Gestión de identidades descentralizadas con Blockchain

SISTEMAS DE AUTENTICACIÓN Y AUTORIZACIÓN

Autor: Xabier Legarda Fernández de Arroyabe

Tutor: Enric Hernández Jiménez

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

Fecha: 31/12/2018

RESUMEN

En este escrito se documenta el trabajo realizado durante el Trabajo de Final de Master. El documento se basa en la siguiente temática: gestión de identidades descentralizada con blockchain.

La gestión de identidades se ha convertido en tema crítico. El método centralizado que se actualmente ha quedado expuesto mostrándose inseguro ante los posibles ataques para el robo de información. Por ello, se plantea una nueva manera de gestionar las identidades de forma descentralizada y donde cada uno tenga el control de sus datos personales.

Se analiza la tecnología blockchain y sus elementos principales. Esta tecnología ha demostrado ser seguro en redes distribuidas públicas. Llegando a formar grandes redes descentralizadas como la del Bitcoin o la de Ethereum.

Ethereum es la plataforma que se ha seleccionado para introducirse en las redes descentralizadas. Creando de forma local una red descentralizada con blockchain basándonos en la plataforma Ethereum. Se explica paso a paso la creación de la red y como interactúa.

Después, se pasa a analizar las diferentes maneras que hay de gestionar las identidades. Centrándose en la gestión descentralizada donde se explica por encima un ejemplo sencillo de cómo funcionaría en una situación real.

Luego, se explica en que consiste el proyecto Hyperledger Indy. Se explica el propósito y el estado de dicho proyecto. A continuación, se define el proceso a seguir para una demo usando la tecnología de Hyperledger Indy y explicando el código fuente que habría que utilizar para llevar a cabo la demo.

Por último, se exponen las conclusiones sacadas del proyecto. Además, se proponen posibles acciones a futuro para poder seguir desarrollando la tecnología explicada.

ABSTRACT

This document explains the work done during the Master's Final Work. The document is base on the following theme: decentralized identity management with blockchain.

Identity management has become a critical issue. The centralized method has been exposed showing insecurity against possible attacks for the theft of information. Therefore, a new way of managing identities is considered in a decentralized way and where each one has control over their personal data.

The blockchain technology and its main elements are analyzed. This technology has proven to be secure in public distributed networks. Reaching large decentralized networks such as Bitcoin or Ethereum.

Ethereum is the platform that has been selected to be introduced in decentralized networks. Creating locally a decentralized network with blockchain based on the Ethereum platform. The creation of the network and how it interacts is explained step by step.

After, it analyzes the different ways of managing identities. Focusing on decentralized management where a simple example of how it would work in a real situation is explained.

Then, it explains what the Hyperledger Indy project consists of. The purpose and status of said project is explained. Next, we define the process to follow for a demo using the technology of Hyperledger Indy and explaining the source code that would be used to carry out the demo.

Finally, the conclusions drawn from the project are presented. In addition, possible future actions are proposed to continue developing the technology explained.

Contenido

Tabla de ilustraciones

1. Introducción

Todo individuo desde que nace tiene derecho a una identidad donde se especifican su nombre, apellidos y otros datos que sirven para diferenciar e identificar un individuo frente a la sociedad. Esta identidad puede estar reflejada en nuestro documento de identidad nacional, el cual, cada uno gestiona y decide con quien compartirlo.

La identidad digital es como se identifica uno en internet o cuando usa diferentes herramientas telemáticas. El control de la identidad digital actualmente se realiza de forma centralizada. Al realizarse de forma centralizada se pierde el control de esa información y es la entidad que almacena nuestra identidad la que decide cómo gestionarlo. Por ejemplo, cuando un usuario se registra en Facebook los datos que se dan para identificarse quedan registrados en sus sistemas y el control de esa información pasa a estar en sus manos. Este método ha tenido muchos incidentes históricamente. Como el caso entre el propio Facebook y Cambridge Analitica o las numerosas veces donde la información de los usuarios se he visto comprometida por fallas de seguridad en los sistemas contra los que se autentican los usuarios mediante una gestión centralizada.

Una gestión descentralizada permite ser los propietarios de la identidad digital y gestionarlo libremente. Permitiendo elegir que se comparte y con quien. Por ejemplo, en la entrada de una discoteca ese entrega al personal de seguridad el DNI cuando realmente lo único que sería necesario compartir sería si es mayor de edad o no. Otros datos como el nombre, lugar de nacimiento etc, no deberían de ser compartidos ni relevantes para ese caso. También, evitaría el tráfico de información que realizan algunas empresas de la información de terceros sin tener conocimiento de ello. Si alguien quisiera comprar información relativa a una persona debería de comprarla directamente a esa persona. Este sistema podría ser muy beneficioso para los usuarios, como por ejemplo, en el caso de los datos médicos. Cada uno gestionaría su historial médico y controlaría quien accede quien no, en vez de estar en un sistema controlado por sanidad donde varios empleados con permisos pueden acceder. Además, en caso de usar los datos médicos para realizar algún estudio o experimento se podría decidir si donarlos o venderlos dependiendo del caso. Para aquellos que dudan de la seguridad de esta red descentralizada, se ha demostrado aplicado a otros ámbitos que se puede crear una red pública descentralizada con blockchain y que sea seguro, por ejemplo, Ethereum con el intercambio de smart contracts.

Actualmente, existen varios proyectos que están basados en la idea del intercambio de identidades descentralizado con blockchain. Una de ellas es Evernym creado bajo la Fundación Sovrin. Aunque esta es una solución híbrida, ya que, sigue almacenando datos de forma centralizada. Otro proyecto interesante es ID2020 que tiene como meta que todo acabar con el problema de la identidad en países pobres. En los cuales muchas personas no están identificadas y no figuran en ningún registro. Por lo tanto, quedan excluidos de la sociedad al no poder hacer uso de ningún servicio. Es una tecnología basada en blockchain que pretende garantizar el derecho humano universal a una identidad. El último proyecto a destacar sería BanQu que ha desarrollado la primera identidad económica mediante tecnología blockchain que tiene como objetivo crear oportunidades económicas para personas en refugiada o en extrema pobreza. Esto permitiría realizar transacciones con personas o identidades de todo el mundo sin necesidad de usar un intermediario como un banco. Toda la información es almacenada mediante blockchains y con tan sólo tener una cartera digital sería possible operar.

En este proyecto, se intentará profundizar en la tecnología de blockchain para la gestión de identidades descentralizada. El primer paso será hacer uso de alguna plataforma descentralizada mediante blockchain que sea pública como Ethereum para entender mejor esta tecnología. Después, se analizarán las diferentes soluciones para gestión de identidades e intentar trabajar con ellas para ver su funcionamiento para finalmente crear una pequeña demostración de una red de gestión de identidades descentralizada.

1.1 Objetivos

A continuación, se definen los objetivos marcados para el trabajo de fin de máster:

- Entender el funcionamiento de la tecnología blockchain y su aplicación en redes descentralizadas.
- Ser capaz de liberar blockchain propio dentro de una red pública descentralizada.
- Entender el funcionamiento de la gestión de identidades descentralizada mediante blockchain.
- Ser capaz de realizar una demostración del funcionamiento de la gestión de identidades descentralizada mediante blockchain.

1.2 Metodología

Para poder alcanzar los objetivos anteriores se definirá una metodología separada en diferentes fases. Mediante el cumplimiento de las diferentes fases definidas se espera conseguir el objetivo final del proyecto

En la primera fase, se afianzarán los conocimientos de la tecnología de blockchain. Se estudiará su funcionamiento de forma técnica. Se analizarán diferentes casos de éxito donde se ha aplicado esta tecnología y como podría aplicarse para la gestión de identidades.

En la siguiente, fase se creará un blockchain propio para deployarlo en alguna de las redes que ya existe, por ejemplo, Ethereum. Se aprenderán los requisitos necesarios para crear un blockhain propio, así como los elementos que lo componen. Para ello, se utilizará una máquina virtual Linux.

Una vez comprendido el funcionamiento de la tecnología blockchain, se analizará la gestión de identidades descentralizada con blockchain. Se utilizarán como referencia algunos de los proyectos existentes como Sovrin. Se hará uso de su tecnología para comprender mejor el funcionamiento.

Por último, se creará una red con una gestión de identidades descentralizada donde cada individuo pueda controlar su identidad. Se realizará una demo donde poder demostrar el funcionamiento de la red y la utilización de la tecnología blockchain dentro de la misma.

1.3 Planificación

En este apartado, se muestra la planificación del proyecto mediante un diagrama de Gantt. En el diagrama, aparecen todas las tareas con sus líneas temporales especificadas.

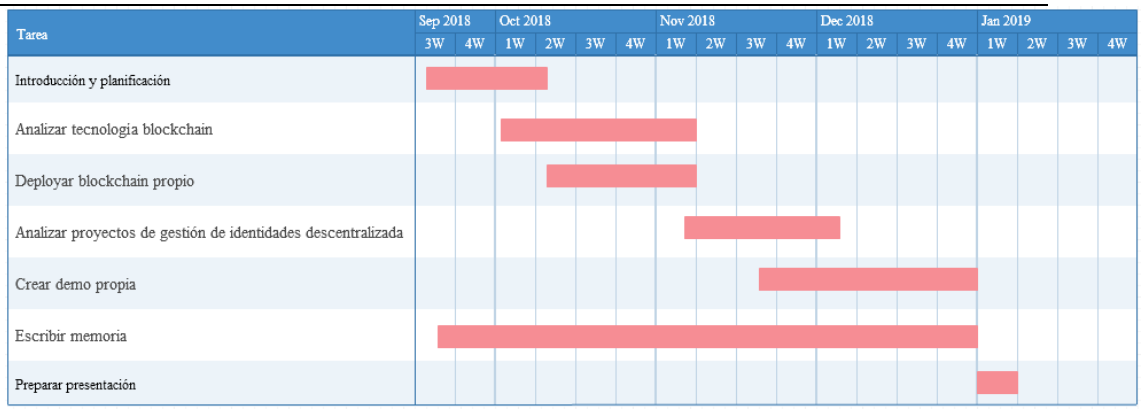

Ilustración 1: Diagrama de Gantt de la planificación de trabajo

2. Blockchain

La tecnología blockchain fue inventada en 2008 por el seudónimo de Satoshi Nakamoto. Se hizo famosa gracias al auge del Bitcoin y las criptomonedas. A pesar de ello, esta tecnología puede ser muy útil para otras funcionalidades como ha sido demostrado. Una frase que podría definir al blockchain es que es un diario casi imposible de falsificar, ya que, se guardan todas las transacciones realizadas dentro de la red distribuida. En este apartado se explicará cómo funciona el blockchain y algunos de los elementos que la componen.

2.1. Principios del blockchain

2.1.1. Red distribuida

La información no pertenece a un ente central, ya que, es distribuida por toda la red y no. Cada uno es propietario de su información y elige con quien compartirla. A diferencia del sistema centralizada nadie controla ni es propietaria de la información de los demás. Por lo tanto, la conexión entre los diferentes nodos es P2P (peer to peer), en vez de usar un nodo central.

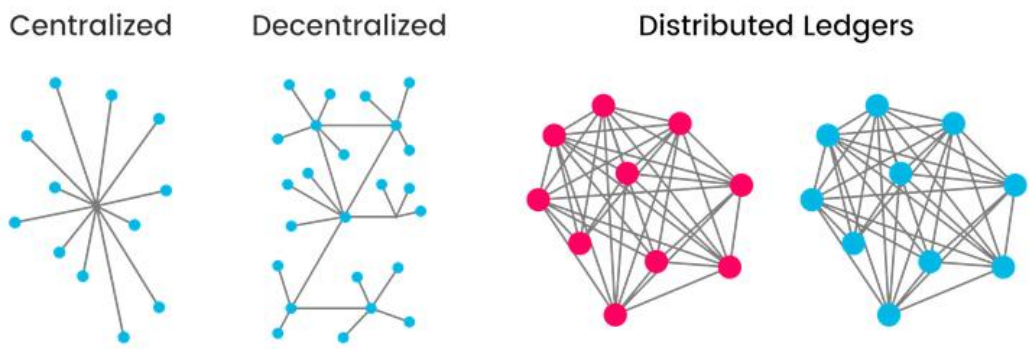

Ilustración 2: Diferencia entre las diferentes arquitecturas de red

2.1.2. Transparencia

Cualquiera que analice la cadena de bloques es capaz de ver cada transacción realizada y sus valores hash. Cada transacción queda guardada y no es posible modificarlo. Todos los nodos de la red pueden observar las transacciones que se están realizando.

2.1.3. Disponibilidad

Mientras uno de los nodos se mantenga conectado la red se mantiene disponible. Si hay nodos distribuidos por todo el mundo es prácticamente imposible que la red se caiga, ya que, habría que desconectar lo nodos localizados por todo el mundo.

2.1.4. Confidencialidad

Cada transacción esta firma mediante un clave pública, la cual, solo se puede descifrar mediante la clave privada emparejada que solo la tiene el propietario de la clave pública. Por lo tanto, tan solo el destinatario de la transacción es capaz de descifrarlo.

2.1.5. Integridad

La integridad del blockchain está asegurada gracias a que cada nodo tiene una copia del mismo y la aprobación de nuevos bloques tiene que estar consensuada por los nodos. Además, cada bloque del blockchain guarda el hash del bloque anterior. Si alguien intentase alterar algún bloque el valor del hash cambiaría.

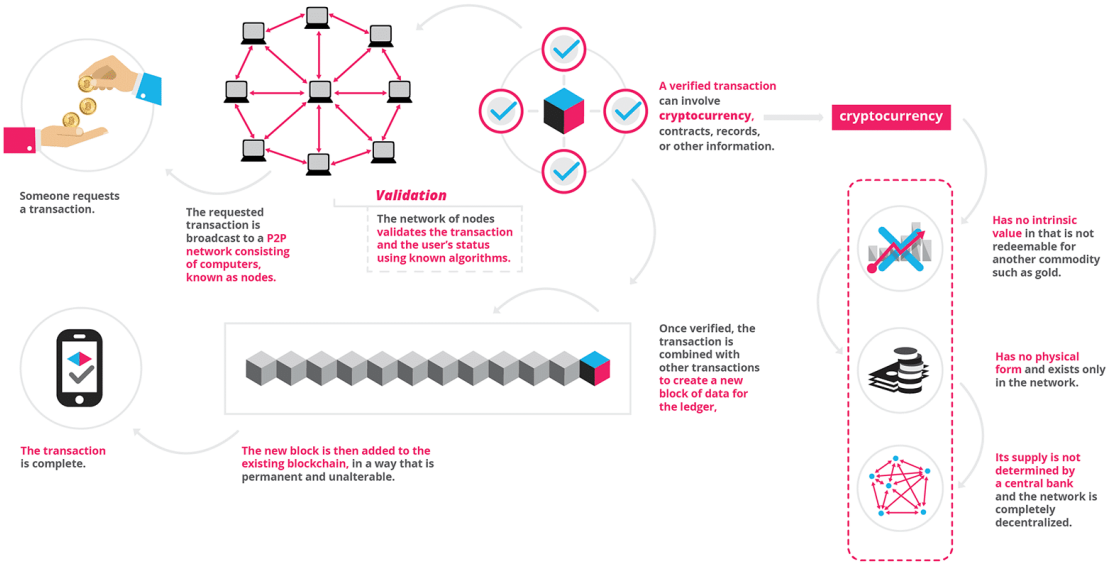

Ilustración 3: Flujo del funcionamiento del blockchain aplicado en criptomonedas

2.2. Elementos del blockchain

2.2.1. Función hash

Una función hash es una función matemática que recibe una cantidad variable de caracteres y las convierte un string con una cantidad fija de caracteres. El que se produzca un pequeño cambio en la cadena recibida generaría un hash completamente diferente. El hash es una cadena de caracteres (números y letras) producidas por una función hash. Los bloques dentro de blockchain se hashean.

2.2.2. Nonce

El término nonce viene de la expresión inglesa (number that can only be used once). En criptografía es un código de un solo uso. Es número aleatorio que se añade al hash para evitar la manipulación de la información de bloques antiguos.

2.2.3. Nodo

Los nodos son los elementos que componen una red distribuida que funcione mediante blockchain. Cada nodo tiene una copia digital de la cadena de bloques para desmantelar una red que funcione mediante blockchain se tendrían que desconectar todos los nodos de la red a la vez. Cada nodo verifica cada transacción, si la mayoría de los nodos dan por válida la transacción se escribe en un bloque.

2.2.4. Bloque

La cadena de todos los bloques forma el blockchain. Como se comenta en el punto anterior cada nodo tiene una copia del blockchain y cuando un bloque alcanza una cantidad de verificaciones válidas se forma un nuevo bloque. El blockchain se actualiza automáticamente no

necesita que un elemento central lance la orden. Una vez que un nuevo bloque entra en la cadena no se puede modificar. Cuando se crea un bloque también se genera un hash. El hash queda almacenado en el bloque con la información de la transacción. Cada bloque almacena el hash del bloque anterior para crear la cadena de bloques. Si algo se modificase dentro de un bloque el hash cambiaría.

2.2.5. Proof-of-work

La prueba de trabajo es un proceso producción de datos difíciles de obtener pero fácil de verificar. En la tecnología blockchain se trata de resolver problemas matemáticos. Si el problema se resuelve correctamente un nuevo bloque se puede añadir al bloque. Entre todos los nodos el primero en resolver el problema es quien puede añadir el siguiente bloque a la cadena. La prueba matemáticas deben ser difíciles de resolver pero fáciles de verificar para evitar trampas.

2.2.6. Cartera

La cartera o wallet en inglés genera una pareja de claves públicas y privadas que asegura la seguridad de las transacciones. Cualquiera puede lanzar una transacción utilizando la clave pública de la dirección de un receptor pero solo el propietario de esa dirección que es quien tiene la clave privada puede acceder al valor de la transacción. Por lo tanto, antes de mandar una transacción se solicitaría la clave pública del receptor y se firmaría con esa clave. De esa manera, sólo el receptor podría descifrar la transacción con la clave privada emparejada a la clave pública. Se le llama cartera por su origen en el bitcoin pero podría almacenar otros elementos pudiendo ser un contenedor de credenciales.

