jsonwebtoken DT

9.0.0 • Public • Published 3 months ago

Readme

Code Beta

4 Dependencies

23,141 Dependents

79 Versions

jsonwebtoken

Build	Dependency
Build Status	Dependency Status

An implementation of **JSON Web Tokens**.

This was developed against `draft-ietf-oauth-json-web-token-08`. It makes use of **node-jws**

Install

```
$ npm install jsonwebtoken
```

Migration notes

- From v8 to v9
- From v7 to v8

Usage

jwt.sign(payload, secretOrPrivateKey, [options, callback])

(Asynchronous) If a callback is supplied, the callback is called with the `err` or the JWT.

(Synchronous) Returns the JsonWebToken as string

`payload` could be an object literal, buffer or string representing valid JSON.

> **Please *note* that** `exp` or any other claim is only set if the payload is an object literal. Buffer or string payloads are not checked for JSON validity.

> If `payload` is not a buffer or a string, it will be coerced into a string using `JSON.stringify`.

`secretOrPrivateKey` is a string (utf-8 encoded), buffer, object, or KeyObject containing either the secret for HMAC algorithms or the PEM encoded private key for RSA and ECDSA. In case of a private key with passphrase an object `{ key, passphrase }` can be used (based on **crypto documentation**), in this case be sure you pass the `algorithm` option. When signing with RSA algorithms the minimum modulus length is 2048 except when the allowInsecureKeySizes option is set to true. Private keys below this size will be rejected with an error.

`options`:

- `algorithm` (default: `HS256`)
- `expiresIn`: expressed in seconds or a string describing a time span vercel/ms.

  > Eg: `60`, `"2 days"`, `"10h"`, `"7d"`. A numeric value is interpreted as a seconds count. If you use a string be sure you provide the time units (days, hours, etc), otherwise milliseconds unit is used by default (`"120"` is equal to `"120ms"`).

- `notBefore`: expressed in seconds or a string describing a time span vercel/ms.

  > Eg: `60`, `"2 days"`, `"10h"`, `"7d"`. A numeric value is interpreted as a seconds count. If you use a string be sure you provide the time units (days, hours, etc), otherwise milliseconds unit is used by default (`"120"` is equal to `"120ms"`).

- `audience`
- `issuer`
- `jwtid`
- `subject`
- `noTimestamp`
- `header`
- `keyid`
- `mutatePayload`: if true, the sign function will modify the payload object directly. This is useful if you need a raw reference to the payload after claims have been applied to it but before it has been encoded into a token.
- `allowInsecureKeySizes`: if true allows private keys with a modulus below 2048 to be used for RSA
- `allowInvalidAsymmetricKeyTypes`: if true, allows asymmetric keys which do not match the specified algorithm. This option is intended only for backwards compatability and should be avoided.

> There are no default values for `expiresIn`, `notBefore`, `audience`, `subject`, `issuer`. These claims can also be provided in the payload directly with `exp`, `nbf`, `aud`, `sub` and `iss` respectively, but you ***can't*** include in both places.

Remember that `exp`, `nbf` and `iat` are **NumericDate**, see related **Token Expiration (exp claim)**

The header can be customized via the `options.header` object.

Generated jwts will include an `iat` (issued at) claim by default unless `noTimestamp` is specified. If `iat` is inserted in the payload, it will be used instead of the real timestamp for calculating other things like `exp` given a timespan in `options.expiresIn`.

Synchronous Sign with default (HMAC SHA256)

```
var jwt = require('jsonwebtoken');
var token = jwt.sign({ foo: 'bar' }, 'shhhhh');
```

Synchronous Sign with RSA SHA256

```
// sign with RSA SHA256
var privateKey = fs.readFileSync('private.key');
var token = jwt.sign({ foo: 'bar' }, privateKey, { algorithm: 'RS256'
```

Sign asynchronously

```
jwt.sign({ foo: 'bar' }, privateKey, { algorithm: 'RS256' }, functior
  console.log(token);
});
```

Backdate a jwt 30 seconds

```
var older_token = jwt.sign({ foo: 'bar', iat: Math.floor(Date.now() /
```

Token Expiration (exp claim)

The standard for JWT defines an `exp` claim for expiration. The expiration is represented as a **NumericDate**:

> A JSON numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds. This is equivalent to the IEEE Std 1003.1, 2013 Edition [POSIX.1] definition "Seconds Since the Epoch", in which each day is accounted for by exactly 86400 seconds, other than that non-integer values can be represented. See RFC 3339 [RFC3339] for details

> regarding date/times in general and UTC in particular.

This means that the `exp` field should contain the number of seconds since the epoch.

Signing a token with 1 hour of expiration:

```
jwt.sign({
  exp: Math.floor(Date.now() / 1000) + (60 * 60),
  data: 'foobar'
}, 'secret');
```

Another way to generate a token like this with this library is:

```
jwt.sign({
  data: 'foobar'
}, 'secret', { expiresIn: 60 * 60 });

//or even better:

jwt.sign({
  data: 'foobar'
}, 'secret', { expiresIn: '1h' });
```

jwt.verify(token, secretOrPublicKey, [options, callback])

(Asynchronous) If a callback is supplied, function acts asynchronously. The callback is called with the decoded payload if the signature is valid and optional expiration, audience, or issuer are valid. If not, it will be called with the error.

