**[did-jwt-vc](https://github.com/decentralized-identity/did-jwt-vc" \l "did-jwt-vc)**

Create and verify W3C Verifiable Credentials and Presentations in JWT format

[**Installation**](https://github.com/decentralized-identity/did-jwt-vc#installation)

npm install did-jwt-vc

[**Usage**](https://github.com/decentralized-identity/did-jwt-vc#usage)

[**Creating JWTs**](https://github.com/decentralized-identity/did-jwt-vc#creating-jwts)

[**Prerequisites**](https://github.com/decentralized-identity/did-jwt-vc#prerequisites)

Create an Issuer object to sign JWTs using, for example [ethr-did](https://github.com/uport-project/ethr-did)

import { EthrDID } from 'ethr-did'

import { Issuer } from 'did-jwt-vc'

const issuer = new EthrDID({

identifier: '0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198',

privateKey: 'd8b595680851765f38ea5405129244ba3cbad84467d190859f4c8b20c1ff6c75'

}) as Issuer

The Issuer object must contain a did attribute, an alg property that is used in the JWT header and a signer function to generate the signature.

[**Creating a Verifiable Credential**](https://github.com/decentralized-identity/did-jwt-vc#creating-a-verifiable-credential)

Specify a payload matching the CredentialPayload or JwtCredentialPayload interfaces. Create a JWT by signing it with the previously configured issuer using the createVerifiableCredentialJwt function:

import { JwtCredentialPayload, createVerifiableCredentialJwt } from 'did-jwt-vc'

const vcPayload: JwtCredentialPayload = {

sub: 'did:ethr:0x435df3eda57154cf8cf7926079881f2912f54db4',

nbf: 1562950282,

vc: {

'@context': ['https://www.w3.org/2018/credentials/v1'],

type: ['VerifiableCredential'],

credentialSubject: {

degree: {

type: 'BachelorDegree',

name: 'Baccalauréat en musiques numériques'

}

}

}

}

const vcJwt = await createVerifiableCredentialJwt(vcPayload, issuer)

console.log(vcJwt)

// eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQi...0CQmqB14NnN5XxD0d\_glLRs1Myc\_LBJjnuNwE

[**Creating a Verifiable Presentation**](https://github.com/decentralized-identity/did-jwt-vc#creating-a-verifiable-presentation)

Specify a payload matching the PresentationPayload or JwtPresentationPayload interfaces, including the VC JWTs to be presented in the vp.verifiableCredential array. Create a JWT by signing it with the previously configured issuer using the createVerifiablePresentationJwt function:

import { JwtPresentationPayload, createVerifiablePresentationJwt } from 'did-jwt-vc'

const vpPayload: JwtPresentationPayload = {

vp: {

'@context': ['https://www.w3.org/2018/credentials/v1'],

type: ['VerifiablePresentation'],

verifiableCredential: [vcJwt]

}

}

const vpJwt = await createVerifiablePresentationJwt(vpPayload, issuer)

console.log(vpJwt)

// eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1ODI1NDc...JNMUzZ6naacuWNGdZGuU0ZDwmgpUMUqIzMqFFRmge0R8QA

[**Verifying JWTs**](https://github.com/decentralized-identity/did-jwt-vc#verifying-jwts)

[**Prerequisites**](https://github.com/decentralized-identity/did-jwt-vc#prerequisites-1)

Create a Resolver using [did-resolver](https://github.com/decentralized-identity/did-resolver) and register the [ethr-did-resolver](https://github.com/decentralized-identity/ethr-did-resolver). When verifying a JWT signed by a DID, it is necessary to resolve its DID Document to check for keys that can validate the signature.

import { Resolver } from 'did-resolver'

import { getResolver } from 'ethr-did-resolver'

// see also https://github.com/decentralized-identity/ethr-did-resolver#multi-network-configuration

const providerConfig = {

rpcUrl: 'https://mainnet.infura.io/v3/<YOUR infura.io PROJECT ID>',

registry: '0xdca7ef03e98e0dc2b855be647c39abe984fcf21b'

}

const resolver = new Resolver(getResolver(providerConfig))

[**Verifying a Verifiable Credential**](https://github.com/decentralized-identity/did-jwt-vc#verifying-a-verifiable-credential)

