USABLE CRYPTOGRAPHY WITH JOSÉ

Nathaniel McCallum

Principal Engineer - Red Hat, Inc.

JOSE != José

(joe-zee) (hoe-say)

JOSE: JSON Object Signing and Encryption

(Standardized JSON Crypto and Identity Assertion Formats)

- RFC 7515 - JSON Web Signature (JWS)
- RFC 7516 - JSON Web Encryption (JWE)
- RFC 7517 - JSON Web Key (JWK)
- RFC 7518 - JSON Web Algorithms (JWA)
- RFC 7519 - JSON Web Token (JWT)
- RFC 7520 - Examples of Protecting Content Using JOSE
- RFC 7638 - JSON Web Key (JWK) Thumbprint

JSON WEB KEY (JWK), SYMMETRIC

```
{
  "kty" : "oct",
  "k"   : "GawgguFyGrWKav7AX4VKUg",
  "alg" : "A128KW"
}
```

In JOSE, binary is always represented as URL-Safe Base64.

JSON WEB KEY (JWK), ELLIPTIC CURVE

```
{
  "kty" : "EC",
  "crv" : "P-256",
  "x"   : "MKBCTNIcK...6KPAqv7D4",
  "y"   : "4Et16SRW2...lbbM4IFyM",
  "d"   : "870MB6gfu...3bVdj3eAE",
  "use" : "enc",
  "kid" : "1"
}
```

JSON WEB KEY (JWK), RSA

```
{
  "kty" : "RSA",
  "p"   : "83i-7IvMG...n7O0nVbfs",
  "q"   : "3dfOR9cuY...Icb6yelxk",
  "qi"  : "GyM_p6JrX...6zTKhAVRU",
  "dq"  : "s9lAH9fgg...w494Q_cgk",
  "dp"  : "G4sPXkc6Y...eiKkTiBj0",
  "n"   : "0vx7agoeb...JzKnqDKgw",
  "d"   : "X4cTteJY_...jfcKoAC8Q",
  "e"   : "AQAB",
  "alg" : "RS256",
  "kid" : "2011-04-29"
}
```

JSON WEB KEY SET (JWKSET)

```
{
  "keys": [<JWK>, <JWK>, ...]
}
```

JSON WEB SIGNATURE (JWS), GENERAL

```
{
  "payload" : "eyJpc3MiO...jp0cnVlfQ",

  "signatures" : [
    {
      "protected" : "eyJhbGciOiJSUzI1NiJ9", /* {"alg":"RS256"} */
      "header"    : { "kid" : "2010-12-29" },
      "signature" : "cC4hiUPoj...UPQGe77Rw"
    },
    {
      "protected" : "eyJhbGciOiJFUzI1NiJ9", /* {"alg":"ES256"} */
      "header"    : { "kid" : "e9bc097a-...de882db0d" },
      "signature" : "DtEhU3ljb...3-Kg6NU1Q"
    }
  ]
}
```

JSON WEB SIGNATURE (JWS), FLAT

```
{
  "payload"   : "eyJpc3MiO...jp0cnVlfQ",

  /* Signature data, flattened into parent object. */
  "signature" : "DtEhU3ljb...3-Kg6NU1Q",
  "protected" : "eyJhbGciOiJFUzI1NiJ9", /* {"alg":"ES256"} */
  "header"    : { "kid" : "e9bc097a-...de882db0d" },
}
```

JSON WEB SIGNATURE (JWS), COMPACT

```
<protected>.<payload>.<signature>
```

Since JOSE always uses URL-Safe Base64, the compact format can be used in a URL.

JSON WEB ENCRYPT. (JWE), GENERAL

```
{
  "protected"     :  "eyJlbmMiO...hTMjU2In0", /* {"enc":"A128CBC-HS256"} */
  "unprotected"   : { "jku" : "https://server.example.com/keys.jwks" },

  "iv"            : "AxY8DCtDaGlsbGljb3RoZQ",
  "ciphertext"    : "KDlTtXchh...HXaI9wOGY",
  "tag"           : "Mz-VPPyU4RlcuYv1IwIvzw",

  "recipients"    : [
    {
      "header" : {
        "alg" : "RSA1_5",
        "kid" : "2011-04-29"
      },
      "encrypted_key" : "UGhIOguC7...qXMR4gp_A"
    },
    {
      "header" : {
        "alg" : "A128KW",
        "kid" : "7"
      },
      "encrypted_key" : "6KB707dM9...2I1rT1oOQ"
    }
  ]
}
```

