

# Blockchain. Teoría e implementación.

Lucas Santander, Fernando Reynoso y Nicolás Maruyama

26 de Mayo 2023

# Prefacio

Este trabajo fue hecho por Lucas Santander, Fernando Reynoso y Nicolas Maruyama pertenecientes al curso de *6AO* en el *Instituto Industrial Luis A. Huergo*, para la materia de *Administracion de Sistemas y Redes*, cuyo profesor es *Ignacion Miguel Garcia*.

En este presente trabajo, abordaremos una descripcion detallada y ciertamente tecnica de una de las tecnologias mas revolucionarias del siglo, partiendo desde la base teorica hasta la implementacion practica y un paso a paso de como se puede implementar la Blockchain con Ethereum, MetaMask y Ganache.

Si bien reconocemos que el abordaje de temas técnicos puede resultar desafiante, estamos convencidos de que es fundamental comprender los aspectos fundamentales de Blockchain para apreciar plenamente su potencial. No obstante, nos esforzaremos por explicar de manera clara y accesible los conceptos más complejos, de modo que cualquier lector, independientemente de su nivel de experiencia previa.

A pesar de esto último, y teniendo en cuenta que este ensayo tiene como público objetivo tanto a nuestros compañeros como a nuestro profesor de la presente materia, asumimos que se posee un conjunto de conocimientos básicos los cuales comprenden: funcionamiento sobre criptografía básica y redes de computadoras.

# Contents

<b>Prefacio</b>	<b>i</b>
<b>1 Funcionamiento</b>	<b>1</b>
1.1 Introduccion . . . . .	1
1.2 Estructura . . . . .	2
1.3 Transacciones . . . . .	2
1.4 Validacion de Transacciones y problema del orden . . . . .	3
1.5 Prueba de trabajo . . . . .	3
1.6 Seguridad . . . . .	4
1.7 La red . . . . .	4
1.8 Incentivo a mineros . . . . .	5
<b>2 Implementación</b>	<b>6</b>
2.1 Explicacion . . . . .	6
2.2 Información sobre las tecnologías usadas . . . . .	6
2.2.1 Ethereum . . . . .	6
2.2.2 Smart Contracts . . . . .	7
2.2.3 Ganache . . . . .	7
2.2.4 MetaMask . . . . .	7
2.2.5 Remix . . . . .	7
2.3 Requisitos previos . . . . .	8
2.4 Set-up de Ganache y Metamask . . . . .	8
2.5 Remix, Metamask y deploy del contrato en Ganache . . . . .	10
2.6 Videos utiles para el set-up . . . . .	12
2.7 Como usar el programa . . . . .	12
<b>A Criptografia</b>	<b>13</b>
A.1 Hash . . . . .	13
A.2 SHA-256 . . . . .	14
<b>Bibliografia</b>	<b>15</b>

# Chapter 1

## Funcionamiento

### 1.1 Introduccion

*Blockchain*, creado por *Satoshi Nakamoto*, una entidad o conjunto de individuos cuya identidad permanece desconocida, representa un mecanismo revolucionario para almacenar y validar transacciones de manera descentralizada en una red de computadoras. La idea fue originalmente propuesta en el paper titulado *Bitcoin: A Peer-to-Peer Electronic Cash System* publicado el 31 de Octubre de 2008. Este innovador enfoque tecnológico ha adquirido una prominencia significativa en diversos campos, desde las criptomonedas hasta la gestión de datos y la seguridad digital.

A diferencia de los sistemas tradicionales basados en intermediarios o entidades centralizadas, el Blockchain opera en una red distribuida de nodos interconectados. Cada nodo de la red posee una copia completa del registro de transacciones, lo que garantiza la transparencia y la confiabilidad del sistema. Al descentralizar la gestión de las transacciones, se elimina la necesidad de un intermediario centralizado y se reducen los riesgos de fraude, corrupción o alteración de los datos.

Este enfoque descentralizado y seguro del Blockchain ha generado un interés generalizado en su aplicación en diversos ámbitos, incluyendo las transacciones financieras, la gestión de la cadena de suministro, los contratos inteligentes y la verificación de identidad. A medida que la tecnología Blockchain continúa evolucionando, se exploran nuevas soluciones y se amplían sus posibilidades, transformando la forma en que interactuamos y confiamos en los sistemas digitales.

El término Blockchain se deriva de la analogía con una cadena, ya que las transacciones se agrupan en bloques y se vinculan en secuencia mediante el uso de funciones criptográficas llamadas *hashes*. Cada bloque contiene un *hash* que identifica de forma única al bloque anterior, creando así una cadena inmutable y resistente a la manipulación.

