El protocolo blockchain y su aplicación en entornos distribuidos.

Luis Alberto Glaría Silva

GRADO EN MATEMÁTICAS. FACULTAD DE CIENCIAS MATEMÁTICAS
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado

Septiembre de 2019

Director:

Luis Llana Díaz

Resumen

Desde la creación del Bitcoin en 2008 y del crecimiento de las inversiones en criptomonedas a partir de 2012, el protocolo Blockchain ha ido despertando cada vez más interés por sus posibles aplicaciones en diversos campos.

En este trabajo nos alejaremos de las cuestiones económicas relacionadas con las criptomonedas para por un lado analizar los fundamentos matemáticos de esta tecnología y por otro lado ver la relación de este protocolo con ciertos problemas de la programación distribuida. Para ello se estudia el funcionamiento y corrección de una posible especificación del Blockchain. También se ha hecho una implementación de este protocolo donde se han podido llevar a la práctica algunas ideas desarrolladas en el trabajo.

Palabras clave

Blockchain, Criptografía de clave pública, Bitcoin, Funciones Hash, Consenso, Prueba de trabajo, Sistemas distribuidos.

Abstract

Since the creation of Bitcoin in 2008, and the growth of investments in crypto-currencies in 2012, the Blockchain protocol has created a great interest for its possible applications in various fields.

This paper will focus on the mathematical foundations of this technology and it's relation to certain problems of distributed programming rather than in the economic features of crypto-currencies. In order to achieve this objective the operation and correctness of a possible Blockchain specification is studied. An implementation of Blockchain was created, where it was possible to develop in practice some ideas of this document.

Keywords

Blockchain, public-key cryptography, Bitcoin, Hash functions, consensus protocol, proof of work, distributed systems.

Índice general

1. Introducción y objetivos.

En 1969 el Departamento de Defensa de los Estados Unidos instaló los primeros nodos de la *Advanced Research Projects Agency Network* (ARPANET), precursora de la actual Internet. Desde este momento surgió la necesidad de garantizar las comunicaciones entre varias computadoras incluso en el supuesto en que algunas de ellas dejaran de funcionar, o se interrumpieran parte de las conexiones que hacían posible la transmisión de información[21]. Nacen así las redes de ordenadores y con ellas el concepto de sistemas distribuidos.

Desde este momento surgieron diversas aplicaciones y protocolos, tales como Napster, Gnutella o Bittorrent [4], que buscaban resolver diferentes problemas de forma cooperativa y distribuida. El sector financiero no estuvo ajeno a estas transformaciones, y ya desde 1983 se teorizó sobre la posibilidad de de utilizar monedas digitales, aunque requiriendo de cierta entidad central que validara las operaciones [18]. También surgió la idea con B-Money [20] de utilizar una moneda digital completamente descentralizada, pero esta propuesta no llegó a ser implementada.

En el año 2008, con la publicación del artículo *Bitcoin: A Peer-to-Peer Electronic Cash System* [29], nace el protocolo Blockchain. Se desconoce la verdadera identidad de la persona u organización oculta bajo el seudónimo de Satoshi Nakamoto, autor de este documento. En el año 2009, el propio Nakamoto publicó el código, escrito en C++, de la criptomoneda Bitcoin. Posteriormente fueron creadas nuevas monedas digitales basadas en la idea original del Bitcoin y a partir del año 2014 se comenzaron a estudiar aplicaciones del Blockchain distintas de las criptomonedas.

Se puede definir entonces al Blockchain como un protocolo que permite mantener un registro distribuido e inmutable de las transacciones o comunicaciones entre los participantes de una determinada red. Y que para garantizar en todo momento que la información almacenada es veraz, se tiene también un mecanismo de consenso que no necesita de ningún tipo de centralización para funcionar. En este punto conviene hacer un apunte sobre la terminología usada, aunque en la literatura se suele utilizar *bitcoin* en minúsculas para hacer referencia al registro donde se almacena la información y reservar *Bitcoin* para hacer referencia a la tecnología o al protocolo, en este trabajo utilizaremos la traducción al español: *cadena de bloques* cuando se haga referencia a la estructura que conforma la base de datos.

Una aspecto importante de las primeras criptomonedas es su carácter completamente público, cualquier usuario de internet podía unirse a la red y participar activamente sin necesidad de ninguna autorización o verificación.

Todas estas implementaciones, independientemente de las variaciones en el protocolo,

tienen en común el carácter distribuido del algoritmo y la inmutabilidad del registro (cadena de bloques). Sin embargo, se pueden establecer 3 distinciones generales en base a como se trate el aspecto público y el nivel de descentralización [11]:

- *Blockchain público:* Cualquiera puede unirse a la red, todos los nodos pueden leer el registro, realizar transacciones y participar en el consenso.
- *Blockchain de consorcio:* En el algoritmo de consenso solo intervienen una parte de los nodos. El acceso a la red y las consultas al registro en principio están abiertas a todos. Este sería el caso de una red pública y centralizada.
- *Blockchain privado:* El acceso a la red no es público y además tanto la lectura del registro como la participión en el algoritmo de consenso pueden encontrarse limitados. Estamos ante una red privada que puede estar en mayor o menor medida centralizada.