JSON WEB ENCRYPT. (JWE), FLAT

```
{
   "protected"     : "eyJlbm...jU2In0", /* {"enc":"A128CBC-HS256"} */
   "unprotected"   : { "jku" : "https://server.example.com/keys.jwks" },

   "iv"            : "AxY8DCtDaGlsbGljb3RoZQ",
   "ciphertext"    : "KDlTtXchh...HXaI9wOGY",
   "tag"           : "Mz-VPPyU4RlcuYv1IwIvzw",

   /* Recipient data, flattened into parent object. */
   "encrypted_key" : "6KB707dM9...2IlrT1oOQ",
   "header"        : { "alg" : "A128KW", "kid" : "7" }
}
```

JSON WEB ENCRYPT. (JWE), COMPACT

```
<protected>.<encrypted_key>.<iv>.<ciphertext>.<tag>
```

Since JOSE always uses URL-Safe Base64, the compact format can be used in a URL.

JSON WEB TOKEN (JWT)

```
{
  "iss": "Red Hat, Inc.",
  "sub": "npmccallum",
  "aud": "DevConf.us",
  "exp": 1486220100,
  "nbf": 1486218600,
  "iat": 1485728305,
  "jti": "377d1083-ce07-44a2-8125-7bc9ac88436c"
}
```

JWTs are wrapped in JWSs or JWEs, possibly recursively.

José = C-library, CLI implementation of JOSE

- Support for all RFC-defined algorithms
- No native C data types
- No JSON parsing
- Inputs/outputs are json_t (jansson)
- API driven by a "template" approach
- Missing parameters:
  - Inferred from keys
  - Inferred from headers
  - Sensible, secure defaults
- Library design:
  - Core implements JOSE logic
  - Crypto implemented as hooks
  - Crypto library agnostic (currently: OpenSSL)
- CLI tool provides thin layer around C API
- Fully tested against the (many) RFC provided test vectors

https://github.com/latchset/jose

```
Fedora 24+: dnf install jose
```

JWK GENERATION

```
bool jose_jwk_gen(jose_cfg_t *cfg, json_t *jwk);


$ jose jwk gen -i '{"alg": "A128GCM"}' -o oct.jwk
$ jose jwk gen -i '{"alg": "RSA1_5"}' -o rsa.jwk
$ jose jwk gen -i '{"alg": "ES256"}' -o ec.jwk

$ jose jwk gen -i '{"kty": "oct", "bytes": 16}' -o oct.jwk
$ jose jwk gen -i '{"kty": "RSA", "bits": 2048}' -o rsa.jwk
$ jose jwk gen -i '{"key": "EC", "crv": "P-256"}' -o ec.jwk

$ jose jwk gen -i '{"alg":"A128GCM"}' -i '{"alg":"RSA1_5"}'
{ "keys": [...] }
```

With the exception of "bits" and "bytes", all input JSON attributes will be retained.

JWK UTILITIES

```
bool jose_jwk_pub(jose_cfg_t *cfg, json_t *jwk);
bool jose_jwk_prm(jose_cfg_t *cfg, const json_t *jwk, bool req, const char *op);
json_t *jose_jwk_thp(jose_cfg_t *cfg, const json_t *jwk, const char *alg);
```

```
$ cat ec.jwk                        # Show a signing key
{ "x": ..., "y": ..., "d": ..., "key_ops": ["sign", "verify"], ... }

$ jose jwk pub -i ec.jwk            # Remove private key material
{ "x": ..., "y": ..., "key_ops": ["verify"], ... }

$ jose jwk use -i ec.jwk -u sign    # Zero exit status
$ jose jwk use -i ec.jwk -u encrypt # Non-zero exit status

$ jose jwk thp -i ec.jwk            # Calculate the key thumbprint
OSpsacaX48B1DI3ehdeh71KMFFg
```

