



## MÓDULO PROYECTO

Ciclo Superior Administración de Sistemas Informáticos y Red

Departamento: Informática y Comunicaciones

IES “María Moliner”

Curso: 2022/2023

Grupo: S2L

Proyecto: Blockchain

**Miguel Guerrero Martín**

Email: miguel.guemar@educa.jcyl.es

Tutor individual: María Lourdes Villoslada Rucio

Tutor colectivo: María Dolores de Miguel de Lucas

Fecha de presentación: 1 de junio de 202



## Índice

1. Identificación del proyecto .....	1
2. Descripción .....	1
2.1 Objetivos.....	1
2.2 Cuestiones metodológicas.....	1
3. Historia .....	2
4. Implementación.....	3
4.1 Nodos.....	3
4.2 Protocolo .....	3
4.3 Criptografía .....	3
4.4 Consenso.....	3
5. El Hash.....	4
6. El bloque.....	6
7. ¿Qué es Blockchain? .....	7
8. El Libro Mayor .....	8
9. Redes P2P .....	9
10. El minado.....	11
10.1 Rango del Nonce .....	13
10.2 Timestamp .....	13
11. Protocolos de Consenso.....	14
11.1 Prueba de trabajo.....	16
11.2 Prueba de participación.....	16
11.3 Prueba de actividad.....	17
11.4 Prueba de quemado .....	17
11.5 Prueba de capacidad .....	17
12. Ataque del 51%.....	18
13. Claves públicas y privadas .....	19
14. Smart Contracts .....	20
15. Ventajas e inconvenientes de Blockchain .....	21
16. Aplicaciones.....	22
17. Ejemplo práctico .....	23
17.1 Prerrequisitos.....	23
17.2 Creación de una Blockchain en Python .....	24

17.3 Ejemplo simple en Python .....	28
17.4 Aplicación web para un sistema de votación .....	33
18. Conclusión .....	46
19. Anexos .....	47
20. Bibliografía .....	49

## 1. Identificación del proyecto

Mi proyecto está basado en comprender y explicar una tecnología actual y en auge como es “Blockchain”. También se detallará brevemente su historia, conceptos relacionados, ventajas e inconvenientes y se realizará una prueba práctica de implementación.

## 2. Descripción

### 2.1 Objetivos

El objetivo principal es poder aplicar los conceptos explicados y comprendidos durante los dos cursos del ciclo. Además de este objetivo principal, se han establecido dos objetivos secundarios.

- El primero es lograr una comprensión clara y detallada del funcionamiento de la tecnología, así como de sus posibles aplicaciones, ventajas y desventajas.
- El segundo es adquirir los conocimientos necesarios para poder llevar a cabo una prueba práctica del proyecto.

### 2.2 Cuestiones metodológicas

Para alcanzar los objetivos propuestos, es necesario realizar una exhaustiva investigación y analizar los distintos aspectos relacionados con Blockchain. La investigación será realizada mediante la lectura de artículos, documentos y estudios relacionados con este tema.

Una vez analizada la información obtenida, se deberá estudiar que herramientas y utilidades se necesitan para implantar esta tecnología.

Por último, se intentará realizar una práctica con los conocimientos adquiridos mediante las fases anteriores.

### 3. Historia

Blockchain aparece por primera vez como concepto en un documento del año 1991 llamado “*How to Time-Stamp a Digital Document*”, redactado por Stuart Haber y W. Scott Stornetta. El artículo se considera uno de los trabajos fundacionales en el campo de la criptografía y que sentó las bases teóricas para la creación de esta tecnología.

En el artículo se aborda el problema de la integridad de los documentos digitales y cómo garantizar que un documento no haya sido modificado después de su creación. Lo que se propone es la utilización de criptografía de clave pública para crear una cadena de bloques (**Blockchain**) en la que cada bloque incluye una marca de tiempo (Timestamp) y un resumen criptográfico de los bloques anteriores.

Sin embargo, en el año 2009 bajo el seudónimo de Satoshi Nakamoto, se publicaría un documento técnico llamado “*Bitcoin: A Peer-to-Peer Electronic Cash System*” que proponía una nueva forma de crear y transferir dinero digitalmente sin la necesidad de un intermediario centralizado.

Dicha propuesta, se basaba en una base de datos descentralizada y distribuida, en la que cada transacción era verificada y registrada por los usuarios de la red en lugar de un tercero confiable (bancos). A esta base de datos se denominó Blockchain debido a que las transacciones se agrupan en bloques y se encadenan de forma secuencial.

Desde entonces, el uso de esta tecnología se ha expandido a otros ámbitos, como la creación de sistemas de votación seguros, cadenas de suministros transparentes, registros de propiedad, patentes...

Se espera que, en el futuro, Blockchain sea una de las tecnologías más importantes y que su uso continúe expandiéndose.

En 2015, Vitalik Buterin fundó “Ethereum”, la primera plataforma programable de Blockchain. Ethereum sirve de base para que desarrolladores puedan desarrollar aplicaciones distribuidas.

Estos dos últimos documentos han sido pilares fundamentales para el desarrollo de Blockchain.

## 4. Implementación

Para que una red Blockchain pueda ser implementada, debe estar compuesta por los siguientes elementos:

### 4.1 Nodos

Son ordenadores conectados a la red con la capacidad de recibir y enviar información. Al ser una red Peer 2 Peer (P2P), los nodos no necesitan un servidor centralizado para validar y registrar transacciones. En cambio, son los propios nodos los que validan y confirman las transacciones, además de participar en el mantenimiento de la base de datos. Para que se pueda establecer una comunicación entre los nodos se debe utilizar un protocolo.

### 4.2 Protocolo

Es un conjunto de convenciones para establecer una comunicación entre ordenadores. El protocolo de comunicación más utilizado es TCP/IP el cual consiste en un mecanismo de transporte para entregar datos entre dispositivos o entre ordenadores y terminales.

### 4.3 Criptografía

Se ocupa de los procedimientos para cifrar la información. Cumple un papel fundamental para evitar la falsificación o manipulación de la misma, además de ser una garantía de privacidad. Para la criptografía se utiliza una función hash.

### 4.4 Consenso

Para que la red sea funcional, todos los nodos participantes deben confiar en la información que se encuentra grabada en Blockchain. La implementación de algoritmos de consenso proporciona seguridad y fiabilidad a la red.

## 5. El Hash

Un hash es una función criptográfica unidireccional que toma como entrada un dato (sea un archivo, texto, imagen...) y lo convierte en una cadena de caracteres de longitud fija. Dicha cadena representa una “huella” digital y única para cada dato. El más utilizado, en el ámbito de Blockchain y de la criptografía, es el **SHA-256**. Esta función, creada por la Agencia de Seguridad Nacional (NSA) y el Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos, genera una cadena de 64 caracteres hexadecimales (256 bits) única para cada dato introducido.



Ilustración 1: Función hash

El hash utilizado para nuestra red debe cumplir los siguientes requisitos:

- Permitir ser ejecutado por cualquier tipo de dato, tamaño y formato.
- Ser determinista, es decir, para la misma entrada siempre debe proporcionar la misma salida.

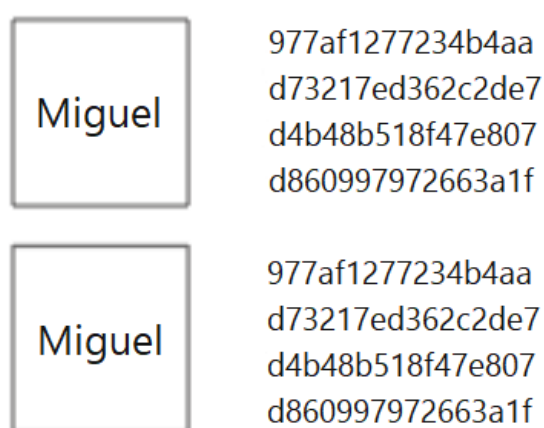
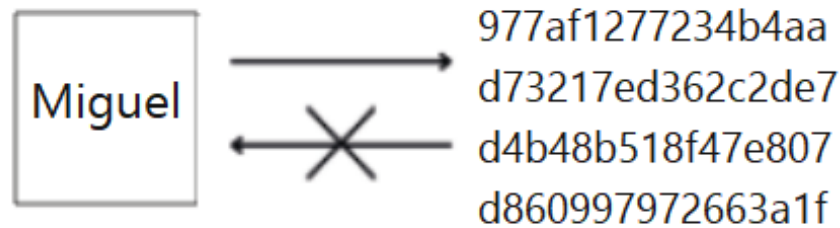


Ilustración 2: Función hash determinista

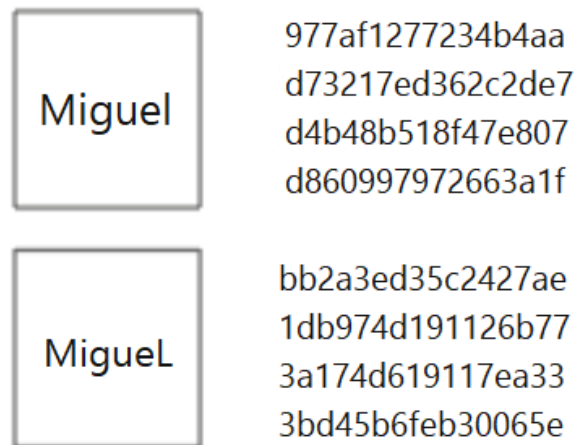


- Ser unidireccional, es decir, una entrada obtiene una salida, pero de la salida no se puede obtener la entrada.



*Ilustración 3: Función hash unidireccional*

- Efecto “cascada”, es decir, a partir de una pequeña variación en la entrada se produce un hash completamente distinto.



*Ilustración 4: Función hash cascada*

- Rápida ejecución, el cálculo del hash debe ser extremadamente rápido, sin importar el tamaño de la entrada.
- Soporte de colisiones. Puede suceder que dos archivos distintos generen una misma salida. El hash SHA-256 puede generar  $2^{256} \approx 1.1579 \cdot 10^{77}$  salidas

## 6. El bloque

En inglés “Block”, es un registro en cuyo interior se almacena información. Cada bloque, además de la información, contiene un hash propio y un hash del bloque anterior, relacionándose ambos bloques. Esta relación es lo que se conoce como Blockchain.

Primero debemos generar un bloque. El primer bloque es llamado “**Bloque Génesis**”. A partir de él, se añaden los siguientes bloques de la cadena.



Ilustración 5: Bloque Génesis

Para añadir un segundo bloque, añadimos el hash del primer bloque. Así se mantendrán los bloques enlazados criptográficamente de tal manera que, si uno es modificado, cambiaría el valor de los siguientes bloques, detectando así que ha habido una alteración en la cadena.