Dado que es la variante más conocida y utilizada, y es a la que pertenece la primera criptomoneda, el Bitcoin, en este trabajo nos centraremos fundamentalmente en el Blockchain de tipo público.

Al igual que no existe una única alternativa en la elección de los niveles de centralización y privacidad, el mecanismo utilizado para alcanzar un acuerdo entre los participantes en la red (consenso) difiere entre las diferentes implementaciones del Blockchain. Sin embargo, para explicar las diferencias entre cada uno de estos algoritmos se deben definir y estudiar ciertas cuestiones referentes a la criptografía.

Será por tanto el primer objetivo de este trabajo dar las nociones matemáticas básicas que fundamentan el Blockchain. De eso tratará el capítulo 2. Una vez sea posible justificar parte del marco teórico de esta tecnología el siguiente paso será explicar su funcionamiento, tarea que será realizada en el 3. En este punto se explicarán de forma más extensa las diferencias entre algunas de las diferentes implementaciones de este protocolo existentes que han sido mencionadas de forma somera esta introducción. Hay que tener en cuenta que el desarrollo de este protocolo ha estado estrechamente ligado a las distintas implementaciones del mismo (o más bien al éxito de estas) y no tanto a un trabajo teórico sistemático.

Un tercer objetivo consistirá en estudiar el Blockchain dentro del contexto específico de la computación distribuida y de la resolución de uno de los problemas fundamentales de este campo. Sobre este tema versará el 4. Para finalizar, se realizará en el 5 una implementación del blockchain en la que se podrá comprobar que es posible llevar a la práctica, de forma relativamente sencilla, las ideas desarrolladas en este trabajo.

2. Criptografía y Blockchain

Garantizar la autenticidad de las transacciones entre los nodos y la integridad del registro es la tarea de la criptografía dentro del Blockchain. Esto es, que siempre será posible identificar el origen de la información de forma segura (*autenticidad*), y que tendremos la certeza que la totalidad o parte de los datos no han sido suprimidos o modificados (*integridad*)

La criptografía de clave pública, donde no es necesaria la existencia de un canal seguro para transmitir la información, será el método criptográfico elegido, pues justamente estamos en un entorno público donde toda la información transmitida es vista por los demás participantes. Este método, desarrollado en la década del 70 del siglo XX, solucionó algunas limitaciones de la criptografía de clave simétrica. En un entorno donde se use una misma clave tanto para cifrar como para descrifrar mensajes se debe disponer un canal seguro a través del cual esta clave sea distribuida.[32] Existen situaciones en las que no se puede garantizar la existencia de tal canal y por tanto el uso de claves simétricas es inviable.

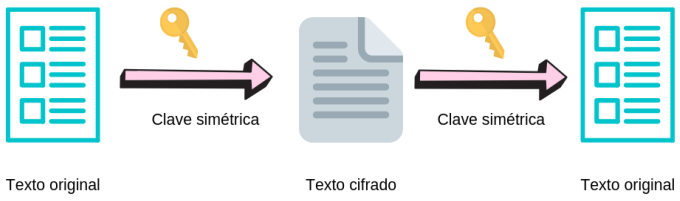


Figura 2.1: Criptografía de clave simétrica

La criptografía de clave pública se fundamentan en la generación de una dupla (p, q) donde p será la clave pública, que se transmitirá a todos los demás participantes, mientras que q es la clave privada solo será conocida por quien haya creado la dupla. A partir de p es computacionalmente infactible deducir q y por tanto cualquiera que haya recibido la clave pública puede usarla para cifrar un mensaje pero solo quien conozca q puede determinar el mensaje original.

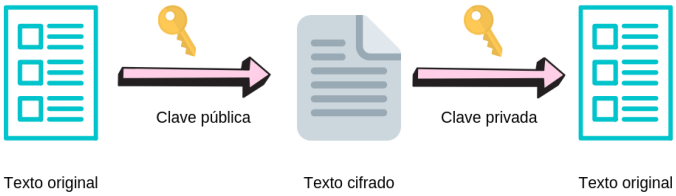


Figura 2.2: Criptografía de clave asimétrica

Dentro del Blockchain sin embargo el objetivo del cifrado no es ocultar información sino verificar la identidad de cada participante. Quien controle la clave privada puede usarla para firmar sus mensajes y la autenticidad de esta firma podrá ser verificada por todos aquellos que hayan recibido la clave pública. [19]

Para especificar el principal algoritmo de clave pública usado en el blockchain (el ECDSA Elliptic Curves Digital Signature Algorithm)[19] y explicar con más detalle el funcionamiento del proceso de autenticación hay que definir una serie de conceptos criptográficos y algebraicos.

