# Desplegar un Directorio Público de Claves

Para poder hace uso de los Public Key Directory (PKD) y Trusted List (TL) 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 correcto entorno de despliegue.

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

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

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

```
Cloning into 'lacchain-pkd'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 21 (delta 3), reused 21 (delta 3), pack-reused 0
Receiving objects: 100% (21/21), 599.68 KiB | 2.35 MiB/s, done.
Resolving deltas: 100% (3/3), done.
```

2. Una vez clonado el repositorio, procedemos a desplegar los contratos inteligentes utilizando la Command Line Interface (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:

```
→  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 lacchain-pkd
? 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 PKD 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: PKD, como se muestra a continuación:

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy PKD
✓ Deployed instance of PKD
0x52663a5679b5126FA23c6f48666BFebd8466c936
```

## Desplegar una Lista de Confianza

A diferencia de un PKD, una TL se puede enlazar de manera jerárquica mediante la asignación de un contrato parent el cual puede corresponder a otra TL o un PKD (usualmente el nodo raiz), por lo tanto el despliegue requiere especificar ese parámetro en el constructor del TL:

```
$ npx oz deploy
```

En este caso, se deberán especificar los siguientes parámetros:

- **red**: lacchain
- **contract**: TrustedList
- **parent**: la dirección del nodo padre del TL (en este caso la dirección del PKD)
- **tlType**: El tipo de Trusted List (véase marco EIDAS)

```
? Choose the kind of deployment regular
? Pick a network lacchain
? Pick a contract to deploy TrustedList
? _parent: address: 0x52663a5679b5126FA23c6f48666BFebd8466c936
? _tlType: bytes32: 0x1
✓ Deployed instance of TrustedList
0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
```

## Registrar una clave pública en el PKD

Dentro del contrato de PKD se pueden registrar las claves publicas asociándolas a un DID y apuntar a la dirección del DID invocando a la función register mediante la herramienta de OpenZeppelin CLI:

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

Para ejecutar el comando, se utilizan los siguientes parámetros:

- **red**: lacchain
- **instancia**: PKD
- **función**: register
- **address**: la dirección de la entidad a registrar (puede ser la dirección ZERO de Ethereum)
- **did**: El DID asociado a la entidad (puede ser cualquier método de DID) donde están las claves públicas
- **expires**: El tiempo de expiración de la clave publica en segundos

```
? Pick a network lacchain
? Pick an instance PKD at 0x52663a5679b5126FA23c6f48666BFebd8466c936
? Select which function register(_entity: address, _did: string, _expires: uint256)
? _entity: address: 0xdca3ebcfbb4e34040f17e6ff21d96d693997183e
? _did: string: did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e
? _expires: uint256: 3600
✓ Transaction successful. Transaction hash: 0x6b2c9b3bdae2cc385f419d2e660165a8a3ea8bf08f252bc6bb98441fefbbd759
Events emitted:
 - PublicKeyAdded(0xdCA3ebcFbB4E34040F17e6ff21D96d693997183E, did:lac:main:0xdca3ebcfbb4e34040f17e6ff21d96d693997183e, 3600)
```

## Registrar una entidad en una Lista de Confianza

Dentro del contrato de PKD se pueden registrar las claves publicas asociándolas a un DID y apuntar a la dirección del DID invocando a la función register mediante la herramienta de OpenZeppelin CLI:

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

Para ejecutar el comando, se utilizan los siguientes parámetros:

- **red**: lacchain
- **instancia**: TrustedList
- **función**: register
- **address**: la dirección de la entidad a registrar (Puede ser la dirección de un sub TL)
- **did**: El DID asociado a la entidad (puede ser cualquier método de DID) donde están las claves públicas de la entidad
- **expires**: El tiempo de expiración de la entidad en segundos

```
? Pick a network lacchain
? Pick an instance TrustedList at 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
? Select which function register(_entity: address, _did: string, _expires: uint256)
? _entity: address: 0x70aab07a3a509fe630760a2c8ef34e8d28064278
? _did: string: did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278
? _expires: uint256: 3600
✓ Transaction successful. Transaction hash: 0x18ede6bf52d7319fee5ce303f1076e6895fc48c1e3e662e62f603983db80b56e
Events emitted:
 - EntityAdded(0x70aAB07A3A509Fe630760a2c8Ef34e8D28064278, did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278, 3600)
```

## Verificar una entidad en una Lista de Confianza

Para consultar si una entidad esta registrada dentro de una Trusted List es muy sencillo, simplemente hay que hacer una llamada al método entities(address) del Smart Contract, como se muestra a continuación:

```
$ npx oz call
```

La llamada al contrato se hace con los siguientes parámetros:

- **red**: lacchain
- **instancia**: TrustedList
- **función**: entities(address)
- **address**: la dirección de la entidad a verificar

Si la entidad existe y esta registrada dentro del TrustedList, el resultado es el siguiente:

```
? Pick a network lacchain
? Pick an instance TrustedList at 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
? Select which function entities(address)
? #0: address: 0x70aab07a3a509fe630760a2c8ef34e8d28064278
✓ Method 'entities(address)' returned: (did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278, 3600, 1, did:lac:main:0x70aab07a3
a509fe630760a2c8ef34e8d28064278, 3600, 1)
Result {
  '0': 'did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278',
  '1': '3600',
  '2': '1',
  did: 'did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278',
  expires: '3600',
  status: '1'
}
```

