

Package ‘jose’

June 14, 2021

Type Package

Title JavaScript Object Signing and Encryption

Version 1.1.0

Description Read and write JSON Web Keys (JWK, rfc7517), generate and verify JSON Web Signatures (JWS, rfc7515) and encode/decode JSON Web Tokens (JWT, rfc7519). These standards provide modern signing and encryption formats that are natively supported by browsers via the JavaScript WebCryptoAPI, and used by services like OAuth 2.0, LetsEncrypt, and Github Apps.

License MIT + file LICENSE

URL <https://datatracker.ietf.org/wg/jose/documents/>
<https://www.w3.org/TR/WebCryptoAPI/#jose>
<https://github.com/r-lib/jose>

BugReports <https://github.com/r-lib/jose/issues>

Depends openssl (>= 1.2.1)

Imports jsonlite

RoxygenNote 7.1.1

VignetteBuilder knitr

Suggests spelling, testthat, knitr, rmarkdown

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Jeroen Ooms [aut, cre] (<<https://orcid.org/0000-0002-4035-0289>>)

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2021-06-13 22:00:02 UTC

R topics documented:

| | |
|----------------------------|---|
| base64url_encode | 2 |
| jwt_claim | 2 |
| jwt_encode_hmac | 3 |
| read_jwk | 4 |

| | |
|--------------|----------|
| Index | 6 |
|--------------|----------|

| | |
|------------------|---------------------------|
| base64url_encode | <i>Base64URL encoding</i> |
|------------------|---------------------------|

Description

The base64url_encode functions are a variant of the standard base64. They are specified in Section 5 of RFC 4648 as a URL-safe alternative. They use different symbols for the 62:nd and 63:rd alphabet character and do not include trailing == padding.

Usage

```
base64url_encode(bin)

base64url_decode(text)
```

Arguments

| | |
|------|----------------------------|
| bin | a binary blob to encode |
| text | a base64url encoded string |

| | |
|-----------|-----------------------|
| jwt_claim | <i>Generate claim</i> |
|-----------|-----------------------|

Description

Helper function to create a named list used as the claim of a JWT payload. See <https://tools.ietf.org/html/rfc7519#section-4.1> for details.

Usage

```
jwt_claim(
  iss = NULL,
  sub = NULL,
  aud = NULL,
  exp = NULL,
  nbf = NULL,
  iat = unclass(Sys.time()),
  jti = NULL,
  ...
)
```

Arguments

| | |
|-----|--|
| iss | (Issuer) Claim, should be rfc7519 'StringOrURI' value |
| sub | (Subject) Claim, should be rfc7519 'StringOrURI' value |
| aud | (Audience) Claim, should contain one or rfc7519 'StringOrURI' values |
| exp | (Expiration Time) Claim, should be rfc7519 'NumericDate' value |
| nbf | (Not Before) Claim, should be rfc7519 'NumericDate' value |
| iat | (Issued At) Claim, should be rfc7519 'NumericDate' value |
| jti | (JWT ID) Claim, optional unique identifier for the JWT |
| ... | additional custom claims to include |

| | |
|-----------------|-----------------------|
| jwt_encode_hmac | <i>JSON Web Token</i> |
|-----------------|-----------------------|

Description

Sign or verify a JSON web token. The `jwt_encode_hmac`, `jwt_encode_rsa`, and `jwt_encode_ec` default to HS256, RS256, and ES256 respectively. See jwt.io or [RFC7519](https://tools.ietf.org/html/rfc7519) for more details.

Usage

```
jwt_encode_hmac(claim = jwt_claim(), secret, size = 256, header = NULL)
```

```
jwt_decode_hmac(jwt, secret)
```

```
jwt_encode_sig(claim = jwt_claim(), key, size = 256, header = NULL)
```

```
jwt_decode_sig(jwt, pubkey)
```

```
jwt_split(jwt)
```

Arguments

| | |
|--------|--|
| claim | a named list with fields to include in the jwt payload |
| secret | string or raw vector with a secret passphrase |
| size | bitsize of sha2 signature, i.e. sha256, sha384 or sha512. Only for HMAC/RSA, not applicable for ECDSA keys. |
| header | named list with additional parameter fields to include in the jwt header as defined in rfc7515 section 9.1.2 |
| jwt | string containing the JSON Web Token (JWT) |
| key | path or object with RSA or EC private key, see openssl::read_key . |
| pubkey | path or object with RSA or EC public key, see openssl::read_pubkey . |

Examples

```
# HMAC signing
mysecret <- "This is super secret"
token <- jwt_claim(name = "jeroen", session = 123456)
sig <- jwt_encode_hmac(token, mysecret)
jwt_decode_hmac(sig, mysecret)

# RSA encoding
mykey <- openssl::rsa_keygen()
pubkey <- as.list(mykey)$pubkey
sig <- jwt_encode_sig(token, mykey)
jwt_decode_sig(sig, pubkey)

# Same with EC
mykey <- openssl::ec_keygen()
pubkey <- as.list(mykey)$pubkey
sig <- jwt_encode_sig(token, mykey)
jwt_decode_sig(sig, pubkey)

# Get elements of the key
mysecret <- "This is super secret"
token <- jwt_claim(name = "jeroen", session = 123456)
jwt <- jwt_encode_hmac(token, mysecret)
jwt_split(jwt)
```

read_jwk

JSON web-keys

Description

Read and write RSA, ECDSA or AES keys as JSON web keys.

Usage

```
read_jwk(file)
```

```
write_jwk(x, path = NULL)
```

Arguments

| | |
|------|---|
| file | path to file with key data or literal json string |
| x | an RSA or EC key or pubkey file |
| path | file path to write output |

Examples

```
# generate an ecdsa key
library(openssl)
key <- ec_keygen("P-521")
write_jwk(key)
write_jwk(as.list(key)$pubkey)

# Same for RSA
key <- rsa_keygen()
write_jwk(key)
write_jwk(as.list(key)$pubkey)
```

Index

`base64url_decode (base64url_encode)`, 2
`base64url_encode`, 2

`jose (jwt_encode_hmac)`, 3
`jwk (read_jwk)`, 4
`jwk_read (read_jwk)`, 4
`jwk_write (read_jwk)`, 4
`jwt (jwt_encode_hmac)`, 3
`jwt_claim`, 2
`jwt_decode_hmac (jwt_encode_hmac)`, 3
`jwt_decode_sig (jwt_encode_hmac)`, 3
`jwt_encode_hmac`, 3
`jwt_encode_sig (jwt_encode_hmac)`, 3
`jwt_split (jwt_encode_hmac)`, 3

`openssl::read_key`, 3
`openssl::read_pubkey`, 3

`read_jwk`, 4

`write_jwk (read_jwk)`, 4