## 1.2 Estructura

Blockchain es una forma de almacenar y validar transacciones de una forma descentralizada dentro de una red de computadoras. El nombre *Blockchain* viene de que el registro se asemeja a una cadena. Las transacciones se agrupan en bloques que se conectan a través del hash del bloque anterior. No hay un ente regulador que gestione las transacciones, por lo tanto, el registro se encuentra en todos los nodos de la red.

Cada bloque está compuesto por un *Header*, un *Body* y un *Nonce*.

- *Body*: contiene el registro de las transacciones. Los bloques tienen un límite de transacciones que pueden almacenar.
- *Header*: se encuentra el Hash producido por el algoritmo SHA-256 de todo el contenido del bloque anterior. Véase el índice 1 para una descripción del algoritmo SHA-256.
- *Nonce*: contiene la “prueba-de-trabajo”.

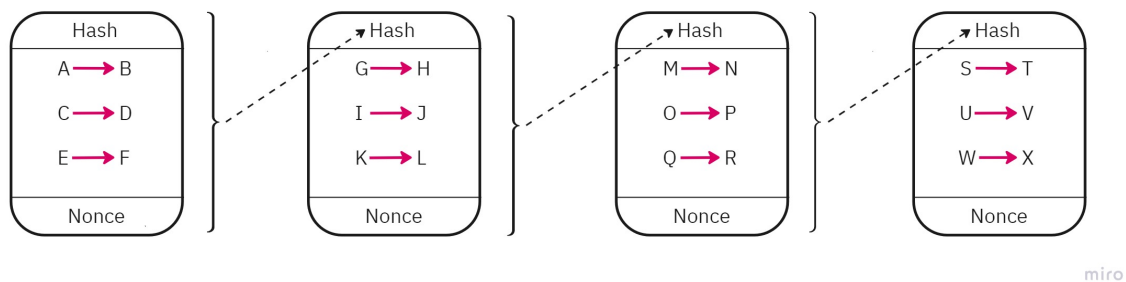


Figure 1.1: Estructura de la Blockchain

Podemos observar cómo se logra el efecto “cadena” a partir de que cada bloque almacena el Hash del bloque anterior.

## 1.3 Transacciones

Un coin es una cadena de firmas digitales. Cada dueño del coin firma el Hash de la transacción con su llave privada y la llave pública del siguiente dueño o payee, la persona a la que va a hacer la transacción.

El problema surge con que el próximo dueño no puede verificar si el dueño anterior no transfirió el coin dos veces. La primera transacción es la que cuenta, por eso no se van a tener en cuenta intentos después del primero de enviar el mismo coin.

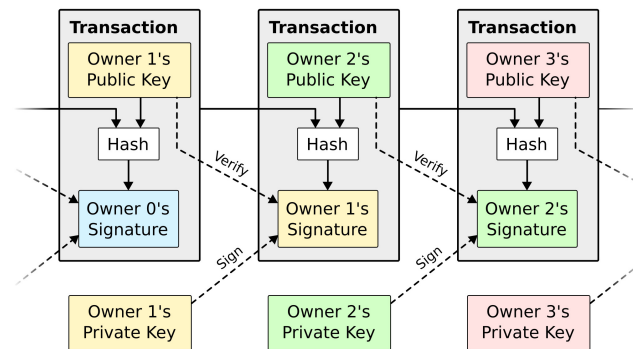


Figure 1.2: Transacciones

## 1.4 Validacion de Transacciones y problema del orden

Para evitar el *double-spending* de un coin, es necesario llegar a una decisión sobre cual transaccion se realizo primero. Para esto las transacciones deben ser públicas, es decir, cuando se realice una transacción se enviará a todos los nodos de la red.

Al ser una red descentralizada las transacciones pueden llegar en diferente orden a cada nodo y no saber cuál fue la primera transacción que se efectuó y, consecuentemente, la transacción lícita. Necesitamos un sistema para que los participantes se pongan de acuerdo en un solo historial del orden en que fueron recibidos.

Una solución al problema del orden podría ser *one-IP-address-one-vote*, es decir, un voto por dirección IP. Este sistema no sería bastante confiable ya que no es difícil simular muchas computadoras con diferentes direcciones IP con un mismo equipo; esto se conoce como *Sybil Attack*. El sistema utilizado es *one-CPU-one-vote*, es decir, un voto por computadora.

Los mineros son los encargados de votar que transacción se realizó primero y por ende, validarla. Un minero dentro de la red es una computadora que está recibiendo todas las transacciones que le llegan y agrupandolas en un bloque, cuando termina un bloque le “avisa” a los demás miembros de la red. Otro problema surge ya que, al ser una red descentralizada no hay una jerarquía en la red, es decir, todos los mineros tienen el mismo poder de decisión y por lo tanto podría haber mineros con bloques terminados pero con las transacciones en diferente orden. Para determinar qué bloque se agregara a la red, los mineros tendrán que resolver un “puzzle criptográfico” y el primero que lo resuelva será en que el que agregara el bloque a la cadena. Este “puzzle” se conoce como *prueba-de-trabajo* o *proof-of-work*.

## 1.5 Prueba de trabajo

El Hash de un bloque está determinado por las transacciones, por el Hash del bloque anterior y por el contenido del “nonce”. La *prueba-de-trabajo* es el uso de un algoritmo

de fuerza bruta para averiguar el contenido del “nonce” de tal manera que el Hash del bloque empiece con N cantidad de ceros. En la figura 3 se puede ver como funcionaria.

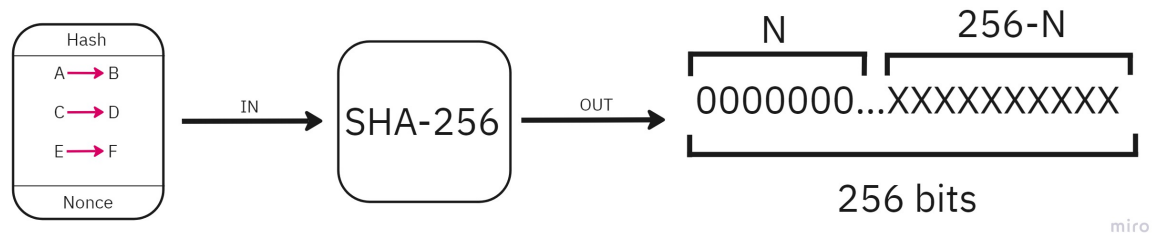


Figure 1.3: Prueba-de-trabajo

## 1.6 Seguridad

La seguridad de la Blockchain radica en que el Hash generado para ese bloque es único por lo que si se cambia el contenido de un bloque, por ejemplo, una transacción, el Hash cambiará también. Esto afectará a todos los bloques siguientes a partir del bloque cambiado, es decir, voy a tener que “rehacer” todos los bloques que se vieron perjudicados por efectuar ese cambio. Esto garantiza el orden de los bloques y la integridad de la cadena. Supongamos el hipotético caso de que un atacante cambie el contenido de un bloque, el mismo tendrá que rehacer ese bloque y todos los siguientes, es decir, tendrá que alcanzar el registro de todos los demás mineros de la red. La decisión mayoritaria está representada por la cadena más larga, que tiene el mayor esfuerzo de prueba de trabajo invertido en ella.

Incluso si esto se logra, no abre el sistema a cambios arbitrarios, como robar dinero o manipular transacciones que no le pertenecen. Los nodos no aceptarán una transacción no válida como pago, y los nodos honestos nunca aceptarán un bloque que los contenga. Un atacante solo puede intentar cambiar una de sus propias transacciones para recuperar el dinero que gastó recientemente.

## 1.7 La red

Los pasos para hacer funcionar la red son los siguientes:

- Las nuevas transacciones se transmiten a todos los nodos.
- Cada nodo recopila nuevas transacciones en un bloque.
- Cada nodo trabaja para encontrar una prueba de trabajo para su bloque.
- Cuando un nodo encuentra una prueba de trabajo, transmite el bloque a todos los nodos.

- Los nodos aceptan el bloque sólo si todas las transacciones en él son válidas y aún no se han gastado.
- Los nodos expresan su aceptación del bloque trabajando en la creación del siguiente bloque en la cadena, utilizando el hash del bloque aceptado como el hash anterior.

Los nodos siempre consideran que la cadena más larga es la correcta y seguirán trabajando para extenderla. Si dos nodos transmiten versiones diferentes del siguiente bloque simultáneamente, algunos nodos pueden recibir uno u otro primero. En ese caso, trabajan en el primero que recibieron, pero guardan la otra rama en caso de que se haga más larga. El empate se romperá cuando se encuentre la siguiente prueba de trabajo y una rama se vuelva más larga; los nodos que estaban trabajando en la otra rama cambiarán a la más larga.

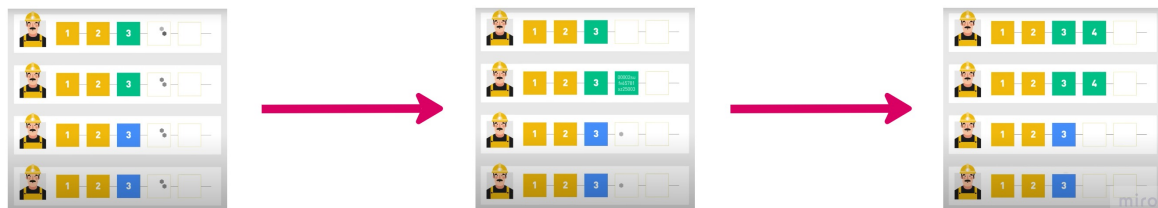


Figure 1.4: Desempate de ramas

## 1.8 Incentivo a mineros

Los mineros reciben una paga por minar bloques. Al eliminar un bloque el sistema agrega una transacción al minero al principio del bloque por ayudar a validar transacciones.