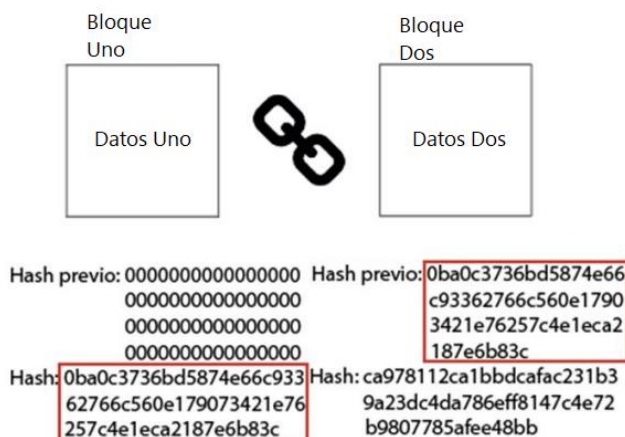


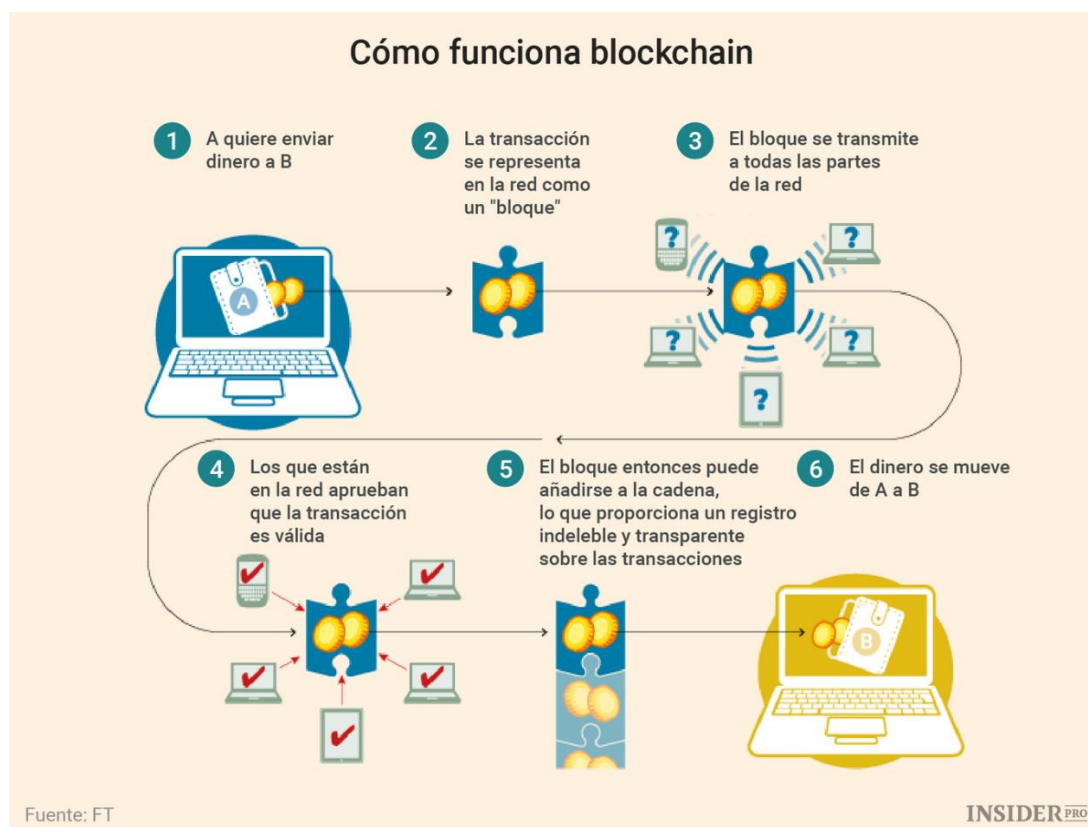
Ilustración 6: Cadena de bloques

## 7. ¿Qué es Blockchain?

En español “Cadena de bloques”, es una tecnología que se utiliza para guardar información de manera segura e inmutable en una base de datos que está distribuida en muchos lugares de la red (nodos) al mismo tiempo. La información se guarda de forma que ningún usuario la puede cambiar ni falsificar porque está protegida con claves criptográficas fuertes. Cada parte de información se guarda en bloques que están unidos entre sí, formando una cadena.

Cada Blockchain necesita tener varios nodos que se encarguen de verificar cada modificación del libro para validarlas y registrarlas.

Esta tecnología es utilizada principalmente en la economía (criptomonedas), pero también puede ser aplicada en diversos campos, como la gestión de identidad, la gestión de cadenas de suministro, la votación electrónica, etc.



*Ilustración 7: Ejemplo de funcionamiento Blockchain en la economía*

## 8. El Libro Mayor

En inglés “Ledger”, es una base de datos donde se registran todas las operaciones o transacciones producidas en el Blockchain, desde su origen (bloque génesis), hasta la actualidad y se va actualizando a tiempo real a medida que se producen nuevas transacciones.

Cada transacción validada por los nodos, se agrega de manera permanente al Libro Mayor y no puede ser modificada sin el consenso de la red, lo que garantiza la integridad de la información. Este hecho convierte al Ledger en uno de los elementos fundamentales para la transparencia, seguridad y privacidad de la cadena de bloques.

El concepto de Libro Mayor se lleva aplicando desde el origen de la escritura para almacenar información como la producción, el dinero, las deudas... El cambio más significativo se produjo en el siglo XIX, de manera que estos libros pasaron a estar centralizados por las grandes burocracias.

Un siglo más tarde, los Ledger sufren otra evolución, la digitalización. Hoy en día, estos libros son bases de datos o grandes ficheros centralizados.

Blockchain ofrece la descentralización de las bases de datos y evita la dependencia de una autoridad para mantener en funcionamiento y la seguridad debido a que el Ledger es distribuido, duplicado y almacenado en cualquier nodo de la red. Como debe existir un consenso de toda la red para modificar el Libro Mayor, se evita que una sola entidad pueda poseer el control absoluto de éste. La seguridad de la red aumenta cuando existen un gran número de nodos y cuantos más bloques contenga la cadena.

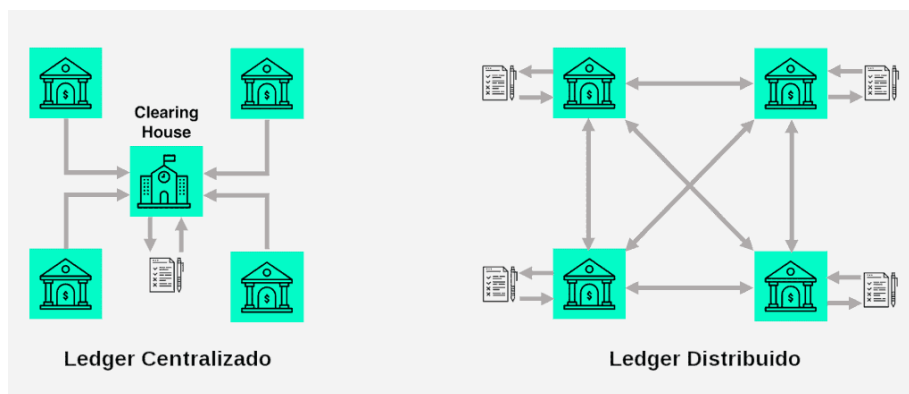
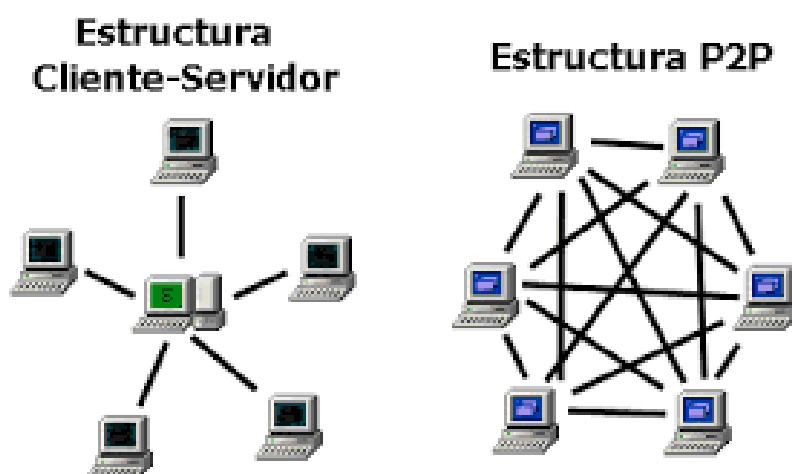


Ilustración 8: Ledger centralizado y distribuido

## 9. Redes P2P

En la década de 1980, las redes informáticas se basaban en una estructura cliente-servidor, en la que un servidor centralizado controlaba el acceso y la distribución de la información. Sin embargo, en 1989, el ingeniero de computación Tim Berners-Lee creó la World Wide Web, que permitió a los usuarios acceder a la información a través de un sistema distribuido.

Las redes “Peer to Peer”, en castellano redes entre iguales, se hicieron populares a partir del año 1999, cuando el programador Shawn Fanning creó **Napster**, un programa de intercambio de archivos que utilizaba una red P2P para permitir compartir música entre los usuarios sin el permiso de los autores, lo que acarreó numerosos problemas legales.

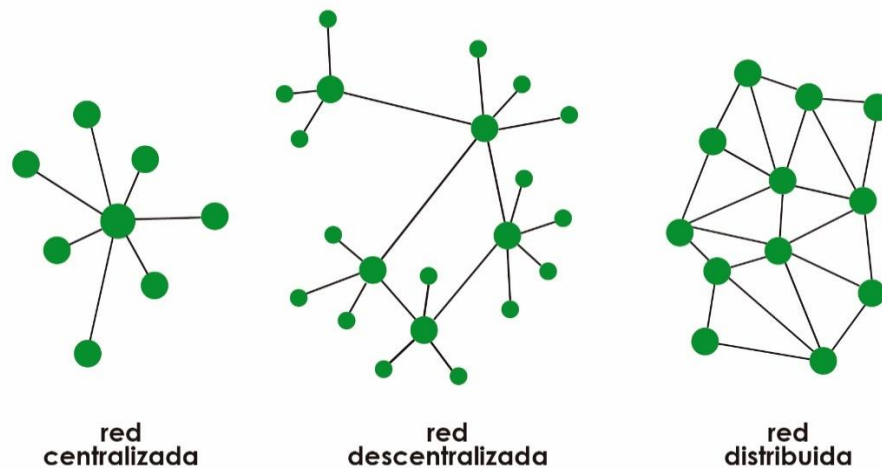


*Ilustración 9: Cliente-Servidor y P2P*

Existen distintos tipos de redes P2P:

- Redes descentralizadas y estructuradas (P2P híbridas): En este tipo de redes no existe un único servidor centralizado que recibe las peticiones. La principal debilidad del sistema centralizado es la falta de disponibilidad debido a que, si falla el nodo principal, nadie puede seguir recibiendo información. Para solucionar el anterior problema, este tipo de redes añaden una serie de nodos que tienen la capacidad de recibir las peticiones, distribuyéndoselas entre ellos. Estos nodos son otros servidores o incluso ordenadores personales. La disponibilidad de la red es directamente proporcional al número de nodos.

- Redes descentralizadas y no estructuradas: En este tipo de redes no existen servidores. Todos los nodos conectados a la red son tratados del mismo modo y poseen las mismas funcionalidades. Esto provoca que exista el mismo número de servidores que de clientes, permitiendo conexiones desde cualquier equipo a cualquier equipo, aumentando la velocidad de respuesta.



*Ilustración 10: Diferencias entre redes P2P*

La descentralización ofrece las siguientes ventajas

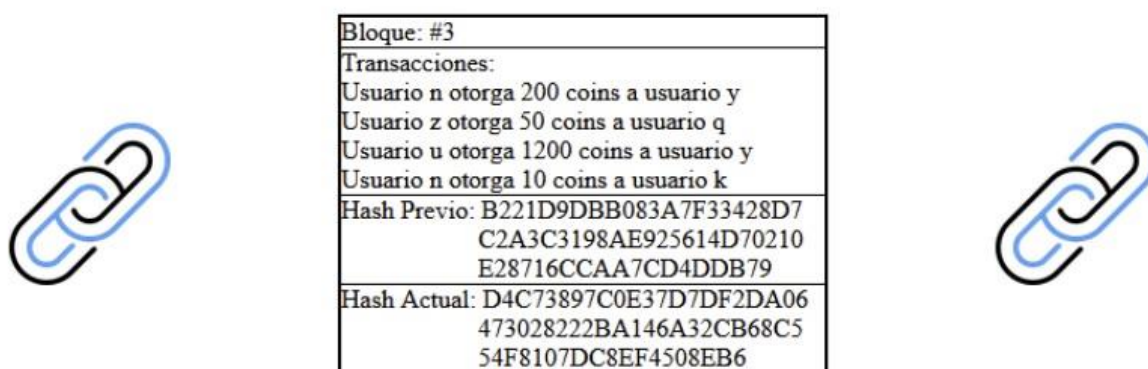
- Tolerancia a los fallos: Este tipo de sistemas tienen una probabilidad menor a fallar porque estadísticamente, es más probable que se caiga un servidor a que se caigan varios nodos de una red. A mayor número de nodos, menor probabilidad de errores.
- Resistencia a ataques: Los sistemas descentralizados son más costosos de atacar, destruir o manipular porque no poseen un único punto para atacar.
- Resistencia a alianzas: Es más complicado que los participantes se pongan de acuerdo para actuar de forma que se beneficien en contra de los otros participantes. En un sistema centralizado, es más probable que un grupo reducido de personas, como la dirección de una empresa, pueda actuar en su beneficio.

