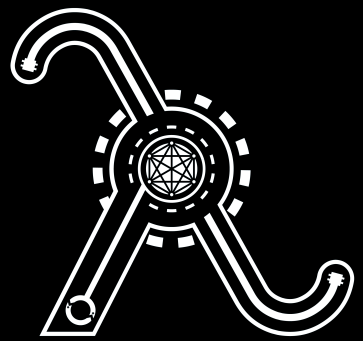
Javascript Object Signing & Encryption

Aaron Zauner
azet@azet.org



lambda.co.at:
Highly-Available, Scalable & Secure Distributed Systems

DevOps/Security and Web Performance Meetup Vienna – 23/03/2015

Javascript Object Signing & Encryption

JWK: JSON Web Key

JWS: JSON Web Signature

JWE: JSON Web Encryption

JWA: JSON Web Algorithms

Examples

JOSE

Definition

JOSE is a framework intended to provide a method to securely transfer claims (such as authorization information) between parties. The JOSE framework provides a collection of specifications to serve this purpose. [...]

JOSE

Couple of new IETF standards being worked on to provide a framework for signatures and/or encryption of JSON data:

- JSON Web Key “JWK”
- JSON Web Signature “JWS”
- JSON Web Encryption “JWE”
- (Algorithms defined in JSON Web Algorithms “JWA”)

..it is..

- End-to-end (E2E)
- **Not** a replacement for TLS!

JWK: JSON Web Key

- Datastructures to represent cryptographic keys
- Used for JWS and JWE
- Keys and Key-Sets

https://tools.ietf.org/html/draft-ietf-jose-json-web-key

JWK: Key

```
{"kty":"EC",
      "crv":"P-256",
      "x":"f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
      "y":"x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0",
      "kid":"Public key used in JWS A.3 example"
}
```

- Key Type: **EC** (Elliptic Curve, Digital Signature Standard)
- Curve: NIST **P-256**
- Curve Points *x* and *y*
- A Key Identifier **kid**

JWK: Key

Other parameters that can be assigned include:

- **use**: Public Key Use (**sig** or **enc**)
- **key_ops**: allowed operations (sign, verify, enc, dec, et cetera)
- **alg**: Intended Algorithm to be used with this Key
- **x5u**: X.509 URL parameter (certificate resource)
- **x5c**: X.509 Certificate Chain
- **x5t** and **x5t#S256**: X.509 SHA-1 and SHA-2 Thumbprints

..might sound familiar to X.509 certificate extensions.

JWK: JWK Set (Key Set)

ECC and RSA Public Keys:

```
{"keys":
  [
    {"kty":"EC",
     "crv":"P-256",
     "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
     "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
     "use":"enc",
     "kid":"1"},

    {"kty":"RSA",
     "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
     "e":"AQAB",
     "alg":"RS256",
     "kid":"2011-04-29"}
  ]
}
```

JWK: JWK Set (Key Set)

Set of Symmetric Encryption Keys (AES key wrap and HMAC):

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

    {"kty":"oct",
     "k":"AyM1SysPpbyDfgZld3umj1qzKObwVMkoqQ-EstJQLr_T-
  1qS0gZH75aKtMN3Yj0iPS4hcgUuTwjAzZr1Z9CAow",
     "kid":"HMAC key used in JWS A.1 example"}
  ]
}
```

JWS: JSON Web Signature

Content signed with:

- Digital Signature .. or;
- Message Authentication Code (MAC)

..thus provides integrity protection.
https://tools.ietf.org/html/draft-ietf-jose-json-web-signature

JWS: JSON Web Signature

- JOSE Header (JWS Protected and Unprotected Headers)
- JWS Payload
- JWS Signature

two serialization formats:

- 'compact': URL-safe (HTTP Auth, URI)
- JWS JSON - JSON Objects (values BASE64URL encoded)

JWS: JSON Web Signature

Compact:

```
BASE64URL(UTF8(JWS Protected Header)) || ,., ||
BASE64URL(JWS Payload) || ,., ||
BASE64URL(JWS Signature)
```

JSON:

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

JWS: JSON Web Signature

JSON Web Token Example

Header

Object:

```
{"typ":"JWT",
  "alg":"HS256"}
```

Encoded:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
```

JWS: JSON Web Signature

JSON Web Token Example

Payload

Object:

```
{"iss":"joe",
     "exp":1300819380,
     "http://example.com/is_root":true}
```

Encoded:

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQo
gImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290IjpOcnVlfQ
```

JWS: JSON Web Signature

JSON Web Token Example

- Header
- Payload
- Signature
- ..seperated by a dot (.)

Encoded:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

JWS: JSON Web Signature

Header Parameters

- **alg**: Algorithm identifier for JWS
- **jku**: JWK Set URL
- **jkw**: JSON Web Key (JWK)
- **kid**: Key ID
- **x5u**: X.509 URL
- **x5c**: X.509 Chain
- **x5t** and **x5t#S256**: X.509 Cert. Thumbprint
- **typ**: "Type" (MIME)
- **cty**: Content Type
- **crit**: "Critical" specifies fields that MUST be protected

JWE: JSON Web Encryption

Format is very similar to JWS, but used for encryption of data

```
BASE64URL(UTF8(JWE Protected Header)) || ,., ||
BASE64URL(JWE Encrypted Key) || ,., ||
BASE64URL(JWE Initialization Vector) || ,., ||
BASE64URL(JWE Ciphertext) || ,., ||
BASE64URL(JWE Authentication Tag)
```

https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption

JWE: JSON Web Encryption

Header Parameters

- **alg**: Algorithm identifier for JWE
- **enc**: Content Encryption Algorithm
- **zip**: Compression algorithm to be used
- **jku**: JWK Set URL
- **jkw**: JSON Web Key (JWK)
- **kid**: Key ID
- **x5u**: X.509 URL
- **x5c**: X.509 Chain
- **x5t** and **x5t#S256**: X.509 Cert. Thumbprint
- **typ**: "Type" (MIME)
- **cty**: Content Type
- **crit**: "Critical" specifies fields that MUST be protected

JWE: JSON Web Encryption

Example (flattened JSON representation):

```
{
"protected":
 "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
"unprotected":
 {"jku":"https://server.example.com/keys.jwks"},
"header":
 {"alg":"A128KW","kid":"7"},
"encrypted_key":
 "6KB707dM9YTIgHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ",
"iv":
 "AxY8DCtDaGlsbGljb3RoZQ",
"ciphertext":
 "KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY",
"tag":
 "Mz-VPPyU4RlcuYv1IwIvzw"
}
```

JWA: JSON Web Algorithms

- JWA Specifies a list of crypto primitives (algorithms) to be used in conjunction with JOSE and their parameters
- Not going into that in this talk. Some of them you've already seen in previous examples, if you want more details on the algorithms that can be used look into the draft
- **DON'T** home-brew your own crypto with these. Use existing, verified, technologies that build on JOSE

https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms

Running Code

Implementations in all popular languages are available on GitHub!

Python

```
import jose

claims = {
    "iss": "http://www.example.com",
    "exp": int(time()) + 3600,
    "sub": 42,
}

jwk = {"k": "password"}
jws = jose.sign(claims, jwk, alg="HS256")
```

http://jose.readthedocs.org/en/latest/

Authentication Protocols

- OAuth
- OpenID / OAuth2.0

..client authentication and authorization can be handeled by JOSE / Web Tokens entirely.
See:

- https://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer
- https://developers.google.com/accounts/docs/OpenIDConnect

Google Wallet

- Google Wallet uses JWT to exchange information between clients (app) and Server

https://developers.google.com/wallet/instant-buy/about-jwts

ACME / Let's Encrypt

- The protocol that Let's Encrypt employs (ACME) uses JOSE for messaging
- i.e. claims for certificates / domains and revocation

https://letsencrypt.github.io/acme-spec/

W3C WebCrypto

- W3C WebCrypto is a JavaScript API for performing basic cryptographic operations in web applications
- W3C WebCrypto employs JOSE (Key Format, Signatures, Algorithms)

Thanks for your patience. Are there any questions?

Twitter:
@a_z_e_t

E-Mail:
azet@azet.org

XMPP:
azet@jabber.ccc.de

GitHub:
https://github.com/azet

GPG Fingerprint:
7CB6 197E 385A 02DC 15D8 E223 E4DB 6492 FDB9 B5D5