2.1. Curvas elípticas y ley de grupo

Definición 2.1 (Curva elíptica)**.** *Una curva elíptica sobre un cuerpo* $\mathbb{K}$ *queda definida por la ecuación*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

con a_i *elementos del cuerpo tales que el discriminante* Δ *de la curva sea distinto de cero. El discriminante de una curva elíptica viene dado por la expresión:*

$$\begin{aligned}
\Delta = & -d_1^2d_4 - 8d_2^3 - 27d_3^2 + 9d_1d_2d_3 \\
d_1 = & \, a_1^2 + 4a_2 \\
d_2 = & \, 2a_4 + a_1a_3 \\
d_3 = & \, a_3^2 + 4a_6 \\
d_4 = & \, a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}$$

Con la condición $\Delta \neq 0$ *garantizamos que la curva sea "suave", es decir, que no tenga ningún punto con más de una recta tangente.*

Para los algoritmos que vamos a estudiar nos restringiremos a los casos en que $\mathbb{K}$ sea un cuerpo finito de característica p, con p un número primo distinto de 2 o 3. Bajo estas condiciones, y haciendo un cambio de variable, la ecuación general de una curva elíptica se puede reescribir como

$$y^2 = x^3 + ax + b \tag{2.1}$$

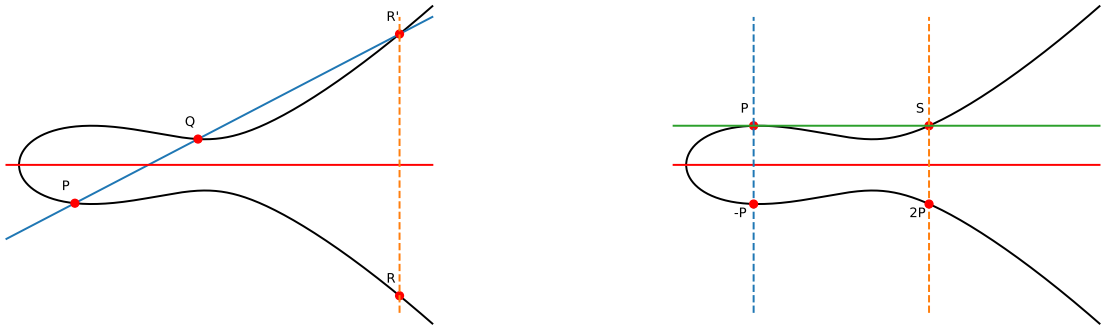
Y para que el discriminante sea distinto de cero los coeficientes a y b deben cumplir $4a^3 + 27b^2 \not\equiv 0 (\text{mod } p)$ En lo adelante cuando se hable de una curva elíptica nos estaremos refiriendo a 2.1 sobre un cuerpo primo de característica distinta de 2 o 3, aunque para dibujar la curva o visualizar algunas operaciones debamos suponer que la tenemos definida sobre un cuerpo de característica 0 como por ejemplo $\mathbb{R}$

A los pares (x, y) que cumplen 2.1 se les añade el punto del infinito y sobre esta estructura proyectiva se define la operación de suma y calculo del inverso como sigue:

Suma Dados dos puntos P, Q de la curva, la recta (proyectiva) que pasa por ellos corta a la curva en otro punto (si se trata de una recta vertical se dice que corta a la curva en el punto del infinito) la reflexión de este otro punto de corte respecto del eje de las abscisas será la suma de P y Q Un caso especial de la suma es cuando se quiere calcular P+P, en esta situación si $P = (x, y)$ con $y \neq 0$ se toma como $2P$ la reflexión respecto del eje x del punto de la corte de la tangente de P con la curva. Si $y = 0$ se toma como $2P$ el punto del infinito.

Inverso El inverso de un punto P es el resultado de su reflexión respecto del eje de las abscisas, así, si P tiene por coordenadas (x, y) entonces $-P$ tendrá coordenadas $(x, -y)$

Esta definición geométrica de las operaciones se aprecia mejor trabajando sobre los números reales, y así es como se suele representar en gráficos. En cualquier caso tal y como se hizo con el inverso, se puede dar una definición algebraica de la suma.



(a) Cálculo de $P + Q = R$ (b) Cálculo de $2P$ y $-P$

Figura 2.3: Curva elíptica $y^2 = x^3 - 3x + 5$

A partir de la definición anterior de las operaciones no es difícil ver que el punto del infinito es el elemento neutro de la suma, que esta operación de suma es conmutativa y que existe el inverso de todo elemento. Sin embargo que la suma está cerrada con esta definición y que se cumple la propiedad asociativa no son resultados inmediatos. Una prueba de esto se puede consultar en [17] y en [35]

Al cumplirse las cuatro propiedades anteriores tenemos que la suma así definida induce un grupo abeliano sobre los puntos de la curva elíptica. Sea $(G, +)$ este grupo y $n = |G|$ su orden. Sabemos que n no es infinito pues la curva está definida sobre $\mathbb{K}$ un cuerpo finito, y por tanto sus puntos vendrán dados por $\{(x, y)|y^2 \equiv x^3 + ax + b(mod p)\}$.

Teorema 2.1 (Teorema de Cauchy)**.** *Sea G un grupo finito y q un primo. Si q divide al orden de G entonces existe un elemento de G de orden q.*

Tenemos por tanto un grupo finito, donde a consecuencia del Teorema de Cauchy para grupos 2.1 podemos encontrar un elemento de orden primo q (si $n = |G|$ ya es primo basta tomar $q = n$) Este elemento g de G genera un subgrupo cíclico $(< q >)$ que será sobre el que se definan los protocolos criptográficos que veremos.

2.2. Funciones Hash

Antes de ver el algoritmo ECDSA hay que definir el concepto de función hash y ver alguna de sus propiedades.

Definición 2.2 (Función Hash)**.** *Una función hash es una aplicación H entre cadenas tal que su conjunto imagen está formado por cadenas de longitud fija, siendo esta magnitud la longitud hash de la función, mientras que su preimagen está formada por cadenas de longitud arbitraria. Se escribe entonces $H : \{0, 1\}^* \to \{0, 1\}^n$ si hablamos de cadenas de bits y denotamos por n a la longitud hash de H.*

Aunque se hable de longitudes arbitrarias pueden existir limitaciones técnicas en cuanto a la longitud del mensaje de entrada. Considerando por ejemplo la función hash SHA-256 (SHA son las siglas de Secure Hash Algorithm) usada ampliamente en diferentes implementaciones del protocolo blockchain, que devuelve cadenas de 256 bits [5]. Tenemos que esta función hash puede procesar mensajes de un máximo de $(2^{64} - 1)$ bits [31]. Es decir, que no se puede obtener el valor de hash de esta función para ninguna cadena que supere esa longitud, pero teniendo en cuenta que este valor equivale a más de 2 millones de terabytes esto no supone un verdadero problema en la práctica. Por otro lado, una importante condición que deben cumplir las funciones hash es que el cálculo de *valores hash*, esto es, obtener el resultado de aplicarle a algún elemento del espacio de partida cierta función hash, debe ser una tarea computacionalmente *simple*.

Una primera consecuencia de la definición 2.2 es que la preimagen de una función hash tiene cardinal estrictamente mayor que su imagen. Por tanto, por el principio del palomar, tendremos que si A es el conjunto preimagen de cierta función hash H y B es su imagen para todo $b \in B$ $\exists\{a_1, a_2\}$ con $a_i \in A$ distintos tales que $H^{-1}(b) = a_i$. Este fenómeno se llama colisión y se define en la resistencia a colisiones de una función hash como la probabilidad de encontrar un par (a_0, a_1) con $a_0 \neq a_1$ tales que se cumpla $H(a_0) = H(a_1)$. A toda función hash H, que usemos en el protocolo blockchain o en el algoritmo ECDSA, le pediremos que ningún algoritmo pueda encontrar una colisión para H en tiempo polinómico, en este caso diremos que H es resistente a colisiones. Otras dos propiedades importantes de las

funciones hash serán la resistencia a preimágenes y a segundas preimágenes. La primera de estas propiedades viene a decir que debe ser *difícil* dado un b en el espacio de llegada encontrar un a en el espacio de partida tal que $H(a) = b$. La segunda propiedad es similar a la definición de resistencia a colisiones, dado un a_0 elegido de forma aleatoria debe ser *difícil* encontrar un a_1 distinto de a_0 tal que $H(a_0) = H(a_1)$. Esta noción de *difícil* viene dada por la imposibilidad de resolver en tiempo polinómico cualquiera de estos problemas.

La propiedad de resistencia a segundas preimágenes es más fuerte que la resistencia a colisiones como consecuencia de la paradoja del cumpleaños. En efecto, si suponemos cadenas binarias y una longitud hash de n nuestra función devolverá exactamente 2^n cadenas distintas. Dadas l cadenas elegidas al azar la probabilidad de no encontrar una colisión viene dada por:

$$NC(2^n, l) = \frac{2^n}{2^n}\frac{2^n-1}{2^n}\cdots\frac{2^n-(l-2)}{2^n}\frac{2^n-(l-1)}{2^n} = \prod_{i=0}^{l-1}(1-\frac{i}{2^n}) \tag{2.2}$$

Así que la probabilidad de encontrar una colisión $C(2^n, l)$ será su complementario:

$$C(2^n, l) = 1 - \prod_{i=0}^{l-1}(1-\frac{i}{2^n}) \tag{2.3}$$

Considerando que el cociente $i/2^n$ es cercano a cero, que es consecuencia de tomar n suficientemente grande, y teniendo en cuenta que la expansión en Serie de Taylor de e^x se puede truncar a partir del término cuadrático para $x \ll 1$. Tenemos que:

$$C(2^n, l) \approx 1 - \prod_{i=0}^{l-1}\exp\left(\frac{-i}{2^n}\right) = 1 - \exp\left(\sum_{i=0}^{l-1}\frac{-i}{2^n}\right) \tag{2.4}$$

Este último sumatorio se puede escribir como $-(l-1)l/2^{n+1}$ usando la fórmula de la suma de los $(l-1)$ primeros naturales. Y que para l suficientemente grande se puede aproximar a $-l^2/2^{n+1}$. Tomando logaritmos y despejando se tiene que:

$$l \approx 2^{n/2}\sqrt{-2\log(1-C(2^n, l))} \tag{2.5}$$

Si se considera una probabilidad $C(2^n, l) = 0{,}5$, es decir que de un total de 2^n elementos distintos la probabilidad de encontrar en un grupo de l elementos elegidos al azar al menos 2 iguales es del 50 % entonces se tiene que $\sqrt{-2\log(1-C(2^n, l))} \approx 1{,}18$. Así $l \approx 1{,}18 * 2^{n/2}$, que es el número mínimo de cadenas binarias necesarias para garantizar una colisión con una probabilidad del 50 %

Sin embargo, la probabilidad de encontrar una segunda preimagen comprobando solo $1{,}18\sqrt{2^n}$ cadenas es de

$$\frac{1{,}18\sqrt{2^n}}{2^n} = \frac{1{,}18}{2^{n/2}} \tag{2.6}$$

Para $n > 20$ la probabilidad anterior prácticamente cero (En la función SHA-256 se tiene que $n = 256$).

Para garantizar una probabilidad de encontrar una preimagen de al menos el 0.5 se tiene que el número de k de cadenas a comprobar debe ser de al menos:

$$k = \frac{1}{2}2^n = 2^{n-1} \tag{2.7}$$

Donde se ha utilizado, al igual que en 2.6, la *Regla de Laplace* (Número de casos favorables entre Número de casos posibles). Como se ve, para n suficientemente grande el coste de encontrar una preimagen es prácticamente el mismo que el de comprobar todas las posibles cadenas que produce nuestra función hash.

Tomando $2^n = 365$, es decir, ahora el número de casos posibles es 365 (ya no estamos mirando cadenas binarias) se obtiene el problema del cumpleaños: para garantizar una colisión (dos individuos que cumplan años el mismo día) con una probabilidad de al menos 50 % basta con reunir 23 personas, este resultado es mucho menor que el valor que podríamos dar intuitivamente, de ahí que se le llame paradoja aunque desde el punto de vista estrictamente lógico no lo sea.

Hay que señalar que los cálculos anteriores solo son válidos bajo ciertos supuestos teóricos que en la práctica no se dan, pues ninguna función hash lleva los elementos del espacio de partida al de llegada siguiendo patrones completamente aleatorios. Una vez han sido encontradas colisiones (o se ha probado que la probabilidad de encontrarlas es *alta*) la función hash deja de ser considerada segura. Un ejemplo de función hash en esta categoría es SHA-1, para la que se conocen colisiones desde 2017 [33], aunque era considerada insegura desde 2005. La función hash usada en la mayor parte de versiones del protocolo Blockchain, así como en la implementación realizada en este trabajo, SHA-256, es considerada segura hasta la fecha actual.

2.3. Algoritmo ECDSA

Ahora podemos especificar el algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm),[19] tanto el usado para la firma de mensajes como el correspondiente algoritmo de verificación. Este algoritmo se basa en la dificultad computacional de resolver el problema del logaritmo discreto para curvas elípticas (ECDLP).

Definición 2.3 (Problema del logaritmo discreto para curvas elípticas)**.** *Sea* $E(\mathbb{K})$ *una curva elíptica sobre un cuerpo* $\mathbb{K}$*. Sea* $P \in E(\mathbb{K})$ *y* $Q \in < P >$*. Encontrar* k *tal que* $Q = kP$*.*

La definición anterior tiene sentido pues como se vio en 2.1 los puntos de una curva elíptica definida sobre un cuerpo finito forman un grupo y por tanto se puede hablar del grupo generado por cualquier elemento la curva. Así, se tiene que $< P >$ es el subgrupo cíclico generado por el punto P de la curva $E(\mathbb{K})$.

No se conoce algoritmo que resuelva este problema en tiempo subexponencial [27]. Sin embargo, aunque aquí no serán especificados se necesitan algoritmos apropiados para crear y representar el cuerpo finito $\mathbb{F}_p$ de forma que se garantice la intratabilidad de este problema.

Antes de poder firmar un mensaje hay que generar las claves públicas y privada. Sea $P = (x_1, y_1)$ un punto de la curva elíptica $E(\mathbb{F}_p)$ que genera un subgrupo cíclico $< P >$ de orden primo n. El par (d, Q) de claves privada y pública se genera usando el algoritmo 1. La elección aleatoria de d es muy importante, en otro caso se pueden tener problemas de seguridad que permitan encontrar la clave privada utilizada.

Algoritmo 1 Generación de claves

1: Se elige $d \in [1, n-1]$ de forma aleatoria
2: Se calcula $Q = dP$
3: Q es la clave pública y d la clave privada.

Una vez se ha generado la clave privada y utilizando una función hash H que devuelva cadenas de longitud menor o igual que n se pueden firmar mensajes usando el algoritmo 2

Algoritmo 2 Generación de firma

1: **repeat**
2: Se elige $k \in [1, n-1]$ de forma aleatoria
3: Se calcula $z_1 = kx_1$
4: Se calcula $r = z_1 \mod n$ si $r = 0$ paso 1.
5: Se calcula $e = H(m)$
6: Se calcula $s = k^{-1}(e + dr) \mod n$. Donde k^{-1} es el inverso de k en el grupo $< P >$
7: **until** $s \neq 0$
8: Se devuelve (r, s)

Ahora, quien recibe un mensaje m con la firma (r, s) puede verificar su procedencia (suponiendo que ha recibido antes la clave pública Q) mediante el algoritmo 3

Algoritmo 3 Validación de firma

1: Verificar que $r, s \in [1, n-1]$, en otro caso **se rechaza la firma**
2: Se calcula $e = H(m)$
3: Se calcula $w = s^{-1} \mod n$
4: Se calcula $u_1 = ew \mod n$, $u_2 = rw \mod n$
5: Se calcula $X = u_1P + u_2Q$, si $X = \infty$ **se rechaza la firma** ▷ $X \in E(\mathbb{F}_p)$
6: Sea $X = (x_1, y_1)$, se calcula $v = x_1 \mod n$
7: Si $v = r$ **se acepta la firma**, en otro caso **se rechaza la firma**

La importancia de la resistencia a colisiones en la función hash H queda explicada con los algoritmos anteriores. La existencia de colisones conocidas (o facilmente calculables) hace que ya no sea posible garantizar la validez de una firma. Un atacante podría usar un mensaje firmado, buscar alguna modificacion de este mensaje que combinada con la firma devuelva el mismo valor hash que en el original. Ahora quien verifique este mensaje falso seria incapaz de distinguirlo del original.

3. Especificación del protocolo blockchain

Una vez se han visto las dos nociones criptográficas fundamentales en las que se fundamenta el blockchain (algoritmo ECDSA y funciones Hash) es posible explicar tanto el funcionamiento de este protocolo como justificar su corrección. Aunque las ideas fundamentales que serán expuestas están basadas en el artículo original del Bitcoin [29] en los puntos en los que existan divergencias entre distintas implementaciones del protocolo se intentarán señalar las diferentes alternativas. Siendo esta una tecnología que se ha desarrollado más con la práctica a través de diversas implementaciones que con trabajos teóricos resulta complicado abarcar en un solo trabajo todas sus variantes.

3.1. Objetivos del Blockchain

El Bitcoin surgió con el objetivo de por un lado eliminar la necesidad de una entidad de confianza que actúe de intermediario en las transacciones económicas digitales y por otro lado de resolver el problema derivado de la ausencia de este ente [29]. En un intercambio basado en artículos físicos a los que se le asigna cierto valor (el papel moneda por ejemplo) los únicos problemas de los que las partes deben preocuparse son aquellos derivados de la estafa o la falsificación. Cuando estas transacciones se realizan sin intervenir medios físicos no se puede garantizar que la parte encargada de entregarle la retribución monetaria a la otra posea los medios económicos que dice tener. Más aún, alguien aún disponiendo de la capacidad de realizar cierta transacción puede enviarle a diferentes individuos cantidades que por separado no superen sus fondos, pero que en conjunto sí lo hagan, este es el llamado problema del doble gasto. Los intermediarios cumplen entonces la función de evitar que esto ocurre al llevar un registro actualizado de las transacciones de cada cliente y por tanto de los fondos de los que dispone.

Pero, ¿qué sentido tiene entonces eliminar el intermediario si con ello solo generamos problemas que debemos resolver? Un sistema descentralizado tolera mucho mejor las particiones que uno centralizado: basta aislar al servidor para dejar inoperativa toda la red. Además, desde el punto de vista de la seguridad cualquier ataque con éxito al servidor supone a su vez la caída de todo el sistema. Incluso suponiendo que se dispone de un intermediario seguro y distribuido en diferentes nodos de forma que la caída de alguno de ellos no suponga mayor

problema para la red, la pregunta que debemos hacernos es cuanto podemos llegar a confiar en esta entidad. En la medida en que controla nuestro dinero puede llegar a actuar de forma más o menos arbitraria con él ya sea por interés propio o al servicio de algún poder externo. Ejemplos en la vida real hay muchos.

El objetivo será por tanto proporcionar un protocolo seguro, que aún estando completamente descentralizado pueda resolver en particular el problema del doble gasto y en general problemas de consenso de otra clase. Por tanto, la corrección de este protocolo vendrá dada por la comprobación de que justamente se han solucionado estas cuestiones.