Blockchain no es más segura por el hecho de utilizar una red P2P. El potencial reside en la unión de estas dos tecnologías para realizar una distribución del Libro Mayor por todos los nodos de la red, convirtiéndose en una base de datos distribuida descentralizada.

## 10. El minado

Los conceptos de “minado” y “minero” se han hecho muy populares debido a la falta de componentes, mayoritariamente tarjetas gráficas, que se ha sufrido durante los últimos años.

Un bloque está formado por su número de bloque (index), los datos que contiene, el hash del bloque anterior y su propio hash calculado a partir de los parámetros anteriores.



Bloque: #3
Transacciones:
Usuario n otorga 200 coins a usuario y
Usuario z otorga 50 coins a usuario q
Usuario u otorga 1200 coins a usuario y
Usuario n otorga 10 coins a usuario k
Hash Previo: B221D9DBB083A7F33428D7 C2A3C3198AE925614D70210 E28716CCAA7CD4DDB79
Hash Actual: D4C73897C0E37D7DF2DA06 473028222BA146A32CB68C5 54F8107DC8EF4508EB6

Ilustración 11: Composición de un bloque

Debido que el cálculo del hash debe ser prácticamente inmediato, es necesario añadir un nuevo elemento al bloque denominado **Nonce**. El “number that can be only used once”, número que solo puede usarse una vez, es un número aleatorio y de características únicas para utilizarse en sistemas criptográficos. El intento de hallazgo de este número es el causante principal de la minería.

Debido a que el hash es irreversible, la única forma de adivinar el Nonce es realizar pruebas de fuerza bruta. Este hecho causó el gran desabastecimiento de tarjetas gráficas debido a que es el hardware que mejor realiza esta tarea.

La función de los mineros es encontrar el Nonce válido con el que el hash cumpla los requisitos, ya que es el único valor del bloque que puede tomar distintos valores. La información y el hash del bloque anterior deben permanecer intactos, pues esa es la función principal de Blockchain.

Las cadenas de bloques marcan un objetivo que facilitan el hallazgo del Nonce válido, denominado **Golden Nonce**.



Supongamos que el hash que necesitamos encontrar para añadir un nuevo bloque debe cumplir el siguiente objetivo: comenzar por 20 ceros desde la parte izquierda. Para encontrar este valor, se debe realizar mediante fuerza bruta como observamos en el siguiente ejemplo.

Para el primer intento, generamos un número aleatorio y probamos a realizar el hash.

El hash producido es superior al objetivo, por tanto, el Nonce no es válido y debemos probar con otro número.

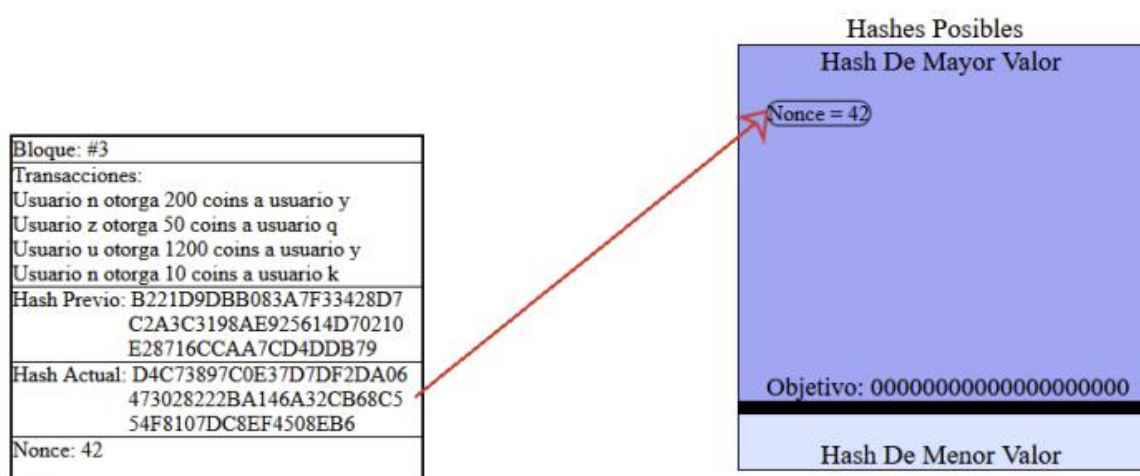


Ilustración 12: Nonce, primer intento

En el segundo intento, el hash producido sigue siendo mayor al objetivo, por lo que sigue sin ser válido.

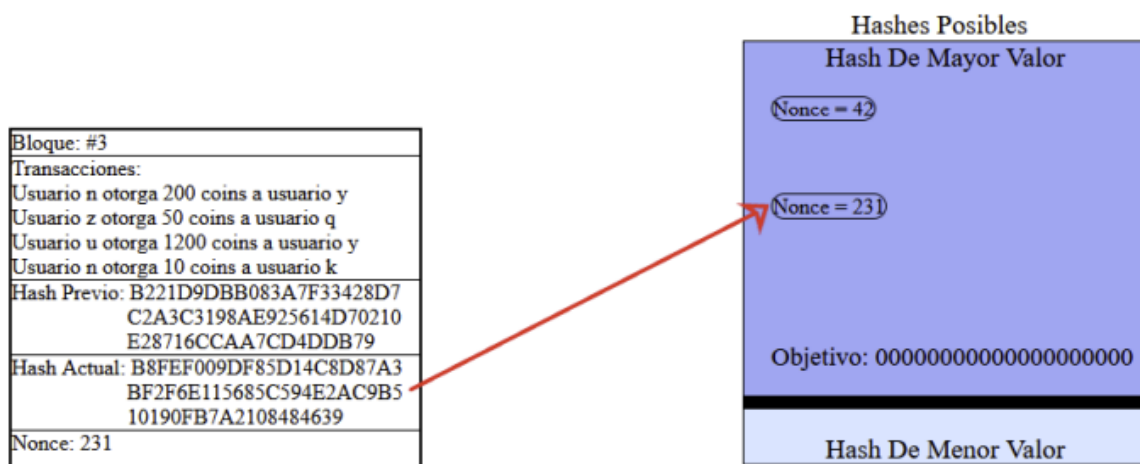


Ilustración 13: Nonce, segundo intento



Se prueba con distintos Nonces hasta que se obtiene un hash que sí cumple con el objetivo. Ahora el bloque se considera “minado” y es añadido a la cadena.

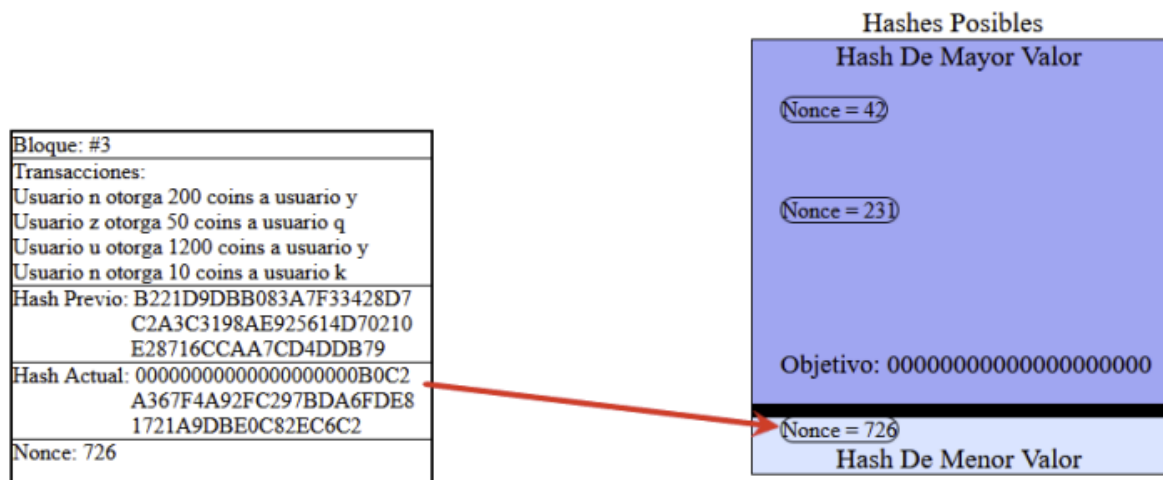


Ilustración 14: Obtención del Golden Nonce

## 10.1 Rango del Nonce

El valor del Nonce no puede ser cualquiera, se suele utilizar un número de 32 bits sin signo, es decir, desde el 0 hasta el 4.294.967.295, ambos incluidos.

Aunque parece una gran cantidad de números, los dispositivos que se encargan de encontrarlo poseen una gran potencia computacional. Un minero con la tecnología actual puede encontrar este número en pocos segundos.

Por esta razón, se añade un nuevo campo al bloque, el **Timestamp**.

## 10.2 Timestamp

El Timestamp es la fecha exacta en el que un bloque está siendo minado. En el Blockchain se utiliza el Unix Timestamp, que cuenta los segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC, por lo que cada segundo, el Timestamp cambia, modificando el resultado del hash.

```
miguel@MiguelPC:~$ date +%s
1684348303
miguel@MiguelPC:~$
```

Ilustración 15: Unix Timestamp en Linux

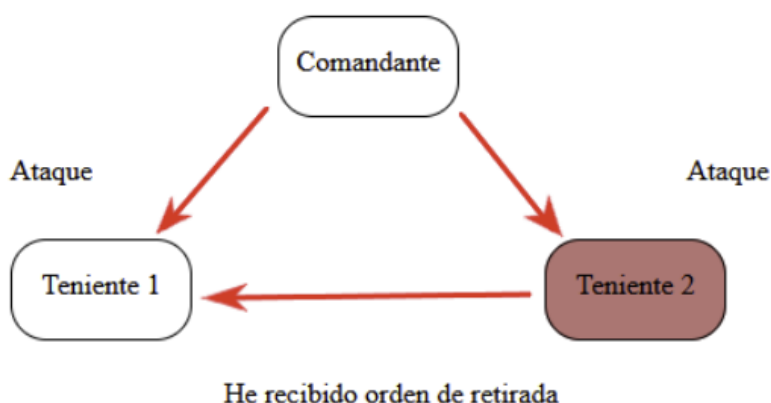
## 11. Protocolos de Consenso

En casi todos los sistemas distribuidos, los nodos pertenecientes a la red deben acordar periódicamente el estado de la Blockchain. Esta acción es realizada mediante consenso de la mayoría de los nodos. Lograr este consenso no es una tarea sencilla de realizar.

A la hora de acordar una decisión mayoritaria, se pueden producir fallos en algunos nodos o incluso que actúen de manera malintencionada. A esto se denomina “problema de los generales bizantinos”.