## Chapter 2

# Implementación

### 2.1 Explicacion

En esta implementación, creamos un programa que simula, de forma muy simplificada, un sistema de votación presidencial electrónico que hace uso de Blockchain. Las elecciones físicas, las que utilizan urnas, cuentan con muchos problemas, los más importantes son los que se relacionan con el fraude. Los votos pueden ser modificados, descartados o repetidos. Nuestro programa busca solucionar estos problemas con las medidas de seguridad que integran al protocolo Blockchain. La información de cada votante, su nombre, apellido, DNI y voto, representaría un bloque dentro del sistema. Y, mediante la lógica utilizada por los Smart Contracts, se aseguraría de registrar a cada ciudadano y de validar las condiciones para poder votar. De esta forma, la información de las elecciones sería completamente intocable e inmodificable por políticos corruptos o agentes maliciosos. Y permitiremos un paso más a una democracia más justa para todos los ciudadanos de la república. Cabe resaltar que esto únicamente es una simulación muy simplificada, y que en la vida real se debería contar con información muy importante, como son el registro de cada ciudadano en la nación para validar la información en su DNI, y se deberían implementar medidas de seguridad más sofisticadas.

### 2.2 Información sobre las tecnologías usadas

#### 2.2.1 Ethereum

Ethereum es una plataforma descentralizada de código abierto que permite a los desarrolladores construir y ejecutar aplicaciones descentralizadas (DApps) y contratos inteligentes. A diferencia de Bitcoin, que se centra principalmente en ser una moneda digital, Ethereum se diseñó para ser una plataforma que permite la creación de aplicaciones descentralizadas. Utiliza su propio lenguaje de programación llamado Solidity, que permite a los desarrolladores escribir contratos inteligentes y DApps en la cadena de bloques de Ethereum. Ethereum se basa en una tecnología de cadena de bloques similar

a Bitcoin, pero con características adicionales. La característica principal y distintiva de Ethereum es la capacidad de ejecutar contratos inteligentes.

### **2.2.2 Smart Contracts**

Es un programa informático autónomo que ejecuta automáticamente, verifica y hace cumplir los términos y condiciones de un contrato acordado entre dos o más partes. Estos contratos están basados en la tecnología de blockchain, que permite la creación de registros inmutables y transparentes.

Los Smart Contracts utilizan un lenguaje de programación específico para definir las reglas y condiciones del contrato. Una vez que se implementa en una plataforma blockchain, el contrato se ejecuta automáticamente cuando se cumplen las condiciones preestablecidas. Al estar basados en tecnología blockchain, los Smart Contracts son seguros y confiables, ya que se verifican y ejecutan de acuerdo con los principios de la cadena de bloques.

### **2.2.3 Ganache**

Ganache es un programa que proporciona un entorno de desarrollo y pruebas local para Ethereum. Es una herramienta desarrollada por Truffle Suite, un conjunto de herramientas populares para el desarrollo de contratos inteligentes en Ethereum. Ganache permite a los desarrolladores emular una red blockchain Ethereum local en sus propias máquinas sin necesidad de conectarse a la red principal de Ethereum. Proporciona una forma rápida y conveniente de desarrollar, implementar y probar contratos inteligentes y aplicaciones descentralizadas.

### **2.2.4 MetaMask**

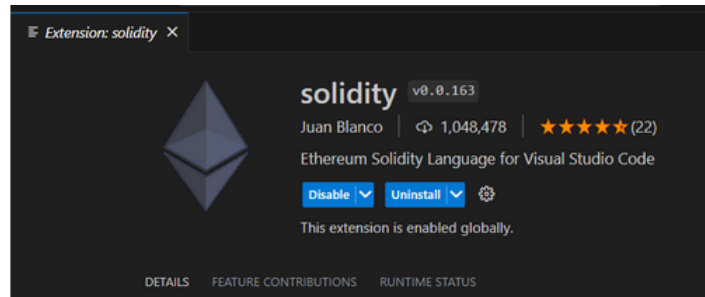
MetaMask es una billetera digital y una extensión de navegador que permite a los usuarios interactuar con aplicaciones descentralizadas basadas en la tecnología blockchain. Es una de las billeteras más populares y ampliamente utilizadas para Ethereum y otras redes blockchain compatibles. Es especialmente útil por permitir a sus usuarios conectar sus propias aplicaciones, como Ganache y Remix, y probar sus smart contracts en un entorno de transacciones simuladas.

### **2.2.5 Remix**

El entorno Remix es una plataforma en línea y un IDE (Integrated Development Environment) utilizado para desarrollar, compilar y depurar contratos inteligentes en Ethereum. Fue creado por Ethereum Foundation y es ampliamente utilizado por desarrolladores para construir y probar aplicaciones descentralizadas (DApps) en la red de Ethereum.

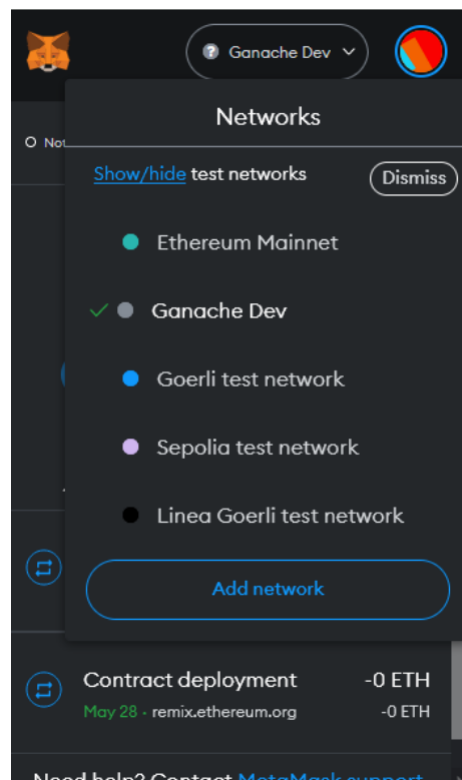
## 2.3 Requisitos previos

- Instalar MetaMask: <https://metamask.io/download/>
- Buscar la página de Remix: <https://remix.ethereum.org/>
- Instalar Ganache: <https://trufflesuite.com/ganache/>
- Instalar NodeJS: <https://nodejs.org/es>
- Instalar Visual Studio Code: <https://code.visualstudio.com/>
- Instalar extensión de Solidity en Visual Studio Code:

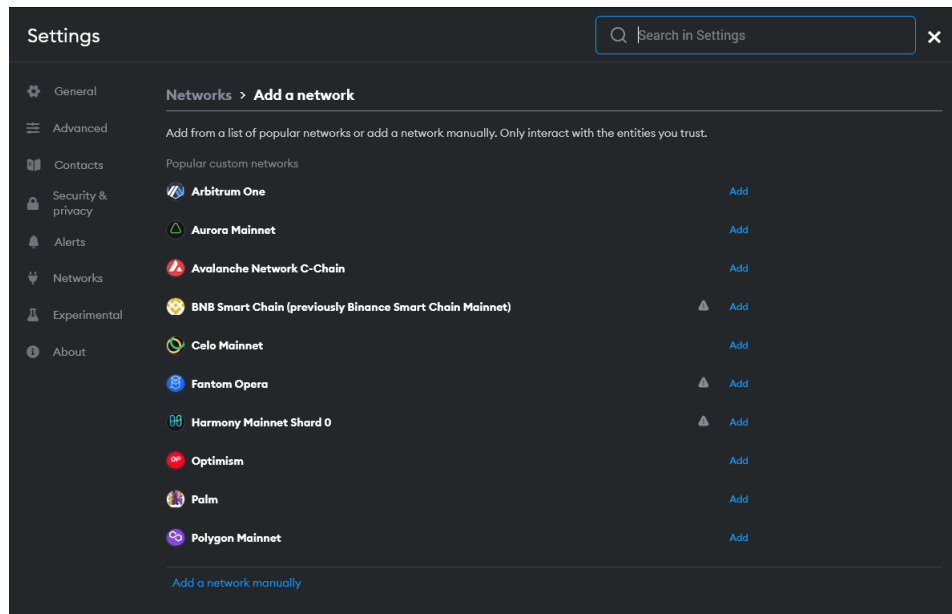


## 2.4 Set-up de Ganache y Metamask

1. Crear una cuenta en MetaMask.
2. Seleccionar la opción “Add network”.



### 3. Seleccionar “Add a network manually”.



### 4. Rellenar con los datos en Ganache.

Networks > Add a network > Add a network manually

**ⓘ** A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

**Network name**

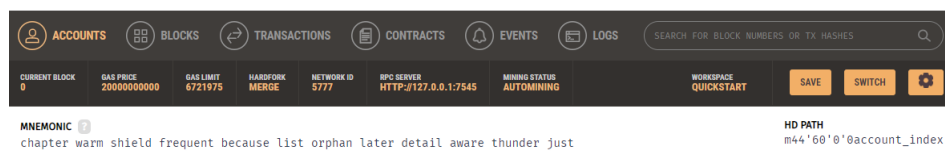
**New RPC URL**

**Chain ID ⓘ**



**Currency symbol**

**Block explorer URL (Optional)**

[Cancel](#) [Save](#)



### 5. Importar una cuenta, usar las que provee Ganache. Usamos la clave privada.

ADDRESS	BALANCE	TX COUNT	INDEX	
0x22e3EE0c7A3ba437f4187c2A32715a3A6Fd45618	100.00 ETH	2	0	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x77F1e7A679Ahd6FAe769701R1E7fdnddAE5f05	0.00 ETH	16	1	

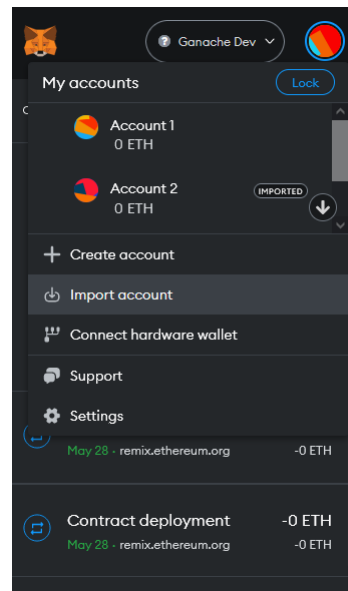
ACCOUNT INFORMATION

ACCOUNT ADDRESS  
0x43882AbB1A3921212Ba805CAA9c64FA91d400476

PRIVATE KEY  
0x0aff225fe96986f71d16d2ed0029e1d6b8721cca2dddf05f328737978fb3fd80

Do not use this private key on a public blockchain; use it for development purposes only!

DONE



Import account

Imported accounts won't be associated with your MetaMask Secret Recovery Phrase. Learn more about imported accounts [here](#).

Select Type Private Key

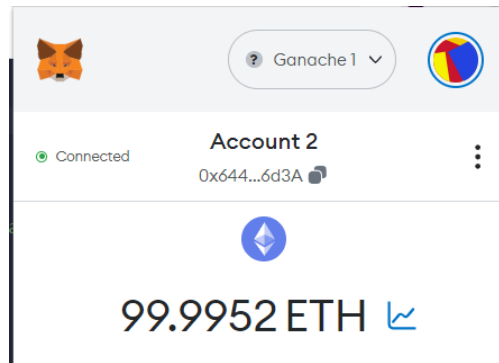
Enter your private key string here:

Cancel Import

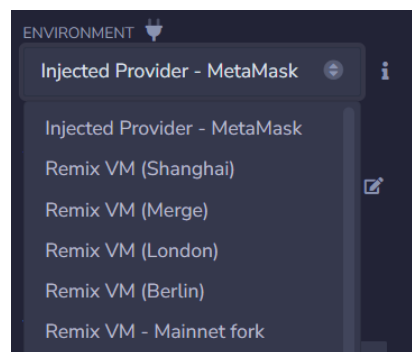
## 2.5 Remix, Metamask y deploy del contrato en Ganache

1. Subir el archivo "*SistemaVotacion.sol*" en la siguiente dirección a Remix.org:  
C:\[...]\Administracion-de-Sistemas-y-Redes-main\implementacion\src\contracts  
\SistemaVotacion\contracts

## 2. Conectar MetaMask con Ganache:

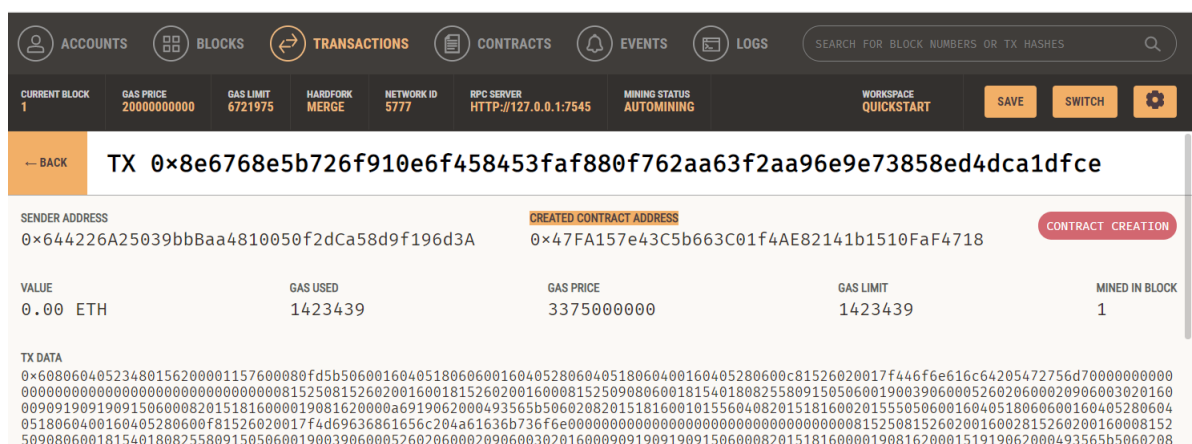


3. En la página de Remix, cambiar el Environment del código a *“Injected Provider - MetaMask”*. En caso de no aparecer, reiniciar la página y fijarse si MetaMask se encuentra conectado. En caso de pedir una url, usar la misma que Ganache.



4. Abrir Ganache y dirigirse a “*Blocks*”

5. Seleccionar un bloque y copiar la clave de “*CREATED CONTRACT ADDRESS*”.



6. Abrir el archivo *"logicaContrato.js"*, con Visual Studio Code, de la dirección:  
C:\[...]\Administracion de Sistema y Redes\Administracion-de-Sistemas-y-Redes-main\Administracion-de-Sistemas-y-Redes-main\implementacion\public\js

7. Modificar las variables “web3” y “addressContrato” con la página que usa Ganache y con la Contract Address que acabamos de copiar.

```
logicaContrato.js > [?] conectarContrato > [?] addressContrato
1  const conectarContrato = async () => {
2    const contratoVotacionJSON = await fetch("/contracts/SistemaVotacion.json").then(res => res.json());
3
4    const web3 = new Web3("http://127.0.0.1:7545");
5    const abi = contratoVotacionJSON.abi;
6    const addressContrato = "0x47FA157e43C5b663C01f4AE82141b1510FaF4718";
7    const contratoVotacion = new web3.eth.Contract(abi, addressContrato);
8
9    return contratoVotacion;
10
11    // const contratoVotacion = TruffleContract(contratoVotacionJSON);
12
13    // contratoVotacion.setProvider(window.ethereum);
14    // return await contratoVotacion.deployed();
15  };
```

## 2.6 Videos utiles para el set-up

- "PROGRAMA BLOCKCHAIN CON GANACHE" <https://youtu.be/JmFmUzJNdGI>
- "Deploying & Testing Smart Contract on Ganache" <https://youtu.be/TwY55ERlmdQ>

## 2.7 Como usar el programa

En la consola de comandos de Visual Studio Code, en el carpeta donde se encuentran los archivos del programa (carpeta implementación), ejecutar los siguientes comandos: “npm i” (para la primera vez) y “node src/server.js”. Buscar la siguiente dirección en su navegador de preferencia: “localhost:5000” o “http://[Mi dirección IP]:5000”. Seguir los pasos descritos en el programa.

# Appendix A

## Criptografia

### A.1 Hash

El funcionamiento de un *Hash* implica una serie de pasos y operaciones que se realizan sobre el mensaje o dato de entrada para generar el valor hash. A continuación, describiré los principales conceptos y procesos involucrados en el funcionamiento interno de un hash:

- Preparación del mensaje: Antes de aplicar el algoritmo de Hash, el mensaje o dato de entrada debe prepararse adecuadamente. Esto puede incluir la normalización del mensaje para eliminar espacios en blanco adicionales, convertirlo a un formato específico o dividirlo en bloques más pequeños, dependiendo del algoritmo de hash utilizado.
- División en bloques: En muchos algoritmos de hash, el mensaje se divide en bloques de tamaño fijo para facilitar el procesamiento. Si el mensaje no es un múltiplo del tamaño del bloque, se puede agregar un relleno específico al final para completar el último bloque.
- Iteración del algoritmo: El algoritmo de hash se aplica iterativamente a cada bloque del mensaje. Cada bloque se procesa independientemente y se combina con el resultado de los bloques anteriores.
- Operaciones de compresión: En cada iteración, el algoritmo de hash utiliza una serie de operaciones matemáticas, como operaciones lógicas, aritméticas y bit a bit, para comprimir la información del bloque actual y actualizar el estado interno del algoritmo. Estas operaciones pueden incluir XOR (OR exclusivo), AND, desplazamientos bit a bit, adiciones modulares, entre otros.
- Funciones no lineales (S-boxes): Muchos algoritmos de hash utilizan funciones no lineales, como *S-boxes* (cajas de sustitución), para introducir no linealidad en el proceso de hash. Estas funciones no lineales aplican transformaciones no reversibles a los datos y mejoran la seguridad del hash.
- Mezcla y permutación: Para garantizar la distribución uniforme de los datos en el



hash resultante, se aplican técnicas de mezcla y permutación. Estas operaciones reordenan y combinan los bits o bytes del estado interno del algoritmo para evitar patrones predecibles y asegurar que pequeños cambios en el mensaje original generen cambios significativos en el valor hash.

- Valor Hash resultante: Después de procesar todos los bloques del mensaje, el algoritmo de hash genera el valor hash resultante. Este valor es una cadena de bits de longitud fija, que suele ser de 128, 160, 256, 384 o 512 bits, dependiendo del algoritmo de hash utilizado.

En resumen, un hash funciona mediante la aplicación de algoritmos matemáticos y operaciones específicas para transformar el mensaje o dato de entrada en un valor hash de longitud fija. El proceso implica la división del mensaje en bloques, la iteración del algoritmo, el uso de operaciones de compresión, funciones no lineales, mezcla y permutación, entre otros. El resultado es un valor hash único que representa de manera segura el mensaje original.

## A.2 SHA-256

El algoritmo SHA-256 (Secure Hash Algorithm 256-bit) es un miembro de la familia de funciones hash criptográficas desarrollado por la Agencia de Seguridad Nacional (NSA) de los Estados Unidos. Su objetivo principal es calcular un resumen o hash de 256 bits a partir de una entrada de datos de cualquier longitud. Esta función hash es ampliamente utilizada en aplicaciones de seguridad, como la firma digital, la autenticación y la integridad de datos.

Paso a paso de cómo funciona el algoritmo SHA-256:

1. Preparación de la entrada: El mensaje original se divide en bloques de 512 bits. Si el mensaje no es un múltiplo de 512 bits, se agrega un padding para completar el bloque final. El padding consiste en un bit '1' seguido de ceros y luego se agrega la longitud del mensaje original en forma de un entero de 64 bits, lo que garantiza la integridad del mensaje.
2. Inicialización de valores iniciales: El algoritmo SHA-256 utiliza una matriz de constantes conocidas como "valores iniciales" o "valores hash intermedios". Estos valores se generan utilizando los primeros 32 bits de la parte decimal de las raíces cuadradas de los primeros 8 números primos.
3. Inicialización de variables de trabajo: Se inicializan ocho variables de 32 bits llamadas "variables de trabajo" (a, b, c, d, e, f, g, h) con los valores iniciales.
4. Procesamiento del mensaje por bloques: Para cada bloque de 512 bits del mensaje, se realiza el siguiente proceso:
  - Expansión del bloque: El bloque se divide en 16 palabras de 32 bits cada una.

- Extensión de palabras: Se extienden las 16 palabras a 64 palabras de 32 bits utilizando una función no lineal llamada "expansión de palabras". Esta función utiliza operaciones lógicas y aritméticas para generar las palabras adicionales.
  - Procesamiento principal: Se realiza un total de 64 rondas de operaciones en las palabras extendidas utilizando las variables de trabajo. Estas rondas involucran diversas operaciones, como rotaciones bit a bit, operaciones lógicas (AND, OR, XOR) y operaciones aritméticas (suma módulo  $2^{32}$ ). Cada ronda modifica las variables de trabajo y produce un nuevo valor hash intermedio.
  - Actualización de variables de trabajo: Después de cada ronda, se actualizan las variables de trabajo con los nuevos valores calculados.
5. Generación del hash final: Después de procesar todos los bloques del mensaje, se concatenan los valores finales de las variables de trabajo (a, b, c, d, e, f, g, h) para formar el hash final de 256 bits. Este valor representa de manera única el mensaje original.

Es importante destacar que el algoritmo SHA-256 es altamente resistente a la colisión, lo que significa que es muy improbable que dos mensajes diferentes produzcan el mismo hash. Además, es computacionalmente costoso revertir el proceso y encontrar el mensaje original a partir del hash, lo que lo hace adecuado para aplicaciones criptográficas.

# Bibliography

- [1] Nakamoto, Satoshi (2008, 31 de Octubre) *Bitcoin: A Peer-to-Peer Electronic Cash System*, Nakamoto Institute. <https://nakamotoinstitute.org/bitcoin/>
- [2] *SHA-256*. Wikipedia. <https://en.wikipedia.org/wiki/SHA-2>
- [3] *Sybil Attack*. Wikipedia. [https://en.wikipedia.org/wiki/Sybil\\_attack](https://en.wikipedia.org/wiki/Sybil_attack)
- [4] IMB *What is blockchain technology?* 3era edición, <https://www.ibm.com/topics/blockchain>
- [5] Santana Vega, Carlos (2021, 23 de Mayo) *HOY SÍ vas a entender QUÉ es el BLOCKCHAIN - (Bitcoin, Cryptos, NFTs y más)*, Dot CSV, <https://www.youtube.com/watch?v=V9Kr2SujqHw>