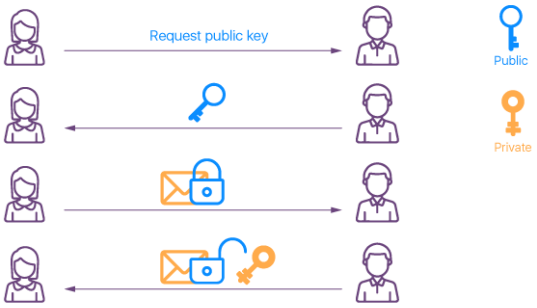

Ilustración 4: Flujo del intercambio de claves

2.2.7. Protocolo

El protocolo es el conjunto de reglas programadas que define el comportamiento de cada elemento dentro de la red. Los protocolos permiten que la red de blockchain funcione como está previsto de forma autónoma y sin que esté controlado por nadie.

3. Crear una red con blockchain mediante Ethereum

3.1. Introducción

Ethereum es una plataforma de software libre basado en blockchain. Permite a los desarrolladores crear aplicaciones descentralizadas donde la criptomoneda Ether es una de las aplicaciones. La cantidad de aplicaciones descentralizadas creadas bajo la plataforma Ethereum sigue creciendo. Entre ellas se pueden encontrar aplicaciones para la votación electrónica, gestión de activos, smart contracts, etc.

En este apartado se explicará con montar una red de Ethereum propia. Se explicarán los pasos que se van dando hasta tener una red propia en funcionamiento. Para ello, se crearán dos nodos en la misma máquina creando una red peer-to-peer en local. Para realizar esta demo, se ha utilizado una instalación limpia de Debian 9 en una máquina virtual. Para poner en marcha la red de Ethereum se ha utilizado Geth (Go implementation of Ethereum protocole) que se puede descargar desde la propia página de Ethereum.

3.2. Área de trabajo

Lo primero será crear los directorios que compondrán el área de trabajo. En este caso, se descomprimen los ficheros descargados de la página de Ethereum (Geth & Tools) en el directorio /opt. Los siguientes pasos, son crear los directorios donde guardaremos los archivos de la red y los nodos.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09# mkdir devnet
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09# cd devnet/
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# mkdir node1
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# mkdir node2
```

Ilustración 5: Crear workspace

3.3. Creación de nodos

Se crean las cuentas (carteras) con las que operarán los nodos. Cada uno de ellos tiene una pareja de claves pública y privada para interactuar dentro de la red. De esta manera, pueden firmar las transacciones y ser identificados dentro de la red.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node1/ account new
INFO [12-02|17:50:07.341] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {f52efe7370d1bf494a3a27dbac58a8e9084c1aea}
```

Ilustración 6: Creación de cuenta para nodo1

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node2/ account new
INFO [12-02|17:54:48.000] Maximum peer count                       ETH=25 LES=0 total=25
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {4a4409d0165494e9302ba982fed1b85f00539ac3}
```

Ilustración 7: Creación de cuenta para nodo2

Se crea una carpeta “keystore” que contiene el archivo de la cuenta. La última parte del nombre de ese fichero es la dirección de la cuenta. Se recomienda guardar las direcciones de cada nodo en un archivo de texto, ya que, más adelante harán falta.

3.4. Creación del fichero génesis

El bloque génesis es el primer bloque del blockchain. Es el único bloque de la cadena que no guarda la dirección de un bloque anterior al ser el primero. Los parámetros del génesis se almacenan en el fichero genesis.json. Geth tiene una herramienta llamada puppeth que permite crear el archivo génesis sin tener que crearlo a mano desde cero. En este caso, el nombre de la red será devnet.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../puppeth
+------------------------------------------------------------+
| Welcome to puppeth, your Ethereum private network manager  |
|                                                            |
| This tool lets you create a new Ethereum network down to   |
| the genesis block, bootnodes, miners and ethstats servers  |
| without the hassle that it would normally entail.          |
|                                                            |
| Puppeth uses SSH to dial in to remote servers, and builds  |
| its network components out of Docker containers using the  |
| docker-compose toolset.                                    |
+------------------------------------------------------------+

Please specify a network name to administer (no spaces or hyphens, please)
> devnet

Sweet, you can set this via --network=devnet next time!

INFO [12-02|18:06:19.066] Administering Ethereum network           name=devnet
WARN [12-02|18:06:19.067] No previous configurations found         path=/root/.puppeth/devnet

What would you like to do? (default = stats)
 1. Show network stats
 2. Configure new genesis
 3. Track new remote server
 4. Deploy network components
> 2
```

Ilustración 8: Creación del génesis

A continuación, se especificarán algunos de los parámetros de cómo será la red a crear:

```
Which consensus engine to use? (default = clique)
 1. Ethash - proof-of-work
 2. Clique - proof-of-authority
> 2

How many seconds should blocks take? (default = 15)
> 5

Which accounts are allowed to seal? (mandatory at least one)
> 0xf52efe7370d1bf494a3a27dbac58a8e9084c1aea
> 0x4a4409d0165494e9302ba982fed1b85f00539ac3
> 0x

Which accounts should be pre-funded? (advisable at least one)
> 0x4a4409d0165494e9302ba982fed1b85f00539ac3
> 0xf52efe7370d1bf494a3a27dbac58a8e9084c1aea
> 0x
```

Ilustración 9: Configuración del génesis

La red utilizará el motor de consenso de tipo proof-of-authority. Al tratarse de una red privada el tener que minar mediante el método proof-of-work es un gasto innecesario de recursos. También, se especifican que cuentas tiene permiso para votar y sellar si un bloque es válido. Se decide que cuentas deberían de tener fondos. Para la demo, se seleccionan ambas cuentas en los dos casos. En el argot de Geth, un node que puede votar se le llama “sealer”.

Por último, habrá que especificar el ID de la cadena/red. Para esta prueba se le da el identificador 8888.

```
Specify your chain/network ID if you want an explicit one (default = random)
> 8888
INFO [12-02|18:11:52.414] Configured new genesis block
```

Ilustración 10: ID de la cadena/red

Una vez definidos los parámetros del génesis se exporta a un fichero. Al fichero habrá que darle el nombre de genesis.json y se coloca en el directorio devnet.

3.5. Inicializar nodos

Utilizando el fichero génesis creado (genesis.json) se crea el bloque génesis. Importante que cada nodo se inicialice con el mismo fichero génesis.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node1/ init genesis.json
INFO [12-02|18:17:28.211] Maximum peer count                       ETH=25 LES=0 total=25
INFO [12-02|18:17:28.212] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node1/geth/chaindata cache=16 handles=16
INFO [12-02|18:17:28.222] Writing custom genesis block
INFO [12-02|18:17:28.289] Persisted trie from memory database      nodes=358 size=52.27kB time=13.642072ms gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [12-02|18:17:28.291] Successfully wrote genesis state         database=chaindata                                                  hash=b320bb…b902e4
INFO [12-02|18:17:28.291] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node1/geth/lightchaindata cache=16 handles=16
INFO [12-02|18:17:28.305] Writing custom genesis block
INFO [12-02|18:17:28.352] Persisted trie from memory database      nodes=358 size=52.27kB time=16.926015ms gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [12-02|18:17:28.353] Successfully wrote genesis state         database=lightchaindata                                             hash=b320bb…b902e4
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node2/ init genesis.json
INFO [12-02|18:17:36.029] Maximum peer count                       ETH=25 LES=0 total=25
INFO [12-02|18:17:36.029] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node2/geth/chaindata cache=16 handles=16
INFO [12-02|18:17:36.038] Writing custom genesis block
INFO [12-02|18:17:36.097] Persisted trie from memory database      nodes=358 size=52.27kB time=9.96731ms gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [12-02|18:17:36.104] Successfully wrote genesis state         database=chaindata                                                  hash=b320bb…b902e4
INFO [12-02|18:17:36.112] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node2/geth/lightchaindata cache=16 handles=16
INFO [12-02|18:17:36.123] Writing custom genesis block
INFO [12-02|18:17:36.160] Persisted trie from memory database      nodes=358 size=52.27kB time=3.802367ms gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [12-02|18:17:36.168] Successfully wrote genesis state         database=lightchaindata                                             hash=b320bb…b902e4
```

Ilustración 11: Inicialización de nodos

En este caso se están inicializando los nodos utilizando el fichero génesis, si se quisiera conectar a una red pública, los parámetros del génesis están definidos en el código fuentes en la carpeta params/.

3.6. Bootnode

Bootnode es un servicio de descubrimiento. El objetivo es que los diferentes nodos sean capaces de encontrarse. La razón de activar el bootnode es que los nodos pueden tener IPs dinámicas o pueden estar siendo apagados y encendidos. El bootnode suele tener una IP estática y funciona como un punto de encuentro donde los nodos saben que ahí se encontrarán con sus compañeros.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../bootnode -genkey boot.key
```

Ilustración 12: Generar bootnode

A continuación, habrá que poner en marcha el bootnode dando así el primer paso para arrancar nuestra red.