(Synchronous) If a callback is not supplied, function acts synchronously. Returns the payload decoded if the signature is valid and optional expiration, audience, or issuer are valid. If not, it will throw the error.

> **Warning:** When the token comes from an untrusted source (e.g. user input or external requests), the returned decoded payload should be treated like any other user input; please make sure to sanitize and only work with properties that are expected

`token` is the JsonWebToken string

`secretOrPublicKey` is a string (utf-8 encoded), buffer, or KeyObject containing either the secret for HMAC algorithms, or the PEM encoded public key for RSA and ECDSA. If `jwt.verify` is called asynchronous, `secretOrPublicKey` can be a function that should fetch the secret or public key. See below for a detailed example

As mentioned in **this comment**, there are other libraries that expect base64 encoded

secrets (random bytes encoded using base64), if that is your case you can pass `Buffer.from(secret, 'base64')`, by doing this the secret will be decoded using base64 and the token verification will use the original random bytes.

`options`

- `algorithms` : List of strings with the names of the allowed algorithms. For instance, `["HS256", "HS384"]`.

  > If not specified a defaults will be used based on the type of key provided
  >
  > - secret - ['HS256', 'HS384', 'HS512']
  > - rsa - ['RS256', 'RS384', 'RS512']
  > - ec - ['ES256', 'ES384', 'ES512']
  > - default - ['RS256', 'RS384', 'RS512']

- `audience` : if you want to check audience (`aud`), provide a value here. The audience can be checked against a string, a regular expression or a list of strings and/or regular expressions.

  > Eg: `"urn:foo"`, `/urn:f[o]{2}/`, `[/urn:f[o]{2}/, "urn:bar"]`

- `complete` : return an object with the decoded `{ payload, header, signature }` instead of only the usual content of the payload.
- `issuer` (optional): string or array of strings of valid values for the `iss` field.
- `jwtid` (optional): if you want to check JWT ID (`jti`), provide a string value here.
- `ignoreExpiration` : if `true` do not validate the expiration of the token.
- `ignoreNotBefore` ...
- `subject` : if you want to check subject (`sub`), provide a value here
- `clockTolerance` : number of seconds to tolerate when checking the `nbf` and `exp` claims, to deal with small clock differences among different servers
- `maxAge` : the maximum allowed age for tokens to still be valid. It is expressed in seconds or a string describing a time span vercel/ms.

  > Eg: `1000`, `"2 days"`, `"10h"`, `"7d"`. A numeric value is interpreted as a seconds count. If you use a string be sure you provide the time units (days, hours, etc), otherwise milliseconds unit is used by default (`"120"` is equal to `"120ms"`).

- `clockTimestamp` : the time in seconds that should be used as the current time for all necessary comparisons.
- `nonce` : if you want to check `nonce` claim, provide a string value here. It is used on Open ID for the ID Tokens. (Open ID implementation notes)
- `allowInvalidAsymmetricKeyTypes` : if true, allows asymmetric keys which do not match the specified algorithm. This option is intended only for backwards compatability and should be avoided.

```
// verify a token symmetric - synchronous
var decoded = jwt.verify(token, 'shhhhh');
console.log(decoded.foo) // bar

// verify a token symmetric
jwt.verify(token, 'shhhhh', function(err, decoded) {
  console.log(decoded.foo) // bar
});

// invalid token - synchronous
try {
  var decoded = jwt.verify(token, 'wrong-secret');
} catch(err) {
  // err
}

// invalid token
jwt.verify(token, 'wrong-secret', function(err, decoded) {
  // err
  // decoded undefined
});

// verify a token asymmetric
var cert = fs.readFileSync('public.pem');  // get public key
jwt.verify(token, cert, function(err, decoded) {
  console.log(decoded.foo) // bar
});

// verify audience
var cert = fs.readFileSync('public.pem');  // get public key
jwt.verify(token, cert, { audience: 'urn:foo' }, function(err, decode
  // if audience mismatch, err == invalid audience
});

// verify issuer
var cert = fs.readFileSync('public.pem');  // get public key
jwt.verify(token, cert, { audience: 'urn:foo', issuer: 'urn:issuer' }
  // if issuer mismatch, err == invalid issuer
});

// verify jwt id
var cert = fs.readFileSync('public.pem');  // get public key
```

```
jwt.verify(token, cert, { audience: 'urn:foo', issuer: 'urn:issuer',
  // if jwt id mismatch, err == invalid jwt id
});

// verify subject
var cert = fs.readFileSync('public.pem');  // get public key
jwt.verify(token, cert, { audience: 'urn:foo', issuer: 'urn:issuer',
  // if subject mismatch, err == invalid subject
});

// alg mismatch
var cert = fs.readFileSync('public.pem'); // get public key
jwt.verify(token, cert, { algorithms: ['RS256'] }, function (err, pay
  // if token alg != RS256,  err == invalid signature
});

// Verify using getKey callback
// Example uses https://github.com/auth0/node-jwks-rsa as a way to fe
var jwksClient = require('jwks-rsa');
var client = jwksClient({
  jwksUri: 'https://sandrino.auth0.com/.well-known/jwks.json'
});
function getKey(header, callback){
  client.getSigningKey(header.kid, function(err, key) {
    var signingKey = key.publicKey || key.rsaPublicKey;
    callback(null, signingKey);
  });
}

jwt.verify(token, getKey, options, function(err, decoded) {
  console.log(decoded.foo) // bar
});
```