SIGNING

```
bool jose_jws_sig(jose_cfg_t *cfg, json_t *jws, json_t *sig, const json_t *jwk);
```

```
$ echo hi | jose jws sig -I- -k ec.jwk -k rsa.jwk # General Serialization
{ "payload": "aGkK", "signatures": [
  { "protected": "...", "signature": "..." },
  { "protected": "...", "signature": "..." } ] }

$ echo hi | jose jws sig -I- -k ec.jwk # Flattened Serialization
{ "payload": "aGkK", "protected": "...", "signature": "..." }

$ echo hi | jose jws sig -I- -c -k ec.jwk # Compact Serialization
eyJhbGciOiJFUzI1NiJ9.aGkK.VauBzVLMesMtTtGfwVOHh9WN1dn6iuEkmebFpJJu...

$ echo hi | jose jws sig -I- -k ec.jwk -O /dev/null # Detached Payload
{ "protected": "...", "signature": "..." }
```

Both jws (-i) and sig (-s) are JWS and Signature object templates, respectively.

The sig (-s) parameter is often simply NULL (C) or omitted (CLI).

VERIFICATION

```
bool jose_jws_ver(jose_cfg_t *cfg, const json_t *jws, const json_t *sig,
                  const json_t *jwk, bool all);
```

```
$ jose jws ver -i msg.jws -k ec.jwk  # Zero Exit Status
hi

$ jose jws ver -i msg.jws -k oct.jwk # Non-Zero Exit Status
No signatures validated!

$ echo hi | jose jws ver -I- -i msg.jws -k ec.jwk # Detached Payload
hi
```

The sig (-s) parameter is a Signature object and is often simply NULL (C) or omitted (CLI).

ENCRYPTION

```
bool jose_jwe_enc(jose_cfg_t *cfg, json_t *jwe, json_t *rcp,
                  const json_t *jwk, const void *pt, size_t ptl);


$ jose jwe enc -I pt.txt -k rsa.jwk -k oct.jwk # Generalized Serialization
{ "ciphertext": "...", "recipients": [{...}, {...}], ...}

$ jose jwe enc -I pt.txt -k rsa.jwk # Flattened Serialization
{ "ciphertext": "...", "encrypted_key": "...", ... }

$ jose jwe enc -c -I pt.txt -k rsa.jwk # Compact Serialization
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.ZBRtX0Z0vaCMMg...

$ jose jwe enc -I pt.txt -k rsa.jwk -O ct.bin # Detached Ciphertext
{ "encrypted_key": "...", ... }
```

The jwk (-k) parameter can be a JSON array of JWKs or a JWKSet.

Both jwe (-i) and rcp (-r) are JWE and Recipient templates, respectively.

The rcp (-r) parameter is often simply NULL (C) or omitted (CLI).

DECRYPTION

```
void *jose_jwe_dec(jose_cfg_t *cfg, const json_t *jwe, const json_t *rcp,
                   const json_t *jwk, size_t *ptl);
```

```
# Encrypt with both an RSA key and a password.
$ echo hi | jose jwe enc -I- -o msg.jwe -p -k rsa.jwk
Please enter a password:
Please re-enter the previous password:

$ jose jwe dec -i ct.jwe -k rsa.jwk # Decrypt with RSA
hi

$ jose jwe dec -i ct.jwe -p # Decrypt with password
Please enter password:
hi

$ jose jwe dec -i ct.jwe -I ct.bin -k rsa.jwk # Detached ciphertext
hi
```

For PBES2* (password) algorithms, the jwk parameter is a JSON String.

The rcp parameter is often NULL.

CLI JSON PARSING

```
$ echo '{"foo": {"bar": "baz"}}' | jose fmt -j- -g foo -g bar -u-
baz

$ echo '{"foo": {"bar": "baz"}}' | jose fmt -j- -g foo -j 17 -s bar -UUo-
{"foo":{"bar":17}}

$ jose fmt -j ct.jwe -g protected -yg enc -u-
A128CBC-HS256
```

Exit status is zero on success. Non-zero indicates the argument that failed.

… AND MUCH MORE!

FUTURE JOSÉ FEATURES

- PKCS#11 (SmartCards, Crypto HW)
- Additional crypto library support
- JSON Web Token functions
- X.509 conversion functions
- Additional RFC features

PULL REQUESTS WELCOME!

QUESTIONS?

Project: https://github.com/latchset/jose

Slides: https://raw.githubusercontent.com/latchset/jose/master/slides.pdf