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../bootnode -nodekey boot.key -verbosity 9 -addr :30310
INFO [12-03|18:49:12.557] New local node record                    seq=1 id=e115acb177264b98 ip=<nil> udp=0 tcp=0
```

Ilustración 13: Arrancar bootnode

Se añade la opción verbosity para poder ver en tiempo real como actúan los nodos. Se puede elegir cualquier puerto para arrancar el bootnode en este caso se elige el 330310, por ejemplo, el 330303 se suele usar en las redes públicas de Ethereum.

3.7. Arrancar los nodos

Arranque del nodo1:

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node1/ --syncmode 'full' --port 30311 --rpc --rpcaddr 'localhost' --rpcport 8501 --rpcapi 'per
sonal,db,eth,net,web3,txpool,miner' --bootnodes 'enode://3ec4fef2d726c2c01f16f0a0030f15dd5a81e274067af2b2157cafbf76aa79fa9c0be52c6664e80cc5b08162ede53279bd70ee10d024fe86613
b0b09e1106c40@127.0.0.1:30310' --networkid 8888 --gasprice '1' -unlock '0xf52efe7370d1bf494a3a27dbac58a8e9084c1aea' --password node1/password.txt --mine
INFO [12-03|18:53:04.758] Maximum peer count                       ETH=25 LES=0 total=25
INFO [12-03|18:53:04.759] Starting peer-to-peer node               instance=Geth/v1.8.19-stable-dae82f09/linux-amd64/go1.11.2
INFO [12-03|18:53:04.759] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node1/geth/chaindata cache=512 handles=512
INFO [12-03|18:53:04.800] Initialised chain configuration          config="{ChainID: 8888 Homestead: 1 DAO: <nil> DAOSupport: false EIP150: 2 EIP155: 3 EIP158: 3 Byzantium:
 4 Constantinople: <nil> Engine: clique}"
INFO [12-03|18:53:04.800] Initialising Ethereum protocol           versions="[63 62]" network=8888
INFO [12-03|18:53:04.895] Loaded most recent local header          number=0 hash=b320bb…b902e4 td=1 age=1d44m51s
INFO [12-03|18:53:04.900] Loaded most recent local full block      number=0 hash=b320bb…b902e4 td=1 age=1d44m51s
INFO [12-03|18:53:04.900] Loaded most recent local fast block      number=0 hash=b320bb…b902e4 td=1 age=1d44m51s
INFO [12-03|18:53:04.900] Regenerated local transaction journal    transactions=0 accounts=0
INFO [12-03|18:53:04.902] Stored checkpoint snapshot to disk       number=0 hash=b320bb…b902e4
INFO [12-03|18:53:04.911] New local node record                    seq=1 id=361db9e7124f55d3 ip=127.0.0.1 udp=30311 tcp=30311
INFO [12-03|18:53:04.917] Started P2P networking                   self=enode://eff57cd5fd351df672752aea955be189ae5e2186ef1c4f59af7f5d08f0242955feecc2114434c5f5214a88a15bff
c1220d4aa867917c305ef7bf4f5dfd3d6922@127.0.0.1:30311
INFO [12-03|18:53:04.921] IPC endpoint opened                      url=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node1/geth.ipc
INFO [12-03|18:53:04.922] HTTP endpoint opened                     url=http://localhost:8501                                  cors= vhosts=localhost
INFO [12-03|18:53:06.298] Unlocked account                         address=0xf52Efe7370d1BF494A3a27DbAc58a8E9084C1aea
INFO [12-03|18:53:06.298] Transaction pool price threshold updated price=1
INFO [12-03|18:53:06.298] Transaction pool price threshold updated price=1
INFO [12-03|18:53:06.298] Etherbase automatically configured       address=0xf52Efe7370d1BF494A3a27DbAc58a8E9084C1aea
INFO [12-03|18:53:06.299] Commit new mining work                   number=1 sealhash=9f97a2…2f17fd uncles=0 txs=0 gas=0 fees=0 elapsed=99.337µs
INFO [12-03|18:53:06.299] Successfully sealed new block            number=1 sealhash=9f97a2…2f17fd hash=22ffca…8ebe12 elapsed=684.181µs
INFO [12-03|18:53:06.299] mined potential block                     number=1 hash=22ffca…8ebe12
INFO [12-03|18:53:06.300] Commit new mining work                   number=2 sealhash=256207…fc2f54 uncles=0 txs=0 gas=0 fees=0 elapsed=367.374µs
INFO [12-03|18:53:06.300] Signed recently, must wait for others
```

Ilustración 14: Arrancar nodo1

- --syncmode 'full' ayuda a prevenir el error Discarded Bad Propagated Block.
- --port 3011 es el puerto para el nodo1 tiene que ser diferente al del bootnode al estar trabajando en local.
- --rpcapi permite usar los modules listados mediante llamadas RPC.
- --bootnodes la dirección del bootnode para que el nodo lo pueda encontrar
- --networkId el id definido al crear el archivo genesis.json.
- --unlock se especifica la dirección de la cuenta a desbloquear para ese nodo
- --password se especifica un fichero de texto donde está almacenada la contraseña de la cuenta
- --mine esta opción sirve para activar el minado en el nodo

El arranque del nodo2 se hará de la misma manera que el nodo1. Cambiando los datos específicos del nodo:

```
root@xlegarda:/opt/geth-linux-amd64-1.8.19-dae82f09/devnet# ../geth --datadir node2/ --syncmode 'full' --port 30312 --rpc --rpcaddr 'localhost' --rpcpo
rt 8502 --rpcapi 'personal,db,eth,net,web3,txpool,miner' --bootnodes 'enode://3ec4fef2d726c2c01f16f0a0030f15dd5a81e274067af2b2157cafbf76aa79fa9c0be52c6
664e80cc5b08162ede53279bd70ee10d024fe86613b0b09e1106c40@127.0.0.1:30310' --networkid 8888 --gasprice '0' --unlock '0x4a4409d0165494e9302ba982fed1b85f00
539ac3' --password node2/password.txt --mine
INFO [12-03|18:56:06.621] Maximum peer count                       ETH=25 LES=0 total=25
INFO [12-03|18:56:06.622] Starting peer-to-peer node               instance=Geth/v1.8.19-stable-dae82f09/linux-amd64/go1.11.2
WARN [12-03|18:56:06.622] Sanitizing invalid miner gas price       provided=0 updated=0
INFO [12-03|18:56:06.622] Allocated cache and file handles         database=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node2/geth/chaindata cache=512
 handles=512
INFO [12-03|18:56:06.809] Initialised chain configuration          config="{ChainID: 8888 Homestead: 1 DAO: <nil> DAOSupport: false EIP150: 2 EIP155: 3
 EIP158: 3 Byzantium: 4 Constantinople: <nil> Engine: clique}"
INFO [12-03|18:56:06.809] Initialising Ethereum protocol           versions="[63 62]" network=8888
INFO [12-03|18:56:08.823] Loaded most recent local header          number=0 hash=b320bb…b902e4 td=1 age=1d47m55s
INFO [12-03|18:56:08.828] Loaded most recent local full block      number=0 hash=b320bb…b902e4 td=1 age=1d47m55s
INFO [12-03|18:56:08.828] Loaded most recent local fast block      number=0 hash=b320bb…b902e4 td=1 age=1d47m55s
INFO [12-03|18:56:08.828] Regenerated local transaction journal    transactions=0 accounts=0
INFO [12-03|18:56:08.838] New local node record                    seq=1 id=6389ba11451fe34a ip=127.0.0.1 udp=30312 tcp=30312
INFO [12-03|18:56:08.842] IPC endpoint opened                      url=/opt/geth-linux-amd64-1.8.19-dae82f09/devnet/node2/geth.ipc
INFO [12-03|18:56:08.843] HTTP endpoint opened                     url=http://localhost:8502                                  cors= vhosts=localho
st
INFO [12-03|18:56:08.861] Started P2P networking                   self=enode://99e6d2050c8cf9d750639692ff539ede558bdb3241cb9712a9298d799c1438e73e36a01
35eeb89076a1885ad951b422e38fefbdaa874a7c8746217d3ba32e4e7@127.0.0.1:30312
INFO [12-03|18:56:09.019] Stored checkpoint snapshot to disk       number=0 hash=b320bb…b902e4
INFO [12-03|18:56:11.195] Unlocked account                         address=0x4A4409d0165494E9302Ba982FED1B85F00539Ac3
INFO [12-03|18:56:11.197] Transaction pool price threshold updated price=0
INFO [12-03|18:56:11.197] Transaction pool price threshold updated price=0
INFO [12-03|18:56:11.198] Etherbase automatically configured       address=0x4A4409d0165494E9302Ba982FED1B85F00539Ac3
INFO [12-03|18:56:11.198] Commit new mining work                   number=1 sealhash=314af3…912cfb uncles=0 txs=0 gas=0 fees=0 elapsed=92.754µs
INFO [12-03|18:56:11.667] Successfully sealed new block            number=1 sealhash=314af3…912cfb hash=de5254…34dbe0 elapsed=469.812ms
INFO [12-03|18:56:11.668] mined potential block                     number=1 hash=de5254…34dbe0
INFO [12-03|18:56:11.672] Commit new mining work                   number=2 sealhash=f50ed1…7ca9b8 uncles=0 txs=0 gas=0 fees=0 elapsed=3.563ms
```

Ilustración 15: Arrancar nodo2

3.8. Análisis de las interacciones

En este punto, se analizarán mediante diferentes capturas de pantallas las interacciones de los nodos.

```
TRACE[12-03|19:02:32.383] >> PING/v4                                addr=127.0.0.1:30312 err=nil
TRACE[12-03|19:02:32.384] << PONG/v4                                addr=127.0.0.1:30312 err=nil
DEBUG[12-03|19:02:32.384] Revalidated node                          b=16 id=6389ba11451fe34a
TRACE[12-03|19:02:33.075] >> NEIGHBORS/v4                           addr=127.0.0.1:30312 err=nil
TRACE[12-03|19:02:33.076] << FINDNODE/v4                            addr=127.0.0.1:30312 err=nil
TRACE[12-03|19:02:34.444] >> PING/v4                                addr=127.0.0.1:30311 err=nil
TRACE[12-03|19:02:34.446] << PONG/v4                                addr=127.0.0.1:30311 err=nil
DEBUG[12-03|19:02:34.446] Revalidated node                          b=16 id=361db9e7124f55d3
```

Ilustración 16: Bootnode en funcionamiento

En esta captura se puede observar cómo se comporta la bootnode. Se puede ver como ambos se van registrando en la bootnode para de esa manera poder ser descubiertos y poder descubrir otros nodos.

```
INFO [12-03|19:05:13.003] Commit new mining work                    number=108 sealhash=229dfc…85f11f uncles=0 txs=0 gas=0 fees=0 elapsed=1.050ms
INFO [12-03|19:05:13.003] Signed recently, must wait for others
INFO [12-03|19:05:18.003] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=317.011µs mgasps=0.000 number=108 hash=4c92b4…f20e94 cache=0.00B
INFO [12-03|19:05:18.003] block reached canonical chain             number=101 hash=df5c4a…eeda15
INFO [12-03|19:05:18.003] Commit new mining work                   number=109 sealhash=1dac77…52a089 uncles=0 txs=0 gas=0 fees=0 elapsed=308.971µs
INFO [12-03|19:05:23.000] Successfully sealed new block            number=109 sealhash=1dac77…52a089 hash=27bc25…35eca0 elapsed=4.997s
INFO [12-03|19:05:23.000] mined potential block                     number=109 hash=27bc25…35eca0
INFO [12-03|19:05:23.001] Commit new mining work                   number=110 sealhash=c17b0e…c65f0b uncles=0 txs=0 gas=0 fees=0 elapsed=612.787µs
INFO [12-03|19:05:23.003] Signed recently, must wait for others
INFO [12-03|19:05:28.003] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=481.18µs  mgasps=0.000 number=110 hash=6d3865…7599bb cache=0.00B
INFO [12-03|19:05:28.003] block reached canonical chain             number=103 hash=aa562f…760a18
INFO [12-03|19:05:28.003] Commit new mining work                   number=111 sealhash=59352e…49cf0f uncles=0 txs=0 gas=0 fees=0 elapsed=338.877µs
INFO [12-03|19:05:33.001] Successfully sealed new block            number=111 sealhash=59352e…49cf0f hash=2665d4…f451bb elapsed=4.997s
INFO [12-03|19:05:33.001] mined potential block                     number=111 hash=2665d4…f451bb
INFO [12-03|19:05:33.003] Commit new mining work                   number=112 sealhash=e3df60…8f772f uncles=0 txs=0 gas=0 fees=0 elapsed=351.174µs
INFO [12-03|19:05:33.003] Signed recently, must wait for others
INFO [12-03|19:05:38.004] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=508.052µs mgasps=0.000 number=112 hash=74532c…ecb17e cache=0.00B
INFO [12-03|19:05:38.005] block reached canonical chain             number=105 hash=e7652c…e4a608
INFO [12-03|19:05:38.005] Commit new mining work                   number=113 sealhash=07988b…7aa6d5 uncles=0 txs=0 gas=0 fees=0 elapsed=575.965µs
INFO [12-03|19:05:43.001] Successfully sealed new block            number=113 sealhash=07988b…7aa6d5 hash=e16a0e…ca94c7 elapsed=4.996s
INFO [12-03|19:05:43.001] mined potential block                     number=113 hash=e16a0e…ca94c7
INFO [12-03|19:05:43.001] Commit new mining work                   number=114 sealhash=7748ad…6c805b uncles=0 txs=0 gas=0 fees=0 elapsed=355.676µs
INFO [12-03|19:05:43.001] Signed recently, must wait for others
```

Ilustración 17: Actividad nodo1

```
INFO [12-03|19:05:13.005] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=759.08µs  mgasps=0.000 number=107 hash=58c714…a231
61 cache=0.00B
INFO [12-03|19:05:13.006] block reached canonical chain             number=100 hash=b00886…219f80
INFO [12-03|19:05:13.006] Commit new mining work                   number=108 sealhash=8f1a55…77817b uncles=0 txs=0 gas=0 fees=0 elapsed=475.761µs
INFO [12-03|19:05:18.001] Successfully sealed new block            number=108 sealhash=8f1a55…77817b hash=4c92b4…f20e94 elapsed=4.994s
INFO [12-03|19:05:18.001] mined potential block                     number=108 hash=4c92b4…f20e94
INFO [12-03|19:05:18.002] Commit new mining work                   number=109 sealhash=30e8e2…a60114 uncles=0 txs=0 gas=0 fees=0 elapsed=681.019µs
INFO [12-03|19:05:18.003] Signed recently, must wait for others
INFO [12-03|19:05:23.002] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=412.797µs mgasps=0.000 number=109 hash=27bc25…35ec
a0 cache=0.00B
INFO [12-03|19:05:23.003] block reached canonical chain             number=102 hash=f39e89…739fbd
INFO [12-03|19:05:23.003] Commit new mining work                   number=110 sealhash=5e2796…40c8e9 uncles=0 txs=0 gas=0 fees=0 elapsed=439.391µs
INFO [12-03|19:05:28.001] Successfully sealed new block            number=110 sealhash=5e2796…40c8e9 hash=6d3865…7599bb elapsed=4.997s
INFO [12-03|19:05:28.001] mined potential block                     number=110 hash=6d3865…7599bb
INFO [12-03|19:05:28.004] Commit new mining work                   number=111 sealhash=f3bf20…3c955f uncles=0 txs=0 gas=0 fees=0 elapsed=515.448µs
INFO [12-03|19:05:28.004] Signed recently, must wait for others
INFO [12-03|19:05:33.002] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=281.922µs mgasps=0.000 number=111 hash=2665d4…f451
bb cache=0.00B
INFO [12-03|19:05:33.003] block reached canonical chain             number=104 hash=256f36…db8d96
INFO [12-03|19:05:33.003] Commit new mining work                   number=112 sealhash=3f79f5…716d53 uncles=0 txs=0 gas=0 fees=0 elapsed=1.000ms
INFO [12-03|19:05:38.001] Successfully sealed new block            number=112 sealhash=3f79f5…716d53 hash=74532c…ecb17e elapsed=4.997s
INFO [12-03|19:05:38.001] mined potential block                     number=112 hash=74532c…ecb17e
INFO [12-03|19:05:38.004] Commit new mining work                   number=113 sealhash=781a6f…7ae754 uncles=0 txs=0 gas=0 fees=0 elapsed=3.079ms
INFO [12-03|19:05:38.005] Signed recently, must wait for others
INFO [12-03|19:05:43.002] Imported new chain segment               blocks=1 txs=0 mgas=0.000 elapsed=259.41µs  mgasps=0.000 number=113 hash=e16a0e…ca94
c7 cache=0.00B
INFO [12-03|19:05:43.002] block reached canonical chain             number=106 hash=a61a59…bfd0b0
INFO [12-03|19:05:43.002] Commit new mining work                   number=114 sealhash=acad68…fd3366 uncles=0 txs=0 gas=0 fees=0 elapsed=170.631µs
INFO [12-03|19:05:48.001] Successfully sealed new block            number=114 sealhash=acad68…fd3366 hash=a03fea…ad518e elapsed=4.998s
```

Ilustración 18: Actividad nodo2

Se puede observar como ambos nodos realizan trabajos de minado y después esperan 5s (tiempo marcado en el génesis) para importar un nuevo segmento de la cadena. Al mismo tiempo se puede ver como los nodos, al estar la red configurada en modo proof-of-authority van sellando los diferentes bloques como válidos.

4. Gestión descentralizada de identidades

4.1. Identidad

La identidad está compuesta por tres ejes principales: relaciones, atributos y agentes.

- Relaciones: las relaciones definen a quien conoce y cuál es el trato con ellos.
- Atributos: los atributos son los que sirven para diferenciarse ante otros. Son la información relativa a la persona y los que definen las características de esa persona.
- Agentes: los agentes son los diferentes elementos que pueda hacer uso de la identidad de una persona o hacerse cargo de la representación de esa persona.

