Despliegue del DIDRegistry

Para poder hacer uso del DIDRegistry como un método de DID basado en blockchain para el registro y control de claves públicas, es necesario clonar el repositorio que contiene los contratos inteligentes e instalar algunas herramientas adicionales. A continuación se muestran los pasos a seguir para tener el entorno que permita hacer un despliegue correcto.

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

```
$ git clone https://github.com/lacchain/lacchain-did-registry
```

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

```
→  iadb git clone https://github.com/lacchain/lacchain-did-registry
Cloning into 'lacchain-did-registry'...
remote: Enumerating objects: 44, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 44 (delta 14), reused 41 (delta 11), pack-reused 0
Receiving objects: 100% (44/44), 242.37 KiB | 961.00 KiB/s, done.
Resolving deltas: 100% (14/14), done.
→  iadb
```

2. Una vez clonado el repositorio, procedemos a desplegar los contratos inteligentes utilizando la Interfaz de Linea de Comandos (CLI) 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:

```
→  lacchain-did-registry git:(master) ✗ npm i @openzeppelin/cli
(⠧⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂⠂) ⸩ rollbackFailedOptional: verb npm-session ce981b3c0078dac1

> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> @web3-js/scrypt-shim@0.1.0 postinstall /private/tmp/iadb/lacchain-did-registry/node_modules/scrypt-shim
> node ./scripts/postinstall.js


> web3@1.2.2 postinstall /private/tmp/iadb/lacchain-did-registry/node_modules/@openzeppelin/upgrades/node_modules/web3
> node angular-patch.js


> web3@1.2.2 postinstall /private/tmp/iadb/lacchain-did-registry/node_modules/@openzeppelin/cli/node_modules/web3
> node angular-patch.js

npm WARN @apollo/client@3.3.19 requires a peer of react@^16.8.0 || ^17.0.0 but none is installed. You must install peer dependenci
es yourself.
npm WARN @lacchain/did-registry@1.0.0 No repository field.

+ @openzeppelin/cli@2.8.2
added 414 packages from 172 contributors and audited 2394 packages in 1348.561s
found 213 vulnerabilities (15 low, 59 moderate, 136 high, 3 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
→  lacchain-did-registry git:(master) ✗
```

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.

```
➜  lacchain-did-registry git:(master) ✗ npx oz init
? Welcome to the OpenZeppelin SDK! Choose a name for your project @lacchain/did-registry
? Initial project version 1.0.0
? Would you like to contribute anonymous usage data to help us improve the OpenZeppelin CLI? Learn more at https://zpl.in/telemetr
y No
Project initialized. Write a new contract in the contracts folder and run 'openzeppelin deploy' to deploy it.
➜  lacchain-did-registry git:(master) ✗
```

5. Después de inicializar el proyecto de OpenZeppelin es posible desplegar el contrato de DIDRegistry 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: DIDRegistry, como se muestra a continuación:

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy DIDRegistry
? _minKeyRotationTime: uint256: 3600
✓ Deployed instance of DIDRegistry
0xD703559f770406FB05986aD7893A87F9c7eAD559
➜  lacchain-did-registry git:(master) ✗
```

Después de ejecutarse el comando OpenZeppelin CLI nos devolverá la dirección del nuevo contrato DIDRegistry desplegado, en este caso **0xD703559f770406FB05986aD7893A87F9c7eAD559**

Despliegue del DIDRegistryRecoverable

Si se desea hacer uso de las funciones de recuperación de clave para un DID, es necesario desplegar el contrato inteligente llamado DIDRegistryRecoverable. Dicho contrato se encuentra en el mismo repositorio que ha sido clonado. Para desplegarlo, ejecutamos el mismo comando que el paso anterior:

```
$ npx oz deploy
```

Para ejecutar el comando, se utilizan los siguientes parámetros:

- **red**: lacchain
- **contrato**: DIDRegistryRecoverable
- **minKeyRotationTime (uint):** El tiempo mínimo (en segundos) para rotar automáticamente el controlador (heredado del Registro DID).
- **maxAttempts (uint):** el número máximo de intentos fallidos en el período de reinicio.
- **minControllers (uint):** el número mínimo de controlador que debe tener la cuenta para poder utilizar esta función.
- **resetSeconds (uint):** el período de tiempo de reinicio (en segundos). Cuando la cuenta excede los maxAttempts, la cuenta debe esperar para alcanzar el tiempo de resetSeconds antes de volver a llamar a la función para recuperar la cuenta. Cuando se alcance este período de tiempo, las claves probadas con éxito para recuperar la cuenta serán eliminadas, en ese caso es necesario volver a probar a los controladores para recuperar la cuenta.

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy DIDRegistryRecoverable
? _minKeyRotationTime: uint256: 3600
? _maxAttempts: uint256: 3
? _minControllers: uint256: 5
? _resetSeconds: uint256: 86400
✓ Deployed instance of DIDRegistryRecoverable
0x140Eb305b9fA01111022ec56Bc2ED52e0b175B0e
➜  lacchain-did-registry git:(master) ✗
```

Después de ejecutarse el comando OpenZeppelin CLI nos devolverá la dirección del nuevo contrato DIDRegistryRecoverable desplegado, en este caso **0x140Eb305b9fA01111022ec56Bc2ED52e0b175B0e**

Crear un nuevo DID

Para interactuar con los contratos inteligentes de una manera más sencilla es posible utilizar una librería en Javascript(Node JS), la cual se encuentra en el siguiente repositorio: https://github.com/lacchain/lacchain-did-js.

La librería esta desarrollada en la versión 14.4 de Node JS, para instalar esta versión se recomienda utilizar el gestor de versiones de NVM(https://github.com/nvm-sh/nvm), ejecutando el siguiente comando:

```
➜  did nvm use 14.4
Now using node v14.4.0 (npm v6.14.5)
➜  did
```

Después de instalar la versión correspondiente, se puede crear un DID utilizando el siguiente fragmento de código:

```
import { DID } from '@lacchain/did'
const did = new DID( {
   registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
   rpcUrl: 'https://writer.lacchain.net',
   network: 'main'
} );
console.log( did.id );
```

El resultado de la ejecución nos dará el identificador asociado al DID, como se muestra en la siguiente imagen:

```
→  did node --experimental-modules --experimental-json-modules index.js
(node:12660) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
did:lac:main:0x3aeddb3c1f45e6a2a0b8518bcaaa1caf16427dc8
```

El identificador del nuevo DID generado es: **did:lac:main:0x3aeddb3c1f45e6a2a0b8518bcaaa1caf16427dc8**

Agregar un nuevo Método de Verificación

Continuando con la versión 14.4 de Node JS, se puede agregar un nuevo método de verificación para el DID utilizando el siguiente fragmento de código:

```
import { DID } from '@lacchain/did'

const did = new DID( {
   registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
   rpcUrl: 'https://writer.lacchain.net',
   network: 'main'
} );

did.addVerificationMethod( {
   type: 'vm',
   algorithm: 'esecp256k1rm',
   encoding: 'hex',
   publicKey: "0x000000000000000000000000",
   controller: did.id,
   expiration: 31536000 // default: 31536000
} ).then( async() => {
   const document = await did.getDocument();
   console.log( document );
} );
```

El resultado de la ejecución nos dará el documento asociado al DID, el cual incluye el nuevo Método de Verificación como se muestra en la siguiente imagen:

```
{
  id: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1',
  controller: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1',
  verificationMethod: [
    {
      id: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1#vm-0',
      type: 'EcdsaSecp256k1VerificationKey2019',
      controller: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1',
      blockchainAccountId: '0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1'
    },
    {
      id: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1#vm-1',
      type: 'EcdsaSecp256k1RecoveryMethod2020',
      controller: 'did:lac:main:0xbc9b80970f7fe7c1651d684d1a9d476a6c9ee4b1',
      publicKeyHex: '000000000000000000000000'
    }
  ],
  . . .
}
```

Agregar un nuevo Controlador

Cada DID puede tener más de un controlador asociado, con los cuales se pueden utilizar las funciones de Rotación Automatica de Claves y ayudar a recuperar el control de un DID (Key Recovery). Para agregar un nuevo controlador al DID, podemos utilizar el siguiente fragmento de código:

```
import { DID } from '@lacchain/did'

const did = new DID( {
        registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

did.addController( '0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712' ).then( async () => {
        const controllers = await did.getControllers();
        console.log( controllers );
} );
```