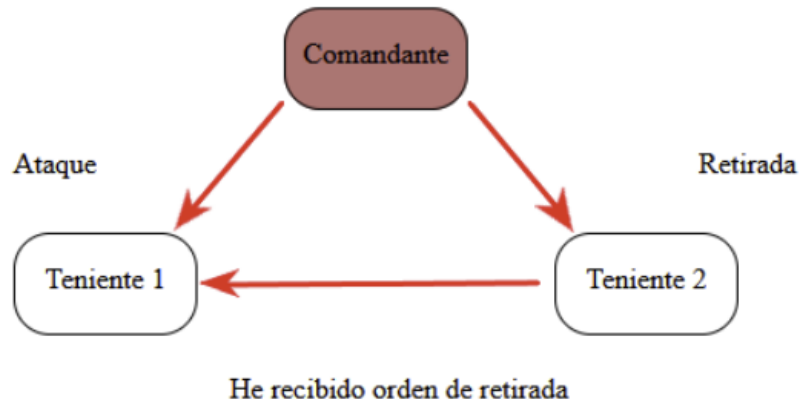
Fue concebido en 1982 por los matemáticos Leslie Lamport, Robert Shostak y Marshall Pease. Este problema se plantea en un escenario donde varios generales bizantinos están ubicados alrededor de una ciudad enemiga y necesitan coordinar un ataque o retirada conjunta. Sin embargo, algunos de los tenientes del ejército pueden enviar información incorrecta o actuar de manera malintencionada. El desafío es cómo pueden los generales llegar a un acuerdo sobre una acción mayoritaria cuando existe la posibilidad de que algunos generales estén dando información falsa o tengan intenciones maliciosas.

En esta primera situación, el comandante es leal y envía una orden de ataque, pero el teniente 2 es un traidor e informa al teniente 1 de que ha recibido la orden de retirada. El teniente 1 debería seguir la orden del comandante, pero al recibir órdenes contradictorias, no existe un consenso de la orden.



*Ilustración 16: Situación 1 del Problema Bizantino*

En este segundo caso, el comandante es un traidor, por lo que conseguir un consenso es imposible a menos que alguno de los tenientes otorgara otra orden por error o por otra razón que el comandante desconoce.



*Ilustración 17: Situación 2 del Problema Bizantino*

En nuestra cadena de bloques, cada teniente representa un nodo de la red. Los nodos deben alcanzar un consenso sobre el estado actual del sistema, es decir, la mayoría de los participantes de la red tienen que estar de acuerdo y ejecutar la misma acción para evitar un fallo en la cadena.

Para evitar este problema, surgen los protocolos de consenso teniendo como objetivo mantener a la cadena de bloques en correcto estado. Las funciones principales de estos algoritmos de consenso son dos:

- Garantizar que el siguiente bloque es la única versión correcta.
- Evitar que numerosos ciberdelincuentes desvíen o bifurquen la cadena añadiendo bloques corrompidos.

Existen numerosos algoritmos de consenso. Los más conocidos son:

### 11.1 Prueba de trabajo

Proof of Work (PoW), es el algoritmo más utilizado en la actualidad, aunque en un futuro puede disminuir su uso.

El fundamento principal de este método es realizar un trabajo duro para poder minar un bloque, pero siendo sencillo de validar.

El algoritmo funciona cuando se intenta añadir un nuevo bloque, evitando que el nuevo bloque haya sido falsificado o que la cadena tenga distintas versiones.

Para evitar un bloque corrupto, el algoritmo realiza una serie de pruebas para ver que el contenido del bloque no ha sido modificado. En el caso de que sí exista modificación, el nodo responsable se quedará sin recompensa y será expulsado de la red.

Otro caso que puede ocurrir es que más de un minero consigan el Golden Nonce en un período de tiempo muy corto, no permitiendo avisar al resto de nodos de la red y produciendo dos versiones de la cadena. Para solucionarlo, el algoritmo deja ambos bloques en la cadena y espera a la inserción de un nuevo bloque. Una vez que se mina el siguiente bloque, se basa en uno de los bloques candidatos anteriores. La cadena que incluye este nuevo bloque se considera válida y la otra se descarta.

Por último, cada vez que un nodo distribuya un nuevo bloque lo único que deberán hacer los nodos de la red será verificar que, al introducir el Nonce, obtenemos un hash que esté dentro del objetivo. Una vez que la mayoría de la red validen el bloque, éste será añadido a la cadena.

### 11.2 Prueba de participación

Es la alternativa principal al anterior algoritmo.

Proof of Stake (PoS) evita tener que invertir en hardware para ser el primero en minar bloques, si no que se elige un “validador”. Dicho validador debe poseer una cierta cantidad de la criptomoneda asociada y un determinado tiempo de participación en la red.

El algoritmo elige a varios validadores de forma aleatoria para crear y validar un nuevo bloque. El validador deberá comprobar que el bloque es correcto, si acepta un bloque malicioso, será expulsado de la red.

Este algoritmo requiere una menor cantidad de energía y es más rápido que el PoW, por ello se prevé que su uso siga aumentando.

### 11.3 Prueba de actividad

Proof of Activity (PoA) combina las dos pruebas anteriores. Primero se inicia con la prueba de trabajo para encontrar el Golden Nonce. Dependiendo de la implementación, los bloques minados no contienen ninguna transacción, sino que solo contiene una cabecera y la dirección de recompensa del minero.

Una vez minado, el sistema cambia a la prueba de participación. Teniendo la cabecera como referencia, se elige un grupo aleatorio de validadores para firmar el nuevo bloque. El modelo se convierte en un bloque completo cuando todos los validadores lo firman.

Si algunos de los validadores seleccionados no está disponible para completar el bloque, se selecciona el siguiente bloque ganador y se elige nuevamente al grupo validador.

### 11.4 Prueba de quemado

Proof of Burn (PoB) es similar a la prueba de participación, pero mientras que en la prueba de participación tiene más probabilidades de ser validador quién tiene más criptomonedas, en este algoritmo tiene más probabilidades quien invierte o “quema” más monedas. Al quemarlas se pierde el acceso a ellas, así el minero puede demostrar su dedicación y compromiso con la red, ya que pierde sus criptomonedas para obtener la capacidad de agregar bloques a la cadena.

### 11.5 Prueba de capacidad

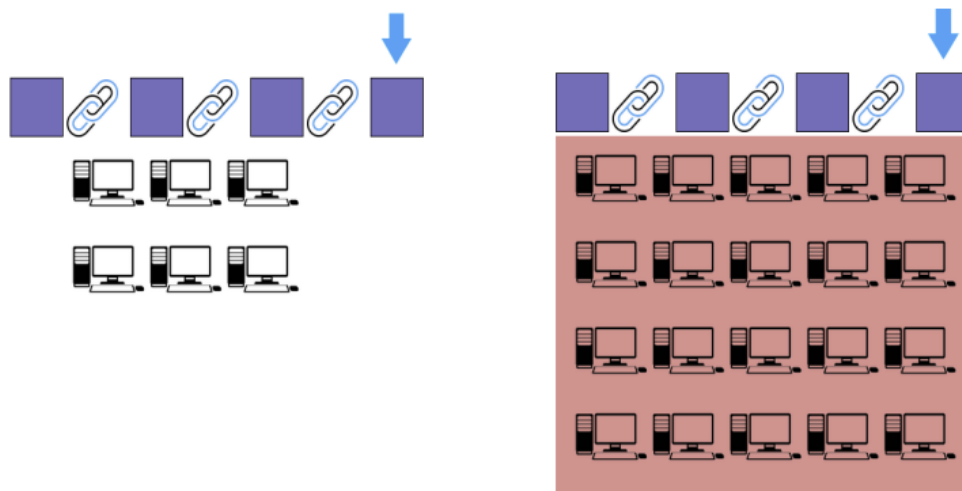
Proof of Capacity (PoC) se basa en el espacio de almacenamiento disponible. Cuanto más espacio en el disco duro, mayor es la probabilidad de minar un bloque y recibir la recompensa.

## 12. Ataque del 51%

No existe ningún sistema ni tecnología que sea 100% segura. El ataque del 51% es la vulnerabilidad más conocida de Blockchain, pero a día de hoy no se ha conseguido realizar con éxito.

Supongamos que una red está compuesta por dos grupos de mineros, los legítimos y los maliciosos. Para que el ataque sea un éxito, el grupo malicioso debe ser estrictamente mayor que el grupo de mineros legítimos.

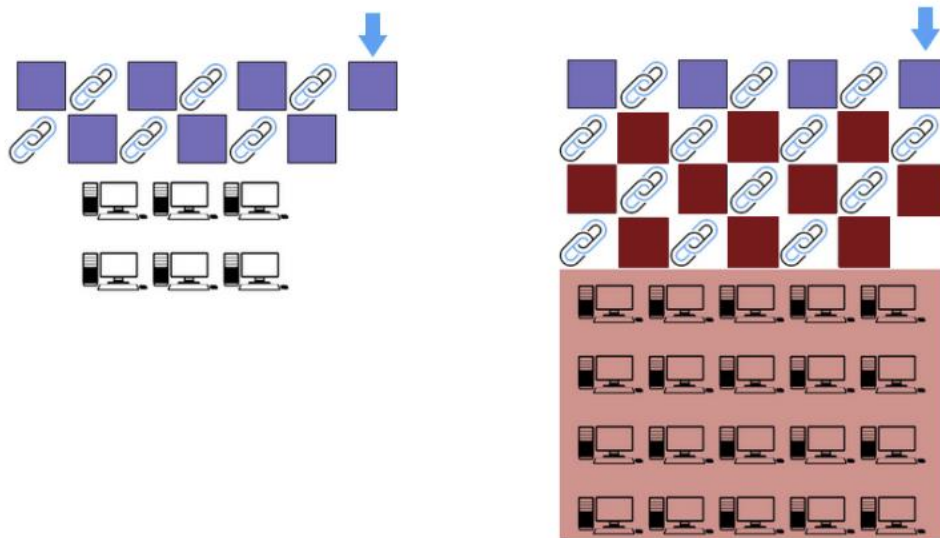
Si un grupo mayoritario de nodos maliciosos se une repentinamente a una red Blockchain, podrían intentar modificar la cadena de bloques. Debido a que el software del sistema no tiene ningún sistema de detección de nodos maliciosos, cada nodo realiza una copia actual de la cadena. Una vez que todos los nodos malintencionados tienen una copia de la cadena, se desconectan de la red y comienzan a añadir sus bloques maliciosos.



*Ilustración 18: Ataque del 51%*

Como el grupo malicioso tiene un mayor número de dispositivos, aumentan su cadena mucho más rápido que la legítima, pero permaneciendo desconectada de nuestra red.

Una vez que añadan los bloques deseados, los mineros maliciosos vuelven a conectarse a nuestro Blockchain, provocando una corrupción en el sistema debido a la existencia de la cadena legítima y la maliciosa.



*Ilustración 19: Ataque del 51% completado*

Como hemos visto anteriormente, una de las principales reglas del Blockchain es validar la cadena de mayor tamaño frente a las demás. En este punto, la red validaría la cadena maliciosa y la red habría caído al ataque.

### 13. Claves públicas y privadas

Blockchain utiliza muchos conceptos de relacionados con la criptografía, por tanto, el uso de claves está muy presente cuando se habla de cadenas de bloques.

El fin de esta tecnología, también denominada cifrado asimétrico, es facilitar una comunicación segura y privada mediante dos firmas digitales en un canal público. De esta manera podemos asegurar que quien ha hecho la transacción es realmente quien dice ser o que solamente ese usuario pueda conocer la información.

El cifrado asimétrico está compuesto de una clave pública (compartida al resto de usuarios) y una clave privada (debe permanecer oculta):

- Clave pública: Si un mensaje es cifrado con la clave pública de una persona, solamente puede descifrarlo con su clave privada.
- Firma digital: Si un mensaje es cifrado por una clave privada, sólo pueden descifrarlo aquellos usuarios que posean la clave pública de ese usuario. De esta manera, el resto de usuarios pueden asegurar que el autor de ese mensaje es quién dice ser.

Las ventajas de la utilización de estas claves son:

- No repudio. Se puede demostrar la participación de las partes (remitente y destinatario) mediante su identificación.
- Comunicación segura y encriptada a través de canales abiertos.
- Autenticación. Permiten a los usuarios de la red demostrar que son los autores originales del mensaje, además de evitar su modificación.

## 14. Smart Contracts

Los contratos Inteligentes son programas que se ejecutan en las cadenas, cuando se cumplen unas condiciones establecidas, para hacer cumplir los términos de acuerdo entre dos partes. El código puede ser visible por cualquier participante.

Están diseñados con el objetivo de automatizar la ejecución de un acuerdo, sin necesidad de que intervenga ningún intermediario ni pérdidas de tiempo.

Los Smart Contracts pueden tener múltiples usos en diferentes áreas. En el ámbito financiero se pueden utilizar para facilitar transacciones, realizar pagos automatizados y establecer contratos de préstamo. En cadenas de suministro, pueden ayudar a rastrear y verificar el movimiento de productos. Además, pueden utilizarse para crear testamentos digitales, acuerdos de propiedad intelectual u otros contratos legales.

Al utilizar los Contratos Inteligentes, se añaden dos nuevos campos a los bloques. En primer lugar, se almacena el historial de todos los contratos, incluidos los contratos de bloques anteriores. Posteriormente, se almacenan los estados actuales de todos los contratos. Al ser un programa, puede tener distintos estados como fallido, ejecutándose, terminado con éxito, etc.

Las ventajas que ofrece este tipo de software son las siguientes:

- Velocidad y precisión. En cuanto se cumple una condición, el contrato se ejecuta de manera inmediata.
- Confianza y transparencia. Todas las transacciones se quedan registradas en la cadena.
- Ahorro. Se abaratan costes debido a la eliminación de intermediarios.



## 15. Ventajas e inconvenientes de Blockchain

Las principales ventajas de Blockchain son las siguientes:

- Seguridad. Debido a la utilización de algoritmos criptográficos y políticas de consenso, los datos almacenados en la cadena son inmutables y resistentes a la modificación.
- Transparencia. Al ser un sistema descentralizado, las cadenas de bloques permiten que todos los participantes tengan acceso a todos los datos en cualquier momento.
- Eliminación de intermediarios. Permite realizar transacciones sin la necesidad de un intermediario, como bancos o gobiernos, gracias a las redes P2P. Esto reduce costos y aumenta el rendimiento.
- Agilidad y eficiencia. Al ser la mayoría de los procesos automáticos y la validación descentralizada, las transacciones en Blockchain son más rápidas y con menos probabilidad de errores.
- Resistencia a la censura. La descentralización evita, precisamente, que un organismo tenga el control absoluto sobre la red, evitando tomar el control de ésta.

Aunque sus ventajas son muy interesantes, ninguna tecnología evita tener algunos inconvenientes. En Blockchain, los principales son:

- Escalabilidad. Las cadenas de bloques públicas con gran cantidad de usuarios se convierten en un desafío a la hora de validar y almacenar cada transacción en todos los nodos de la red. Esto puede limitar la capacidad de procesamiento en comparación con los sistemas centralizados.
- Costes. El mantenimiento y operación de Blockchain puede ser costoso, especialmente en consumo de energía y capacidad de almacenamiento.
- Privacidad. Los datos, al ser accesibles por todos los participantes de la red, pierden su privacidad, especialmente importante en datos sensibles.
- Inmutabilidad. Aunque es una de sus principales ventajas, si se produce un error o algún dato insertado es incorrecto, hace casi imposible su edición.

## 16. Aplicaciones

Esta tecnología puede ser aplicada a distintos ámbitos y sectores.

- Criptomonedas. Posiblemente sea el uso más conocido. Bitcoin o Ethereum, entre otras, utilizan Blockchain para almacenar las transacciones realizadas con sus divisas, permitiendo pagos rápidos, seguros y anónimos, evitando intermediarios como los bancos.
- Economía. También puede ser aplicada al ámbito de finanzas y banca, pudiendo realizar transferencias de dinero internacionales de manera rápida y económica. Además, facilita la trazabilidad y mejora los procesos de verificación de identidad y prevención de fraude.
- Cadenas de suministro. Permite rastrear y verificar el origen y la procedencia de productos a lo largo de una cadena de suministro. Esto garantiza la autenticidad de los productos y mejora la eficiencia en la logística.
- Votaciones. Las cadenas de bloques pueden ser utilizadas para asegurar la integridad y transparencia en la realización de votaciones, garantizando que los resultados sean inmutables y confiables.
- Propiedad intelectual y derechos de autor. En el último año, este ámbito se ha convertido en una de las aplicaciones más conocidas de Blockchain, por detrás de las criptomonedas. Se pueden registrar y proteger derechos de propiedad intelectual como patentes, derechos de autor...

## 17. Ejemplo práctico

Como ejemplo práctico, he decidido realizar una simulación de cómo sería un sistema de votación digital basado en Blockchain.

Para ello, voy a crear una sencilla aplicación web escrito en el lenguaje de programación “Python”, además de una interfaz HTML con CSS y JavaScript para conectarnos a dicha aplicación.

### 17.1 Prerrequisitos

Los archivos necesarios para replicar los siguientes ejemplos prácticos pueden ser encontrados en el siguiente repositorio de [GitHub](#).

Primero debemos instalar Python. En la página oficial de [Python](#), podemos encontrar y descargar el instalador. Una vez descargado, ejecutamos y seguimos los pasos que muestre el instalador.

Para evitar conflictos con otros entornos de trabajo u otras aplicaciones, crearemos y activaremos un entorno virtual. Para ello, se utiliza el comando “python.exe -m venv <nombre\_entorno>”. Para activar el entorno virtual creado, usamos el comando “.<nombre\_entorno>\Scripts\activate”. Para desactivar el entorno, escribimos “deactivate”.

```
PS D:\#Proyecto\python> python.exe -m venv .venv
● PS D:\#Proyecto\python> .\.venv\Scripts\activate
● (.venv) PS D:\#Proyecto\python> deactivate
○ PS D:\#Proyecto\python> 
```

Una vez creado y activado nuestro entorno virtual (en mi caso llamado “**.venv**”) procedemos a instalar las librerías necesarias para ejecutar mi aplicación. Con el comando “python.exe -m pip install Flask”, instalamos la librería de Flask. Flask es un framework escrito en Python que permite crear y ejecutar aplicaciones web rápidamente y con pocas líneas de código.

```
PS D:\#Proyecto\python> .\.venv\Scripts\activate
(.venv) PS D:\#Proyecto\python> python -m pip install Flask
Requirement already satisfied: Flask in d:\#proyecto\python\ve
)
```

## 17.2 Creación de una Blockchain en Python

En el archivo “**blockchain.py**”, se procede a desarrollar el código necesario para crear nuestra cadena de bloques.

Primero importamos las librerías que vamos a utilizar.

```
import datetime
import hashlib
```

- datetime: nos permitirá realizar el Timestamp de nuestros bloques en el momento de ser minados.
- hashlib: Nos permitirá realizar hashes con distintos algoritmos. En este ejemplo se utiliza el SHA-256

Una vez importadas las librerías, procederemos a crear una clase para nuestro **Bloque**, agregando los atributos y las funciones que queremos que utilice.

```
class Bloque:
    def __init__(self, index, timestamp, data, prev_hash):
        ...

        Iniciar un bloque.
        Índice
        TimeStamp
        Datos
        Hash del bloque anterior
        Nonce
        Hash del bloque
        ...

        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.prev_hash = prev_hash
        self.nonce = 0
        self.hash = self.calcular_hash()
```

La función **\_\_init\_\_** inicializa los atributos de nuestro bloque.

- index: Almacena el índice de nuestros bloques en la cadena.
- timestamp: Almacena, en Unix Time, la fecha y hora exacta en el que el bloque es minado, es decir, unido a nuestro Blockchain

- data: Este atributo almacena la información que queremos guardar de manera inmutable. En este ejemplo, se almacena el DNI del votante y el candidato al que vota.
- prev\_hash: Hash del bloque anterior de la cadena. Este atributo es el que enlaza un bloque con el anterior.
- nonce: Número que debe ser hallado para encontrar un hash que cumpla con el objetivo.
- hash: Atributo que almacena el hash resultante de los anteriores atributos. En este ejemplo, el hash a hallar debe comenzar por 3 ceros "000xxxxxxxxxxx..."

Una vez detallado los atributos que contendrá nuestro bloque, procedemos a crear una función que calcule el hash del bloque.

```
def calcular_hash(self):
    """
    Calcula el hash del bloque
    """
    bloque_hash = str(self.index) + str(self.timestamp) + str(self.data) + str(self.prev_hash) + str(self.nonce)
    return hashlib.sha256(bloque_hash.encode()).hexdigest()
```

- Esta función genera y devuelve un hash a partir de los otros atributos.

La siguiente función será utilizada para minar el bloque, es decir, añadirlo a nuestra Blockchain.

```
def minar_bloque(self):
    """
    Minado de un bloque para añadirlo a la cadena.
    Aumentar cant_ceros para mayor dificultad o viceversa
    """
    cant_ceros = 4
    n_ceros = "0" * cant_ceros

    # Búsqueda del Nonce para encontrar un hash válido
    while self.hash[:cant_ceros] != n_ceros:
        self.nonce += 1
        self.timestamp = datetime.datetime.now().timestamp()
        self.hash = self.calcular_hash()
        print(f"{self.nonce}: {self.hash}") # Mostrar el Nonce y el hash generado
    print('Minado realizado con éxito')
```

- cant\_ceros: Variable que almacena el número de 0 por la que debe comenzar un hash para cumplir los requisitos. Esta variable establece la "dificultad" de encontrar un hash válido. A mayor número de 0, mayor potencia computacional requerida.

- El bucle es ejecutado hasta que se encuentra un hash que cumpla el objetivo. Para ello, el Nonce se incrementa con paso 1 (en la realidad se genera aleatoriamente). Una vez que hallamos un hash que cumpla el objetivo, el bloque será añadido a la cadena.

Después de generar la clase para nuestros bloques, creamos la clase **Blockchain**.

```
class Blockchain:
    def __init__(self):
        """
        Iniciar cadena con el Bloque Génesis
        """
        self.chain = [self.crear_bloque_genesis()]

    def crear_bloque_genesis(self):
        """
        Creación del Bloque Génesis
        """
        return Bloque(0, datetime.datetime.now().timestamp(), {'Bloque génesis': True}, "0")
```

- **\_\_init\_\_**: Inicia nuestra cadena de bloques.
- **crear\_bloque\_genesis**: Esta función genera el primer bloque de nuestra cadena. A partir de él, se unirán los nuevos bloques.

```
def obtener_bloque_anterior(self):
    """
    Devuelve el bloque anterior de la cadena
    """
    return self.chain[-1]
```

- **obtener\_bloque\_anterior**: Función que devuelve el bloque anterior de la cadena.

```
def add_bloque(self, bloque_nuevo):
    """
    Añade un nuevo bloque a la cadena después de minar
    """
    # Obtener hash del bloque previo y nuevo index
    bloque_nuevo.prev_hash = self.obtener_bloque_anterior().hash
    bloque_nuevo.index = self.obtener_bloque_anterior().index + 1

    # Minar y añadir
    bloque_nuevo.minar_bloque()
    self.chain.append(bloque_nuevo)
```

- **add\_bloque:** Función que añade el bloque a la cadena una vez que el minado haya sido realizado con éxito.

```
def validar_cadena(self):
    """
    Comprobar si la cadena es válida o ha sido modificada.
    Se comprueban todos los bloques de la cadena
    """
    for i in range(1, len(self.chain)):
        bloque_actual = self.chain[i]
        bloque_anterior = self.chain[i - 1]

        # Comparar hash actual con el hash almacenado en el bloque actual
        if bloque_actual.hash != bloque_actual.calcular_hash():
            return False

        # Comparar hash del bloque anterior con el hash del bloque anterior almacenado en el bloque actual
        if bloque_actual.prev_hash != bloque_anterior.hash:
            return False

    return True
```

- **validar\_cadena:** Función que comprueba la validez de la cadena, es decir, que no existen modificaciones. Para ello, se comprueba, en todos los bloques de la cadena, que el hash actual del bloque es el mismo que el hash almacenado en la cadena. También se comprueba que el hash del bloque anterior coincide con el hash previo del bloque actual.