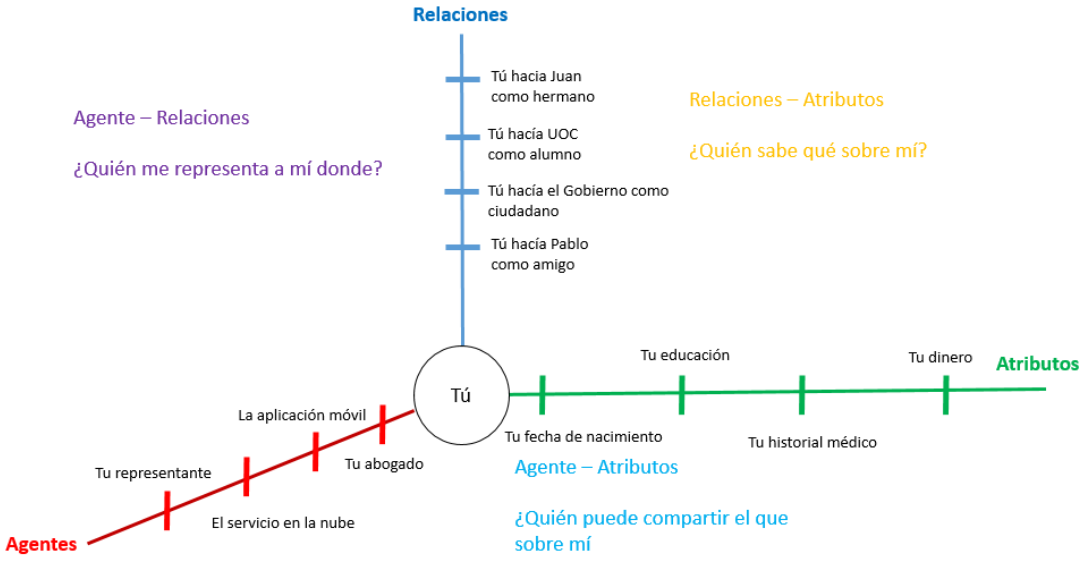

Ilustración 19: Ejes de la identidad

4.2. Arquitecturas de gestión de identidades

4.2.1. Gestión centralizada

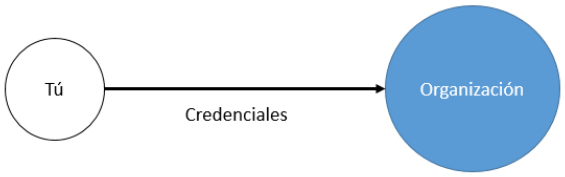

Ilustración 20: Gestión centralizada

Esta es la arquitectura más extendida. El usuario se identifica ante una organización central, a través de unas credenciales. Esas credenciales están almacenadas en la infraestructura de la organización.

4.2.2. Gestión federada

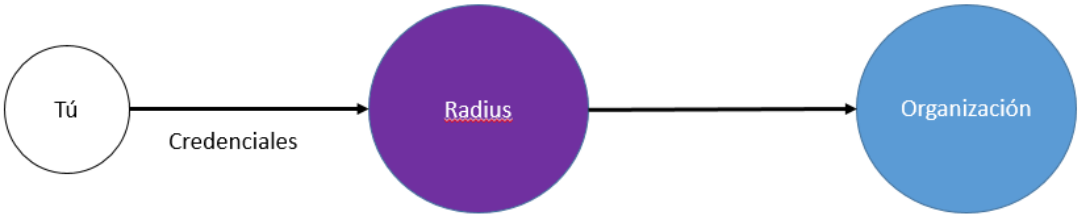

Ilustración 21: Gestión federada

Esta solución es frecuente en grandes organizaciones. Un sistema intermedio se encarga de la gestión de las identidades. Este sistema conecta después con los diferentes servicios ofrecidos, trabajando como intermediario. Esta forma de identificarse permite utilizar varios servicios sin necesidad de identificarse directamente ante cada uno de ellos. Un ejemplo son aquellas arquitecturas que cuenta un servidor RADIUS para gestionar las identidades. En este caso las credenciales también están en poder de la organización.

4.2.3. Gestión descentralizada

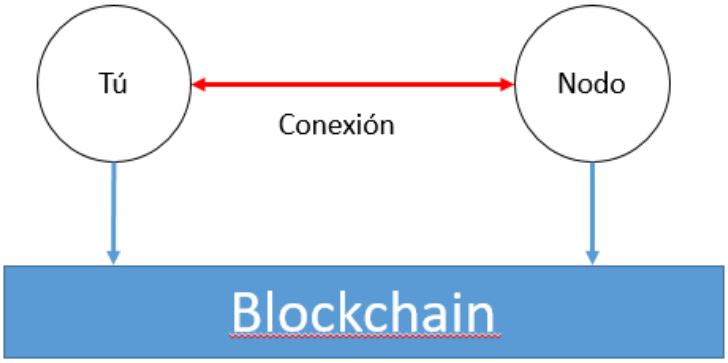

Ilustración 22: Gestión descentralizada

En este caso, se realiza una conexión punto a punto contra el ente al que hay que identificarse. Al mismo tiempo ambos son nodos de un sistema descentralizado que cuenta con libro mayor de todas las transacciones que se están realizando en la red, también, conocido como blockchain. El que realiza la petición decide que datos comparte con el otro nodo realizando una transacción encriptada que guardada en el blockchain. El nodo encargado de verificar la identidad desencriptará la información para obtener los datos. A continuación, se muestra cómo sería el proceso de entrada a una discoteca utilizando un sistema descentralizado. En este proceso tomarán parte tres agentes: el cliente, el portero y el Gobierno.

- El cliente intenta entrar a la discoteca y el portero le pide la identificación para comprobar que sea mayor de edad. En este punto, se establece la conexión entre ambos.
- El cliente consulta su cartera (wallet) y coge su fecha nacimiento. Utiliza la clave pública del portero para encriptar la información y lanzar la transacción.
- La información enviada por el cliente previamente ha tenido que ser verificada en otra transacción por el Gobierno sino ese dato no tendría validez.
- El portero recoge la transacción la desencripta utilizado su pareja de claves pública / privada y obtiene la fecha de nacimiento del cliente. Si el cliente es mayor de 18 años el portero le dejará pasar, en caso contrario, el cliente se quedará fuera.

5. Hyperledger Indy

5.1. Introducción

Hyperledger Indy es un proyecto de código abierto para la gestión de identidades descentralizada. En estos momentos, está en la fase de incubación. Es parte del proyecto Hyperledger que busca avanzar en el uso de la tecnología blockchain dentro de la industria. El proyecto Hyperledger está alojado por The Linux Foundation. En este apartado, se muestra como es la creación de un sistema descentralizado de identidades utilizado la tecnología del proyecto Hyperledger Indy.

5.2. Preparación del entorno

En este caso, se ha utilizado una máquina virtual con la versión Ubuntu Xenial 16.04.05. Para la demo, habrá que crear una pool de nodos local. El primer paso es, añadir el repositorio de Sovrin. Para ello se hará uso de los siguientes comandos:

```
root@xlegarda:~# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 68DB5E88

root@xlegarda:~# add-apt-repository "deb https://repo.sovrin.org/sdk/deb xenial stable"
```

Ilustración 23: Añadir repositorio Sovrin

Después, hay que actualizar la lista mediante el comando apt update. Para instalar el paquete de libindy hay que usar el comando:

```
root@xlegarda:~# apt-get install -y libindy
```

Ilustración 24: Instalar librería Indy

Esto sería lo esencial para poder desarrollar una aplicación para Hyperledger Indy.

5.3. Creación de demo

En la demo se enseña los pasos a seguir y el código fuente para que Xabi un exalumno de la UOC puedo solicitar su título universitario en formato digital a la Universidad Oberta de Catalunya.

El primer trozo irá destinado a la creación del bloque génesis, el cual, contiene el código del agente Steward. Los agentes Steward son los encargados de asignar las identificaciones a los diferentes agentes que se conectan a la red. Todo el código está escrito en lenguaje JSON.

```
await pool.set_protocol_version(2)

  pool_ = {'name': 'pooll'}
  pool_['genesis_txn_path'] = get_pool_genesis_txn_path(pool_['name'])
  pool_['config'] = json.dumps({"genesis_txn": str(pool_['genesis_txn_path'])})
  await pool.create_pool_ledger_config(pool_['name'], pool_['config'])
  pool_['handle'] = await pool.open_pool_ledger(pool_['name'], None)
```

Ilustración 25: Creación génesis

En la genesis_txn_path se determina el fichero de configuración donde están definadas las transacciones génesis. La función `pool.create_pool_ledger_config` permite crear una configuración del pool asignándole un nombre. Después de crear la configuración, se pueden conectar los nodos siguiendo lo marcado en la configuración con la función `pool.open_pool_ledger`.