Pass in a VC JWT along with the resolver to verify using the verifyCredential function:

import { verifyCredential } from 'did-jwt-vc'

const verifiedVC = await verifyCredential(vcJwt, resolver)

console.log(verifiedVC)

/\*

{

"payload": {

// the original payload of the signed credential

},

"doc": {

// the DID document of the credential issuer (as returned by the `resolver`)

},

"issuer": "did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198", //the credential issuer

"signer": {

//the publicKey entry of the `doc` that has signed the credential

},

"jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjY...Sx3Y2IdWaUpatJQA", // the original credential

//parsed payload aligned to the W3C data model

"verifiableCredential": {

"@context": [Array],

"type": [ "VerifiableCredential", "UniversityDegreeCredential" ],

"issuer": {

"id": "did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198"

},

"issuanceDate": "2019-07-12T16:51:22.000Z",

"credentialSubject": {

"id": "did:ethr:0x435df3eda57154cf8cf7926079881f2912f54db4"

"degree": {

"type": "BachelorDegree",

"name": "Baccalauréat en musiques numériques"

},

},

"proof": {

// proof type for internal use, NOT a registered vc-data-model type

"type": "JwtProof2020",

"jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjY...Sx3Y2IdWaUpatJQA"

}

}

}

\*/

[**Verifying a Verifiable Presentation**](https://github.com/decentralized-identity/did-jwt-vc#verifying-a-verifiable-presentation)

Pass in a VP JWT along with the resolver to verify using the verifyPresentation function:

import { verifyPresentation } from 'did-jwt-vc'

const verifiedVP = await verifyPresentation(vpJwt, resolver)

console.log(verifiedVP)

/\*

{

//original JWT payload

payload: {

iat: 1568045263,

vp: {

'@context': [Array],

type: ['VerifiablePresentation'],

verifiableCredential: [

'eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjY5...lpNm51cqSx3Y2IdWaUpatJQA'

]

},

iss: 'did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198'

},

doc: {

// the DID document of the presentation issuer (as returned by the `resolver`)

},

signer: {

//the publicKey entry of the `doc` that has signed the presentation

},

issuer: 'did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198',

jwt: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjgwNDUyNjMsInZwIjp7...ViNNCvoTQ-swSHwbELW7-EGPAcHLOMiIwE',

// parsed payload aligned to the W3C data model

verifiablePresentation: {

verifiableCredential: [

{

iat: 1566923269,

credentialSubject: {

degree: { type: 'BachelorDegree', name: 'Baccalauréat en musiques numériques' },

id: 'did:ethr:0x435df3eda57154cf8cf7926079881f2912f54db4'

},

issuer: { id: 'did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198' },

type: ['VerifiableCredential', 'UniversityDegreeCredential'],

'@context': [Array],

issuanceDate: '2019-07-12T16:51:22.000Z',

proof: {

type: 'JwtProof2020',

jwt: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjY5...lpNm51cqSx3Y2IdWaUpatJQA'

}

}

],

holder: 'did:ethr:0xf1232f840f3ad7d23fcdaa84d6c66dac24efb198',

type: ['VerifiablePresentation'],

'@context': [Array],

issuanceDate: '2019-09-09T16:07:43.000Z',

proof: {

// proof type for internal use, NOT a registered W3C vc-data-model proof type

type: 'JwtProof2020',

jwt: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NkstUiJ9.eyJpYXQiOjE1NjgwNDUyNjMsInZwI...ViNNCvoTQ-swSHwbELW7-EGPAcHLOMiIwE'

}

}

}

\*/

[**Notes on verification and proof properties**](https://github.com/decentralized-identity/did-jwt-vc#notes-on-verification-and-proof-properties)

The result of the verification methods, when successful, also conveniently contain the decoded and parsed payloads, in a format that closely matches the [W3C data model](https://www.w3.org/TR/vc-data-model/) for verifiable credentials and presentations. This makes it easier to work with both credential encodings in the same system. This parsed payload also shows a proof property that lists the full JWT credential or presentation.

The JwtProof2020 is a synthetic proof type, usable for differentiating credentials by type. It is not a registered W3C VC Data Model algorithm and should not be treated as such.

Also note that the @context fields that appear in this parsed payload are the same as the ones in the incoming JWT. This means that the parsed payload will probably not be suitable for an LD-processor.

Please see #54 for more information.