Con este sencillo código podemos crear una cadena, crear bloques, añadirlos a la cadena y validar la secuencia de bloques.

### 17.3 Ejemplo simple en Python

En el archivo “**example.py**” se procede a generar una cadena, minar 5 bloques y mostrarlos.

En primer lugar, debemos importar las librerías necesarias:

```
from blockchain import Bloque, Blockchain
```

- blockchain: de nuestro archivo creado anteriormente, importamos la clase Bloque y la clase Blockchain

Una vez importadas las librerías, inicializamos la cadena (se genera el bloque génesis) y procedemos a minar los primeros 5 bloques.

```
# Generar cadena
miguel_chain = Blockchain()

# Agregar bloques
miguel_chain.add_bloque(Bloque(0, 0, {'usuario': 'U_001', 'voto': 'Candidato A'}, ""))
miguel_chain.add_bloque(Bloque(0, 0, {'usuario': 'U_002', 'voto': 'Candidato A'}, ""))
miguel_chain.add_bloque(Bloque(0, 0, {'usuario': 'U_003', 'voto': 'Candidato B'}, ""))
miguel_chain.add_bloque(Bloque(0, 0, {'usuario': 'U_004', 'voto': 'Candidato B'}, ""))
miguel_chain.add_bloque(Bloque(0, 0, {'usuario': 'U_005', 'voto': 'Candidato A'}, ""))
```

En la primera parte del ejemplo, se puede observar cada Nonce intentado y el hash generado de cada bloque minado. Se puede comprobar que el minado se completa cuando se encuentra un hash que comience por 3 ceros.

```
1198: 3f5f5cfd1bc44ca2ffb15a2deff97e78131e00dec8dece3cdf35bb9d7544ba2
1199: 17d1fa094d16438548823616ecdb0c42fe99657a112067e4294d5cad0a9cf6d6
1200: 2238fb952c25c22f35ef7f1075a90284e7611372a653135f4490431176985c79
1201: 340ffa6a75031fdd8dd8dd5ad4ce0f875cedc4f13c914e4cdbc1f93a432f230
1202: 000f4243e8387ea185b8b8184bb40ceeb65b5fd6a85d6d770dfae257d6ba41cf
Minado realizado con éxito
```

```
3456: 79241b53de02593313690b6f6b4245bf9ec73b65610c9ff593b6863ad75b24d3
3457: 861118f6161e5a4ab7a51a40f1a4b010279c372b579da7aeca777fcc866653c9
3458: ffb214fea29de10468a36a91f4934efead5c41e3f373bcff3f2d154476fbd0e
3459: a66d3e30ca5452c5a2acf3aaa5ae32cff17c4168f1dc78bbf6ffa89afcb08e9d
3460: 86ea899892a81f1b119d3394dde461c9c7fe83c7af000055dd77e6e5f7a3b0e6
3461: 1aecbdaa2dec96c958f9851a537a4f6c58f926c33fa9ca3bfd40203ecd0fa21b
3462: 5152a54472aa9c4c642d880fe06a4c6dd552130e83c3617cee57374946760787
3463: 00013058076eb7466e7076796de857f67da9ee07f07310122145afd5a87b115b
Minado realizado con éxito
```



Con la cadena creada y compuesta de varios bloques, procedemos a la visualización de los bloques.

```
# Mostrar bloques
print()
print("Miguel Chain:")
print("-----")
for block in miguel_chain.chain:
    print("Index:", block.index)
    print("Timestamp:", block.timestamp)
    print("Data:", block.data)
    print("Previous Hash:", block.prev_hash)
    print("Hash:", block.hash)
    print("-----")
```

En primer lugar, aparece el bloque génesis, observamos que al primer bloque de la cadena no se exige el requisito de los 3 ceros del hash.

```
Mostrar cadena:
-----
Index: 0
Timestamp: 1684782857.398783
Data: {'Bloque génesis': True}
Previous Hash: 0
Hash: dc63bea6b4086338574d8a3a932a7b1f3ec959baa5e050de938214ad9b102259
```

A partir del bloque génesis, se unen los siguientes bloques

```
Mostrar cadena:
-----
Index: 0
Timestamp: 1684782857.398783
Data: {'Bloque génesis': True}
Previous Hash: 0
Hash: dc63bea6b4086338574d8a3a932a7b1f3ec959baa5e050de938214ad9b102259
-----
Index: 1
Timestamp: 1684782858.935886
Data: {'usuario': 'U_001', 'voto': 'Candidato A'}
Previous Hash: dc63bea6b4086338574d8a3a932a7b1f3ec959baa5e050de938214ad9b102259
Hash: 000a7c0beae8fc59c496e2f38f9c083aff6bb1409469535d0493ce49b33cea02
-----
Index: 2
Timestamp: 1684782859.123411
Data: {'usuario': 'U_002', 'voto': 'Candidato A'}
Previous Hash: 000a7c0beae8fc59c496e2f38f9c083aff6bb1409469535d0493ce49b33cea02
Hash: 0002c1d5a0ffabba2bd70a132b9716209995c541d6d0511fb9c3d432969823a1
-----
Index: 3
Timestamp: 1684782859.829934
Data: {'usuario': 'U_003', 'voto': 'Candidato B'}
Previous Hash: 0002c1d5a0ffabba2bd70a132b9716209995c541d6d0511fb9c3d432969823a1
Hash: 00006728ee5826fef3c2e84239c854134a6596405db2214f4b30b8c3b635ac55
-----
Index: 4
Timestamp: 1684782860.504935
Data: {'usuario': 'U_004', 'voto': 'Candidato B'}
Previous Hash: 00006728ee5826fef3c2e84239c854134a6596405db2214f4b30b8c3b635ac55
Hash: 000204de004465e96cab19516005d04ce11f2168e857a2f826f842462ca9be49
-----
Index: 5
Timestamp: 1684782861.815014
Data: {'usuario': 'U_005', 'voto': 'Candidato A'}
Previous Hash: 000204de004465e96cab19516005d04ce11f2168e857a2f826f842462ca9be49
Hash: 000bee051b5b60872dba777c433bfd066cdda5cfbe79f258f8280b131e0dcfd4
```

Como se puede observar, el hash del bloque anterior es almacenado en el bloque siguiente. De esta manera todos los bloques de la cadena quedan enlazados criptográficamente.

Por último, comprobamos la validez de la cadena.

```
# Verificar cadena
print()
print("¿La cadena es válida?", miguel_chain.validar_cadena())
input('Pulsa una tecla para continuar')
```

Podemos observar que se cumplen los requisitos, validándose la cadena.

```
¿La cadena es válida? True
```

Ahora que tenemos una Blockchain totalmente funcional, ¿qué ocurre si una persona malintencionada modifica uno de nuestros bloques?

```
# Modificación de un bloque
print()
print("*****")
print("¿Qué ocurre si se modifica el valor de un dato?")
modificar_index = int(input("Index del bloque a modificar: "))
modificar_dato = input("Dato modificado: ")
print("*****")
print()

# Modificación y minado de un bloque
bloque_modificar = miguel_chain.chain[modificar_index]
bloque_modificado = Bloque(bloque_modificar.index, 0, modificar_dato, bloque_modificar.prev_hash)
bloque_modificado.minar_bloque()
miguel_chain.chain[modificar_index] = bloque_modificado
```

En este ejemplo modifico el bloque cuyo índice es el **2**, modificando el contenido del voto de **“Candidato A”** a **“Candidato B”**.

```
¿Qué ocurre si se modifica el valor de un dato?
Index del bloque a modificar: 2
Dato modificado: Candidato B
```

Se puede observar que el minado se ha realizado con éxito, cambiando la información del bloque 2 por la nueva del bloque modificado.

```
7843: b48ec75fa6c01c72c30b235901045624a27656e1248a8e5baff718e9e066a63a
7844: 8570430dc4bb68e00ff5cabe3771c228f2c1df9982906ab23902d45d32d55bda
7845: 000eb457e2757d4c71235c8a772047889e87e9e0e36f5325c8f5c94f2c4cb2a6
Minado realizado con éxito
```

Al mostrar de nuevo el contenido de la cadena, se puede observar que el hash previo del bloque **3** no coincide con el hash del bloque modificado, el bloque 2, produciendo una discrepancia en Blockchain.

```
Index: 1
Timestamp: 1684782858.935886
Data: {'usuario': 'U_001', 'voto': 'Candidato A'}
Previous Hash: dc63bea6b4086338574d8a3a932a7b1f3ec959baa5e050de938214ad9b102259
Hash: 000a7c0beae8fc59c496e2f38f9c083aff6bb1409469535d0493ce49b33cea02
-----
Index: 2
Timestamp: 1684783400.009651
Data: Candidato B
Previous Hash: 000a7c0beae8fc59c496e2f38f9c083aff6bb1409469535d0493ce49b33cea02
Hash: 000eb457e2757d4c71235c8a772047889e87e9e0e36f5325c8f5c94f2c4cb2a6
-----
Index: 3
Timestamp: 1684782859.829934
Data: {'usuario': 'U_003', 'voto': 'Candidato B'}
Previous Hash: 0002c1d5a0ffabba2bd70a132b9716209995c541d6d0511fb9c3d432969823a1
Hash: 00006728ee5826fef3c2e84239c854134a6596405db2214f4b30b8c3b635ac55
```

Si validamos de nuevo la cadena, podemos observar que la validación no se cumple.

```
¿La cadena es válida? False
```

## 17.4 Aplicación web para un sistema de votación

En este apartado se mostrará una simulación de cómo se podría aplicar la tecnología Blockchain para realizar un sistema de votación electrónica mediante una aplicación web.

En el archivo `__main__.py`, comenzamos importando las librerías necesarias:

```
from flask import Flask, jsonify, request, render_template
from blockchain import Bloque, Blockchain
import random
```

- flask: Framework que nos permitirá crear una aplicación web.
- blockchain: Importa las clases necesarias para generar nuestra Blockchain.
- random: Será utilizado para generar votos aleatorios.

A continuación, creamos una app de Flask e iniciamos la cadena.

```
# Creación de una app Flask
app = Flask(__name__)

# Inicio de la cadena
miguel_chain = Blockchain()
```

Después de iniciar la cadena, procedemos a minar 24 bloques con votos aleatorios para añadirlos a la cadena.

```
# Primeros 24 votos aleatorios
candidatos = ['A','B','C','D']
for i in range(1,25):
    candidato_votado = candidatos[random.randint(0,3)]
    miguel_chain.add_bloque(
        Bloque(0, 0, {'usuario': 'U_'+str(i), 'voto': 'Candidato '+candidato_votado}, ''))
    )
```

El siguiente paso es crear las rutas de nuestra aplicación web y mostrar sus respectivos documentos HTML almacenados en la carpeta **templates**. El código de los documentos HTML será explicado más adelante.

```
# Rutas html
@app.route('/')
def index_html():
    return render_template('index.html')

@app.route('/editar')
def editar_html():
    return render_template('editar.html')

@app.route('/login')
def login_html():
    return render_template('login.html')

@app.route('/resultado')
def resultado_html():
    return render_template('resultado.html')

@app.route('/votar/<dni>')
def votar_html(dni):
    return render_template('votar.html', dni=dni)
```

A continuación, creamos las rutas para solicitar o añadir información a nuestra aplicación (API).