Libindy tiene el concepto *wallet o cartera*. La cartera es un almacenamiento seguro donde guardar DID (descentralized indentifier), claves… Lo primero que se crea es una cartera para el Steward.

```
steward = {
      'name': "Sovrin Steward",
      'wallet_config': json.dumps({'id': 'sovrin_steward_wallet'}),
      'wallet_credentials': json.dumps({'key': 'steward_wallet_key'}),
      'pool': pool_['handle'],
      'seed': '000000000000000000000000Steward1'
  }

  await wallet.create_wallet(steward['wallet_config'], steward['wallet_credentials'])
  steward['wallet'] = await wallet.open_wallet(steward['wallet_config'], steward['wallet_credentials'])

  steward['did_info'] = json.dumps({'seed': steward['seed']})
  steward['did'], steward['key'] = await did.create_and_store_my_did(steward['wallet'], steward['did_info'])
```

Ilustración 26: Creación de wallet para Steward

Utilizando la función `wallet.create_wallet` se crea la cartera para Steward el siguiente paso es crear un DID para almacenarlo en la cartera con la función `did.create_and_store_my_did`.

Una vez creado el Steward el siguiente paso es que la UOC se conecte. El Steward crea un nuevo DID y lo almacena en su cartera. Este identificador lo utiliza para interactuar de manera segura con la UOC.

```
(steward['did_for_uoc'], steward['key_for_uoc']) = await did.create_and_store_my_did(steward['wallet'], "{}")
```

Ilustración 27: Creación de DID para UOC

El Steward lanza la transacción NYM al blockchain para ello llama a la función `ledger.build_nym_request` para crear la petición NYM y lo envía mediante la función `ledger.sign_and_submit_request`. Las transacciones NYM se usan para la creación de nuevos DID.

```
nym_request = await ledger.build_nym_request(steward['did'], steward['did_for_uoc'], steward['key_for_uoc'], None, role)
await ledger.sign_and_submit_request(steward['pool'], steward['wallet'], steward['did'], nym_request)
```

Ilustración 28: Envío de petición NYM

El Steward ha creado una petición de conexión que contiene un DID y un nonce.

```
connection_request = {
    'did': steward['did_for_faber'],
    'nonce': 123456789
}
```

Ilustración 29: Datos de la conexión

La UOC acepta la conexión del Steward y se crea una cartera `wallet.create_wallet`. La UOC crea un nuevo registro DID en su cartera para interactuar con el Steward `did.create_and_store_my_did`. También se crea, una respuesta de la conexión con el DID, el nonce y la clave del Steward (Verkey) el cual se lo preguntará al Ledger (libro mayor).

```
connection_response = json.dumps({
    'did': uoc['did_for_steward'],
    'verkey': uoc['key_for_steward'],
    'nonce': connection_request['nonce']
})
```

Ilustración 30: Respuesta de UOC

Esta respuesta se envía encriptada con la clave del Steward.

```
uoc['steward_key_for_faber'] = await did.key_for_did(uoc['pool'], uoc['wallet'], connection_request['did'])
anoncrypted_connection_response = await crypto.anon_crypt(uoc['steward_key_for_uoc'], connection_response.encode('utf-8'))
```

Ilustración 31: Petición de clave y firma

El Steward descencripta la información y hace una comparación del nonce para autenticar a la UOC.

```
decrypted_connection_response = \
    (await crypto.anon_decrypt(steward['wallet'], steward['key_for_uoc'], anoncrypted_connection_response)).decode("utf-8")

    assert connection_request['nonce'] == decrypted_connection_response['nonce']
```

Ilustración 32: Desencriptación y autenticación

El Steward manda una transacción NYM para el DID de la UOC al libro mayor. Aunque lo envíe el Steward el propietario del DID es la UOC, ya que, esta encriptado con la verkey de la UOC.

```
nym_request = await ledger.build_nym_request(steward['did'], decrypted_connection_response['did'], decrypted_connection_response['verkey'], None, role)
await ledger.sign_and_submit_request(steward['pool'], steward['wallet'], steward['did'], nym_request)
```

Ilustración 33: Envío de DID de la UOC

En este punto el Steward y la UOC tienen una conexión peer-to-peer segura. El DID creado anteriormente no es lo mismo que una SSI (self-sovereign identity). Ese DID sólo debe ser utilizado para conexiones con el Steward. Ahora UOC tendrá que crear un registro DID que le sirve para verificarse en el Ledger. Para ello, crea un nuevo registro DID con `did.create_and_store_my_did` y lo guarda en su cartera. UOC prepara un mensaje con su DID y su verkey. En este mensaje lo encripta y se lo envía al Steward.

```
uoc['did_info'] = json.dumps({
    'did': uoc['did'],
    'verkey': uoc['key']
})

authcrypted_uoc_did_info_json = \
    await crypto.auth_crypt(uoc['wallet'], uoc['key_for_steward'], uoc['steward_key_for_uoc, uoc['did_info'].encode('utf-8'))
```

Ilustración 34: Envío del nuevo DID a Steward

El Steward desencripta el mensaje y solicita al Ledger el verkey asociado al DID de la UOC. Comprueba que el verkey recibido por parte de la UOC y el obtenido desde el Ledger sean iguales.

```
sender['uoc_key_for_steward'], authdecrypted_uoc_did_info_json = \
    await crypto.auth_decrypt(steward['wallet'], steward['key_for_uoc'], authcrypted_uoc_did_info_json)
uoc_did_info = json.loads(authdecrypted_uoc_did_info_json)

steward['uoc_key_for_steward'] = await did.key_for_did(steward['pool'], steward['wallet'], ['uoc_did_for_steward'])

assert sender_verkey == steward['uoc_key_for_steward']
```

Ilustración 35: Comprobación del verkey

El Steward crear una transacción NYM con el rol Trust Anchor. En este punto, la UOC ya tiene un DID relacionado con su identidad en el Ledger.

```
nym_request = await ledger.build_nym_request(steward['did'], decrypted_uoc_did_info_json['did'],
                                             decrypted_uoc_did_info_json['verkey'], None, 'TRUST_ANCHOR')
await ledger.sign_and_submit_request(steward['pool'], steward['wallet'], steward['did'], nym_request)
```

Ilustración 36: Envío del NYM con el nuevo DID

En este paso crearemos el esquema de credenciales. Este esquema describe la lista de atributos que una credencial particular puede tener. Cualquiera con el rol de Trust Anchor puede crearlo. Anteriormente, se ha visto como el Steward le ha otrogado el rol Trust Anchor a UOC al enviar la transacción NYM. A continuación, la UOC creará un esquema de credenciales para el título universitario llamado title.

```
title = {
    'name': 'title',
    'version': '1.2',
    'attributes': ['first_name', 'last_name', 'degree', 'status', 'year', 'average', 'ssn']
}
(uoc['title_schema_id'], uoc['title_schema']) = \
    await anoncreds.issuer_create_schema(uoc['did'], title['name'], title['version'],
                                         json.dumps(title['attributes']))
title_schema_id = uoc['title_schema_id']
```

Ilustración 37: Definición de atributos del título

La UOC envía el esquema definido al Ledger con la transacción Schema llamando la función `ledger.build_schema_request` y para enviarlo utiliza `ledger.sign_and_submit_request`.

```
schema_request = await ledger.build_schema_request(uoc['did'], uoc['title_schema'])
await ledger.sign_and_submit_request(uoc['pool'], uoc['wallet'], uoc['did'], schema_request)
```

Ilustración 38: Envío del esquema de credenciales

Después de crear el esquema de credenciales, hay que crear la definición de la credencial. La UOC crea la definición para la credencial title pero cualquiera con el rol Trust Anchor podría crearlo. El primer paso, es solicitar el esquema de credenciales al Ledger.

```
get_schema_request = await ledger.build_get_schema_request(uoc['did'], title_schema_id)
get_schema_response = await ledger.submit_request(uoc['pool'], get_schema_request)
uoc['title_schema_id'], uoc['title_schema'] = await ledger.parse_get_schema_response(get_schema_response)
```

Ilustración 39: Solicitud del esquema al Ledger

Una vez obtenido el esquema, hay que crear la definición de la credencial. Se utiliza la función `anoncreds.issuer_create_and_store_credential_def` esto devuelve una parte pública de la definición de la credencial. La parte privada queda almacenada en la cartera.

```
title_cred_def = {
    'tag': 'TAG1',
    'type': 'CL',
    'config': {"support_revocation": False}
}
(uoc['title_cred_def_id'], uoc['title_cred_def']) = \
    await anoncreds.issuer_create_and_store_credential_def(uoc['wallet'], uoc['did'],
                                                           uoc['title_schema'], title_cred_def['tag'],
                                                           title_cred_def['type'],
                                                           json.dumps(title_cred_def['config']))
```

Ilustración 40: Creación de la definición del credencial

La UOC envía la definición creada al Ledger mediante las funciones `ledger.build_cred_def_request` y `ledger.sign_and_submit_request`.

```
cred_def_request = await ledger.build_cred_def_request(uoc['did'], uoc['title_cred_def'])
await ledger.sign_and_submit_request(uoc['pool'], uoc['wallet'], uoc['did'], cred_def_request)
```

Ilustración 41: Envío de la definición del credencial

En este momento, entra Xabi un exalumno de la UOC que quiere obtener su título universitario. Esa credencial tiene que estar sellada por alguna entidad que en este caso es la UOC de otro modo no tendría validez. Xabi y la UOC establecen una conexión. Una vez establecida la conexión la UOC le crea una oferta de credenciales a Xabi con su título.

```
uoc['title_cred_offer'] = await anoncreds.issuer_create_credential_offer(uoc['wallet'], uoc['title_cred_def_id'])
```

Ilustración 42: Oferta de la credencial title

Xabi quiere ver los atributos de la credencial title que le está ofreciendo la UOC. Estos atributos son conocidos porque anteriormente se ha enviado al Ledger el esquema de credenciales para title.

```
get_schema_request = await ledger.build_get_schema_request(xabi['did_for_uoc'], xabi['title_cred_offer']['schema_id'])
get_schema_response = await ledger.submit_request(xabi['pool'], get_schema_request)
title_schema = await ledger.parse_get_schema_response(get_schema_response)

print(title_schema['data'])
# title Schema:
{
    'name': 'title',
    'version': '1.2',
    'attr_names': ['first_name', 'last_name', 'degree', 'status', 'year', 'average', 'ssn']
}
```

Ilustración 43: Atributos de la credencial title

Xabi quiere usar esa credencial, por lo tanto, la tiene que solicitar. Para ello tiene que crear un Master Secret en su cartera. Un Master Secret es un elemento de datos privado usado para garantizar que esa credencial solo aplica a él. También, necesita obtener la definición de la credencial. Con estos datos Xabi puede realizar la solicitud del título.

```
xabi['master_secret_id'] = await anoncreds.prover_create_master_secret(xabi['wallet'], None)

get_cred_def_request = await ledger.build_get_cred_def_request(xabi['did_for_uoc'], xabi['title_cred_offer']['cred_def_id'])
get_cred_def_response = await ledger.submit_request(xabi['pool'], get_cred_def_request)
xabi['title_cred_def'] = await ledger.parse_get_cred_def_response(get_cred_def_response)

(xabi['title_cred_request'], xabi['title_cred_request_metadata']) = \
    await anoncreds.prover_create_credential_req(xabi['wallet'], xabi['did_for_uoc'], xabi['title_cred_offer'],
                                                 xabi['title_cred_def'], xabi['master_secret_id'])
```

Ilustración 44: Petición del title a la UOC

La UOC prepara los datos del título codificados y en plano para cada atributo en el esquema de credenciales. Lo rellena con los datos para Xabi.

```
title_cred_values = json.dumps({
    "first_name": {"raw": "Xabi", "encoded": "113948171645748869017221791627810333 5"},
    "last_name": {"raw": "Legarda", "encoded": "532164278024179012358790245678912345 2"},
    "degree": {"raw": "Computer Science engineering", "encoded": "12434523576212321"},
    "status": {"raw": "graduated", "encoded": "2213454313412354"},
    "ssn": {"raw": "123-45-6789", "encoded": "3124141231422543541"},
    "year": {"raw": "2018", "encoded": "2018"},
    "average": {"raw": "7", "encoded": "7"}
})

uoc['title_cred_def'], _, _ = \
    await anoncreds.issuer_create_credential(uoc['wallet'], uoc['title_cred_offer'], uoc['title_cred_request'],
                                             title_cred_values, None, None)
```

Ilustración 45: La credencial title de Xabi

Una vez recibida la credencial title sellada, Xabi lo guarda en su cartera para su posterior uso.

```
await anoncreds.prover_store_credential(xabi['wallet'], None, uoc['title_cred_request'], uoc['title_cred_request_metadata'],
                                        xabi['title_cred'], xabi['title_cred_def'], None)
```

Ilustración 46: Guarda la credencial en su cartera

Xabi quiere usar esa credencial, por lo tanto, la tiene que solicitar. Para ello tiene que crear un Master Secret en su cartera. Un Master Secret es un elemento de datos privado usado para garantizar que esa credencial solo aplica a él. También, necesita obtener la definición de la credencial. Con estos datos Xabi puede realizar la solicitud del título.

6. Conclusión

Hoy en día, es muy importante controlar la identidad digital. Hay que tener cuidado con lo que se publica y comparte, ya que, el mundo esta cada vez más conectado y esa información se puede distribuir muy rápidamente. Actualmente, prácticamente cualquier servicio se gestiona de forma digital. Para darse de alta en todos estos servicios es necesario dar los datos personales. El problema viene cuando se pierde la pista de los datos personales de cada uno y pasan a estar en manos de un tercero.

El sistema de gestión de identidades más extendido es el sistema centralizado. Por lo tanto, cuando un usuario se da de alta en un servicio sus datos pasan a estar alojados en los servidores de la empresa proveedora. Se ha demostrado en los últimos años como estas empresas no son capaces de mantener seguros los datos. Se han dado muchos casos como el robo de contraseñas a eBay o el sonado caso del portal de citas Ashley Madison. Debido a estas brechas de seguridad, han quedado expuestos todos los datos de los usuarios sin que ellos pudieran controlar nada. Más graves son aún los casos recientes de Facebook donde la compañía norteamericana ha vendido los datos personales de sus usuarios a terceras empresas. Se ha lucrado con datos de sus usuarios sin conocimiento de los propietarios reales y sin que estos hayan recibido nada a cambio.

Estos escándalos hacen necesario que haya que plantearse un cambio en el sistema actual de la gestión de identidades. Hay que buscar un sistema donde el propietario real de la información tenga el control sobre sus datos personales. De esta forma, cada uno decide que comparte y con quien comparte la información personal, sin que terceros se queden los datos en propiedad dentro de su infraestructura. Para ello, habría que implementar un sistema de gestión de identidades descentralizado.

Se ha demostrado que puede haber un sistema de descentralizado seguro, por ejemplo, con las criptomonedas. La más conocida el Bitcoin utiliza la tecnología blockchain que ha demostrado ser segura. Aun así, durante la realización de este trabajo, se ha visto que el uso de esta tecnología para la gestión de identidades esta aún muy poco desarrollada. Siendo la fundación Sovrin y el proyecto Hyperledger Indy los que más están apostando por una gestión de descentralizada de identidades abierto. Aunque Hyperledger tiene ya algunos proyectos funcionales como Hyperledger Fabric. Sin embargo, su solución dedicada para la gestión de identidades descentralizada está aún en la fase de incubación.

Se puedo concluir con la reflexión de que es necesario un cambio en la gestión de la identidad para mejorar la exposición que tienen los usuarios y que cada uno pueda controlar su información. Por otro lado, aún no hay soluciones que se puedan aplicar de forma sencilla a un entorno productivo, lo cual, dificulta que esta tecnología se pueda usar. Además, la información mueve muchos intereses y dinero y no ayuda que las grandes compañías apoyen su desarrollo. Hoy en día, la mayoría de estos proyectos son liderados por organizaciones sin ánimo de lucro.

7. Líneas futuras

El tema tratado durante el trabajo ofrece un amplio abanico posibilidades de cara al futuro, ya que, aún le queda mucho camino por recorrer.

Podría ser interesante intentar conectarse a la red de Sovrin Foundation y poder interactuar con otros nodos que pertenezcan a la red descentralizada. Esto permitiría hacerse una idea de cómo funciona un sistema descentralizado de gestión de identidades que está abierto y en funcionamiento.

Este proyecto deja abierto la posibilidad de crear alguna solución mediante la tecnología de Hyperledger Indy. Un paso importante para dominar esta librería sería crear una aplicación que sea accesible por usuarios externos, el cual, use un sistema descentralizado de identidades. Para ello, habría que explorar más las utilidades de la librería Hyperledger Indy y ver cómo podrían integrarse sus funcionalidades en una aplicación externa. Por ejemplo, llevar a la realidad la demo donde un alumno consigue un certificado emitido por una institución académica. Habría que tener en cuenta que el alumno no tendría por qué tener conocimientos técnicos. Esta solución es compleja, pero sería una forma de consolidar y llevar un caso real un sistema descentralizado de gestión de identidades.

Hay que tener en cuenta que Hyperledger Indy es un proyecto en incubación y que irá mejorando. Además, al no haber una solución estándar o predominante, seguro que salen otros proyectos para dar solución a este problema. Por lo tanto, habría que estar atento e investigar posibles nuevos proyectos, sobre todos si estos proyectos son de código abierto. De esta manera, pudiendo utilizar lo desarrollado en estos proyectos o sugerir posibles mejoras.

8. Bibliografía

Sovrin. (2018). *Home - Sovrin*. [online] Available at: https://sovrin.org/ [Accessed 31 Dec. 2018].

BanQu. (2018). *BanQu*. [online] Available at: https://banqu.co/ [].

Evernym. (2018). *Index - Evernym*. [online] Available at: https://www.evernym.com/

ID2020. (2018). *ID2020*. [online] Available at: https://id2020.org/

CoinDesk. (2018). *CoinDesk - Leader in blockchain news.*. [online] Available at: https://www.coindesk.com/

Medium. (2018). *Medium – a place to read and write big ideas and important stories*. [online] Available at: https://medium.com/ [Accessed 31 Dec. 2018].

Cointelegraph. (2018). *Latest Bitcoin, Blockchain and Ethereum news*. [online] Available at: https://cointelegraph.com/

Geth.ethereum.org. (2018). *Go Ethereum*. [online] Available at: https://geth.ethereum.org/

Hyperledger. (2018). *Hyperledger Indy – Hyperledger*. [online] Available at: https://www.hyperledger.org/projects/hyperledger-indy