signature (JWS)
  - JSON Web Encryption (JWE)
  - JSON Web Key (JWK)

Design Philosophy

- Make simple things simple
- Make complex things possible

Design Goals

- Easy to use in all modern web development environments
- Compact, URL-safe representation

Background (1)

- In October 2010, there were numerous proposed JSON-based token and crypto formats:
  - JSON Simple Sign and JSON Simple Encrypt
  - Canvas Applications Signatures
  - JSON Tokens
- Clear that agreement would better serve all
- Mike Jones surveyed features, design decisions
  - Proposed consensus feature set
  - based on discussions including Google, Facebook, AOL, NRI, Microsoft, and independent contributors
- Extensive review, discussions at IIW, Nov 2010
- Consensus JWT draft published December 2010

Background (2)

- JavaScript Message Security Format (JSMS) published in March 2011 by Eric Rescorla & Joe Hildebrand
  - JSON-based signing and encryption format
- JWS published in March 2011 (split off from JWT)
- OAuth JWT Bearer Token Profile published March 2011
- At IETF 80 (March 2011), JSMS and JWS authors agreed to work together on unified specs
- JWK published in April 2011
- JWE published in September 2011
  - Encryption features from JSMS using JWS-based syntax

Background (3)

- JWT, JWS, JWE, JWK updated October 2011
  - Incorporating feedback from WOES/JOSE members
- JWT, etc. specs already in use
  - Google, Microsoft, others
  - Deployments planned by numerous parties
- OpenID Connect uses JWT, JWS, JWE, JWK
  - At least 7 independent implementations

JSON Web Signature (JWS)

- http://tools.ietf.org/html/draft-jones-json-web-signature
- Sign arbitrary content using compact JSON-based representation
  - Includes both true digital signatures and HMACs
- Representation contains three parts:
  - Header
  - Payload
  - Signature
- Parts base64url encoded and concatenated, separated by period ('.') characters
  - URL-safe representation

JWS Header Example

- JWS Header:

  -
    ```
    {"typ":"JWT",
     "alg":"HS256"}
    ```
  - Specifies use of HMAC SHA-256 algorithm
  - Also contains optional content type parameter
- Base64url encoded JWS Header:

  -
    ```
    eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
    ```

JWS Payload Example

- JWS Payload (before base64url encoding):

  -
    ```
    {"iss":"joe",
     "exp":1300819380,
     "http://example.com/is_root":true}
    ```
- JWS Payload (after base64url encoding):

  -
    ```
    eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQo
    gImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
    ```

JWS Signing Input

- Signature covers both Header and Payload
- Signing input concatenation of encoded Header and Payload, separated by period
  - Enables direct signing of output representation
- Example signing input:
  -
    ```
    eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3
    MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0d
    HA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
    ```

JWS Signature

- Example base64url encoded HMAC SHA-256 value:
  - `dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk`

JWS Header Parameters

- `"alg"` – Signature Algorithm (REQUIRED)
- `"jku"` – JSON Web Key URL
- `"x5u"` – X.509 Public Key URL
- `"x5t"` – X.509 Certificate Thumbprint
- `"kid"` – Key Identifier
- `"typ"` – Type for signed content

JWS Algorithm Identifiers

- Compact algorithm ("alg") identifiers:
  - "HS256" – HMAC SHA-256
  - "RS256" – RSA SHA-256
  - "ES256" – ECDSA with P-256 curve and SHA-256
- Other hash sizes also defined:
  - 384, 512
- Other algorithms, identifiers MAY be used

JSON Web Encryption (JWE)

- http://tools.ietf.org/html/draft-jones-json-web-encryption
- Encrypt arbitrary content using compact JSON-based representation
- Representation contains three parts:
  - Header
  - Encrypted Key
  - Ciphertext
- Parts base64url encoded and concatenated, separated by period ('.') characters
  - URL-safe representation

JWE Header Example

- JWS Header:

  -
    ```
    {"alg":"RSA1_5",
     "enc":"A256GCM",
     "iv":"__79_Pv6-fg",
     "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"}
    ```
  - RSA-PKCS1_1.5 used to encrypt JWE Encrypted Key
  - AES-256-GCM used to encrypt Plaintext
  - Initialization Vector value specified
  - X.509 Certificate Thumbprint specified
- Header base64url encoded just like JWS

JWE Header Parameters

- `"alg"` – Encryption Algorithm for JWE Encrypted Key (REQUIRED)
- `"enc"` – Encryption Algorithm for Plaintext (REQUIRED)
- `"iv"` – Initialization Vector
- `"epk"` – Ephemeral Public Key
- `"zip"` – Compression algorithm
- `"jku"` – JSON Web Key URL
- `"x5u"` – X.509 Public Key URL
- `"x5t"` – X.509 Certificate Thumbprint
- `"kid"` – Key Identifier
- `"typ"` – Type for encrypted content

JWE Key Encryption Alg Identifiers

- Algorithm ("`alg`") identifiers:
  - "`RSA1_5`" – RSA using RSA-PKCS1-1.5 padding
  - "`RSA-OAEP`" – RSA using Optimal Asymmetric Encryption Padding (OAEP)
  - "`ECDH-ES`" – Elliptic Curve Diffie-Hellman Ephemeral Static
  - "`A128KW`", "`A256KW`" – AES Key Wrap with 128, 256 bit keys
  - "`A128GCM`", "`A256GCM`" – AES Galois/Counter Mode (GCM) with 128, 256 bit keys
- Other algorithms, identifiers MAY be used

JWE Plaintext Encryption Alg Identifiers

- Algorithm ("enc") identifiers:
  - "A128CBC","A256CBC" – AES Cipher Block Chaining (CBC) mode with 128, 256 bit keys
  - "A128GCM","A256GCM" – AES Galois/Counter Mode (GCM) with 128, 256 bit keys
- Other algorithms, identifiers MAY be used

JSON Web Key (JWK)

- http://tools.ietf.org/html/draft-jones-json-web-key
- JSON representation of public keys
- *Representation of private keys out of scope*

JWK Example

```
{"keyvalues":
 [
  {"algorithm":"EC",
   "curve":"P-256",
   "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
   "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
   "use":"encryption",
   "keyid":"1"},

  {"algorithm":"RSA",
   "modulus": "0vx7ag(omitted)Cur-kEgU8awapJzKnqDKgw",
   "exponent":"AQAB",
   "keyid":"2011-04-29"}
 ]
}
```

Refactoring for JOSE

- JOSE charter specifies four deliverables:
  - Signing
  - Encryption
  - Public Key Representation
  - Algorithms Profile
- If accepted by WG, would refactor JWS, JWE, JWK to move algorithms into separate doc

Open Issues

- Do we also want pure JSON representations?
  - Not base64url encoded so not URL-safe
  - Would be usable in HTTP bodies, etc.
- Do we need additional header parameters?
  - Public keys by value (rather than by reference)
  - X.509 certs by value (rather than by reference)
- Do we specify representation combining encryption and integrity operations?
  - Would be more compact than nested operations
  - Would likely result in four-part representation

Related Work

- W3C Web Cryptography Working Group
  - http://www.w3.org/wiki/IdentityCharter
  - Specifying JavaScript APIs for cryptography
  - JOSE specs should underlie this WG's APIs

Next Steps

- Decide whether to accept JWS, JWE, JWK as JOSE working group documents
- Determine coordination strategy with W3C Web Cryptography WG
- Refactor current documents per JOSE charter
- Submit IETF -00 drafts