▶ Need to peek into a JWT without verifying it? (Click to expand)

Errors & Codes

Possible thrown errors during verification. Error is the first argument of the verification callback.

TokenExpiredError

Thrown error if the token is expired.

Error object:

- name: 'TokenExpiredError'
- message: 'jwt expired'
- expiredAt: [ExpDate]

```
jwt.verify(token, 'shhhhh', function(err, decoded) {
  if (err) {
    /*
      err = {
        name: 'TokenExpiredError',
        message: 'jwt expired',
        expiredAt: 1408621000
      }
    */
  }
});
```

JsonWebTokenError

Error object:

- name: 'JsonWebTokenError'
- message:
  - 'invalid token' - the header or payload could not be parsed
  - 'jwt malformed' - the token does not have three components (delimited by a `.`)
  - 'jwt signature is required'
  - 'invalid signature'
  - 'jwt audience invalid. expected: [OPTIONS AUDIENCE]'
  - 'jwt issuer invalid. expected: [OPTIONS ISSUER]'
  - 'jwt id invalid. expected: [OPTIONS JWT ID]'
  - 'jwt subject invalid. expected: [OPTIONS SUBJECT]'

```
jwt.verify(token, 'shhhhh', function(err, decoded) {
  if (err) {
    /*
      err = {
        name: 'JsonWebTokenError',
        message: 'jwt malformed'
      }
    */
  }
});
```

NotBeforeError

Thrown if current time is before the nbf claim.

Error object:

- name: 'NotBeforeError'
- message: 'jwt not active'
- date: 2018-10-04T16:10:44.000Z

```
jwt.verify(token, 'shhhhh', function(err, decoded) {
  if (err) {
    /*
      err = {
        name: 'NotBeforeError',
        message: 'jwt not active',
        date: 2018-10-04T16:10:44.000Z
      }
    */
  }
});
```

Algorithms supported

Array of supported algorithms. The following algorithms are currently supported.

alg Parameter Value	Digital Signature or MAC Algorithm
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm
RS256	RSASSA-PKCS1-v1_5 using SHA-256 hash algorithm
RS384	RSASSA-PKCS1-v1_5 using SHA-384 hash algorithm
RS512	RSASSA-PKCS1-v1_5 using SHA-512 hash algorithm
PS256	RSASSA-PSS using SHA-256 hash algorithm (only node ^6.12.0 OR >=8.0.0)
PS384	RSASSA-PSS using SHA-384 hash algorithm (only node ^6.12.0 OR >=8.0.0)
PS512	RSASSA-PSS using SHA-512 hash algorithm (only node ^6.12.0 OR >=8.0.0)
ES256	ECDSA using P-256 curve and SHA-256 hash algorithm

alg Parameter Value	Digital Signature or MAC Algorithm
ES384	ECDSA using P-384 curve and SHA-384 hash algorithm
ES512	ECDSA using P-521 curve and SHA-512 hash algorithm
none	No digital signature or MAC value included

Refreshing JWTs

First of all, we recommend you to think carefully if auto-refreshing a JWT will not introduce any vulnerability in your system.

We are not comfortable including this as part of the library, however, you can take a look at **this example** to show how this could be accomplished. Apart from that example there are **an issue** and **a pull request** to get more knowledge about this topic.

TODO

- X.509 certificate chain is not checked

Issue Reporting

If you have found a bug or if you have a feature request, please report them at this repository issues section. Please do not report security vulnerabilities on the public GitHub issue tracker. The **Responsible Disclosure Program** details the procedure for disclosing security issues.

Author

Auth0

License

This project is licensed under the MIT license. See the **LICENSE** file for more info.

Install

```
> npm i jsonwebtoken
```

Keywords

jwt

Repository

github.com/auth0/node-jsonwebtoken

Homepage

github.com/auth0/node-jsonwebtoken#readme

Weekly Downloads

10,980,177

Version
9.0.0

License
MIT

Unpacked Size
43.1 kB

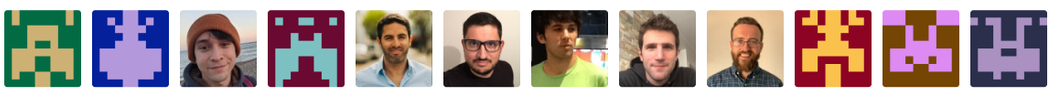
Total Files
15

Issues
94

Pull Requests
36

Last publish
3 months ago

Collaborators



Try on RunKit

Report malware

Support

Help

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy