Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Información

Trabajo Fin de Grado

Comparativa de algoritmos hash para criptomonedas

Autor: Sergio Moreno Solera

Tutor: José María Gutiérrez Martínez

2023

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Información

Trabajo Fin de Grado

Comparativa de algoritmos hash para criptomonedas

Autor: Sergio Moreno Solera

Tutor: José María Gutiérrez Martínez

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA: 20/09/2023

RESUMEN

Este trabajo de fin de grado ha consistido en estudiar los distintos métodos de consenso utilizados en sistemas de criptomonedas. Con el resultado de este análisis se seleccionaron los algoritmos hashing que tenían mayor relevancia para el estudio deseado. A continuación, se determinaron ciertos parámetros de rendimiento aplicables a los algoritmos seleccionados, para finalmente reforzar este análisis comparativo con una demo demostrativa del funcionamiento de estos.

ABSTRACT

This undergraduate thesis has consisted of analyzing the different consensus methods used in cryptocurrency systems. Based on the results of this analysis, the hashing algorithms that were most relevant for the desired study were selected. Next, certain performance parameters applicable to the selected algorithms were determined, in order to finally reinforce this comparative analysis with a demonstrative demo of their functioning.

PALABRAS CLAVE

Criptomonedas, hash, método de consenso, algoritmo PoW, SHA-256, Blake2b, Scrypt.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

En la era digital en la que vivimos, las criptomonedas se han posicionado como una innovadora forma de intercambio de valor. Estas monedas digitales utilizan criptografía para garantizar la seguridad en las transacciones y controlar la creación de nuevas unidades. A diferencia de las monedas tradicionales, las criptomonedas se basan en la tecnología blockchain, que es un registro digital descentralizado e inmutable. Este registro recoge de manera transparente todas las transacciones realizadas con una criptomoneda en particular.

En los últimos años, las criptomonedas han experimentado un crecimiento significativo en su popularidad. Cada vez más personas y empresas confían en ellas debido a su seguridad, eficiencia y a las ventajas que ofrecen en comparación con los sistemas financieros tradicionales. Una de las principales características de las criptomonedas es su naturaleza descentralizada, lo que significa que no están controladas por entidades gubernamentales o bancos centrales. Esto proporciona a los usuarios una mayor autonomía y control sobre sus activos financieros.

Un aspecto fundamental en las criptomonedas es la creación de hash. Este proceso desempeña un papel crucial en la generación de nuevas unidades de la moneda y en la validación de las transacciones en la blockchain. La creación de hash consiste en convertir los datos de las transacciones y bloques en una cadena de caracteres alfanuméricos única. Esta cadena, conocida como hash, se utiliza para identificar y validar las transacciones o bloques, garantizando la integridad de la blockchain.

El proceso de creación de hash se lleva a cabo mediante el uso de algoritmos hash específicos. Estos algoritmos toman los datos de entrada y los procesan de manera única, generando un resultado irreproducible. La eficiencia en la creación de hash es de vital importancia en los sistemas de criptomonedas, ya que influye directamente en la rapidez con la que se pueden crear nuevas unidades y validar transacciones en la blockchain. Además, la eficiencia en la creación de hash está estrechamente relacionada con la seguridad y la resistencia de la blockchain ante posibles ataques o manipulaciones.

En este trabajo de fin de grado, se realizará un análisis exhaustivo de los distintos algoritmos utilizados en la creación de hash en los sistemas de criptomonedas. El objetivo principal será comparar estos algoritmos y establecer parámetros de rendimiento aplicables a cada uno de ellos. A través de esta investigación, se buscará comprender en profundidad la importancia de la creación de hash en las criptomonedas y cómo afecta tanto a la eficiencia como a la seguridad de estos sistemas.

Además del análisis teórico, se llevará a cabo una demostración práctica del funcionamiento de los algoritmos seleccionados. Esta demostración permitirá reforzar el análisis comparativo realizado y brindará una visión más completa sobre el rendimiento y la efectividad de los algoritmos de hash en el contexto de las criptomonedas.

2. OBJETIVOS

El **objetivo principal** de este TFG es **analizar los distintos algoritmos para la generación de hash usados en sistemas de criptomonedas, para comparar y establecer los parámetros de rendimiento aplicables a cada uno de ellos.**
Para la consecución del objetivo principal, se han establecido los siguientes **objetivos específicos:**

- Obtener una comparación de los algoritmos más utilizados para la generación de hash.
- Realizar una demo demostrativa de los parámetros encontrados en dos o más sistemas.
- Generar un análisis con los datos obtenidos en la ejecución de los algoritmos.

Para la consecución de estos objetivos se han determinado los siguientes **objetivos operativos:**

- Investigar las características de cada algoritmo de consenso.
- Analizar los distintos sistemas de criptomonedas existentes, realizando un estudio que recoja la relevancia de la eficiencia de los algoritmos hash.
- Seleccionar los algoritmos más relevantes para posteriormente determinar los parámetros de rendimiento que permitirán la comparación entre ellos.
- Diseñar pruebas para evidenciar los parámetros del análisis.

3. ESTADO DEL ARTE

Este tercer apartado de la memoria servirá para entender los distintos elementos que están involucrados en el campo de estudio del trabajo.

Primero en el apartado 3.1 se da una visión general del concepto de criptomoneda y se definen los conceptos más importantes.

Después en el apartado 3.2 nos adentramos en el concepto de hash, el cual se explica en detalle para comprender su funcionamiento y características.

El apartado 3.3 sirve como presentación de los apartados 3.4 y 3.5, en este punto se da una visión general del gran número de criptomonedas y algoritmos que existen, listando los algoritmos de consenso más importantes.

Una vez introducidos los algoritmos en el apartado 3.3, pasamos a definir de manera breve cada uno de ellos para poder entender su funcionamiento. En el apartado 3.4 se encuentran los métodos de consenso más comunes, que guardan relación entre ellos, y en el apartado 3.5 se explican los algoritmos con menos relevancia y que no guardan casi similitudes entre ellos. Con esta visión del funcionamiento de cada método de consenso podremos determinar más adelante cuales tienen una generación de hash más interesante para el estudio de este trabajo.

3.1 CRIPTOMONEDAS

Las criptomonedas son monedas digitales que utilizan la criptografía para garantizar la seguridad en las transacciones y controlar la creación de nuevas unidades. Se basan en la tecnología blockchain, que es un registro digital inmutable y descentralizado que registra todas las transacciones de una criptomoneda en particular.

El uso de criptomonedas ha ido en aumento en los últimos años, ya que cada vez son más las personas y empresas que confían en su seguridad y eficiencia para realizar transacciones. Además, las criptomonedas ofrecen una alternativa a los sistemas financieros tradicionales, que suelen estar controlados por bancos y entidades gubernamentales. Uno de los aspectos más importantes en las criptomonedas es la creación de hash, que es un proceso fundamental en la creación de nuevas unidades de la moneda y en la validación de las transacciones en la blockchain.

El proceso de creación de hash en criptomonedas se utiliza para convertir los datos de las transacciones y bloques en una cadena de caracteres alfanuméricos única, que se utiliza para identificar y validar la transacción o bloque. Esto se logra mediante el uso

de una función hash, que toma los datos de entrada y los procesa de tal manera que produce un resultado único e irreproducible. La eficiencia en la creación de hash es crucial en los sistemas de criptomonedas, ya que afecta directamente a la rapidez con la que se pueden crear nuevas unidades y validar transacciones en la blockchain. Además, la eficiencia en la creación de hash también está relacionada con la seguridad y la resistencia de la blockchain a posibles ataques o manipulaciones.

Por lo tanto, en este trabajo de grado se analizarán los distintos algoritmos utilizados en la creación de hash en sistemas de criptomonedas, con el objetivo de compararlos y establecer parámetros de rendimiento aplicables a cada uno de ellos. Con los resultados obtenidos, se realizará una demostración práctica de algunos de los parámetros encontrados en dos o más sistemas.

3.1.1 Blockchain

Blockchain (también denominado cadena de bloques) es la tecnología que permite la inmutabilidad e integridad de los datos. Es una base de datos donde la información almacenada no se puede modificar ni eliminar, esto se debe a que todas las transacciones que se producen dentro de una red quedan registradas y existen mineros (nodos en la red) que se encargan de validar y procesar estas transacciones realizadas para agregarlas a la cadena de bloques.

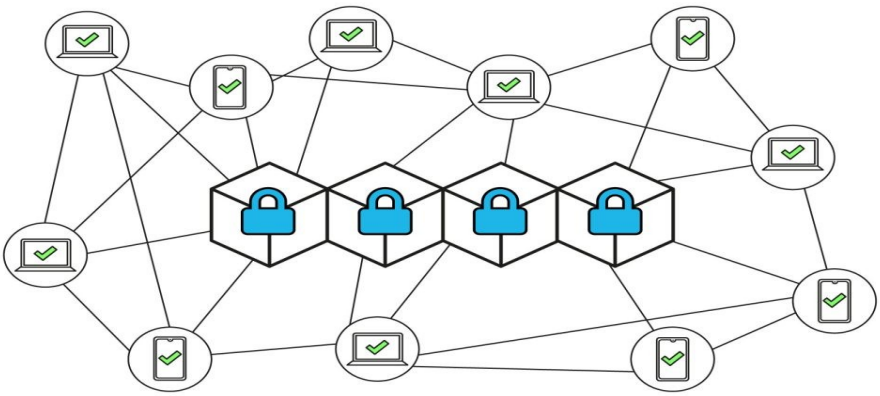

Ilustración 1. Blockchain (tomada de https://cryptoconexion.com/blockchain/)

3.1.2 Mineros

Los mineros de blockchain son nodos en la red de una criptomoneda que se encargan de validar y procesar transacciones realizadas en la red. En los sistemas de criptomonedas que serán objetos de este estudio los mineros tienen que resolver complejos algoritmos criptográficos que promueven la validez de las transacciones y en caso de conseguirlo recibirán una recompensa en forma de criptomoneda. Sin embargo, en otros sistemas la validación de transacciones no requiere un esfuerzo computacional como veremos en el funcionamiento de los distintos métodos/algoritmos de consenso.

Este proceso de validar las transacciones es esencial para garantizar la seguridad e integridad en la cadena de bloques. Este proceso varía entre los distintos sistemas de criptomonedas ya que existen diferentes métodos de consenso que son utilizados, y lo mineros de cada red son los encargados de que se cumplan las reglas del consenso.

3.1.3 Algoritmos de Consenso

Los algoritmos son conjuntos de reglas o procedimientos que se siguen para resolver un problema o realizar una tarea específica. En el contexto de las criptomonedas, se utilizan algoritmos (denominados algoritmos de consenso) para crear y validar las transacciones y para generar nuevas monedas.

Los algoritmos de consenso son los mecanismos utilizados en las redes blockchain para garantizar que todos los nodos están de acuerdo con las transacciones realizadas, dando validez a la transacción. La integridad de la cadena de bloques se mantiene gracias a la generación de hashes asociados a los bloques que se van añadiendo, cada método de consenso tiene su propio algoritmo.

3.1.4. Hash

Un **hash es una función matemática que toma una entrada de cualquier tamaño y la convierte en una salida de tamaño fijo**. Se utiliza en las criptomonedas para verificar la integridad de los datos y para generar un "sello digital" único para cada transacción. Es decir, es lo que ofrece seguridad y da validez a las transacciones en la cadena de bloques de la criptomoneda que corresponda.

3.2 ¿QUÉ ES UN ALGORITMO HASH?

Una vez hemos definido los términos claves de este trabajo podemos profundizar más en detalle para poder comprender la utilidad que tienen los algoritmos hash. Para ello a continuación se hará una explicación en este apartado de las características de un algoritmo hash y después estudiaremos la metodología de los algoritmos de consenso más importantes.

Para poder analizar los distintos algoritmos usados para la generación de hash, el primer paso será definir el significado de estos algoritmos. El concepto de algoritmo está más comúnmente asociado a la informática, sin embargo, **los algoritmos son la serie de pasos que todos utilizamos a diario para llegar a la solución de un problema**, por ejemplo, cuando leemos las instrucciones para armar un mueble o una receta de cocina. **Un algoritmo informático es un conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema o para realizar una tarea.** En el ámbito que nos compete de la programación un algoritmo supone el paso previo para **escribir un código, para encontrar la solución a un problema (por ejemplo, los que se solucionan para minar un bloque).** Se define un algoritmo para que después el código pueda indicarle a la computadora las acciones que debe llevar a cabo.

Estas son las partes que constituyen un algoritmo:

- Input o entrada: Es la información que damos al algoritmo para que trabaje y ofrezca la solución al problema.
- Proceso: Este es el conjunto de pasos para que a partir de la entrada llegue a la solución.
- Output o salida: Aquí obtenemos el resultado a partir de la información de los valores de la entrada y que son utilizados en el proceso.

Asimismo, los algoritmos son procesos con características en común, las cuales son:

- Precisos: Objetivos, es decir no son ambiguos.
- Ordenados: presentan una secuencia clara y precisa para llegar a la solución.
- Finitos: Siempre tienen un número determinado de pasos.
- Concretos: Ofrecen una solución determinada para un problema planteado.
- Definidos: Los resultados no son variables cuando se da una entrada específica.

Una vez vista la definición general de algoritmo, nos centramos en los algoritmos para la generación de un hash. **El hashing, o hash, es un término frecuente cuando se**

habla de la tecnología de blockchain. El hashing se refiere a la transformación y generación de datos de entrada de cualquier longitud en una cadena de tamaño fijo, que se realiza mediante un algoritmo específico.

Un algoritmo de hashing toma un número infinito de bits, realiza cálculos y genera un número fijo de estos bits. Más allá de la longitud de los datos de entrada, la salida siempre será rectificada. Por lo tanto, los datos originales se denominan de **entrada** y la transformación final se llama **hash**. En la actualidad, muchos algoritmos de hashing se diferencian solo en la forma de procesar la información, y es el punto clave que atañe este trabajo.

Concretamente se van a estudiar los distintos algoritmos de hash criptográficos, lo que ya hemos denominado previamente como algoritmos de consenso y que deben de cumplir una serie de propiedades que se describen a continuación:

- **Determinista:** Indica que para un mismo conjunto de datos introducidos siempre se ha de obtener el mismo valor del código hash. Está propiedad es muy importante, ya que si el código hash fuese diferente cada vez sería imposible emplearlos para verificar que los datos de origen son los mismos.

- **Computacionalmente eficiente:** El algoritmo ha de ser computacionalmente eficiente. Esto significa que la función ha de obtener el valor del código hash rápidamente. En el caso de que el proceso no sea lo suficientemente rápido, las aplicaciones desarrolladas simplemente no serán eficientes.

- **No reversible:** No ha de existir una forma directa de obtener el conjunto de datos original a partir del código hash. Aunque siempre existe la posibilidad de conseguirlo indirectamente, mediante un ataque de fuerza bruta se puede buscar el conjunto de datos que genere el código hash de forma indirecta.

- **Pequeños cambios en el input cambian completamente el código hash**: Al realizar un pequeño cambio en la entrada, por ejemplo, un solo bit, los cabios en el código hash han de ser enormes. Esto significa que los algoritmos han de ser sensibles a cualquier mínimo cambio en el conjunto de datos de entrada.

- **Resistente a colisiones:** Mediante la resistencia a colisiones se hace referencia a que la probabilidad de que dos conjuntos de datos diferentes generen el mismo código hash debe ser ínfima. Esta propiedad nos indica que si obtenemos el mismo código hash para dos conjuntos de datos podemos asegurar con una gran probabilidad que ambos son el mismo. Por ejemplo, al generar el hash de dos contraseñas podemos asegurar que ambas son la misma cuando ambos códigos coinciden. El problema de la colisión no se puede evitar completamente. Esto es debido a que el tamaño de los códigos hash es finito y,

por lo tanto, el número de posibilidades esta acotado. Por otro lado, el conjunto de datos de entrada es prácticamente infinito.

3.3 MARCO TEÓRICO

Existen 23.022 criptomonedas según CoinMarketPlace con fecha 23/02/2023, cada una de ellas sigue un algoritmo de consenso. Los algoritmos de consenso son el método a través de cual en una determinada criptomoneda se validan y añaden a la blockchain las transacciones realizadas en la red. Entre los distintos algoritmos o métodos de consenso podemos clasificar las criptomonedas y los algoritmos de hashing que utilizan. Muchos de los algoritmos hash son compartidos entre distintas criptomonedas, incluso algunas combinan varios algoritmos para producir el hash. Los mineros o validadores son los encargados de descifrar el hash correcto, asegurando el funcionamiento de la cadena de bloques, procesando transacciones y recibiendo recompensas en forma de monedas de una criptomoneda en particular.

Partiendo de esta visión general de las criptomonedas y el funcionamiento de las cadenas de bloque, es momento de profundizar y ver en qué consisten los métodos de consenso más relevantes que existen. La producción de hash en cada uno de ellos no tiene la misma relevancia. Por tanto, una vez comprendido el funcionamiento de los algoritmos de consenso más relevantes se realizará una descripción de cada uno de ellos, poniendo especial atención en la prueba de trabajo ya que observaremos que es el método de consenso del cual nos interesa analizar los algoritmos hashing por el contexto de este estudio.

A continuación, se listan los algoritmos de consenso más relevantes que hemos recogido en el estudio:

- Proof of Work (PoW)
- Proof of Stake (PoS)
- Delegated Proof of Stake (DPoS)
- Leased Proof of Stake (LPoS)
- Proof of Elapsed Time (PoET)
- Practical Byzantine Fault Tolerance (PBFT)
- Simplified Byzantine Fault Tolerance (SBFT)
- Delegated Byzantine Fault Tolerance (DBFT)
- Directed Acyclic Graphs (DAG)

- Proof-of-Activity
- Proof-of-Importance
- Proof-of-Capacity
- Proof-of-Burn
- Proof-of-Weight

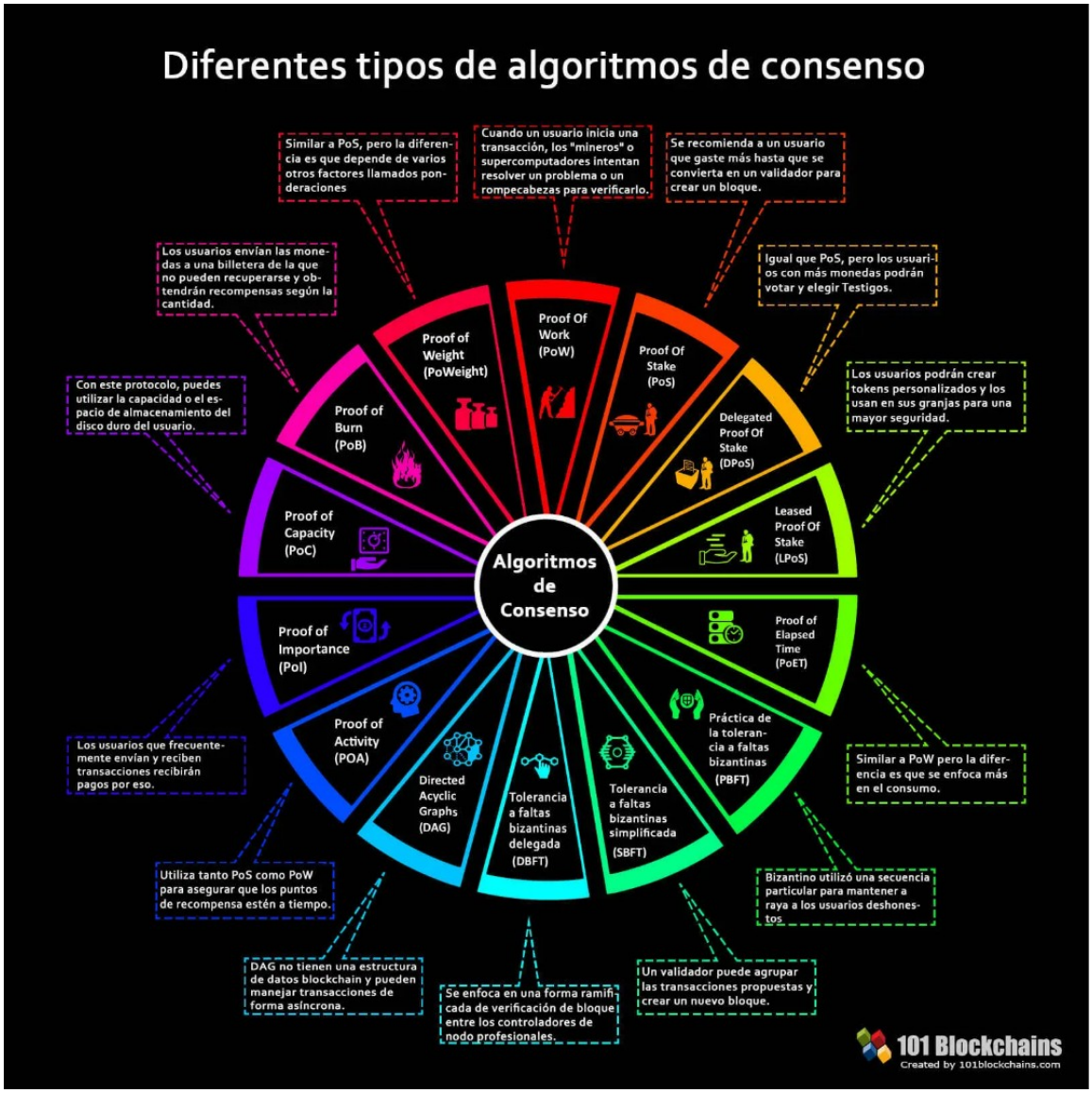

Ilustración 2. Algoritmos de consenso (tomada de: https://101blockchains.com/es/algoritmos-de-consenso-blockchain/)

3.4 ALGORITMOS DE CONSENSO

Los algoritmos de consenso son procesos de toma de decisiones para un grupo, donde cada individuo dentro del grupo construye y apoya la decisión que funcione mejor para ellos. Es una forma de resolución donde los miembros deben apoyar la decisión mayoritaria, les guste o no.

Los algoritmos de consenso no solo están de acuerdo con la mayoría de los votos, sino que también están de acuerdo con aquel que los beneficie a todos. Así que, siempre es una victoria para la red. Los modelos de consenso de blockchain son métodos usados para crear igualdad y equidad. Los sistemas de consenso utilizados para este acuerdo se denominan teoremas de consenso. Estos modelos de consenso de blockchain consisten en algunos objetivos particulares, tales como:

- **Llegar a un acuerdo:** El mecanismo reúne todos los acuerdos del grupo tanto como puede.
- **Colaboración:** Cada uno en el grupo apunta a un mejor acuerdo que resulte en los intereses colectivos del grupo.
- **Cooperación:** Cada miembro trabajara en equipo y dejara de lado sus propios intereses.
- **Igualdad de derechos:** Cada uno de los participantes tiene el mismo valor en la votación. Esto significa que el voto de cada persona es importante.
- **Participación:** Todos los que están dentro de la red deben de participar en la votación. Nadie puede quedarse fuera de la votación.
- **Actividad:** Cada miembro del grupo es igualmente activo. No hay miembros con más responsabilidades que otros en el grupo.

Una vez definidas las características generales de los algoritmos/métodos de consenso, se va a describir de manera resumida el funcionamiento de los algoritmos más usados para poder entender porque serán o no elegidos los algoritmos hashing usados en los sistemas de criptomonedas que hagan uso de ellos.

3.4.1 Proof of Work (PoW) – Prueba de Trabajo

La Prueba de Trabajo es el primer algoritmo de consenso utilizado en la tecnología blockchain. Muchas criptomonedas utilizan este modelo de consenso para confirmar las transacciones y producir bloques relevantes en la red. La responsabilidad de la generación de bloques y la confirmación de las transacciones recae en los nodos individuales llamados mineros. El principio central de la PoW es resolver problemas matemáticos complejos para producir soluciones de manera eficiente. Sin embargo, la PoW tiene ciertas limitaciones, ya que el aumento de la red exige más poder computacional, lo que aumenta la sensibilidad general del sistema.

¿Por qué el aumento de la red conlleva un mayor poder computacional? Esto se debe a que la dificultad del problema matemático está ajustada para que se resuelva en un tiempo determinado, por ejemplo, en el caso de Bitcoin es de aproximadamente cada 10 minutos, independientemente de cuántos mineros estén participando por tanto para que ese tiempo se mantenga estable la dificultad que conlleva resolver el problema aumentará paralelamente al aumento de la red. Como resultado, el aumento en la competencia y la potencia de cómputo necesaria puede tener algunas implicaciones y limitaciones:

Consumo de energía: El aumento en el poder computacional requerido para la PoW puede llevar a un mayor consumo de energía. Los equipos de minería necesitan una gran cantidad de electricidad para resolver los problemas matemáticos, lo que puede resultar en un impacto ambiental significativo y altos costos energéticos.

Centralización del poder: A medida que la red se vuelve más competitiva y costosa de operar, la minería de criptomonedas tiende a centralizarse en manos de unos pocos actores con recursos considerables. Esto puede llevar a una concentración de poder en ciertos grupos o regiones, lo que va en contra del principio de descentralización de muchas criptomonedas.

Tiempo de confirmación de transacciones: A medida que la red crece, el aumento en la potencia computacional necesaria puede llevar a un mayor tiempo de confirmación de transacciones. Esto se debe a que los bloques se generan aproximadamente cada 10 minutos, y si hay muchos usuarios compitiendo por agregar transacciones a la cadena, podría haber congestión y retrasos en la confirmación de las transacciones.

La secuencia de consenso de la blockchain depende de datos e informaciones precisas, lo que significa que la velocidad del sistema es crítica. Si un problema se vuelve demasiado complejo, se tardará más tiempo en generar un bloque, lo que puede retrasar las transacciones y detener el flujo de trabajo. Por otro lado, si el problema es demasiado fácil, el sistema será propenso a los ataques DdoS (denegación distribuida de servicio). Además, la solución debe ser verificada de manera precisa para que la red mantenga su característica principal: la transparencia.

La PoW se implementa mediante la resolución de problemas matemáticos complejos por parte de los mineros. Después de resolver un problema, se crea un nuevo bloque y se confirman las transacciones. Los bloques contienen la función hash del bloque anterior, lo que agrega una capa adicional de protección y previene cualquier tipo de violación. En la siguiente ilustración se describe a alto nivel el funcionamiento de PoW.

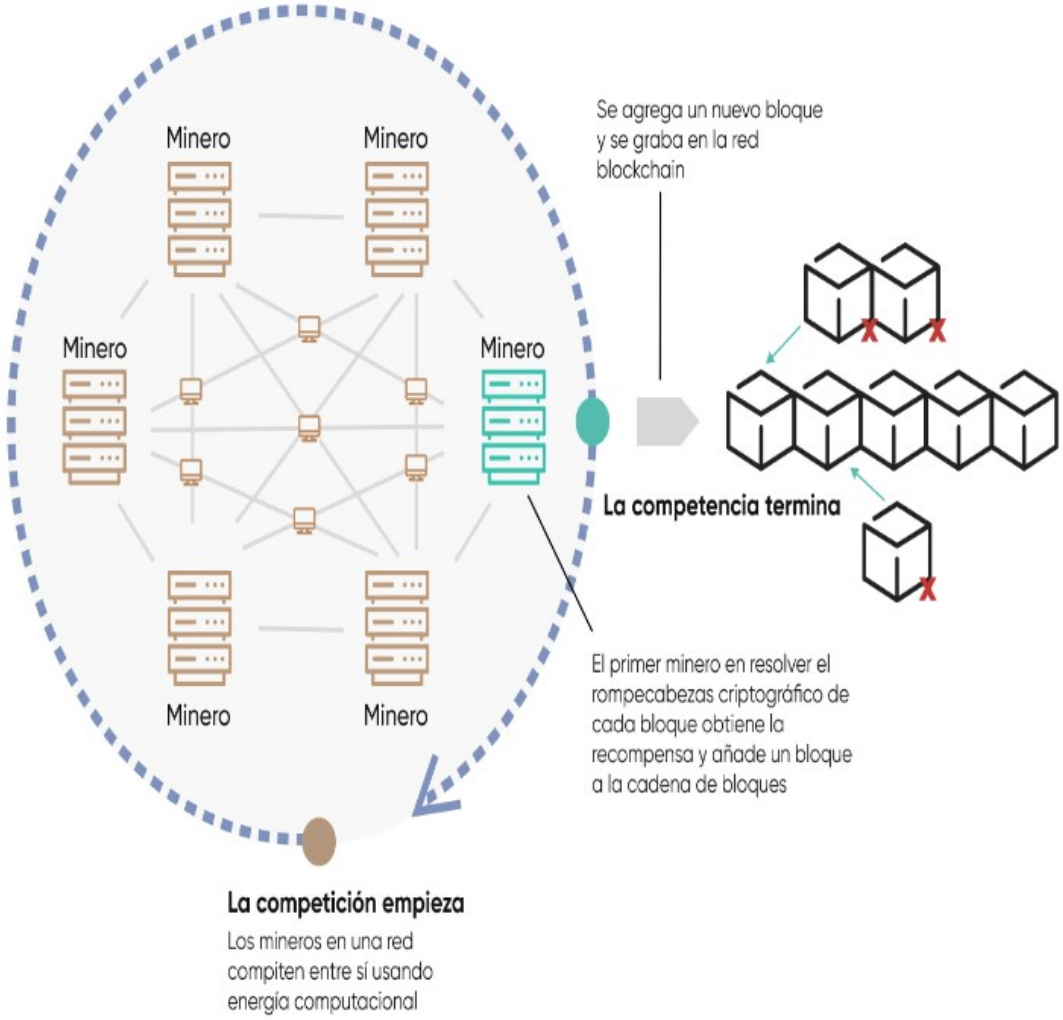

Ilustración 3. ¿Cómo funciona PoW? (tomada de https://capital.com/es/20-terminos-criptomonedas)

El algoritmo de consenso PoW es ampliamente utilizado en criptomonedas como Bitcoin y Litecoin. Ethereum también lo utilizó en algunos proyectos, aunque ha pasado a la Prueba de Participación (PoS). La PoW ofrece protección DDoS y reduce el impacto de la participación minera, lo que la convierte en una opción segura para la tecnología blockchain.

Sin embargo, la PoW tiene algunos problemas, como el gran consumo de energía que requiere para mantener la red, lo que puede ser costoso para los mineros y un problema para el mundo, que se enfrenta a una escasez de energía. Además, la PoW puede llevar a la centralización de mineros, lo que es otro gran problema para la red descentralizada.

3.4.2 Proof of Stake (PoS) --Prueba de participación

La Prueba de Participación es un algoritmo de consenso utilizado en la tecnología blockchain que asigna la tarea de validar transacciones a los participantes que poseen una cantidad determinada de criptomoneda. Estos participantes, conocidos como validadores, son elegidos aleatoriamente por el protocolo para agregar bloques a la cadena de bloques y validar transacciones en la red. El depósito de criptomoneda que se requiere para convertirse en un validador actúa como garantía y el validador recibe una recompensa por su trabajo. Si un validador actúa de manera malintencionada, puede perder su depósito como sanción. Por tanto, la Prueba de Participación utiliza la cantidad de criptomonedas poseídas por un participante como medida de su capacidad para validar transacciones y crear nuevos bloques en la cadena de bloques. A continuación, se muestran las diferencias más relevantes en la mecánica de PoS comparado con PoW:

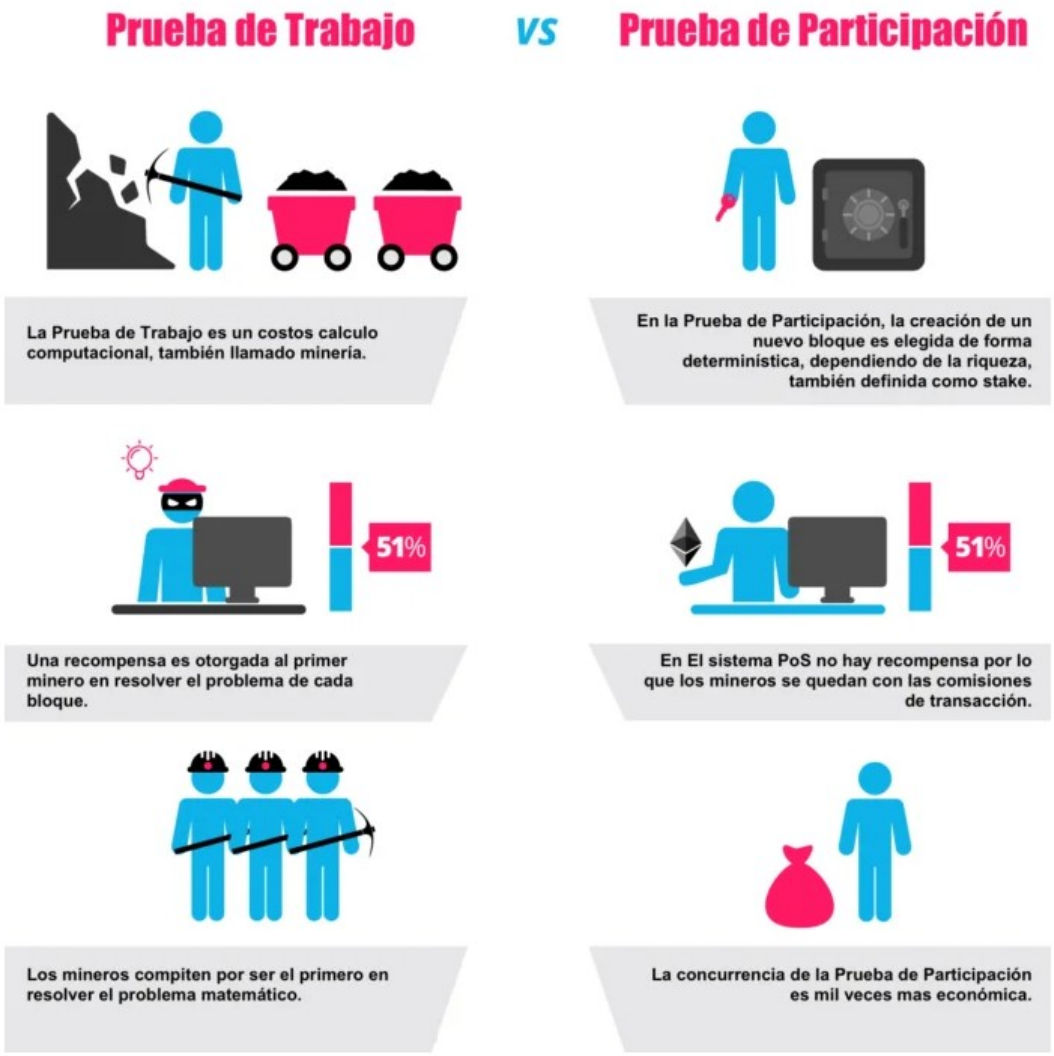

Ilustración 4. Comparación entre PoW y PoS.

3.4.3 Delegated Proof of Stake (DPoS) -- Prueba de Participación Delegada

La Prueba de Participación Delegada es una variante del algoritmo de Prueba de Participación. En DPoS, los participantes de la red eligen delegados para validar transacciones y agregar bloques a la cadena de bloques en su nombre. Estos delegados son elegidos mediante votación y se les asigna una cantidad determinada de criptomoneda como garantía.

Los delegados reciben una recompensa por validar transacciones y agregar bloques a la cadena de bloques en nombre de los participantes que los eligieron. Si un delegado actúa de manera malintencionada, como intentar validar transacciones falsas, puede perder su garantía y ser removido de su posición.

La Prueba de Participación Delegada se utiliza comúnmente en cadenas de bloques con grandes cantidades de usuarios y transacciones, ya que permite una mayor escalabilidad y eficiencia en comparación con la Prueba de Trabajo o la Prueba de Participación tradicionales

3.4.4 Leased Proof os Stake (LpoS) -- Prueba de participación arrendada

La prueba de participación arrendada es otra variante de la prueba de participación que busca resolver las limitaciones de la prueba de participación clásica. Introducida por la plataforma Waves, LPoS utiliza muy poca energía y garantiza una funcionalidad superior.

En la prueba de participación original, las personas con una cantidad limitada de monedas a menudo no podían participar, lo que llevaba a la creación de comunidades centralizadas dentro de una plataforma descentralizada. LPoS aborda este problema permitiendo que los pequeños propietarios de monedas arrienden sus monedas a la red y obtengan beneficios de esta manera.

Desde su introducción, la prueba de participación arrendada ha resuelto las limitaciones del sistema anterior y ha permitido que los pequeños inversionistas participen en igualdad de condiciones. Esto establece la transparencia como el tema principal de los algoritmos de consenso, ya que todos tienen la oportunidad de obtener beneficios en igualdad de condiciones.

3.4.5 Proof of Elapsed Time (PoET) -- Prueba de tiempo transcurrido

El algoritmo de Prueba de Tiempo Transcurrido (PoET), es un algoritmo de consenso que se utiliza principalmente en redes blockchain privadas que requieren permiso para acceder a la red. En estas redes privadas, es necesario decidir sobre los derechos de minería o los principios de votación.

Para garantizar la transparencia en la red, PoET utiliza una táctica específica. Además, los algoritmos de consenso también garantizan un inicio de sesión seguro en el sistema, ya que la red requiere una identificación antes de permitir que los mineros se unan.

Este algoritmo de consenso proporciona la oportunidad de elegir de manera justa a los ganadores. La estrategia principal de PoET implica que cada persona en la red debe esperar una cantidad aleatoria de tiempo antes de que el participante que haya terminado su parte justa de tiempo de espera esté en el registro para crear un nuevo bloque.

Para justificar estos escenarios, el algoritmo debe considerar dos cosas: en primer lugar, si el ganador eligió un tiempo aleatorio y no un tiempo más corto para obtener la victoria, y, en segundo lugar, si la persona realmente esperó el tiempo que se le asignó.

3.4.6 Practical Byzantine Fault Tolerance (PBFT) -- Práctica de la tolerancia a faltas bizantinas

La Tolerancia a Faltas Bizantinas es otro método que se utiliza para garantizar la seguridad y confiabilidad de la red. A diferencia de la Prueba de Trabajo (PoW) y la Prueba de Participación (PoS), la PBFT se basa en la cooperación de los validadores para llegar a un acuerdo sobre el estado de la red y las transacciones realizadas. En este algoritmo, cada validador tiene el mismo peso en el proceso de toma de decisiones y la red puede tolerar hasta un tercio de los validadores que fallen o actúen de manera malintencionada sin afectar significativamente el consenso. La Tolerancia a Faltas Bizantinas es una práctica de ingeniería utilizada en sistemas críticos donde la confiabilidad y la integridad son fundamentales para la seguridad y el éxito del sistema

3.4.7 Simplified Byzantine Fault Tolerance (SBFT) -- Tolerancia a faltas bizantinas simplificada

La Tolerancia de Prácticas Bizantinas Simplificadas es un método de consenso simplificado que se utiliza en criptomonedas para garantizar la seguridad y confiabilidad de la red en presencia de fallas o ataques malintencionados. En el SBFT, un conjunto más pequeño de nodos validadores llega a un acuerdo sobre el estado de la red y las transacciones realizadas, lo que simplifica el proceso de toma de decisiones. A diferencia de la Prueba de Participación, el SBFT no otorga una ventaja adicional a los validadores con una mayor cantidad de criptomoneda, ya que todos los validadores tienen un peso igual en el proceso de toma de decisiones.

3.4.8 Delegated Byzantine Fault Tolerance (DBFT) -- Tolerancia a Faltas Bizantizas delegada

La Tolerancia a Faltas Bizantinas Delegada es un algoritmo utilizado para garantizar que la red pueda operar de manera segura y confiable incluso en presencia de fallas o ataques malintencionados. En dBFT, los nodos validadores son seleccionados para participar en el proceso de toma de decisiones y votan en cada ronda de consenso para

llegar a un acuerdo sobre el estado de la red y las transacciones realizadas. A diferencia de otros algoritmos de consenso, como la PoW o la PoS, el dBFT es más eficiente en términos de energía y no requiere un alto poder de procesamiento. Además, el dBFT permite la delegación de votos, lo que significa que los poseedores de criptomonedas pueden delegar su derecho a votar en un nodo validador de su elección, lo que les permite participar en el proceso de consenso sin necesidad de ejecutar un nodo completo.

3.4.9 Directed Acyclic Graphs (DAG) -- Grafo acíclico dirigido

El Grafo Acíclico Dirigido es un método que utiliza una estructura de grafo acíclico dirigido para organizar las transacciones en la red. En el DAG, cada transacción se representa como un nodo en el grafo, y las transacciones que dependen de otras transacciones se conectan a ellas mediante arcos dirigidos. Cada nodo del DAG es validado y confirmado por otros nodos en la red, lo que crea una cadena de bloques parcialmente ordenada en la que no todas las transacciones tienen el mismo nivel de validez. Los nodos validadores en el DAG no necesitan resolver problemas matemáticos complejos o poseer grandes cantidades de criptomonedas para validar transacciones, lo que lo hace más accesible y eficiente que otros algoritmos de consenso como PoW y PoS.

3.5 OTROS TIPOS DE ALGORITMOS DE CONSENSO

Este apartado 3.5 continua el apartado inmediatamente anterior (3.4), sin embargo se ha querido hacer una diferenciación debido a que los algoritmos que se explican a continuación tiene diferencias reseñables en forma, ya que los recogidos en el punto anterior encuentran similitudes en su método y estos cinco métodos de consenso no guardan relación.

3.5.1 Proof of Activity (PoA) – Prueba de Actividad

La Prueba de Actividad es un método de consenso que combina PoW y PoS para lograr un consenso descentralizado y eficiente. En PoA, los mineros primero deben demostrar que han llevado a cabo una cantidad significativa de trabajo computacional en un bloque de transacciones específico, similar a PoW. Una vez que se completa el trabajo, se utiliza la prueba de participación para validar y confirmar el bloque, similar a PoS. La validación de bloques en PoA es más rápida y eficiente que en PoW, ya que no es necesario resolver problemas matemáticos complejos. Además, el PoA es más seguro que el PoS ya que requiere que los mineros realicen trabajo computacional, lo que evita que un nodo malintencionado tome el control de la red.

3.5.2 Proof of Importance (PoI) – Prueba de Importancia

La Prueba de Importancia asigna importancia a los usuarios según su participación y actividad en la red. En PoI, los usuarios con una mayor participación y actividad en la

red tienen más peso en la toma de decisiones y en la validación de transacciones. La importancia se calcula en función de varios factores, como el número de tokens que posee un usuario, la cantidad de transacciones realizadas y la cantidad de nodos a los que está conectado. Al asignar importancia a los usuarios, PoI fomenta la participación y la actividad en la red, lo que promueve una mayor descentralización y seguridad. Además, PoI es más escalable y eficiente que otros métodos de consenso, como PoW, ya que no requiere de grandes cantidades de poder de procesamiento.

3.5.3 Proof of Capacity (PoC) – Prueba de Capacidad

En la Prueba de Capacidad, los mineros demuestran que tienen una cierta cantidad de espacio de almacenamiento disponible. Este espacio se utiliza para almacenar una gran cantidad de datos precalculados y encriptados llamados "tramas de parcelas". Los mineros seleccionan aleatoriamente un número de tramas y realizan cálculos en ellas para demostrar que tienen acceso al espacio de almacenamiento y que han invertido un cierto esfuerzo en crear las tramas. Los mineros que demuestran que tienen acceso a la cantidad necesaria de espacio de almacenamiento y que han invertido suficiente esfuerzo en la creación de tramas tienen más posibilidades de agregar un nuevo bloque a la cadena de bloques y recibir una recompensa.

3.5.4 Proof of Burn (PoB) – Prueba de Quemado

La prueba de quemado implica la destrucción intencional de criptomonedas para demostrar la inversión en la red. En PoB, los participantes de la red envían sus tokens a una dirección específica, lo que los quema o destruye permanentemente. Esta acción demuestra el compromiso del participante con la red y se utiliza como una forma de votar en el proceso de toma de decisiones en la red. Cuanto más se "queman" los tokens, más influencia tiene el participante en la red. Además, PoB reduce la oferta total de tokens en circulación, lo que puede aumentar el valor de las criptomonedas restantes. Sin embargo, PoB también puede tener limitaciones, como la necesidad de incentivar a los participantes para que "quemen" sus tokens y la falta de participación de los inversores a largo plazo que no quieren perder sus criptomonedas. En resumen, PoB es un método de consenso innovador que se centra en la demostración de inversión en la red mediante la destrucción de tokens, lo que puede promover la toma de decisiones y reducir la oferta total de tokens en circulación.

3.5.5 Proof of Weight (PoWeight) – Prueba de peso

Prueba de peso es un método de consenso utilizado en criptomonedas que se basa en el concepto de "peso" de los nodos en la red. A diferencia de otros métodos de consenso, como PoW o PoS, PoWeight se centra en la importancia de los nodos individuales en lugar de la cantidad de poder de procesamiento o la cantidad de tokens que poseen. En PoWeight, cada nodo de la red tiene un peso específico asignado en función de su importancia y participación en la red. Este peso se utiliza para determinar la probabilidad de que el nodo sea elegido para validar transacciones en la red. Cuanto mayor sea el peso de un nodo, mayor será la probabilidad de que se seleccione para

validar las transacciones. El peso se puede calcular utilizando diferentes factores, como la cantidad de tokens que posee un nodo, la cantidad de transacciones realizadas o la cantidad de nodos a los que está conectado. PoWeight es un método de consenso escalable y eficiente que fomenta la participación y la actividad en la red, lo que puede mejorar la seguridad y la descentralización de la red de criptomonedas.

4. DESARROLLO

Una vez estudiados los métodos de consenso más relevantes en criptomonedas descritos en el apartado anterior, se ha determinado que los algoritmos hashing que utilizan la prueba de trabajo (PoW) serán objeto de estudio en este trabajo. Esta elección se debe a que, aunque en todos los sistemas de criptomonedas la generación de hash es crucial para la creación de bloques seguros, es en PoW donde se requiere un esfuerzo significativo para generar hash y, por tanto, los parámetros de rendimiento que son clave en este trabajo se centran en este método de consenso.

4.1 ALGORITMOS SELECCIONADOS

El objetivo de esta sección es analizar en detalle los algoritmos hash más relevantes utilizados en criptomonedas PoW, con el fin de entender cada uno de ellos y poder comparar su rendimiento.

Para ello, se describirá el funcionamiento de cada uno de los algoritmos seleccionados, se evaluará su seguridad y su eficiencia en términos de tiempo y energía necesarios para su ejecución. Además, se discutirán las ventajas y desventajas de cada algoritmo en términos de su resistencia a los ataques de minería y su escalabilidad.

4.1.1 SHA-256

El algoritmo SHA-256 (Secure Hash Algorithm 256 bits) es una función hash criptográfica que se usa en distintos sistemas de criptomonedas, como Bitcoin. Fue desarrollado por la Agencia de Seguridad Nacional de Estados Unidos (NSA) en 2001 y se ha convertido en uno de los algoritmos de hash más populares y seguros.

El algoritmo SHA-256 toma un mensaje de entrada de longitud variable y lo convierte en un hash de 256 bits de longitud fija. El hash es único para el mensaje de entrada y se utiliza para verificar la integridad de los datos. El proceso de creación del hash implica la aplicación de una serie de operaciones criptográficas a los datos de entrada. El resultado de cada operación se utiliza como entrada para la siguiente operación, lo que garantiza que cualquier cambio en los datos de entrada produzca un hash completamente diferente. En la siguiente imagen se puede apreciar en detalle las operaciones que realiza el algoritmo:

Ilustración 5. Diagrama funcionamiento SHA-256 (tomada de https://sha256algorithm.com/)

En el contexto de las criptomonedas, SHA-256 se utiliza en el proceso de minería, que es el proceso de validar transacciones y crear nuevos bloques en la cadena de bloques de una criptomoneda. Los mineros resuelven un problema matemático complejo que requiere una gran cantidad de poder de procesamiento para producir un hash que cumpla con un cierto requisito de dificultad, que generalmente implica que el hash debe comenzar con un número determinado de ceros.

El algoritmo SHA-256 se utiliza en varias criptomonedas importantes, como Bitcoin, Bitcoin Cash y Syscoin. En Bitcoin, por ejemplo, los mineros utilizan SHA-256 para encontrar un hash que cumpla con un requisito de dificultad específico para crear un nuevo bloque en la cadena de bloques. La dificultad se ajusta automáticamente para garantizar que los bloques se creen a una velocidad constante y predecible.

Las ventajas del algoritmo SHA-256 incluyen su seguridad y resistencia a la colisión, lo que significa que es extremadamente difícil encontrar dos mensajes de entrada diferentes que produzcan el mismo hash. Además, el algoritmo es rápido y eficiente, lo

que lo hace ideal para su uso en sistemas de criptomonedas que requieren una alta velocidad de procesamiento de hash.

Sin embargo, una posible desventaja del algoritmo es que su uso intensivo de recursos lo hace más propenso a los ataques de minería de ASIC, que son dispositivos de hardware especializados diseñados para el minado de criptomonedas. Estos dispositivos pueden realizar cálculos de hash más rápidamente que las CPU o GPU, lo que les da una ventaja competitiva en la minería de criptomonedas que utilizan SHA-256. Además, el uso de SHA-256 en sistemas de criptomonedas ha sido criticado por su alto consumo de energía y su impacto ambiental.

4.1.2 SCRYPT

Scrypt es un algoritmo que se utiliza en criptomonedas como Litecoin y Dogecoin. Funciona de manera similar al algoritmo SHA-256, pero con algunas diferencias.

Primero, en Scrypt, en lugar de simplemente ejecutar una función matemática una vez para obtener el hash, se ejecuta la función muchas veces. Esto se llama "iteración". Cuantas más iteraciones haya, más difícil será generar el hash.

Segundo, Scrypt utiliza mucha más memoria que SHA-256. Esto significa que los dispositivos ASIC (circuitos integrados específicos de la aplicación) que se utilizan para generar hashes en SHA-256 no funcionan tan bien con Scrypt. Esto hace que sea más difícil para alguien con un ASIC generar muchos hashes en poco tiempo.

Para obtener el hash en Scrypt, se toma un conjunto de datos (por ejemplo, una transacción) y se le aplica una función llamada "key derivation function" (KDF). Esta función es similar a la función matemática en SHA-256, pero se ejecuta muchas veces con diferentes parámetros para obtener el hash.

La dificultad en Scrypt se mide en función del número de iteraciones y de la cantidad de memoria necesaria para ejecutar el algoritmo. Cuantas más iteraciones y memoria se necesiten, más difícil será generar el hash.

4.1.3 ETHASH

Ethash es un algoritmo utilizado en la criptomoneda Ethereum para generar nuevos bloques en su blockchain. Al igual que otros algoritmos de PoW, utiliza la creación de hash para resolver un problema y obtener una recompensa. A diferencia de algunos algoritmos que solo utilizan la potencia de procesamiento de la CPU o GPU, ethash también tiene en cuenta la memoria del sistema para hacer el proceso más resistente a ASIC.

La creación del hash en ethash es un proceso que consta de varios pasos. En primer lugar, se toman los datos del bloque que se va a minar, como la transacción, el número de bloque anterior, el nonce y otros datos relevantes. A partir de estos datos, se crea una semilla inicial que se utiliza para generar un conjunto de valores de datos aleatorios.

Luego, se realiza una operación llamada "mezcla", que consiste en combinar los datos aleatorios generados anteriormente con la semilla inicial. Este proceso se repite varias veces para generar un conjunto de datos de mezcla finales. Estos datos de mezcla se utilizan luego para realizar una operación de hashing criptográfico, que genera el hash final del bloque.

El proceso de creación de hash en ethash también incluye una dificultad variable que se ajusta automáticamente en función del poder de procesamiento de la red. Cuanto mayor sea la potencia de procesamiento de la red, mayor será la dificultad para generar un bloque y, por lo tanto, mayor será la recompensa por hacerlo. Esto ayuda a mantener la estabilidad y seguridad de la red.

En términos de ventajas y desventajas, ethash es un algoritmo que se centra en la resistencia a ASIC, lo que significa que no es tan fácil de minar utilizando hardware especializado. Esto lo hace más accesible para mineros individuales y descentraliza la red. Sin embargo, también requiere más memoria y recursos del sistema para su procesamiento, lo que puede hacer que sea menos eficiente en algunas configuraciones de hardware. Además, la alta dificultad y la competencia en la red pueden dificultar la rentabilidad del minado.

4.1.4 EQUIHASH

Equihash es un algoritmo de prueba de trabajo diseñado para ser resistente a los circuitos integrados de aplicación específica, lo que significa que es más difícil para los mineros utilizar equipos especializados para obtener ventaja en la minería de criptomonedas. Este algoritmo fue creado por Alex Biryukov y Dmitry Khovratovich en 2016 y se utiliza en criptomonedas como Zcash, Komodo y Bitcoin Private.

El proceso de creación de un hash en Equihash comienza con la selección de un valor nonce (número arbitrario utilizado solo una vez) y una semilla (un valor constante definido en el código fuente de la criptomoneda). Luego se divide la semilla en múltiples bloques y se utilizan estos bloques junto con el nonce para generar una matriz de datos.

A continuación, se realiza un proceso de optimización de memoria llamado "acceso aleatorio a la memoria duro", que utiliza una cantidad específica de memoria RAM para

generar una secuencia de números aleatorios. La secuencia de números aleatorios se utiliza para generar una solución única para el problema de Equihash.

Una vez que se genera la solución, se verifica su validez comparándola con un objetivo específico. Si la solución es válida, se agrega al bloque de la cadena de bloques y se otorga al minero una recompensa en forma de criptomoneda.

La dificultad del proceso de Equihash se mide en función del tamaño de la matriz de datos utilizada para la generación del hash, así como de la cantidad de memoria RAM necesaria para la optimización de memoria HRAM. Como resultado, Equihash es muy eficiente en términos de uso de memoria, lo que lo hace más resistente a los ASIC.

Una de las principales ventajas de Equihash es que no requiere una gran cantidad de energía para generar hashes, lo que lo hace más eficiente y ecológico en comparación con otros algoritmos de PoW. Sin embargo, una posible desventaja de Equihash es que puede requerir más tiempo para generar hashes en comparación con otros algoritmos de PoW debido al proceso de HRAM.

4.1.5 CRYTONIGHT

El proceso de creación de hash en CryptoNight consta de varias etapas:

Preparación de la entrada: se toma la información que se desea convertir en un hash, que en el caso de CryptoNight es el contenido de un bloque de transacciones. Se agrega un nonce, que es un número arbitrario utilizado para modificar el hash, y se mezcla con otros datos como la dirección del minero y la marca de tiempo.

Cálculo del hash: se aplica el algoritmo CryptoNight a la entrada preparada en el paso anterior. El algoritmo utiliza una función de reducción de memoria resistente a ASIC, lo que significa que está diseñado para ser difícil de implementar en hardware especializado y favorecer el uso de CPU y GPU. Esta función utiliza una gran cantidad de memoria para evitar la creación de circuitos ASIC especializados.

Verificación de la dificultad: el hash resultante se verifica para asegurarse de que cumpla con los requisitos de dificultad de la red. La dificultad se establece por la red y determina como de difícil es resolver el rompecabezas de hash para crear un nuevo bloque.
Validación: si el hash cumple con los requisitos de dificultad, se agrega el bloque a la cadena de bloques y se recompensa al minero con la criptomoneda de la red.

Una de las ventajas de CryptoNight es su resistencia a los circuitos ASIC especializados, lo que permite a los mineros utilizar hardware más común como CPU y GPU para participar en el proceso de minería. Sin embargo, esta ventaja también puede

ser una desventaja, ya que la resistencia ASIC puede hacer que el proceso de minería sea más lento y costoso en términos de energía.

Otra característica importante de CryptoNight es su capacidad para mantener la privacidad de las transacciones. El algoritmo utiliza técnicas de ofuscación para evitar que se rastreen las direcciones de envío y recepción de la criptomoneda.

4.1.6 X11

X11 es un algoritmo de consenso de PoW utilizado por varias criptomonedas, como Dash, PIVX y StartCoin. Fue lanzado en 2014 y diseñado para ser resistente a los ASIC.

X11 es un algoritmo de hash que utiliza 11 funciones criptográficas diferentes en su proceso de generación de hash. Estas funciones incluyen SHA-256, BLAKE, Grøstl, JH, Keccak, Skein, Luffa, CubeHash, SHAvite-3, SIMD y Echo. Cada una de estas funciones se utiliza en un orden específico para producir el hash final.

El proceso de generación de hash en X11 es muy similar a otros algoritmos de hash. Se comienza con un bloque de datos y se realiza una serie de operaciones criptográficas sobre ese bloque para producir un valor hash de 256 bits. La dificultad del proceso se ajusta automáticamente para mantener un tiempo de generación de bloque constante.

Una de las principales ventajas de X11 es su resistencia a los ASIC. Al utilizar múltiples funciones criptográficas en lugar de solo una, es mucho más difícil para los ASIC especializados realizar el proceso de hash de manera más eficiente que los CPU o GPU convencionales. Esto ayuda a mantener la descentralización de la red y evita que un pequeño número de mineros con ASIC controlen la mayoría del poder de hash en la red.

Sin embargo, una de las desventajas de X11 es que es un algoritmo muy intensivo en memoria. Para mantener la resistencia a los ASIC, el algoritmo requiere una gran cantidad de memoria para completar el proceso de hash. Esto significa que el proceso de minería de X11 es más eficiente en hardware con una gran cantidad de memoria, como las GPU de gama alta.
En general, X11 es un algoritmo de hash interesante y eficaz utilizado por varias criptomonedas. Su resistencia a los ASIC y su capacidad para mantener la descentralización de la red lo hacen atractivo para los mineros que buscan una alternativa a los algoritmos más comunes como SHA-256 y Scrypt.

4.1.7 BLAKE2B

Blake2b es un algoritmo de hash criptográfico que es una versión mejorada de su predecesor, Blake. Fue creado por los mismos desarrolladores que crearon el algoritmo original, pero con el objetivo de mejorar la velocidad, la eficiencia y la seguridad.

El algoritmo Blake2b utiliza una estructura de árbol hash llamada "árbol Merkle", que es similar a la utilizada en el algoritmo SHA-256. El árbol Merkle divide los datos en bloques, y cada bloque se procesa individualmente para crear un hash. Luego, los hashes resultantes se combinan en pares y se vuelven a procesar para crear un hash final.

Blake2b también incluye varias mejoras de seguridad en comparación con Blake, incluyendo la resistencia a las colisiones y la resistencia a los ataques de longitud extendida.

Una de las principales ventajas de Blake2b es su velocidad y eficiencia en términos de uso de recursos. Es más rápido que muchos otros algoritmos de hash, lo que lo hace ideal para su uso en aplicaciones que requieren un procesamiento rápido y eficiente de grandes cantidades de datos.

En términos de criptomonedas, Blake2b se utiliza como algoritmo de hash en varias monedas alternativas, incluyendo Siacoin (SC) y Verge (XVG). La dificultad del proceso de creación de hash depende de varios factores, como la potencia de procesamiento disponible en la red y el tamaño del bloque.

Entre las desventajas de Blake2b se encuentra su relativa falta de adopción en comparación con otros algoritmos de hash más populares como SHA-256 o Scrypt. Además, como es un algoritmo relativamente nuevo, todavía no ha sido completamente probado en términos de seguridad a largo plazo.

4.2 PARAMETROS

Una vez comprendido el funcionamiento de los algoritmos descritos en el punto anterior es momento de realizar las comparaciones entre los mismo de las que se viene hablado durante toda la memoria. A continuación, se muestran los parámetros seleccionados debido a su relevancia en el entorno de las criptomonedas:

Velocidad de hashing: se refiere a la cantidad de hashes que un dispositivo de minería puede calcular en un segundo. Este parámetro es importante para evaluar la eficiencia de los algoritmos de hashing en términos de tiempo y recursos.

Consumo de energía: se refiere a la cantidad de energía eléctrica que consume un dispositivo de minería para calcular los hashes. Este parámetro es importante para evaluar la eficiencia de los algoritmos de hashing en términos de consumo de energía y costos operativos.

Resistencia a ASICs: se refiere a la capacidad de un algoritmo de hashing para resistir la creación y uso de dispositivos de minería ASICs (Application-Specific Integrated Circuit). Los ASICs son dispositivos de minería especializados que pueden ser más eficientes que las CPU o GPU en términos de velocidad y consumo de energía, lo que puede llevar a una centralización de la minería en manos de unos pocos grandes actores. Un algoritmo de hashing resistente a ASICs busca promover una mayor distribución y descentralización de la minería.

Resistencia a GPUs: se refiere a la capacidad de un algoritmo de hashing para resistir el uso de tarjetas gráficas (GPUs) en la minería. Las GPUs también pueden ser más eficientes que las CPU en términos de velocidad y consumo de energía, lo que puede llevar a una centralización de la minería en manos de unos pocos grandes actores. Un algoritmo de hashing resistente a GPUs busca promover una mayor distribución y descentralización de la minería.

Distribución de minería: se refiere a la capacidad de un algoritmo de hashing para fomentar una distribución más equitativa de la minería entre diferentes actores. Una distribución más equitativa puede mejorar la seguridad y la resistencia a la manipulación de la red.

Seguridad: se refiere a la capacidad de un algoritmo de hashing para proteger la integridad y la seguridad de la red. Los algoritmos de hashing deben ser resistentes a diferentes tipos de ataques, como ataques de 51%, ataques de doble gasto, entre otros.

Precio por hash: se refiere al costo de adquirir y mantener la infraestructura de minería necesaria para calcular los hashes. El precio por hash puede variar dependiendo de factores como el costo del hardware, el costo de la energía, entre otros.
Teniendo como referencia estos parámetros definidos, se ha elaborado una tabla que contiene los 7 algoritmos explicados y una valoración que no deja de ser subjetiva que puntúa del 1 al 10 cada una de las características de los algoritmos. No necesariamente una puntuación alta es indicativa de un valor positivo ya que en consumo de energía cuanto mayor es la puntuación peor es la valoración por tanto hay que tener en cuenta puntuación y criterio. Si bien es cierto que al algoritmo SHA-256 tiene valoraciones superiores en la mayoría de los parámetros, en todos ellos los algoritmos mantienen una linealidad. Sin embargo, en la velocidad de hashing, el consumo de energía y la resistencia a ASICs es donde se observan los valores más dispares. Es por ello por lo que vamos a centrar las pruebas de este trabajo en la velocidad de hashing que guarda, aunque no de manera directa relación con el precio por hash y la resistencia a ASICs o GPUs.

Antes de ver la tabla vamos a describir brevemente la composición de las valoraciones.

Velocidad de Hashing:

- SHA-256: Tiene una alta velocidad de hashing y es eficiente en términos de rendimiento.
- Scrypt: Es más lento que SHA-256 debido a su enfoque en la memoria y la CPU.
- Ethash: Tiene una velocidad de hashing moderada y se adapta bien a GPUs.
- Equihash: Es más lento que Ethash y se enfoca en resistir ASICs.
- CryptoNight: Es relativamente lento y está diseñado para resistir ASICs y GPUs.
- X11: Tiene una velocidad de hashing moderada y es conocido por su eficiencia energética.
- Blake2b: Es muy rápido y eficiente en términos de velocidad de hashing.

Consumo de Energía:

- SHA-256: Consume menos energía en comparación con algunos otros algoritmos.
- Scrypt: Es más eficiente en términos de energía que Ethash y Equihash.
- Ethash: Tiene un consumo de energía moderado.
- Equihash: Puede ser más eficiente en términos de energía que Ethash.
- CryptoNight: Es eficiente en energía debido a su resistencia a hardware especializado.
- X11: Es conocido por su eficiencia energética.
- Blake2b: Es eficiente en términos de energía debido a su velocidad.

Resistencia a ASICs:

- SHA-256: Es vulnerable a ASICs y la minería está dominada por ellos.
- Scrypt: Fue diseñado para resistir ASICs, pero algunos han sido desarrollados.
- Ethash: Es resistente a ASICs, lo que fomenta la minería descentralizada.
- Equihash: Inicialmente resistente a ASICs, pero algunos ASICs han aparecido.
- CryptoNight: Resiste ASICs, promoviendo una minería más amplia.
- X11: Ha sido resistente a ASICs, lo que fomenta la minería con GPUs.
- Blake2b: Inicialmente resistente a ASICs, pero algunos ASICs han aparecido.

Resistencia a GPUs:

- SHA-256: Ampliamente minado con ASICs, no es adecuado para GPUs.
- Scrypt: Originalmente diseñado para resistir ASICs, es amigable con GPUs.
- Ethash: Diseñado para GPUs, fomenta la minería descentralizada.
- Equihash: Inicialmente resistente a GPUs, pero algunos son eficientes.
- CryptoNight: Resiste GPUs, promoviendo una minería más justa.
- X11: Fomenta la minería con GPUs, aunque algunos ASICs están disponibles.
- Blake2b: Inicialmente resistente a GPUs, pero algunos ASICs han aparecido.

Distribución de Minería:

- SHA-256: Dominado por grandes operaciones de minería, menos descentralizado.
- Scrypt: Tiende a ser más descentralizado debido a la resistencia a ASICs.
- Ethash: Fomenta la minería descentralizada con GPUs ampliamente disponibles.
- Equihash: Inicialmente descentralizado, pero algunos ASICs han afectado.
- CryptoNight: Promueve la minería más equitativa debido a la resistencia.
- X11: Más descentralizado que SHA-256, pero algunos ASICs.
- Blake2b: Inicialmente descentralizado, pero algunos ASICs han aparecido.

Seguridad:

Todos los algoritmos son seguros en términos de protección de datos y resistencia a ataques. La seguridad de cada uno está relacionada con su resistencia a ASICs, GPUs y la centralización de la minería, lo que contribuye a la protección de la red y a la equidad en la minería.

Precio por Hash:

- SHA-256: Precio por hash generalmente alto debido a la competencia.

- Scrypt: Precio por hash moderado, ya que no requiere hardware costoso.
- Ethash: Precio por hash moderado debido a la disponibilidad de GPUs.
- Equihash: Precio por hash generalmente moderado.
- CryptoNight: Precio por hash generalmente bajo debido a la resistencia.
- X11: Precio por hash moderado, pero puede variar según la demanda.
- Blake2b: Precio por hash moderado, pero puede verse afectado por ASICs.

Parámetros	SHA-256	Scrypt	Ethash	Equihash	CryptoNight	X11	Blake2b
Velocidad de hashing	10	5	6	4	3	7	8
Consumo de energía	9	7	6	5	3	8	7
Resistencia a ASICs	2	9	10	9	8	7	8
Resistencia a GPUs	8	6	6	7	7	6	6
Distribución de minería	10	7	6	7	7	6	6
Seguridad	10	9	8	8	8	8	8
Precio por hash	9	6	7	7	8	6	7

Tabla 1. Valoración de algoritmos PoW

En base a estas valoraciones, a la estructura de los algoritmos y el acceso más o menos limitados se han seleccionado tres de ellos para ejecutar las pruebas, estos algoritmos son SHA-256, SCRYPT y BLAKE2B.

4.3 PRUEBAS

Para cada uno de los tres algoritmos seleccionados en el punto anterior se ha diseñado un script que represente la generación de un hash válido en un sistema de criptomonedas. Para SHA-256 se ha aplicado el algoritmo de consenso de Bitcoin, para Scrypt se ha aplicado el de Litecoin y para Blake2b dado que no es fácil acceder a los algoritmos internos, en concreto el de ajuste de dificultad en Siacoin, se ha creado un script que haga uso del algoritmo Blake2b pero con el control de dificultad de Bitcoin, de ese modo podremos comparar directamente la velocidad de ambos algoritmos ante el mismo problema. Por último, para Scrypt se ha diseñado un test que represente la generación de un hash válido en Litecoin. Todas las pruebas han sido realizadas en una CPU con estas características:

- Modelo: Intel Core i7-6700HQ
- Arquitectura: Skylake
- Número de núcleos: 4 núcleos
- Número de hilos: 8 hilos (admite Hyper-Threading)
- Frecuencia base: 2.60 GHz
- Frecuencia turbo máxima: 3.50 GHz
- Caché: 6 MB SmartCache
- Gráficos integrados: Intel HD Graphics 530
- Tecnología de proceso: 14nm

4.3.1 TEST ALGORITMO SHA-256

Comenzamos con el algoritmo SHA-256. En Bitcoin el tiempo medio de creación de bloque es de 10 minutos, los mineros para encontrar un hash válido deben probar con un nonce en cada intento (el nonce es el dato variable), hasta encontrar un hash con los datos de entrada correctos que cumpla los requisitos actuales, en concreto en Bitcoin este requisito es que al inicio del hash haya un número de 0 determinados. Esto se ajusta de manera automática en función de la capacidad de cómputo de la red.

Para nuestro caso, hemos incluido una variable con la que modificar el número de ceros al inicio del hash y así ir haciendo pruebas del tiempo que tardamos en generar un hash. El código es el siguiente:

```python
import hashlib
import time

def minar_bloque(datos, dificultad_objetivo):
    objetivo = "0" * dificultad_objetivo
    nonce = 0
    while True:
        datos_bloque = datos.encode() + str(nonce).encode()
        hash_sha256 = hashlib.sha256(datos_bloque).hexdigest()
        if hash_sha256[:dificultad_objetivo] == objetivo:
            return nonce, hash_sha256
        nonce += 1

datos = "TEST_SHA-256"
dificultad_objetivo = 5  # Dificultad objetivo (número de ceros al principio del hash)

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Ilustración 6. Código base test SHA-256 (producción propia)

Se han realizado múltiples ejecuciones variando los datos de entrada, así como la dificultad. Al realizar las pruebas se observa que aumentar en un cero la dificultad eleva exponencialmente el tiempo necesario para resolver el acertijo. También es muy notoria la diferencia entre hashes ante una mínima variación en los datos de entrada, vamos a ir viendo los resultados de las pruebas realizadas y se irán comentando los detalles más relevantes.

En la primera prueba de SHA-256, comprobamos que ante una misma entrada de datos el hash generado es siempre el mismo, por otro lado, aunque los datos y el problema sea el mismo, el tiempo de ejecución varía mínimamente, esto es debido a los recursos usados por la CPU. En este caso los datos de entrada son “TEST_SHA-256” y la dificultad es 5, es decir se debe encontrar un nonce que con estos datos de entrada nos del hash que cumpla la característica de al menos 5 ceros al inicio:

```
========== RESTART: C:\Users\mserg\Downloads\TEST_SHA-256.py ==========
Nonce encontrado: 5212972
Hash encontrado: 000007bf77d97ec09daf0ec947373ca8efff8fb505a0dafbe62b035e5557676a
Tiempo transcurrido: 8.801279 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SHA-256.py ==========
Nonce encontrado: 5212972
Hash encontrado: 000007bf77d97ec09daf0ec947373ca8efff8fb505a0dafbe62b035e5557676a
Tiempo transcurrido: 9.738288 segundos
```

Ilustración 7. Prueba determinista SHA-256 (producción propia)

En la segunda prueba de SHA-256, hemos decidido variar el último dígito de los datos de entrada, esto supone una variación enorme en el hash generado, y también supone que el tiempo de ejecución cambie. Esto es normal ya que, aunque el problema a resolver sea el mismo, la solución puede ser encontrada de manera aleatoria en un rango de tiempo. Para esta prueba se ha variado el último digito de los datos de entrada, siendo SHA-567, SHA-258 y SHA-259 respectivamente. Con este ligero cambio vemos cómo cambia radicalmente el hash generado y también el tiempo en encontrar el hash, esto es normal ya que el número de intentos para conseguir difiere de uno a otro. Se mantiene la dificultad en 5 y en la primera prueba vemos que salen 6 ceros, esto es mera casualidad, pero al cumplir los requisitos es el primer hash válido.

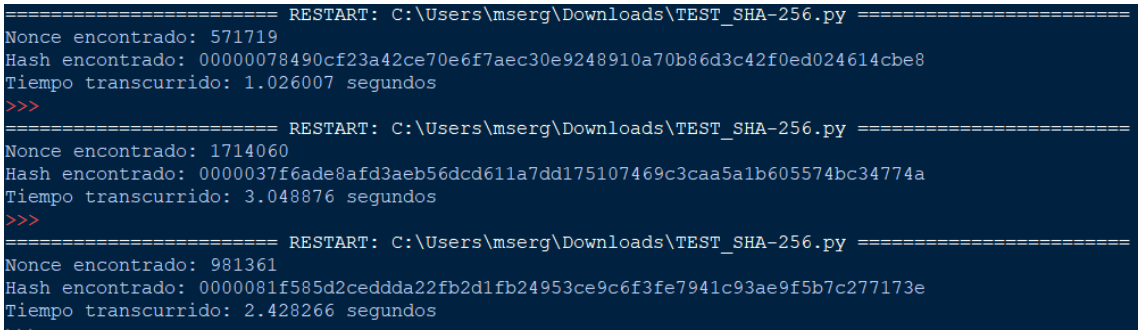

Ilustración 8. Prueba variando datos de entrada SHA-256 (producción propia)

En la última de las pruebas de este algoritmo se observa como al ir aumentando el requisito de número de ceros al inicio del hash, el tiempo de creación de hash, es decir el tiempo que se tarda en encontrar el hash válido aumenta exponencialmente, aunque nunca de manera lineal. Midiendo estos tiempos de generación de bloque al encontrar hashes válidos es como se ajusta la dificultad de la red para que el tiempo medio de generación de bloque esté siempre próximo a los 10 minutos. En este ejemplo las dificultades aplicadas han sido 3, 4, 5, 6 y 7 respectivamente.

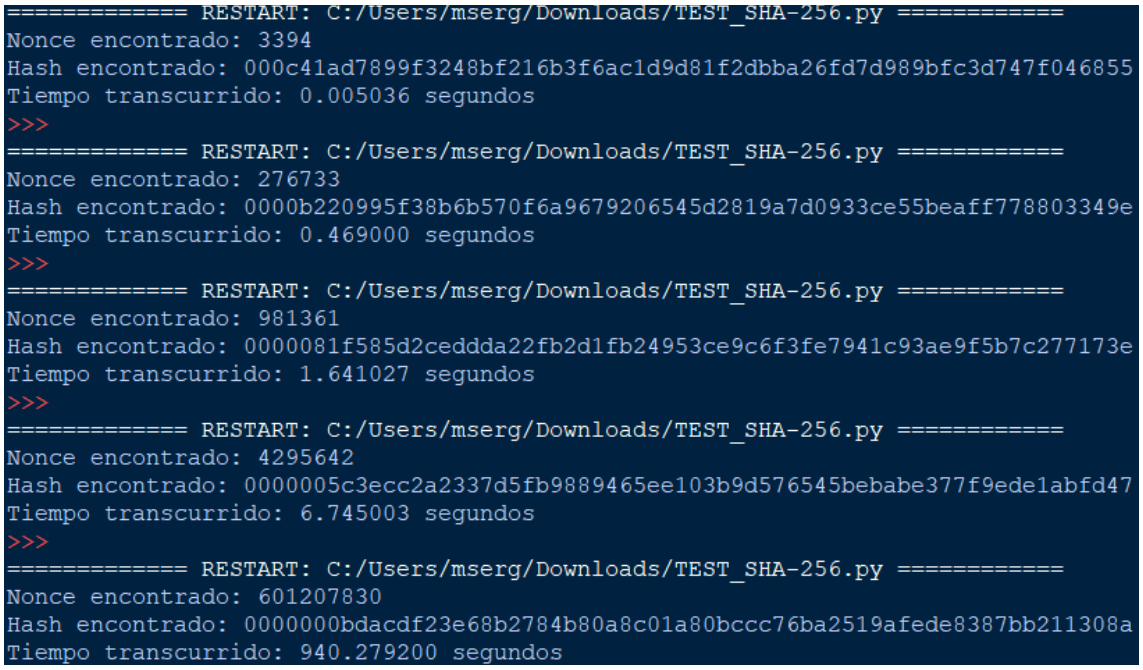

Ilustración 9. Prueba aumentando dificultad de hash válido SHA-256 (producción propia)

4.3.2 TEST ALGORITMO BLAKE2B

El segundo de los algoritmos que han sido testeados es BLAKE2B, se ha elegido como segunda opción ya que tiene más similitudes con SHA-256 que SCRYPT, por ejemplo, el tiempo de generación de bloque es aproximadamente 10 minutos al igual que en BITCOIN. La principal diferencia con SHA-256 es la longitud del hash que es de 512 bits en lugar de 256, esto nos servirá para comparar de una manera más evidente la velocidad de ambos algoritmos. Como se ha indicado al inicio del apartado para controlar la dificultad se sigue el método de requerir un número de ceros determinados al inicio del hash. Este es el código utilizado:

```python
import hashlib
import time

def minar_bloque(datos, dificultad_objetivo):
    objetivo = "0" * dificultad_objetivo
    nonce = 0
    while True:
        datos_bloque = datos.encode() + str(nonce).encode()
        hash_blake2b = hashlib.blake2b(datos_bloque, digest_size=64).hexdigest()
        if hash_blake2b[:dificultad_objetivo] == objetivo:
            return nonce, hash_blake2b
        nonce += 1

datos = "TEST_BLAKE2B"
dificultad_objetivo = 5  # Dificultad objetivo(número de ceros al principio del hash)

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Ilustración 10. Código base test BLAKE2B (producción propia)

En la primera prueba comprobamos que aumentado la dificultad en un 0 el tiempo también crece exponencialmente. En esta prueba los datos de entrada han sido "TEST_BLAKE2B" y la dificultad ha sido de 5, 6, 7 y 8 respectivamente. Recordamos que el número de dificultad en este algoritmo representa el número de ceros con los que debe iniciar el hash generado para ser válido. Entre la dificultad 6 y 7 vemos que el tiempo se incrementa mínimamente, sin embargo, no es el patrón general y vemos como en el siguiente nivel, cuando la dificultad es 8 el tiempo para encontrar ese hash es muy diferencial con el anterior.

```
========== RESTART: C:/Users/mserg/Downloads/TEST_BLAKE2b.py ==========
Nonce encontrado: 778532
Hash encontrado: 000008b3c1f8ced8a7f91ba65307f2d5fc4a360e3b6c8f34cdda9e21eeba8561d45
c07739d1fb890d9bf5228e1fa8088d1f5ddb17a8abaf9bc92f00f67c13d90
Tiempo transcurrido: 1.135995 segundos
>>>
========== RESTART: C:/Users/mserg/Downloads/TEST_BLAKE2b.py ==========
Nonce encontrado: 11471132
Hash encontrado: 00000031fb0113eb9ec54b94757ca3d52f53d73f733b48e804d255b9b47ab6c43a7
7e0903b9d284a19fee4dc9b2f1dafa45cf3ae827d108bab3d3c4063f0fa54
Tiempo transcurrido: 14.290992 segundos
>>>
========== RESTART: C:/Users/mserg/Downloads/TEST_BLAKE2b.py ==========
Nonce encontrado: 13047099
Hash encontrado: 0000000f123b9a106bba8244835a7c5133af0c0828203106363584603aff7ed8686
996dc20c579d2b9343b78862f8c3881265692cf54de28036281a371fa1552
Tiempo transcurrido: 15.907089 segundos
>>>
========== RESTART: C:/Users/mserg/Downloads/TEST_BLAKE2b.py ==========
Nonce encontrado: 1552403678
Hash encontrado: 0000000054556ef3d7ba0981631f80ad8cc336618ac6e5411fc8a7c050439324bca
20f1bb2e39c52a9cf7d1950dafe120e670653ba738639b9cdea3d6d6cd9b4
Tiempo transcurrido: 2009.272774 segundos
```

Ilustración 11. Prueba aumentando la dificultad de hash válido BLAKE2B (producción propia)

En la segunda prueba, se validad otro de los principios de estos algoritmos y es que, ante una mínima variación de los datos de entrada, el hash de la salida cambia radicalmente. En este caso se ha mantenido una dificultad constante de 4, y se han variado los datos de entrada siendo "TEST_Blake2b", "TEST_BLAKE2c" y "TEST_BLAKE2C" respectivamente.

```
========== RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==========
Nonce encontrado: 735
Hash encontrado: 0000aa368b3dee339f539c3b32794d82c6a2a978ed9bf2166df81f2cfd13b6c
e9962047365762dfeb9fc97109925c183a5a1ff41f6eefaa6a3876b44cf91151b
Tiempo transcurrido: 0.001000 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==========
Nonce encontrado: 21592
Hash encontrado: 0000fa326c7791ce55be671dbdd0425b12d109c1651e25c30dbd5d37247be99
c5be9c3383b9e6525d95bbd32a2a7c166636fc5b08c7a218c2d044339d61a5db7
Tiempo transcurrido: 0.025998 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==========
Nonce encontrado: 21349
Hash encontrado: 00004b60da27a142045e97aa015aab70aeef9582251023db137aa6d2c66ca2a
0aca0634d5b1f4423fe324f7a8928cb3b452ad121b7e1fe78f6b79f236cbc1845
Tiempo transcurrido: 0.029015 segundos
```

Ilustración 12. Prueba variando datos de entrada BLAKE2B (producción propia)

En la tercera de las pruebas de BLAKE2B se ha comprobado que cumple con una de las características que le definen como algoritmo que es la de determinista, vemos como ante los mismos datos de entrada el hash generado de salida es el mismo (datos de

entrada “TEST_BLAKE2B”, dificultad 4). La única diferencia entre las tres ejecuciones es el tiempo que no es exactamente igual, como ya hemos visto con SHA-256.

```
================ RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==============
Nonce encontrado: 79153
Hash encontrado: 0000162ec50d686c20e95196770d9e604915fb8556476cae24fb44f13dde06c
6c1bf88cf62c292cb452260f1e90bfc2b07e6ec02416e5969855f7b3bcd93b027
Tiempo transcurrido: 0.107060 segundos
>>>
================ RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==============
Nonce encontrado: 79153
Hash encontrado: 0000162ec50d686c20e95196770d9e604915fb8556476cae24fb44f13dde06c
6c1bf88cf62c292cb452260f1e90bfc2b07e6ec02416e5969855f7b3bcd93b027
Tiempo transcurrido: 0.105013 segundos
>>>
================ RESTART: C:\Users\mserg\Downloads\TEST_BLAKE2b.py ==============
Nonce encontrado: 79153
Hash encontrado: 0000162ec50d686c20e95196770d9e604915fb8556476cae24fb44f13dde06c
6c1bf88cf62c292cb452260f1e90bfc2b07e6ec02416e5969855f7b3bcd93b027
Tiempo transcurrido: 0.103996 segundos
```

Ilustración 13. Prueba determinista BLAKE2B (producción propia)

4.3.3 TEST ALGORITMO SCRYPT

El tercero de los algoritmos es SCRYPT. En este caso el nivel de dificultad se regula de manera diferente, hay dos variables que se ajustan automáticamente la N que representa el promedio de intentos para encontrar u bloque válido, en nuestra prueba lo dejaremos fijo y por otro lado en Litecoin para que un hash sea válido el requisito que debe cumplir es ser más pequeño que un número dado, cuando se quiere mantener el tiempo promedio de generación de bloque que en Litecoin es de 2,5 minutos se hace más pequeño este número y así se dificulta la generación de bloques. El código del test es el siguiente:

```python
import scrypt
import time
import binascii

def minar_bloque(datos, dificultad_objetivo):
    nonce = 0
    while True:
        datos_bloque = datos.encode() + str(nonce).encode()
        hash_scrypt = scrypt.hash(datos_bloque, datos_bloque, N=16384, r=8, p=1, buflen=32)
        hash_scrypt_hex = binascii.hexlify(hash_scrypt).decode()
        hash_int = int(hash_scrypt_hex, 16)
        if hash_int < 2**(256 - dificultad_objetivo):
            return nonce, hash_scrypt_hex
        nonce += 1

datos = "TEST_SCRYPT"
dificultad_objetivo = 4  # Dificultad objetivo

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Ilustración 14. Código base test SCRYPT (producción propia)

Comenzamos comprobando que se cumplen el principio de determinista. En la siguiente imagen vemos que el hash de salida es el mismo con la misma entrada de datos (TEST_SCRYPT):

```
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 84
Hash encontrado: 01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d
Tiempo transcurrido: 5.207045 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 84
Hash encontrado: 01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d
Tiempo transcurrido: 5.307015 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 84
Hash encontrado: 01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d
Tiempo transcurrido: 5.523003 segundos
```

Ilustración 15. Prueba determinista SCRYPT (producción propia)

Ahora modificamos levemente los datos de entrada y se observan grandes cambios en el hash, los inputs son “TEST-SCRYPT”, “TEST_SCRYPT” y TEST-scrypt” respectivamente:

```
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 52
Hash encontrado: 03e4a75cb06e4d3fae5f9490fc63f12a4f075dad8f11c3917aef958ddb369511
Tiempo transcurrido: 3.400037 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 84
Hash encontrado: 01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d
Tiempo transcurrido: 5.460531 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 64
Hash encontrado: 01750571f1aa7240d5a9c1fba7f2610934d7ea5e34f21edcf518c81443ea1dd5
Tiempo transcurrido: 4.846040 segundos
```

Ilustración 16. Prueba variando los datos de entrada SCRYPT (producción propia)

En la tercera prueba de SCRYPT variamos la dificultad, aumentándola en cada intento. A lo largo de las pruebas de este algoritmo vemos que el número de nonce probados hasta encontrar el correcto es muy inferior a los intentos de los otros dos algoritmos, aun así, vemos como aumentando levemente la dificultad conseguimos elevar en gran medida el tiempo necesario para encontrar un hash válido (las dificultades usadas han sido 3, 4, 6, 8, 10 y 12 respectivamente):

```
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 9
Hash encontrado: 130ab81c71f724f5232cd8c5ff1b012ca872990e646c59f5a4ef1c031802e50a
Tiempo transcurrido: 0.661033 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 27
Hash encontrado: 0d4429f5584afb2924fa965226c84f2b3c3c0024a0486faf6aadad0f6fb4693d
Tiempo transcurrido: 1.804994 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 185
Hash encontrado: 00a8a6aa5ed962399183635877605cccb72810b8950c3b3f12cfc1a253ba4ebc
Tiempo transcurrido: 11.559464 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 185
Hash encontrado: 00a8a6aa5ed962399183635877605cccb72810b8950c3b3f12cfc1a253ba4ebc
Tiempo transcurrido: 11.548403 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 2519
Hash encontrado: 002fcebbf22679fd737b5d1830d96c89ab60c4374714559e359a9c7f75ee8086
Tiempo transcurrido: 157.875748 segundos
>>>
========== RESTART: C:\Users\mserg\Downloads\TEST_SCRYPT.py ==========
Nonce encontrado: 5642
Hash encontrado: 0001138ec18b5dcf39cbaf0f89024494ec2886ae8f5f3b079e45b217a1ba5551
Tiempo transcurrido: 333.134564 segundos
```

Ilustración 17. Prueba aumentando la dificultad de hash válido SCRYPT (producción propia)

4.4 RESULTADOS

Con todas las pruebas ya realizadas en este apartado se recogen los resultados obtenidos en tablas y están ordenados por algoritmo y prueba para proceder a su posterior análisis. Con los datos recogidos en tablas se podrá observar las conclusiones que se ha ido explicando en el apartado anterior, además veremos que datos o conclusiones relevantes se pueden extraer de los resultados.

4.4.1 RESULTADOS SHA-256

El primero de los algoritmos testeados es SHA-256 y estos son los resultados:

Resultados test SHA-256 Determinista:

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_SHA-256	5	5212972	000007bf77d97ec09daf0ec947373ca8efff8fb505a0dafbe62b035e5557676a	8.801279
TEST_SHA-256	5	5212972	000007bf77d97ec09daf0ec947373ca8efff8fb505a0dafbe62b035e5557676a	9.738288

Tabla 2. Resultados test SHA-256 Determinista

Resultados test SHA-256 con variación de inputs

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_SHA-257	5	571719	00000078490cf23a42ce70e6f7aec30e9248910a70b86d3c42f0ed024614cbe8	1.026007
TEST_SHA-258	5	1714060	0000037f6ade8afd3aeb56dcd611a7dd175107469c3caa5a1b605574bc34774a	3.048876
TEST_SHA259	5	981361	0000081f585d2ceddda22fb2d1fb24953ce9c6f3fe7941c93ae9f5b7	2.428266

			c277173e	

Tabla 3. Resultados test SHA-256 con variación de inputs

Resultados test SHA-256 aumento de dificultad

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_SHA-259	3	3394	000c41ad7899f3248bf216b3f6ac1d9d81f2dbba26fd7d989bfc3d747f046855	0.005036
TEST_SHA-259	4	276733	0000b220995f38b6b570f6a9679206545d2819a7d0933ce55beaff778803349e	0.469000
TEST_SHA259	5	981361	0000081f585d2ceddda22fb2d1fb24953ce9c6f3fe7941c93ae9f5b7c277173e	1.641027
TEST_SHA259	6	4295642	0000005c3ecc2a2337d5fb9889465ee103b9d576545bebabe377f9ede1abfd47	6.745003
TEST_SHA259	7	601207830	0000000bdacdf23e68b2784b80a8c01a80bccc76ba2519afede8387bb211308a	940.279200

Tabla 4. Resultados test SHA-256 aumento de dificultad

4.4.2 RESULTADOS BLAKE2B

El siguiente algoritmo testeado es BLAKE2B.

Resultados test BLAKE2B aumento de dificultad

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_BLAKE2B	5	778532	000008b3c1f8ced8a7f91ba65307f2d5fc4a360e3b6c8f34cdda9e21eeba8561d45c07739d1fb890d9bf5228e1fa8088d1f5ddb17a8abaf9b	1.135995

			c92f00f67c13d90	
TEST_BLAKE2B	6	11471132	00000031fb0113eb9ec54b94757ca3d52f53d73f733b48e804d2553b9b47ab6c43a77e0903b9d284a19fee4dc9b2f1dafa45cf3ae827d108bab3d3c4063f0fa54	14.290992
TEST_BLAKE2B	7	13047099	0000000f123b9a106bba8244835a7c5133af0c08282031063635846 03aff7ed8686996dc20c579d2b9343b78862f8c3881265692cf54de28036281a371fa1552	15.907089
TEST_BLAKE2B	8	1552403678	0000000054556ef3d7ba0981631f80ad8cc336618ac6e5411fc8a7c050439324bca20f1bb2e39c52a9cf7d1950dafe120e670653ba738639b9cdea3d6d6cd9b4	2009.272774

Tabla 5. Resultados test BLAKE2B aumento de dificultad

Resultados test BLAKE2b con variación de inputs

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_Blake2b	4	735	0000aa368b3dee339f539c3b32794d82c6a2a978ed9bf2166df81f2cfd13b6ce9962047365762dfeb9fc97109925c183a5a1ff41f6eefaa6a3876b44cf91151b	0.001000
TEST_BLAKE2c	4	21592	0000fa326c7791ce55be671dbdd0425b12d109c1651e25c30dbd5d37247be99c5be9c3383b9e6525d95bbd32a2a7c166636fc5b08c7a218c2d044339d61a5db7	0.025998
TEST_BLAKE2C	4	21349	00004b60da27a142045e97aa015aab70aeef9	0.029015

			582251023db137aa6d 2c66ca2a0aca0634d5b 1f4423fe324f7a8928c b3b452ad121b7e1fe7 8f6b79f236cbc1845	

Tabla 6. Resultados test BLAKE2B con variación de inputs

Resultados test BLAKE2B Determinista

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_BLAKE2B	4	79153	0000162ec50d686c20 e95196770d9e604915 fb8556476cae24fb44f 13dde06c6c1bf88cf62 c292cb452260f1e90bf c2b07e6ec02416e596 9855f7b3bcd93b027	0.107060
TEST_BLAKE2B	4	79153	0000162ec50d686c20 e95196770d9e604915 fb8556476cae24fb44f 13dde06c6c1bf88cf62 c292cb452260f1e90bf c2b07e6ec02416e596 9855f7b3bcd93b027	0.105013
TEST_BLAKE2B	4	79153	0000162ec50d686c20 e95196770d9e604915 fb8556476cae24fb44f 13dde06c6c1bf88cf62 c292cb452260f1e90bf c2b07e6ec02416e596 9855f7b3bcd93b027	0.103996

Tabla 7. Resultados test BLAKE2B Determinista

4.4.3 RESULTADOS SCRYPT

Resultados test SCRYPT Determinista

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST_SCRYPT	6	84	01bd080243cd202783 d51e5c4e2cc8ddeadfff 542d8dadbceff5cf20c	5.207045

			458ce6d	
TEST_SCRYPT	6	84	01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d	5.307015
TEST_SCRYPT	6	84	01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d	5.523003

Tabla 8. Resultados tests SCRYPT Determinista

Resultados test SCRYPT con variación de inputs

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
TEST-SCRYPT	6	52	03e4a75cb06e4d3fae5f9490fc63f12a4f075dad8f11c3917aef958ddb369511	3.400037
TEST_SCRYPT	6	84	01bd080243cd202783d51e5c4e2cc8ddeadfff542d8dadbceff5cf20c458ce6d	5.460531
TEST_scrypt	6	64	01750571f1aa7240d5a9c1fba7f2610934d7ea5e34f21edcf518c81443ea1dd5	4.846040

Tabla 9. Resultados test SCRYPT con variación de inputs

Resultados test SCRYPT aumento de dificultad

Datos de entrada	Dificultad	Intentos	HASH	Tiempo (segundos)
SCRYPT	3	9	130ab81c71f724f5232cd8c5ff1b012ca872990e646c59f5a4ef1c031802e50a	0.661033
SCRYPT	4	27	0d4429f5584afb2924fa965226c84f2b3c3c0024a0486faf6aadad0f6fb4693d	1.804994
SCRYPT	6	185	00a8a6aa5ed962399183635877605cccb72810b8950c3b3f12cfc1a253ba4ebc	11.559464

SCRYPT	8	185	00a8a6aa5ed962399183635877605cccb72810b8950c3b3f12cfc1a253ba4ebc	11.559464
SCRYPT	10	2519	002fcebbf22679fd737b5d1830d96c89ab60c4374714559e359a9c7f75ee8086	157.875748
SCRYPT	12	5642	0001138ec18b5dcf39cbaf0f89024494ec2886ae8f5f3b079e45b217alba5551	333.134564

Tabla 10. Resultados test SCRYPT aumento de dificultad

Una vez analizados los resultados obtenidos tras múltiples pruebas realizadas hemos podido comprobar la velocidad de hashing de los tres algoritmos testeados, además hemos podido asegurar que cumplen con las características que definen los algoritmos de hash criptográfico como son:

- **Determinista:** Hemos comprobado que ante unos mismos datos de entrada el hash generado es el mismo, esencial para verificación de datos.
- **Computacionalmente eficiente:** Los algoritmos, aunque con diferencias entre ellos generan los hashes de manera relativamente rápida.
- **No reversible:** No existe un camino directo para determinar los datos de origen teniendo el hash como información, el único camino es mediante fuerza bruta.

- **Pequeños cambios en el input cambian completamente el código hash**: Al realizar un pequeño cambio en la entrada, hemos visto en las pruebas que cambiando una sola letra o incluso esa misma letra solo cambiando de minúscula a mayúscula hace que el hash de salida cambie completamente, sin guardar relación ninguna, aunque dos datos de entrada sean muy similares.

- **Resistente a colisiones:** Al ser el número de posibilidades de entrada esta característica por definición no se puede cumplir al 100%, sin embargo, sí que con una alta garantía se puede afirmar que cuando obtenemos el mismo código hash de salida, los datos de entrada serán iguales.

En cuanto a la velocidad se ha evidenciado que, aunque el tiempo de generación de hash entre SHA-256 y BLAKE2B es muy similar, al ser es hash generado por BLAKE2B de 512 bits frente a los 256bits de SHA-256, podemos concretar que la velocidad de Blake2b es superior.

Por otro lado, SCRYPT muestra unas diferencias claras en cuando a hash generados en un mismo intervalo de tiempo si lo comparamos con los otros dos algoritmos, de esto podemos concluir que SCRYPT es mucho más complejo, y también que la GPU utilizada no es propicia para este algoritmo.

Tras este breve análisis se ha recogido la información obtenida a través de las pruebas realizadas en la siguiente tabla.

Algoritmo	Tamaño hash salida	Hashrate(Hash/s)
SHA-256	256 bits	584.774
BLAKE2B	512 bits	762.443
SCRYPT	256 bits	16

Tabla 11. Datos relevantes extraidos de los test

Una vez extraídos los datos de las pruebas realizadas y analizados, vemos el hashrate (hashes generados por segundo) de cada uno de los algoritmos con la CPU que se han realizado los test de prueba. Lógicamente el hashrate los mineros de cada una de las criptomonedas que usan estos algoritmos es muy superior ya que tendrán ASICs, GPUs o máquinas enfocadas en esta función. Pero a nosotros para hacer los test y comparar nos sirve como simulador.

Al ver estos parámetros lo más destacado es el bajo hashrate de SCRYPT, esto evidencia una de las máximas del algoritmo que es uno de los más complejos y que trata de evitar que se fabriquen ASICs especializadas. Entre Blake2b y SHA-256 vemos que las diferencias no son muy importantes, aunque si queda claro que Blake2b es más rápido, primero porque tiene un hashrate superior y también por que el hash de salida tiene el doble de longitud lo que hace darle más valor al número de hashrate.

5. CONCLUSIONES

En este trabajo de fin de grado se han seguido los objetivos marcados inicialmente y que daban forma a la estructura de la memoria. Partiendo de un campo muy amplio se ha ido reduciendo caso a caso y apartado a apartado el punto de interés, alcanzando finalmente unos resultados que eran la manera deseada de completar el trabajo. Aun así, siendo tan amplio este mundo de algoritmos, tecnología blockchain y criptomonedas, se queda aún abierto a futuros trabajos donde aún se puede acotar más el área de conocimiento permitiendo así profundizar en un objetivo fijo.

La metodología seguida para el estudio y análisis obtenido ha sido inicialmente una búsqueda de información general que diera contexto al trabajo. Una vez asentadas las bases, se fueron decidiendo que partes generaban un mayor interés y deberían estar descritas en la memoria. Así, se comenzó con el estado del arte para poner punto de partida acerca de las criptomonedas y como los algoritmos de consenso tienen una gran relevancia en ella. El siguiente paso fue comprender los distintos métodos de consenso para poder determinar cuál de ellos era más interesante para el trabajo, una vez definido el método de consenso que fue PoW, se realizó una investigación de distintos algoritmos hash que trabajan de esta manera para finalmente hacer una selección de siete de ellos. Siguiendo la misma metodología fueron definidos todos ellos, y se decidió realizar la demo demostrativa con tres de estos algoritmos. Con los test realizados se recogieron los datos, y se llegó a la parte final del trabajo donde se realizaba un análisis en detalle de los resultados obtenidos.

Este trabajo fue planteado con una idea ambiciosa, y en su desarrollo se han visto ciertas limitaciones. Hemos visto que el mundo de las criptomonedas, con la tecnología blockchain y todos los algoritmos hash que existen tiene un alcance y detalle que hace inviable recoger toda esa información en un trabajo de esta envergadura. Además, no todos los datos y códigos son públicos y esto es una de las limitaciones más granes. Aún con estas dificultades, se ha conseguido adaptar el estudio y desarrollo de este trabajo para que de una manera representativa permita comprender en detalle el funcionamiento, las características y los métodos que usan los algoritmos, con todo esto ha sido posible dar respuesta a las incógnitas generadas cuando comenzó este TFG.

A nivel personal ha sido complicado el proceso hasta obtener el resultado final. Este trabajo se ha prolongado en el tiempo más de lo esperado en un inicio, debido a los cambios de estructura de la memoria, de datos e información se han realizado tratando de sintetizar y clarificar al máximo cada uno de los apartados. Una vez concluido puedo sentirme orgullo del resultado final, ya que han surgido dudas de si sería posible llegar al objetivo marcado, pero las dificultades han podido ser superadas.

Finalmente me gustaría destacar los puntos más importantes recogidos en el trabajo y la conclusión que he generado. Partiendo de un punto de partida muy genérico se ha conseguido abarcar mucha información de la manera más sintetizada posible, esto hace referencia a los algoritmos y métodos de consenso que han sido descritos. Este punto ha sido complicado por que reducir toda la extensión que supone la comprensión de cada uno de ellos no era una tarea sencilla, y era un punto clave para comprender el posterior desarrollo. Con la parte teórica clara, fue más sencillo definir las pruebas que se iban a realizar, ya que estaba claro que características y funcionalidades se querían comprobar. Por último los resultados obtenidos han dado validez al estudio previo, se ha podido demostrar que efectivamente los algoritmos hash cumplían con las características que su nombre les atribuye y se ha conseguido realizar la comparación pensada inicialmente entre algoritmos.

6. BIBLIOGRAFÍA

IEBSchool: "Blockchain: cómo la cadena de bloques revoluciona el sector financiero". Recuperadode:https://www.iebschool.com/blog/blockchain-cadena-bloques-revoluciona-sector-financiero-finanzas/

Revista Tecnológica ESPOL: "Mineros de criptomonedas en el contexto de la cadena de bloques". Recuperado de: http://rte.espol.edu.ec/index.php/tecnologica/article/view/828/520

INESDI Blog: "Algoritmo de consenso en blockchain: tipos y características". Recuperado de: https://www.inesdi.com/blog/algoritmo-consenso-blockchain/#:~:text=Un%20algoritmo%20de%20consenso%20es,los%20nodos%20tienen%20que%20seguir.

CoinMarketCap: "Estado del arte de las criptomonedas". Recuperado de: https://coinmarketcap.com/es/all/views/all/

Nakamoto, S. (2008). "Bitcoin: A peer-to-peer electronic cash system". Recuperado de: https://bitcoin.org/bitcoin.pdf

Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). "Ouroboros: Un protocolo de blockchain de prueba de participación con seguridad demostrable". Recuperado de: https://eprint.iacr.org/2016/889.pdf

Open Access: "Análisis y evaluación de algoritmos de consenso en blockchain y su aplicación en el desarrollo de aplicaciones descentralizadas". Recuperado de: https://openaccess.uoc.edu/handle/10609/127926

DSpace UCE (Universidad Central del Ecuador): "Estudio comparativo de algoritmos de consenso en blockchain". Recuperado de: http://www.dspace.uce.edu.ec/handle/25000/21832

Bit2Me Academy: "SHA-256: el algoritmo de Bitcoin". Recuperado de: https://academy.bit2me.com/sha256-algoritmo-bitcoin/

Bit2Me Academy: "¿Qué es el algoritmo de minería Ethash?". Recuperado de: https://academy.bit2me.com/que-es-algoritmo-de-mineria-ethash/

UVaDoc (Universidad de Valladolid): "Análisis comparativo de algoritmos de consenso en sistemas de criptomonedas". Recuperado de:

https://uvadoc.uva.es/bitstream/handle/10324/38340/TFG-J-95.pdf?sequence=1&isAllowed=y

HowToMine.com: "Litecoin Mining: Complete Beginner's Guide". Recuperado de https://howtomine.com/es/litecoin/

MDPI: "Consensus Algorithms in Blockchain: A Comparative Analysis". Recuperado de https://www.mdpi.com/1911-8074/13/11/263

UPM Open Access: "Sistema de blockchain para el registro de títulos académicos". Recuperado de https://oa.upm.es/70939/?trk=public_post-text

Universidad de Zaragoza: "Fuzzy hashing: proceso de identificación de ficheros basado en el análisis de contenido". Recuperado de http://webdiis.unizar.es/~ricardo/files/TFGs/FuzzyHashingProcesos.pdf.

7. ANEXO

En este anexo se recoge el código usado en el TFG.

Código test SHA-256:

```
import hashlib
import time

def minar_bloque(datos, dificultad_objetivo):
        objetivo = "0" * dificultad_objetivo
        nonce = 0
        while True:
                datos_bloque = datos.encode() + str(nonce).encode()
                hash_sha256 = hashlib.sha256(datos_bloque).hexdigest()
                if hash_sha256[:dificultad_objetivo] == objetivo:
                        return nonce, hash_sha256
                nonce += 1

datos = "TEST_SHA-256"
dificultad_objetivo = 7 # Dificultad objetivo (número de ceros al principio del hash)

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Código test BLAKE2B:

```
import hashlib
import time

def minar_bloque(datos, dificultad_objetivo):
        objetivo = "0" * dificultad_objetivo
        nonce = 0
        while True:
                datos_bloque = datos.encode() + str(nonce).encode()
                hash_blake2b = hashlib.blake2b(datos_bloque,
                digest_size=64).hexdigest()
                if hash_blake2b[:dificultad_objetivo] == objetivo:
                        return nonce, hash_blake2b
                nonce += 1

datos = "TEST_BLAKE2B"
dificultad_objetivo = 4 # Dificultad objetivo (número de ceros al principio del hash)

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Código test SCRYPT:

```
import scrypt
import time
import binascii

def minar_bloque(datos, dificultad_objetivo):
        nonce = 0
        while True:
                datos_bloque = datos.encode() + str(nonce).encode()
                hash_scrypt = scrypt.hash(datos_bloque, datos_bloque, N=16384, r=8,
                p=1, buflen=32)
                hash_scrypt_hex = binascii.hexlify(hash_scrypt).decode()
                hash_int = int(hash_scrypt_hex, 16)
                if hash_int < 2**(256 - dificultad_objetivo):
                        return nonce, hash_scrypt_hex
                nonce += 1

datos = "TEST_SCRYPT"
dificultad_objetivo = 1 # Dificultad objetivo

tiempo_inicio = time.time()
nonce_encontrado, hash_encontrado = minar_bloque(datos, dificultad_objetivo)
tiempo_fin = time.time()

print(f"Nonce encontrado: {nonce_encontrado}")
print(f"Hash encontrado: {hash_encontrado}")
print(f"Tiempo transcurrido: {tiempo_fin - tiempo_inicio:.6f} segundos")
```

Universidad de Alcalá