Desplegar el CredentialRegistry

Para poder hacer uso del Credential Registry como método de registro y verificación de credenciales descentralizado es preciso clonar el repositorio que contiene los contratos inteligentes e instalar algunas herramientas adicionales. A continuación se muestran los pasos a seguir para tener un correcto entorno de despliegue.

1. Para clonar el repositorio se debe ejecutar el siguiente comando:

```
$ git clone https://github.com/lacchain/vc-contracts
```

El resultado del comando debería descargar la carpeta de contratos desde el repositorio, como se muestra en la siguiente imagen:

```
➜  ~ git clone https://github.com/lacchain/vc-contracts
Cloning into 'vc-contracts'...
remote: Enumerating objects: 97, done.
remote: Counting objects: 100% (97/97), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 97 (delta 44), reused 77 (delta 24), pack-reused 0
Receiving objects: 100% (97/97), 1.36 MiB | 450.00 KiB/s, done.
Resolving deltas: 100% (44/44), done.
```

2. Una vez clonado el repositorio, procedemos a desplegar los contratos inteligentes utilizando la herramienta de linea de comandos de openzeppelin. Para ello se debe ejecutar el siguiente comando:

```
$ npm i @openzeppelin/cli
```

Lo anterior instalará el CLI de openzeppelin en la carpeta donde se ejecute el comando:

```
➜  vc-contracts git:(master) ✗ npm i @openzeppelin/cli
npm WARN deprecated truffle-config@1.1.16: WARNING: This package has been renamed to @truffle/config.
npm WARN deprecated truffle-provider@0.1.16: WARNING: This package has been renamed to @truffle/provider.
npm WARN deprecated truffle-error@0.0.5: WARNING: This package has been renamed to @truffle/error.
npm WARN deprecated truffle-interface-adapter@0.2.5: WARNING: This package has been renamed to @truffle/interface-adapter.
```

Nota: Si se desea, es posible instalar la herramienta de forma global agregando el flag -g al comando anterior, lo cual permitirá ejecutar la linea de comandos de openzeppelin desde cualquier otro proyecto.

3. Una vez instalando OpenZeppelin CLI es necesario editar la configuración de la red a utilizar para el despliegue. Dentro del repositorio renombramos el archivo de configuración de ejemplo llamado truffle-config.default por truffle-config.js

```
$ mv truffle-config.default truffle-config.js
```

Y editamos el archivo truffle-config.js para incluir la configuración de la red de LACChain. Considere el siguiente código:

```
const HDWalletProvider = require('@truffle/hdwallet-provider');

module.exports = {
        networks: {
                lacchain: {
                        provider: () => new HDWalletProvider(
                                '862d035b908235f1d0af51713a9e6fc8a1108ac98a77a0be91131d226aa8f4d0',
                                'https://writer.lacchain.net'
                        ),
                        gas: 5000000,
                        gasPrice: 0,
                        network_id: '*'
                }
        },

        mocha: {
                timeout: 100000
        },

        compilers: {
                solc: {
                        version: "0.6.12"
                }
        }
};
```

4. Una vez guardado el archivo truffle-config.js, procedemos a inicializar el proyecto de OpenZeppelin mediante el siguiente comando:

```
$ npx oz init
```

El comando solicitara un nombre al proyecto: el cual usualmente es el mismo nombre del repositorio y la version: la cual es normalmente 1.0.0.

```
? Welcome to the OpenZeppelin SDK! Choose a name for your project vc-contracts
? Initial project version 1.0.0
Project initialized. Write a new contract in the contracts folder and run 'openzeppelin deploy' to deploy it.
```

5. Después de inicializar el proyecto de OpenZeppelin es posible desplegar el contrato de CredentialRegistry con el siguiente comando:

```
$ npx oz deploy
```

El comando anterior invocara el CLI de openzeppelin que se haya instalado sobre el directorio. Si la herramienta se instalo de manera global, es posible omitir el prefijo npx. Al momento de desplegar el contrato OpenZeppelin nos pedirá seleccionar el tipo de despliegue: regular, la red: lacchain, y el contrato a desplegar: CredentialRegistry, como se muestra a continuación:

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy CredentialRegistry
✓ Deployed instance of CredentialRegistry
0xB53676F362b5a684A44E786fE8dc12CAA2cc9e03
```

6. Después de desplegar el CredentialRegistry, ahora vamos a desplegar el Smart Contract ClaimsVerifier, el cual es el contrato encargado de interactuar con las aplicaciones. Volvemos a ejecutar el siguiente comando:

```
$ npx oz deploy
```

Seleccionando el mismo tipo de despliegue y red, ahora seleccionamos el contrato ClaimsVerifier y ponemos la dirección del CredentialRegistry generado en el paso anterior como se muestra en la siguiente imagen:

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy ClaimsVerifier
? _registryAddress: address: 0xB53676F362b5a684A44E786fE8dc12CAA2cc9e03
✓ Deployed instance of ClaimsVerifier
0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C
```

7. Debido a que los contratos inteligentes hacen uso del Sistema de Control de Acceso de OpenZeppelin y el ClaimsVerifier funciona como un Facade del CredentialRegistry, es necesario dar permiso al ClaimsVerifier para interactuar con el CredentialRegistry asignando la dirección del ClaimsVerifier con el role issuer dentro del CredentialRegistry. Para asignar el role al contrato, se puede hacer uso del CLI de OZ mediante el siguiente comando:

```
$ npx oz send-tx
```

Para ejecutar el comando, se utilizan los siguientes parámetros:
- **red**: lacchain
- **instancia**: CredentialRegistry
- **función**: grantRole
- **role**: 0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122
- **account**: la dirección del ClaimsVerifier

```
➜  vc-contracts git:(master) ✗ npx oz send-tx
? Pick a network lacchain
? Pick an instance CredentialRegistry at 0xB53676F362b5a684A44E786fE8dc12CAA2cc9e03
? Select which function grantRole(role: bytes32, account: address)
? role: bytes32: 0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122
? account: address: 0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C
✓ Transaction successful. Transaction hash: 0x2228c1f30050d7d02ae0101d570852e84156d02be785f456d72cadb6b8222337
Events emitted:
 - RoleGranted(0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122, 0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C, 0x7
6393AD6569272385963Bc9A135356456bBe3F83)
```

8. De igual forma, se necesita asignar las cuentas de issuer al contrato ClaimsVerifier mediante el mismo comando:

```
$ npx oz send-tx
```

Para ejecutar el comando, se utilizan los siguientes parámetros:
- **red**: lacchain
- **instancia**: ClaimsVerifier
- **función**: grantRole
- **role**: 0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122
- **account**: la dirección del issuer de la VC

```
➜  vc-contracts git:(master) ✗ npx oz send-tx
? Pick a network lacchain
? Pick an instance ClaimsVerifier at 0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C
? Select which function grantRole(role: bytes32, account: address)
? role: bytes32: 0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122
? account: address: 0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712
✓ Transaction successful. Transaction hash: 0xe0a4d4d808150917f1c91c912f1bfd02c6d774f2d5b75079d2ef0fe308f9c882
Events emitted:
 - RoleGranted(0x114e74f6ea3bd819998f78687bfcb11b140da08e9b7d222fa9c1f1ba1f2aa122, 0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712, 0x76393AD6569272385963Bc9A135356456bBe3F83)
```

9. Opcionalmente, se pueden asignar cuentas contrato ClaimsVerifier como signers mediante el mismo comando:

```
$ npx oz send-tx
```

Para ejecutar el comando, se utilizan los siguientes parámetros:

- **red**: lacchain
- **instancia**: ClaimsVerifier
- **función**: grantRole
- **role**: 0xe2f4eaae4a9751e85a3e4a7b9587827a877f29914755229b07a7b2da98285f70
- **account**: la dirección del issuer de la VC

```
➜  vc-contracts git:(master) ✗ npx oz send-tx
You can improve web3's peformance when running Node.js versions older than 10.5.0 by installing the (deprecated) scrypt package in your project
You can improve web3's peformance when running Node.js versions older than 10.5.0 by installing the (deprecated) scrypt package in your project
? Pick a network lacchain
? Pick an instance ClaimsVerifier at 0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C
? Select which function grantRole(role: bytes32, account: address)
? role: bytes32: 0xe2f4eaae4a9751e85a3e4a7b9587827a877f29914755229b07a7b2da98285f70
? account: address: 0xdca3ebcfbb4e34040f17e6ff21d96d693997183e
✓ Transaction successful. Transaction hash: 0x70e5bc88816dc4a71c21c6762d2f52be202941503b1635b7ca945b3968ee6777
Events emitted:
 - RoleGranted(0xe2f4eaae4a9751e85a3e4a7b9587827a877f29914755229b07a7b2da98285f70, 0xdCA3ebcFbB4E34040F17e6ff21D96d693997183E, 0x76393AD6569272385963Bc9A135356456bBe3F83)
```

Registrar una Credencial Verificable

Una vez desplegados los Smart Contracts y configurados los roles, se puede ahora registrar, firma y verificar una Credencial. Para registrar una Credencial es necesario generar algunas firmas digitales con base en EIP 712 utilizando código en NodeJS, como se muestra a continuación:

```
import ethers from "ethers";
import moment from "moment";
import { CLAIMS_VERIFIER_ABI, getCredentialHash, signCredential } from "@lacchain/vc-contracts-utils";

const CLAIMS_VERIFIER_ADDRESS = "0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C";
const ISSUER_ADDRESS = "0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712";
const ISSUER_PRIVATE_KEY = "60090a13d72f682c03db585bf6c3a296600b5d50598a9ceef3291534dede6bea";

const vc = {
        "@context": "https://www.w3.org/2018/credentials/v1",
        id: "73bde252-cb3e-44ab-94f9-eba6a8a2f28d",
        type: "VerifiableCredential",
        issuer: `did:lac:main:${ISSUER_ADDRESS}`,
        issuanceDate: '2021-12-12T07:17:34.479Z',
        expirationDate: '2022-12-12T07:17:34.479Z',

        credentialSubject: {
                id: `did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e`,
                data: 'anything'
        },
        proof: []
}

Async function register() {
        const claimsVerifier = new ethers.Contract( CLAIMS_VERIFIER_ADDRESS, CLAIMS_VERIFIER_ABI,
                new ethers.Wallet( '0x' + ISSUER_PRIVATE_KEY, new ethers.providers.JsonRpcProvider( "https://
writer.lacchain.net" ) ) );

        const credentialHash = getCredentialHash( vc, ISSUER_ADDRESS, CLAIMS_VERIFIER_ADDRESS );
        const signature = await signCredential( credentialHash, ISSUER_PRIVATE_KEY );

        await claimsVerifier.registerCredential( 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
credentialHash,
                Math.round( moment( vc.issuanceDate ).valueOf() / 1000 ),
                Math.round( moment( vc.expirationDate ).valueOf() / 1000 ),
                signature, { from: ISSUER_ADDRESS } );

        vc.proof.push( {
                id: vc.issuer,
                type: "EcdsaSecp256k1Signature2019",
                proofPurpose: "assertionMethod",
                verificationMethod: `${vc.issuer}#vm-0`,
                domain: CLAIMS_VERIFIER_ADDRESS,
                proofValue: signature
        } );

        console.log( vc );
```

Para ejecutar el código anterior hay que ejecutar los siguientes comando:

```
$ npm install ethers
$ npm install moments
$ npm install @lacchain/vc-contracts-utils
$ node --experimental-modules index.mjs
```

El resultado de la ejecución nos dará la Credencial Verificable con la firma del issuer, como se muestra en la siguiente imagen:

```
{
  '@context': 'https://www.w3.org/2018/credentials/v1',
  id: '73bde252-cb3e-44ab-94f9-eba6a8a2f28d',
  type: 'VerifiableCredential',
  issuer: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712',
  issuanceDate: '2021-12-12T07:17:34.479Z',
  expirationDate: '2022-12-12T07:17:34.479Z',
  credentialSubject: {
    id: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
    data: 'anything'
  },
  proof: [
    {
      id: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712',
      type: 'EcdsaSecp256k1Signature2019',
      proofPurpose: 'assertionMethod',
      verificationMethod: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712#vm-0',
      domain: '0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C',
      proofValue: '0x9027802c40a9a8f0a6b8005c9132c6b9bb38f85fe201e15d8d2df44b7f5fa88a3397ff458eb542b5312d7218f423f76dcf3130fdc8a66
a1300215ede225b40f81b'
    }
  ]
}
```