El resultado de la ejecución nos mostrará la lista de controladores del DID, el cual incluye la dirección del nuevo controlador (0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712) como se muestra en la siguiente imagen:

```
(node:71812) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
[
  '0x6b1A8169e55F4337DBe2E09122dd0691e367a4E0',
  '0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712'
]
```

Cambiar el Controlador activo

Cada DID solo tiene un controlador activo, el cual tiene la capacidad de modificar el DID Document. Después de agregar un nuevo controlador, es posible cambiar el controlador activo mediante el siguiente fragmento de código:

```
import { DID } from '@lacchain/did'

const did = new DID( {
        registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

did.addController( '0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712' ).then( async () => {
        const controllers = await did.getControllers();
        console.log( controllers );
} );
```

El resultado de la ejecución nos mostrará el controlador activo actual del DID, el cual corresponde con la dirección del nuevo controlador agregado (0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712), como se muestra en la siguiente imagen:

```
(node:72778) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712
```

Habilitar la Rotación Automatica de Claves

Una de las funciones que distinguen al método lac de DID, es la capacidad de rotar de forma automática el controlador activo de cada DID. Cada DID puede especificar un tiempo diferente de rotación automática de su controlador activo siempre y cuando tenga mas de un controlador registrado. La rotación automática se puede habilitar especificando el tiempo de rotación (en segundos) considerando como condición que sea mayor o igual al parámetro minKeyRotationTime que se definió al momento del despliegue del contrato DIDRegistry o DIDRegistryRecoverable. A continuación se muestra el fragmento de código que permite habilitar la rotación automática de claves para un DID:

```
import { DID } from '@lacchain/did'

const sleep = seconds => new Promise( resolve => setTimeout( resolve, seconds * 1e3 ) );

const did = new DID( {
        registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

did.addController( '0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712' ).then( async () => {
        console.log( await did.getController() );
        await did.enableKeyRotation( 10 );
        await sleep( 11 );
        console.log( await did.getController() );
} );
```

El resultado de la ejecución nos mostrará el controlador activo actual del DID y después de 10 segundos el nuevo controlador agregado: 0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712, como se muestra en la siguiente imagen:

```
(node:74027) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
0x5B14f618A386b6100855eF3e581e362F864B8a1C
0x2Da061c6cFA5C23828e9D8dfbe295a22e8779712
```

Recuperar el control de un DID

Es posible hacer uso de la función de recuperación de claves del contrato DIDRegistryRecoverable haciendo uso de la misma librería de javascript, cumpliendo las siguientes condiciones:

1. Agregar al DID el número mínimo de controladores especificado en el Smart Contract al momento de su despliegue (minControllers). En este caso especifico: 5
2. No tener habilitada la función de rotación automática de claves
3. Poseer las claves privadas de al menos N/2 + 1 controladores, donde N es el número de controladores del DID. En este caso deberían probarse 5/2 + 1 controladores = 3

El siguiente fragmento de código muestra cómo es posible recuperar el acceso al DID estableciendo como controlador activo el último controlador del cual probamos tener la clave privada:

```
import { DIDRecoverable } from '@lacchain/did'

const did = new DIDRecoverable( {
        registry: '0x140Eb305b9fA01111022ec56Bc2ED52e0b175B0e',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

const controllers = [
        { address: '0x9a0d6fcbe696b30aadb8b10f71a8d14502b38325', privateKey:
'e4cae20a84c2b06f733928ac8fe7c8fea2bf56340e945441a5c500a70a9f9444' },
        { address: '0x5ac03d827dc1caad73933d375f0f85a77efd8514', privateKey:
'6c782722b2311eed153ca657efe5132271f0ccd4e7a88015927db30eeddd39c8' },
        { address: '0x6a3bf40f418ad5af57e58c1db6e65b7458f6e421', privateKey:
'27ebca688eb924168ba68cd71247a5ecb73aa738f0b595ecdf875e7531e1187d' },
        { address: '0xfce1066dfd086c03fd0b83d81160ad808983bc88', privateKey:
'adbff22951acb7680cbba6c3ff7639f88498e8c4991d2dcd266026b2c27553c1' },
]


async function recover() {
        await did.addController( controllers[0].address );
        await did.addController( controllers[1].address );
        await did.addController( controllers[2].address );
        await did.addController( controllers[3].address );

        const currentController = await did.getController();
        console.log( { currentController } );
        await did.recover( controllers[3].address, controllers[3].privateKey );
        await did.recover( controllers[2].address, controllers[2].privateKey );
        await did.recover( controllers[1].address, controllers[1].privateKey );
        const recoveredController = await did.getController();
        console.log( { recoveredController } );
}

recover();
```