- **add\_bloque:** Si el usuario no ha realizado ningún voto, el bloque es minado y añadido a la cadena.

```
# Rutas API
@app.route('/add_bloque', methods=['POST'])
def add_bloque():
    """
    Añade un bloque a la cadena
    """
    data_api = request.get_json()

    # Comprueba si el votante ha votado
    usuario = data_api['usuario']
    for bloque in miguel_chain.chain:
        if usuario in bloque.data.values():
            return jsonify({'message': 'El usuario ya ha votado'})

    # Crea, mina y añade un bloque a la cadena
    data = {
        'usuario': usuario,
        'voto': data_api['voto']
    }
    miguel_chain.add_bloque(Bloque(0, 0, data, ""))

    return jsonify({'message': 'El bloque se ha añadido correctamente'})
```

- **get\_chain:** Comprueba la validez de la cadena y retorna todos los bloques excepto el bloque génesis.

```
@app.route('/get_chain', methods=['GET'])
def get_chain():
    """
    Devuelve el contenido de la cadena y comprueba si es válida
    """
    data = []

    # Comprobar validez de la cadena
    if not miguel_chain.validar_cadena():
        data.append(
            {
                'error': 'cadena modificada!'
            }
        )

    # Devolver cadena excepto el bloque génesis
    for bloque in miguel_chain.chain[1:]:
        data.append(bloque.data)

    return jsonify(data)
```

- **modify/<index>/<voto>**: Se utilizará para realizar la prueba de modificación de un bloque. El parámetro index indica el índice del bloque a modificar. El parámetro voto almacena el dato modificado.

```
@app.route('/modify/<index>/<voto>')
def modificar(index, voto):
    ...

    Modifica el contenido de un bloque cuyo índice es un parámetro
    ...

    # Modificación de un bloque
    modificar_index = int(index)
    modificar_data = {
        'usuario': 'Editado',
        'voto': 'Candidato '+voto.upper()
    }
    bloque_modificar = miguel_chain.chain[modificar_index]
    bloque_modificado = Bloque(
        bloque_modificar.index, 0, modificar_data, bloque_modificar.prev_hash
    )

    # Minar bloque modificado
    bloque_modificado.minar_bloque()
    miguel_chain.chain[modificar_index] = bloque_modificado

    return jsonify(['Editado con éxito'])
```

Por último, iniciamos la aplicación web

```
# Programa
if __name__ == '__main__':
    app.run()
```

Una vez iniciado el programa, podremos observar como son minados los 24 votos realizados aleatoriamente. Una vez minados, podremos acceder a la aplicación web. Para ello, desde cualquier navegador, introducimos <http://127.0.0.1:5000>.

```
5027: 33a31acca21714d31c0531a2a40c55dab24fc7717bfc2a57e4a6f7052074a59c
5028: 473d7050885b0b51b60658439dc6f599d97c59ff55a90928f35a17ed6014d5b5
5029: 00012adc2840fcefafa235678d1ecceaa4f314e5abcc433b3a399cd8ad9703720
Minado realizado con éxito
* Serving Flask app '__main__'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



En **/resultado** se puede observar el recuento actual de votos.

## Resultado de la votación

Candidato A: 5
Candidato B: 6
Candidato C: 7
Candidato D: 6

Un votante que desee realizar una votación, deberá acceder a **/login** e iniciar sesión. En este ejemplo los DNI válidos son "U\_<1-100>" y la contraseña es "12345".

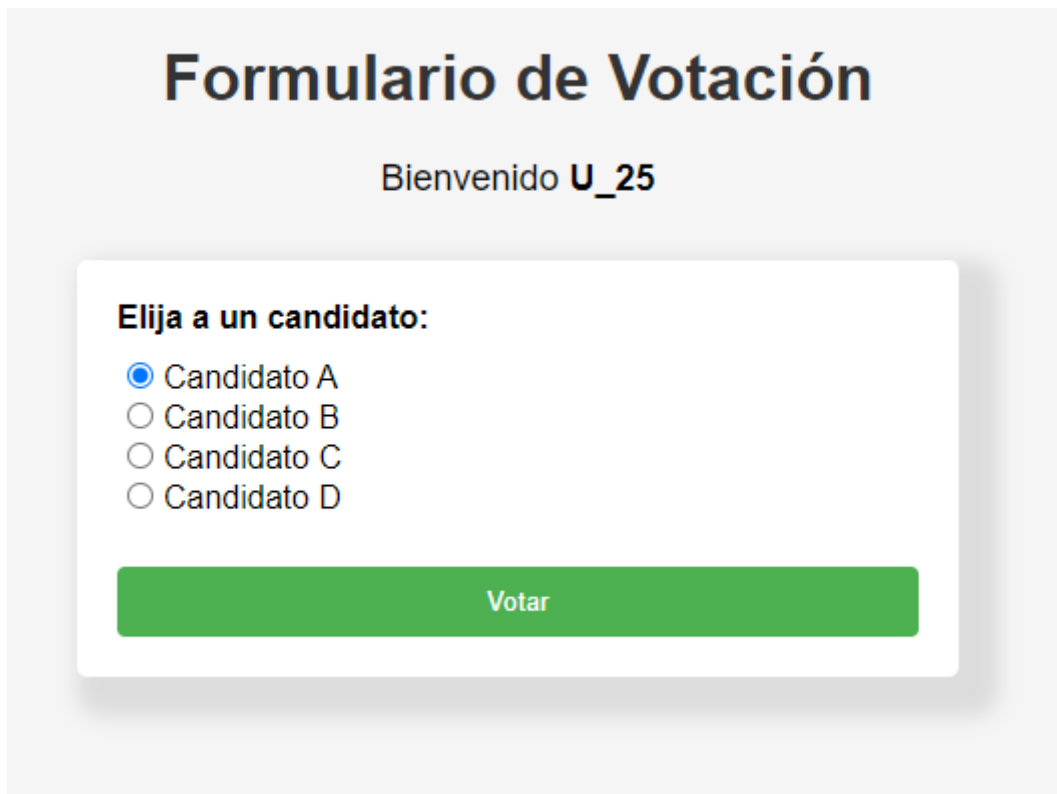
## Acceso a votación

**DNI:**

**Contraseña:**

Login

Al introducir los datos correctos, el usuario accederá a un formulario donde poder realizar su voto. En este ejemplo seleccionamos al Candidato A.



## Formulario de Votación

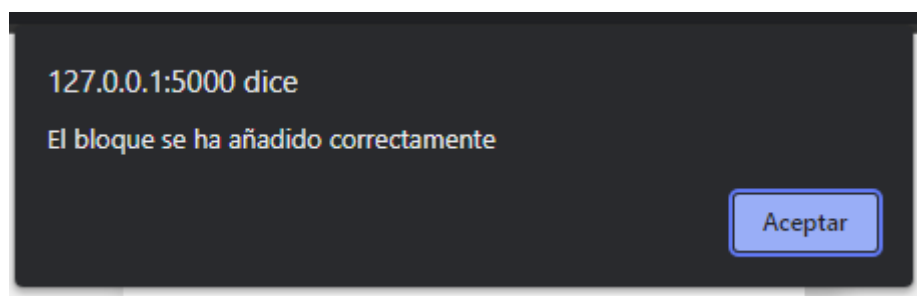
Bienvenido U\_25

**Elija a un candidato:**

- ☒ Candidato A
- ☐ Candidato B
- ☐ Candidato C
- ☐ Candidato D

Votar

Se comprueba que el votante no ha realizado un voto anteriormente y se mina el bloque.



127.0.0.1:5000 dice

El bloque se ha añadido correctamente

Aceptar

La cadena contiene un nuevo bloque (voto).

U_25	Candidato A
------	-------------

Para simular una modificación malintencionada de un bloque, accedemos a **/editar**. En el formulario, introducimos el index del bloque que queremos editar y en Candidato elegimos la letra del candidato a votar. Por ejemplo, modificamos el voto del U\_8 para votar al Candidato A.

U_8	Candidato B
-----	-------------

## Editar un bloque

**Index del bloque a editar:**

**Candidato:**

**Editar Bloque**

Una vez minado de nuevo el bloque, la modificación se añade a la cadena.

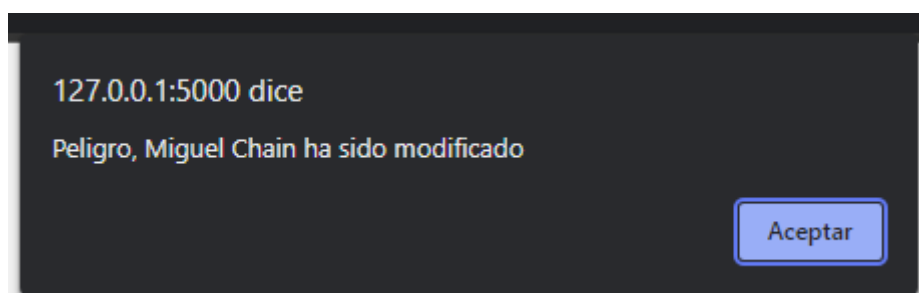
127.0.0.1:5000 dice

Bloque modificado

**Aceptar**

¿Qué ocurre si volvemos a observar el recuento de votos? Como vimos en el `example.py`, la cadena detecta que el hash del bloque modificado no coincide con el hash previo del siguiente bloque, invalidando la cadena.

Al visualizar de nuevo los resultados en **/resultados**, nos aparecerá una alerta notificando que existe una modificación.



## Votos de cada participante

Votante	Voto
Warning!	Modificación detectada
U_1	Candidato C
U_2	Candidato A
U_3	Candidato C
U_4	Candidato D
U_5	Candidato C
U_6	Candidato B
U_7	Candidato B
Editado	Candidato A

Si en vez de realizarse en un entorno local, fuese en uno de los miles o millones de nodos de una red, la cadena modificada sería desechada por consenso, conservándose la cadena sin modificar.

A continuación, se muestra el código de los documentos HTML almacenados en el directorio **templates**.

- **resultado:** En este documento se muestran los bloques almacenados en la cadena y el recuento de votos.

```
<body>
  <h1>Resultado de la votación</h1>
  <ul>
    <li id="candidatoA">Candidato A: </li>
    <li id="candidatoB">Candidato B: </li>
    <li id="candidatoC">Candidato C: </li>
    <li id="candidatoD">Candidato D: </li>
  </ul>

  <h1>Votos de cada participante</h1>
  <table id="resultados">
    <thead>
      <tr>
        <th>Votante</th>
        <th>Voto</th>
      </tr>
    </thead>
    <tbody></tbody>
  </table>
```

Mediante código JavaScript, solicitamos el listado de bloques de nuestra cadena. Primero inicializamos los contadores de votos de cada candidato.

```
<script>
  // Contadores candidatos
  let n_A = 0;
  let n_B = 0;
  let n_C = 0;
  let n_D = 0;
```

Solicitamos la cadena de bloques a la aplicación y mostramos los bloques en la tabla.