Agregar una firma adicional a una Credencial Verificable

Después que credencial ha sido registrada es necesario que sea firmada por todos los Signers registrados en ClaimsVerifier Smart Contract. Es posible registrar una firma de un Signer mediante el siguiente código:

```
import ethers from "ethers";
import { CLAIMS_VERIFIER_ABI, getCredentialHash, signCredential } from "@lacchain/vc-contracts-utils";

const CLAIMS_VERIFIER_ADDRESS = "0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C";
const ISSUER_ADDRESS = "0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712";
const SIGNER_ADDRESS = "0xdca3ebcfbb4e34040f17e6ff21d96d693997183e";
const SIGNER_PRIVATE_KEY = "b4a2142e9b0a034ff0ab245ab80f3369480790082007907 87b917a4ce9ae0a98";

const vc = {
        "@context": "https://www.w3.org/2018/credentials/v1",
        id: "73bde252-cb3e-44ab-94f9-eba6a8a2f28d",
        type: "VerifiableCredential",
        issuer: `did:lac:main:${ISSUER_ADDRESS}`,
        issuanceDate: '2021-12-12T07:17:34.479Z',
        expirationDate: '2022-12-12T07:17:34.479Z',
        credentialSubject: {
                id: `did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e`,
                data: 'anything'
        },
        proof: []
}

async function sign() {
        const claimsVerifier = new ethers.Contract( CLAIMS_VERIFIER_ADDRESS, CLAIMS_VERIFIER_ABI,
                new ethers.Wallet( '0x' + SIGNER_PRIVATE_KEY, new ethers.providers.JsonRpcProvider( "https://
writer.lacchain.net" ) ) );

        const credentialHash = getCredentialHash( vc, ISSUER_ADDRESS, CLAIMS_VERIFIER_ADDRESS );
        const signature = await signCredential( credentialHash, SIGNER_PRIVATE_KEY );

        await claimsVerifier.registerSignature( credentialHash, ISSUER_ADDRESS, signature, { from: SIGNER_ADDRESS } );

        vc.proof.push( {
                id: `did:lac:main:${SIGNER_ADDRESS}`,
                type: "EcdsaSecp256k1Signature2019",
                proofPurpose: "assertionMethod",
                verificationMethod: `did:lac:main:${SIGNER_ADDRESS}#vm-0`,
                domain: CLAIMS_VERIFIER_ADDRESS,
                proofValue: signature
        } );

        console.log( vc );
}

sign();
```

El resultado de la ejecución nos dará la Credencial Verificable con la firma del signer en la sección de **poof**, como se muestra en la siguiente imagen:

```
{
  '@context': 'https://www.w3.org/2018/credentials/v1',
  id: '73bde252-cb3e-44ab-94f9-eba6a8a2f28d',
  type: 'VerifiableCredential',
  issuer: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712',
  issuanceDate: '2021-12-12T07:17:34.479Z',
  expirationDate: '2022-12-12T07:17:34.479Z',
  credentialSubject: {
    id: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
    data: 'anything'
  },
  proof: [
    {
      id: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
      type: 'EcdsaSecp256k1Signature2019',
      proofPurpose: 'assertionMethod',
      verificationMethod: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e#vm-0',
      domain: '0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C',
      proofValue: '0x088b6f9d64bb74807d84a502247f4370a3d292482eea5bce411249ebc323c83a4ec9a0f47b143b1a44307541cf0fac0bbd418b9036849
489cb22eb16abd780f41c'
    }
  ]
}
```

Verificar una Credencial Verificable

Una vez registrada y firmada la Credencial, se deben concatenar los proofs del issuer y de los signers en el mismo arreglo, como se muestra a continuación:

```
const vc = {
        '@context': 'https://www.w3.org/2018/credentials/v1',
        id: '73bde252-cb3e-44ab-94f9-eba6a8a2f28d',
        type: 'VerifiableCredential',
        issuer: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712',
        issuanceDate: '2021-12-12T07:17:34.479Z',
        expirationDate: '2022-12-12T07:17:34.479Z',
        credentialSubject: {
                id: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
                data: 'anything'
        },
        proof: [
                {
                        id: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712',
                        type: 'EcdsaSecp256k1Signature2019',
                        proofPurpose: 'assertionMethod',
                        verificationMethod: 'did:lac:main:0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712#vm-0',
                        domain: '0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C',
                        proofValue:
'0x9027802c40a9a8f0a6b8005c9132c6b9bb38f85fe201e15d8d2df44b7f5fa88a3397ff458eb542b5312d7218f423f76dcf3130fdc8a66a1300215
ede225b40f81b'
                },
                {
                        id: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e',
                        type: 'EcdsaSecp256k1Signature2019',
                        proofPurpose: 'assertionMethod',
                        verificationMethod: 'did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e#vm-0',
                        domain: '0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C',
                        proofValue:
'0x088b6f9d64bb74807d84a502247f4370a3d292482eea5bce411249ebc323c83a4ec9a0f47b143b1a44307541cf0fac0bbd418b9036849489cb22e
b16abd780f41c'
                }
        ]
}
```