3.2. Configuración de la red

El protocolo Blockchain se define e implementa sobre una red basada en la arquitectura peer-to-peer (red entre pares). El término peer-to-peer (en lo adelante P2P) viene a decir que al contrario que en un sistema cliente/servidor aquí todos los nodos realizan potencialmente las mismas funciones. Este punto puede no ser del todo cierto en determinadas implementaciones como por ejemplo en Blockchain de consorcio o privados, pero en estos casos es el protocolo elegido y no la estructura de la red quien restringe las funciones de algunos nodos. Estas restricciones no limitan en ningún caso la capacidad de enviar o recibir información directamente entre los participantes, sino que está controlada que clase de información determinados nodos pueden enviar y como los demás interpretan estos mensajes. El Bitcoin es el ejemplo más importante de red blockchain donde no hay ninguna clase de jerarquía

Aunque todos los nodos teóricamente puedan realizar las mismas funciones en la práctica suelen elegir que tareas realizar y cuales ignorar. Así, algunas máquinas pueden dedicarse a ejecutar las labores más complejas del algoritmo de consenso (más adelante veremos en que consiste este algoritmo), otras almacenan parte (o la totalidad) de la cadena de bloque, mientras que otros nodos ejecutan tareas de verificación. Existen otras especializaciones (o combinaciones de especializaciones) que pueden variar entre diferentes implementaciones del blockchain.

Un problema importante en las redes es P2P es como los participantes se encuentran unos con otros para establecer una conexión. Los nodos mantienen un registro privado de las direcciones IP con las que se han comunicado, y pueden compartir este registro con otros miembros que lo soliciten, esta sería la forma estándar de ampliar la agenda de contactos. La primera vez que un nodo se une a la red dispone de una lista de direcciones IP prefijada, además, en ciertos protocolos blockchain que han alcanzado un número de participantes considerable existe también la posibilidad de conectarse a varias DNS *Seed* (DNS son las siglas de *Domain Name System*) donde se recibe una lista generada aleatoriamente de otros nodos que han estado activos en la red en un rango de tiempo determinado. Los servidores DNS sirven para asociar direcciones IP con nombres de dominio, en el caso del protocolo blockchain en general y del Bitcoin en particular, las direcciones IP almacenadas se corresponden con nodos de la red. Este punto, dado que es el único en el que existe algo de centralización, entraña ciertos riesgos, pues alguno de estos DNS *Seed* puede caer bajo el control de una

entidad maliciosa que le proporcione a los nuevos participantes solo direcciones ip de nodos bajo su control, de esta forma se podría crear una red independiente de la principal donde existe una cadena de bloques generada de forma arbitraria. Por ello se mantiene cierto control de forma pública y abierta sobre estas DNS *Seed* y se puede igualmente usar direcciones IP recibidas a través de cualquier otra vía.

3.3. Cadena de bloques

El componente fundamental del blockchain es la estructura de datos donde se almacena toda la información que los nodos que participan en el protocolo han acordado que es válida. En las criptomonedas este registro cumple la función de libro de cuentas, sin embargo, se pueden guardar todo tipo de datos.

Cada uno de los bloques que conforman esta cadena además de la propia información que pueda contener mantiene una referencia al bloque anterior. La excepción a esto es el bloque inicial, que se genera de forma diferente al resto. Esta estructura enlazada no es suficiente para garantizar la integridad de la cadena de bloques, pues un adversario podría modificar la información almacenada en un bloque dejando intacta la referencia al bloque anterior. Aquí es donde se utilizan por primera vez las funciones hash definidas en 2.2: además de la información almacenada y la referencia al bloque anterior guardaremos también el resultado de aplicar cierta función hash al conjunto de los datos del bloque (incluido quien es el bloque anterior). Si la función elegida es resistente frente a segundas preimágenes se puede verificar que el bloque es correcto simplemente aplicándole la función Hash a sus datos y comparando con la almacenada. Y la referencia al bloque anterior en lugar de ser únicamente un índice incluirá también el valor hash de ese bloque.

Sin embargo, aún con este añadido seguimos sin tener garantizado que un adversario pueda hacer modificaciones en la cadena. Es cierto que ahora para hacer cambios en un único bloque tendría que generar nuevamente toda la cadena a partir de ese bloque, pero el coste de operar con funciones hash es lineal, así que es factible hacer estas modificaciones en un tiempo razonable. Además de la referencia al bloque anterior y la información del bloque actual en los datos que sobre los que se aplica la función hash se incluirá un valor generado de forma aleatoria, así se garantiza que la cadena hash resultante tenga ciertas propiedades definidas en el protocolo. Por ejemplo, en el Bitcoin a la cadena hash se le pide que comience por cierto número de ceros. El cálculo de este valor aleatorio, denominado nonce (number used once), es la tarea principal del algoritmo de prueba de trabajo que se verá en el siguiente apartado.

Otra información que se incluye en los bloques, asociada con la función hash, es una marca temporal (*timestamp*) del momento en que fue creado el bloque. De esta forma se puede probar que la información existía cuando se calculó el valor hash. Mantener un registro temporal también servirá en el algoritmo de consenso para rechazar directamente aquellos bloques con marcas de tiempo superiores al momento actual.

Por último, y con el objetivo de reducir el uso de espacio, el conjunto de operaciones almacenadas en un bloque no se guardan explícitamente. En su lugar se almacena la raíz

del árbol Merkle resultante. En la siguiente sección esta estructura de datos será analizada.

Lo anterior se puede considerar más o menos estándar en todas las implementaciones del blockchain, pero se pueden incluir otros elementos dentro de cada bloque.