El código agrega primero 4 controladores adicionales que sumados al controlador inicial suman 5, después invoca a la función recover para probar que se tiene la clave privada de 3 controladores adicionales. Finalmente, cuando el **DIDRegistryRecoverable** detecta que se han cumplido las condiciones, establece el controlador activo al último controlador que se probo (en este caso el controllers[1] que corresponde a la dirección: 0x5ac03d827dc1caad73933d375f0f85a77efd8514), como se muestra en la siguiente imagen:

```
(node:75157) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
{ currentController: '0x971c3aDb8D119B734c180D40194DaE64aBC26902' }
{ recoveredController: '0x5aC03D827dC1CaaD73933D375f0F85a77EFD8514' }
```

Revocar un Método de Verificación

El proceso de revocación de un Método de Verificación es muy sencillo y consiste en llamar a la función revokeVerificationMethod del DIDRegistry o DIDRegistryRecoverable, como se muestra en el siguiente fragmento de código:

```
import { DID } from '@lacchain/did'

const did = new DID( {
        registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

const vm = {
        type: 'vm',
        algorithm: 'esecp256k1rm',
        encoding: 'hex',
        publicKey: "0x000000000000000000000000",
        controller: did.id,
        expiration: 31536000
};

did.addVerificationMethod( vm ).then( async() => {
        console.log( 'With VM', ( await did.getDocument() ).verificationMethod );
        await did.revokeVerificationMethod( vm );
        console.log( 'Without VM', ( await did.getDocument() ).verificationMethod );
} );
```

En la siguiente imagen podemos ver cómo desaparece el Método de Verificación del DID Document cuando los revocamos.

```
(node:75493) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
With VM [
  {
    id: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24#vm-0',
    type: 'EcdsaSecp256k1VerificationKey2019',
    controller: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24',
    blockchainAccountId: '0xa0486f9748519ec5bdfda2ed048ff9dea6412d24'
  },
  {
    id: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24#vm-1',
    type: 'EcdsaSecp256k1RecoveryMethod2020',
    controller: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24',
    publicKeyHex: '000000000000000000000000'
  }
]
Without VM [
  {
    id: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24#vm-0',
    type: 'EcdsaSecp256k1VerificationKey2019',
    controller: 'did:lac:main:0xa0486f9748519ec5bdfda2ed048ff9dea6412d24',
    blockchainAccountId: '0xa0486f9748519ec5bdfda2ed048ff9dea6412d24'
  }
]
```

Resolver el Documento de un DID

La forma de resolver el Documento del DID se basa en explorar los eventos emitidos por el Smart Contract (véase ERC 1066). Sin embargo, la librería de javascript simplifica el proceso de resolución simplemente invocando el método getDocument del DIDRegistry, como se muestra a continuación:

```
import { DID } from '@lacchain/did'

const did = new DID( {
        registry: '0xbDa1238272FDA68885564449Cb77A87Fc8205E8ba',
        rpcUrl: 'https://writer.lacchain.net',
        network: 'main'
} );

async function getDocument(){
        const document = await did.getDocument();
        console.log( document.document );
}

getDocument();
```

El resultado que obtiene es la estructura original de un DID Document, apuntando mediante la referencia a los métodos de verificación, como se muestra en la siguiente imagen:

```
(node:75991) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
{
  '@context': 'https://w3id.org/did/v1',
  id: 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1',
  controller: 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1',
  verificationMethod: [
    {
      id: 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0',
      type: 'EcdsaSecp256k1VerificationKey2019',
      controller: 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1',
      blockchainAccountId: '0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1'
    }
  ],
  authentication: [ 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0' ],
  assertionMethod: [ 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0' ],
  keyAgreement: [ 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0' ],
  capabilityInvocation: [ 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0' ],
  capabilityDelegation: [ 'did:lac:main:0xb5d80e484b0d90c5ae3210222270c35ba07cdaf1#vm-0' ]
}
```