La función para verificar una VC se muestra en el siguiente código:

```
import ethers from "ethers";
import moment from "moment";
import { CLAIMS_VERIFIER_ABI, getRSV, sha256 } from "@lacchain/vc-contracts-utils";

const CLAIMS_VERIFIER_ADDRESS = "0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C";

async function verify() {
        const claimsVerifier = new ethers.Contract( CLAIMS_VERIFIER_ADDRESS, CLAIMS_VERIFIER_ABI,
                new ethers.providers.JsonRpcProvider( "https://writer.lacchain.net" ) );

        const data = `0x${sha256( JSON.stringify( vc.credentialSubject ) )}`;
        const rsv = getRSV( vc.proof[0].proofValue );
        const result = await claimsVerifier.verifyCredential( [
                vc.issuer.replace( 'did:lac:main:', '' ),
                vc.credentialSubject.id.replace( 'did:lac:main:', '' ),
                data,
                Math.round( moment( vc.issuanceDate ).valueOf() / 1000 ),
                Math.round( moment( vc.expirationDate ).valueOf() / 1000 )
        ], rsv.v, rsv.r, rsv.s );

        const credentialExists = result[0];
        const isNotRevoked = result[1];
        const issuerSignatureValid = result[2];
        const additionalSigners = result[3];
        const isNotExpired = result[4];

        console.log( { credentialExists, isNotRevoked, issuerSignatureValid, additionalSigners, isNotExpired } );
}

verify();
```

La salida de la ejecución del código anterior es la siguiente:

```
{
  credentialExists: true,
  isNotRevoked: true,
  issuerSignatureValid: true,
  additionalSigners: true,
  isNotExpired: true
}
```

En el caso de que se hayan definido signers adicionales, se debe ejecutar el siguiente código para verificar la firma de cada uno de ellos de manera individual:

```
import ethers from "ethers";
import moment from "moment";
import { CLAIMS_VERIFIER_ABI, sha256 } from "@lacchain/vc-contracts-utils";

const CLAIMS_VERIFIER_ADDRESS = "0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C";

async function verifySignature() {
        const claimsVerifier = new ethers.Contract( CLAIMS_VERIFIER_ADDRESS, CLAIMS_VERIFIER_ABI,
                new ethers.providers.JsonRpcProvider( "https://writer.lacchain.net" ) );

        const data = `0x${sha256( JSON.stringify( vc.credentialSubject ) )}`;

        const isValidSignature = await claimsVerifier.verifySigner( [
                vc.issuer.replace( 'did:lac:main:', '' ),
                vc.credentialSubject.id.replace( 'did:lac:main:', '' ),
                data,
                Math.round( moment( vc.issuanceDate ).valueOf() / 1000 ),
                Math.round( moment( vc.expirationDate ).valueOf() / 1000 )
        ], vc.proof[1].proofValue );


        console.log( { isValidSignature } );
}
```

Si la firma es correcta, la salida de la ejecución del código anterior es la siguiente:

```
{ isValidSignature: true }
```

El proceso de debe ejecutar para cada firma adicional.

Revocar una Credencial Verificable

Finalmente como parte del proceso, es posible revocar una VC invocando la función **revokeCredential** del Smart Contract mediante el siguiente código:

```
import ethers from "ethers";
import { CREDENTIAL_REGISTRY_ABI, getCredentialHash } from "@lacchain/vc-contracts-utils";

const CREDENTIAL_REGISTRY_ADDRESS = "0xB53676F362b5a684A44E786fE8dc12CAA2cc9e03";
const CLAIMS_VERIFIER_ADDRESS = "0x1A1a5e43B3a29cD8C0A1631d31CfBA595646074C";
const ISSUER_ADDRESS = "0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712";
const ISSUER_PRIVATE_KEY = "60090a13d72f682c03db585bf6c3a296600b5d50598a9ceef3291534dede6bea";


async function revoke() {
        const credentialRegistry = new ethers.Contract( CREDENTIAL_REGISTRY_ADDRESS, CREDENTIAL_REGISTRY_ABI,
                new ethers.Wallet( '0x' + ISSUER_PRIVATE_KEY, new ethers.providers.JsonRpcProvider( "https://
writer.lacchain.net" ) ) );

        const credentialHash = getCredentialHash( vc, ISSUER_ADDRESS, CLAIMS_VERIFIER_ADDRESS );

        const tx = await credentialRegistry.revokeCredential( credentialHash );

        console.log( { hash: tx.hash } );
}

revoke();
```

El resultado de la ejecución del comando solamente regresa el hash de la transacción.

```
{
  hash: '0xb0ca21f4474bc3e009ef01018da76e61c5eca2b063f36b295efa2aeac922b282'
}
```

Una vez revocada la credencial, si volvemos a ejecutar el proceso de verificación nos dará el siguiente resultado:

```
{
  credentialExists: true,
  isNotRevoked: false,
  issuerSignatureValid: true,
  additionalSigners: true,
  isNotExpired: true
}
```

El resultado indica que la credencial existe, las firmas son validas y no ha expirado, pero si esta revocada (isNotRevoked: false)