## Verificar la Cadena de Confianza de una entidad

La Cadena de Confianza o Root-of-Trust (RoT) consiste en la verificación de la entidad sobre una TL, pero siguiendo jerárquicamente el path hasta llegar a un nodo raíz (usualmente un PKD) el cual sea de confianza y

garantice la identidad de toda la cadena. Para verificar el RoT es necesario ejecutar el procedimiento anterior de manera recursiva obteniendo el nodo padre mediante la siguiente llamada al contrato de TL:

```
$ npx oz call
```

La llamada al contrato se hace con los siguientes parámetros:

- **red**: lacchain
- **instancia**: TrustedList
- **función**: parent()

La ejecución del comando regresara la dirección del contrato padre: TL o PKD, el cual se utiliza para repetir el mismo proceso de verificación. Sin embargo, en este caso, la dirección corresponde a un PKD.

```
? Pick a network lacchain
? Pick an instance TrustedList at 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
? Select which function parent()
✓ Method 'parent()' returned: 0x52663a5679b5126FA23c6f48666BFebd8466c936
0x52663a5679b5126FA23c6f48666BFebd8466c936
```

Una vez obtenida la dirección del PKD, podemos verificar si la TL esta registrada como una entidad dentro del PKD. Recordemos que la dirección del TL es: 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C. Así que procederemos a buscar esa dirección haciendo una llamada a la función isActive del contrato PKD, como se muestra a continuación.

La llamada al contrato se hace con los siguientes parámetros:

- **red**: lacchain
- **instancia**: TrustedList
- **función**: isActive
- **entity**: la dirección de la entidad (en este caso, la dirección de la TL)

Si la entidad se encuentra registrada y no ha sido revocada, el resultado sera un valor boolean (true).

```
? Pick a network lacchain
? Pick an instance PKD at 0x52663a5679b5126FA23c6f48666BFebd8466c936
? Select which function isActive(entity: address)
? entity: address: 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
✓ Method 'isActive(address)' returned: true
true
```

De esta manera, aunque una entidad pueda registrar un nodo padre como la dirección de un PKD valido, el proceso de verificación de RoT consiste en preguntar al PKD si realmente la entidad o TL esta registrado y es valida.

## Revocar una entidad en una Lista de Confianza

El proceso de revocación de una entidad registrada en una TL se debe realizar invocando a la función **revoke** del Smart Contract asociado a la TL desde la cuenta owner que desplegó el contrato.

La invocación al contrato se hace con los siguientes parámetros:

- **red**: lacchain
- **instancia**: TrustedList

- **función**: revoke
- **entity**: la dirección de la entidad (en este caso: 0x70aab07a3a509fe630760a2c8ef34e8d28064278)

```
? Pick a network lacchain
? Pick an instance TrustedList at 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
? Select which function revoke(_entity: address)
? _entity: address: 0x70aab07a3a509fe630760a2c8ef34e8d28064278
✓ Transaction successful. Transaction hash: 0xebff9e12ccc8298ab333ef5430adbd03f684c61188a954ebdb0c06d073f7abfd
Events emitted:
 - EntityRevoked(0x70aAB07A3A509Fe630760a2c8Ef34e8D28064278)
```

Una vez revocada la entidad, podemos volver a verificar su estatus llamando a la función entices del TL, como se puede ver en la siguiente imagen:

```
? Pick a network lacchain
? Pick an instance TrustedList at 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
? Select which function entities(address)
? #0: address: 0x70aab07a3a509fe630760a2c8ef34e8d28064278
✓ Method 'entities(address)' returned: (did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278, 3600, 2, did:lac:main:0x70aab07a3
a509fe630760a2c8ef34e8d28064278, 3600, 2)
Result {
  '0': 'did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278',
  '1': '3600',
  '2': '2',
  did: 'did:lac:main:0x70aab07a3a509fe630760a2c8ef34e8d28064278',
  expires: '3600',
  status: '2'
}
```

El resultado muestra un status=2, lo cual indica que la entidad ha sido revocada. La lista de status que se maneja dentro de una TL es:

- 0: Sin registrar
- 1: Registrada
- 2: Revocada

## Revocar una clave pública en un PKD

El proceso de revocación de una clave pública en un PKD (la cual puede estar asociada también a una TL) es similar a revocar una entidad dentro de una TL.

La invocación al contrato se hace con los siguientes parámetros:

- **red**: lacchain
- **instancia**: PKD
- **función**: revoke
- **entity**: la dirección de la clave publica(en este caso, el de la TL: 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C)

```
? Pick a network lacchain
? Pick an instance PKD at 0x52663a5679b5126FA23c6f48666BFebd8466c936
? Select which function revoke(_entity: address)
? _entity: address: 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
✓ Transaction successful. Transaction hash: 0xab7d817527e14b0401332e3d94c7ed8e180a7021af4970ae230aead5186d6a8b
Events emitted:
 - PublicKeyRevoked(0x4f706ebB346C05DC8591Ee68C5576c725a75D94C)
```

Una vez revocada la clave publica asociada a una TL, podemos volver a verificar su estatus llamando a la función **isActive** del PKD, como se muestra a continuación.

```
? Pick a network lacchain
? Pick an instance PKD at 0x52663a5679b5126FA23c6f48666BFebd8466c936
? Select which function isActive(entity: address)
? entity: address: 0x4f706ebB346C05DC8591Ee68C5576c725a75D94C
✓ Method 'isActive(address)' returned: false
false
```

Como podemos ver ahora, el resultado de la verificación es **false**, lo cual indica que la entidad ha sido revocada.