```
// Solicitud de la cadena
fetch('/get_chain')
.then(response => response.json())
.then(data => {
    // Obtener la referencia a la tabla
    const table = document.getElementById('resultados');
    e = 0;

    // Mostrar resultados en la tabla
    data.forEach(item => {
        const row = table.insertRow();
        const chainDNI = row.insertCell();
        const chainVoto = row.insertCell();
        chainDNI.textContent = item.usuario;
        chainVoto.textContent = item.voto;
        if (item.error){
            chainDNI.textContent = 'Warning!';
            chainVoto.textContent = 'Modificación detectada';
            e = 1;
        }
        if (item.voto === 'Candidato A'){
            n_A++;
        }
        if (item.voto === 'Candidato B'){
            n_B++;
        }
        if (item.voto === 'Candidato C'){
            n_C++;
        }
        if (item.voto === 'Candidato D'){
            n_D++;
        }
    });

    // Recuento total de votos
    document.getElementById('candidatoA').textContent += n_A;
    document.getElementById('candidatoB').textContent += n_B;
    document.getElementById('candidatoC').textContent += n_C;
    document.getElementById('candidatoD').textContent += n_D;

    // Advertencia de modificación
    if (e === 1) {
        alert('Peligro, Miguel Chain ha sido modificado')
    }
});
```

- **login:** El código HTML es un formulario sencillo con dos inputs, uno de tipo texto para introducir el DNI y otro de tipo password para la contraseña. Ambos obligatorios.

```
<body>
  <h1>Formulario de Registro</h1>

  <form id="login" onsubmit="return validar()">
    <div id="error" class="error"></div>
    <label for="dni">DNI:</label>
    <input type="text" id="dni" name="dni" required>
    <br>
    <label for="password">Contraseña:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <input type="submit" value="Login">
  </form>
```

En JavaScript comprobamos si el DNI y la contraseña son válidos. Si es correcto, se redirecciona al usuario al formulario para realizar el voto. Si alguno de los campos es incorrecto, se notifica al usuario.

```
<script>
  // DNI y contraseña de los votantes
  votantes = {}
  for (let i = 1; i <= 100; i++) {
    votantes['U_'+i] = '12345'
  }

  // Validación y redirección
  function validar() {
    var dni = document.getElementById('dni').value;
    var password = document.getElementById('password').value;
    var error = document.getElementById('error');

    if (votantes[dni] === password) {
      window.location.href = '/votar/'+dni;
      return false;
    } else {
      error.textContent = '*DNI o contraseña incorrectos';
      return false;
    }
  }
</script>
```

- **votar:** En este documento, el usuario puede seleccionar al candidato y realizar el voto.

```
<body>
  <h1>Formulario de Votación</h1>
  <p>Bienvenido <b>{{ dni }}</b></p>

  <form id="votacion">
    <input type="text" id="usuario" name="usuario" value="{{ dni }}" hidden>

    <label for="voto">Elija a un candidato:</label>
    <input type="radio" name="voto" id="voto" value="A" checked>Candidato A<br>
    <input type="radio" name="voto" id="voto" value="B">Candidato B<br>
    <input type="radio" name="voto" id="voto" value="C">Candidato C<br>
    <input type="radio" name="voto" id="voto" value="D">Candidato D

    <input type="submit" value="Votar">
  </form>
```

Con el código JavaScript, enviamos los datos del formulario a nuestra aplicación. Si el usuario no ha realizado ningún voto anteriormente, se procede a minar y añadir el bloque a la cadena. Una vez minado, se muestran los resultados actuales de las votaciones.

```
<script>
  document.getElementById("votacion").addEventListener("submit", function(event) {
    event.preventDefault();

    // Valores del formulario
    var voto = document.querySelector('input[name="voto"]:checked').value;
    var usuario = document.getElementById("usuario").value;

    // Datos a enviar
    var data = {
      usuario: usuario,
      voto: 'Candidato '+voto
    };

    // Enviar datos para añadir un bloque a la cadena
    fetch('/add_bloque', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    })

    // Respuesta
    .then(response => response.json())
    .then(data => {
      alert(data.message);
      window.location.href = '/resultado';
    })
    .catch(error => {
      alert('Error');
    });
  });
</script>
```



- **editar:** En este código se muestra un pequeño formulario en el que nos permite introducir el index del bloque a modificar y añadir el contenido que queramos. En este caso, elegir a otro candidato.

```
<body>
  <h1>Editar un bloque</h1>
  <form id="editar">
    <label for="index">Index del bloque a editar:</label>
    <input type="number" id="index" name="index">

    <label for="voto">Candidato:</label>
    <input type="text" id="voto" name="voto">

    <input type="submit" value="Editar Bloque">
  </form>
```

Con JavaScript enviamos los datos introducidos a nuestra aplicación para minar el bloque editado.

```
<script>
  document.getElementById('editar').addEventListener('submit', (event) => {
    event.preventDefault();

    // Valores del formulario
    const index = document.getElementById('index').value;
    const voto = document.getElementById('voto').value;

    // Enviar datos para modificar un bloque de la cadena
    fetch('/modify/'+index+'/'+voto)
    .then(response => {
      if (response.ok) {
        alert('Bloque modificado');
      } else {
        alert('Error')
      }
    })
    .catch(error => {
      alert('Error');
    });
  });
</script>
```

## 18. Conclusión

Después de llevar a cabo la investigación, mi conclusión es que Blockchain es una tecnología muy interesante que puede tener numerosas aplicaciones en distintas áreas.

El punto fuerte para implementar las cadenas de bloques es la seguridad y transparencia que ofrece, sobre todo en escenarios relacionados con la economía, patentes, propiedades y cadenas de suministro al permitir la creación de registros digitales inmutables y confiables.

Si se implementa de manera adecuada y se solucionan sus defectos, esta tecnología tiene el potencial para ser la base de las interacciones digitales seguras y confiables.

El desarrollo de inteligencias artificiales puede beneficiar a las cadenas de bloques y aumentar el desarrollo e implementación de Blockchain en empresas, gobiernos, etc.

## 19. Anexos

- API: Es un intermediario entre dos sistemas, que permite que una aplicación se comunique con otra y pida datos o acciones específicas.
- Block (Bloque): En informática, unidad de información que contiene datos.
  - Bloque génesis: Es el primer bloque de la cadena. Los siguientes bloques son añadidos a partir de él.
- Blockchain (Cadena de bloques): Secuencia continua de bloques enlazados mediante criptografía.
- Consenso: En informática, es el proceso mediante el cual los participantes de una Blockchain acuerdan el estado válido de la cadena.
- Criptografía: Rama de la informática encargada de la seguridad, integridad y autenticidad de los datos en una red.
- CSS: Lenguaje de diseño utilizado en desarrollo web para la presentación de un documento.
- Flask: Es un framework para crear aplicaciones web con Python.
- GitHub: Es una plataforma de alojamiento de repositorios y control de versiones.
- Hash: Función matemática que toma de entrada un dato y retorna una cadena de caracteres de longitud fija.
- HTML: Lenguaje de marcas utilizado para crear y estructurar contenido en páginas y aplicaciones web.
- JavaScript: Lenguaje de programación de alto nivel orientado a objetos utilizado principalmente en desarrollo web.
- Ledger (Libro Mayor): Es un libro, registro o sistema digital utilizado para almacenar y registrar transacciones financieras.
- Minado: En Blockchain, proceso de validación y registro de un bloque a la cadena.
- Nonce: En criptografía, es un número utilizado para operaciones criptográficas.
  - Golden Nonce: Es el número que permite añadir un bloque a la cadena.
- Nodo: Dispositivo conectado a una red.
- P2P: Red descentralizada diseñada para la compartición de recursos entre dispositivos de una red.

- Protocolo: Conjunto de normas que permiten la comunicación entre dispositivos de una red.
- Python: Lenguaje de programación de alto nivel utilizado para desarrollo web, inteligencia artificial, desarrollo de aplicaciones...
- Smart Contracts (Contratos Inteligentes): Programas informáticos cuyo objetivo es automatizar la verificación de los bloques de una cadena.
- Timestamp: Es una referencia temporal utilizado en informática.
  - Unix Timestamp: Representa los segundos transcurridos desde el 1 de enero de 1970 a las 0:00:00 UTC.

## 20. Bibliografía

- Academy, B. (2023). “¿Qué es SHA-256? Bit2Me Academy”. Disponible en <https://academy.bit2me.com/sha256-algoritmo-bitcoin/>
- BBVA. (2023). “Claves para entender la tecnología 'blockchain'”. Disponible en <https://www.bbva.com/es/innovacion/claves-para-entender-la-tecnologia-blockchain/>
- Escobero, G. (2022). “Redes peer-to-peer y tecnología blockchain”. Disponible en <https://www.teldat.com/es/blog/redes-peer-to-peer-blockchain-bitcoin/>
- Ethereum. (2023). “Ethereum guides”. Disponible en <https://ethereum.org/en/guides/>
- Fernández, Y. (2022). “Bitcoin, guía a fondo: qué es, cómo funciona y cómo conseguirlos”. Disponible en <https://www.xataka.com/basics/bitcoin-guia-a-fondo-que-como-funciona-como-conseguirlos>
- Flask. (s.f.). “Welcome to Flask — Flask Documentation (2.3.x)”. Disponible en <https://flask.palletsprojects.com/en/2.3.x/>
- Haber, S. y Scott Stornetta, W. (1991). “How to Time-Stamp a Digital Document”. Disponible en [http://www.staroceans.org/e-book/Haber\\_Stornetta.pdf](http://www.staroceans.org/e-book/Haber_Stornetta.pdf)
- Hurtado, J. S. (2023). “Qué es Blockchain y cómo funciona la tecnología Blockchain”. Disponible en <https://www.iebschool.com/blog/blockchain-cadena-bloques-revoluciona-sector-financiero-finanzas/>
- IBM. (s.f.). “¿Qué es la tecnología Blockchain?”. Disponible en <https://www.ibm.com/es-es/topics/blockchain>
- Inesdi. (2022). “Algoritmo de consenso en Blockchain: Protocolos y mecanismos”. Disponible en <https://www.inesdi.com/blog/algoritmo-consenso-blockchain>
- J2logo. (2022). “Tutorial de Flask en español: Desarrollando una aplicación web en Python”. Disponible en <https://j2logo.com/tutorial-flask-espanol/>
- Maldonado, J. (2020). “¿Qué es el nonce? Un número vital en Bitcoin”. Disponible en <https://es.cointelegraph.com/explained/what-is-the-nonce-a-vital-number-in-bitcoin>

- Mejía Herrera, D. S. y Múnera Sánchez, J. P. (2022). “*BlockID diseño de un sistema de votaciones basado en la tecnología blockchain*”. Disponible en <https://repositorio.utp.edu.co/server/api/core/bitstreams/4eb0a214-00c2-4403-bc1d-7f0fa54a328e/content>
- Nakamoto, S. (2008). “*Bitcoin: A Peer-to-Peer Electronic Cash System*”. Disponible en <https://bitcoin.org/bitcoin.pdf>
- Nastasache, A. (2023). “*Smart contracts: ¿Qué son, cómo funcionan y qué aportan?*”. Disponible en <https://academy.bit2me.com/que-son-los-smart-contracts/>
- Python (2023). “*Welcome to Python.org*”. Disponible en <https://www.python.org/doc/>
- Raúl (2022). “*Guía de protocolos de consenso de Blockchain*”. Disponible en <https://coinmotion.com/es/guia-protocolos-consenso-blockchain/>
- Rubio, J. (2022). “*Blockchain*”. Disponible en [https://dit.gonzalonazareno.org/gestiona/proyectos/2021-22/blockchain\\_jesus\\_rubio.pdf](https://dit.gonzalonazareno.org/gestiona/proyectos/2021-22/blockchain_jesus_rubio.pdf)
- Ruiz, A. (2022). “*Blockchain: Qué es y para qué sirve*”. Disponible en <https://ticnegocios.camaravalencia.com/servicios/tendencias/blockchain-que-es-y-que-ventajas-tiene/>
- Santander. (2022). “*Smart contracts, ¿qué son y para qué sirven?*”. Disponible en <https://www.santander.com/es/stories/smart-contracts#>