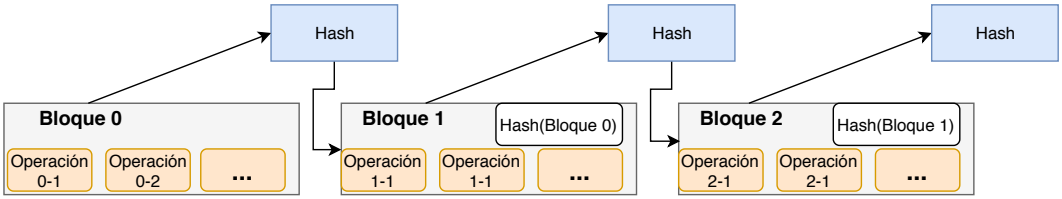


Figura 3.1: Diagrama Proceso Blockchain

3.4. Árboles Merkle

Los árboles Merkle, patentados en 1979 por Ralph Merkle[28] han sido usados activamente en la verificación de bases de datos. En el caso del Blockchain para cada transacción se calcula su valor hash, combinando estos resultados en pares se reduce a la mitad el número de valores que hay que almacenar. Repitiendo este proceso, ahora para los valores resultantes se llega a un valor hash final, llamado raíz hash, que se ha calculado de forma determinística a partir de los valores originales (transacciones).

Por ejemplo, si se desean guardar las transacciones A,B,C,D en cierto nodo calculamos en primer lugar, H(A), H(B), H(C), H(D) donde H es la función hash que estemos utilizando. Posteriormente se calcula H((H(A),H(B) y H(H(C),H(D)), por último queda aplicarle la función H a estos dos valores, y se tiene entonces la raíz Merkle que se incluirá en el bloque.

Por las propiedades de las funciones hash vistas antes, se garantiza la integridad de las transacciones almacenadas, pues cualquier ligera modificación en las mismas produciría cambios en la raíz Merkle resultante y por tanto ese bloque se rechazaría. Ademas, la propia estructura de los arboles Merkle nos permite encontrar que transacción ha cambiado, pues basta descender desde la raíz comparando cada uno de los valores hash almacenados hasta encontrar la transacción, que viene a ser una hoja del árbol, que ha sido modificada.

Los árboles Merkle permiten de esta forma hacer verificaciones sobre el estado de la cadena de forma rápida y permitiendo un menor uso de almacenamiento.

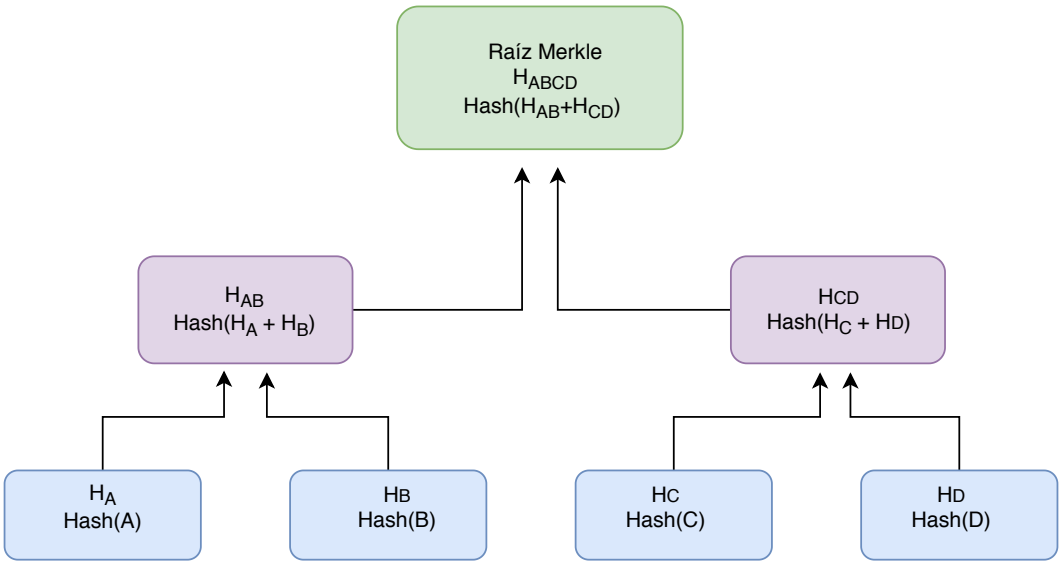


Figura 3.2: Diagrama Árbol Merkle

3.5. Algoritmo de prueba de trabajo

La estructura en cadena de bloques enlazados usando funciones hash no resuelve por si sola el problema del doble gasto visto en 3.1, para eso hay que establecer un algoritmo que permita a los participantes decidir que información se puede incluir en la cadena de bloques, validando de esta forma las transacciones correctas. En el Bitcoin se estableció como tal sistema el algoritmo de prueba de trabajo (Proof of Work) [29], en posteriores implementaciones del Blockchain, salvo ciertas modificaciones se ha mantenido en general este mismo algoritmo. La idea de la prueba de trabajo se originó para resolver el problema del uso abusivo de ciertos recursos como el correo electrónico mediante el envío masivo de mensajes de spam. Hashcash [16], una de las primeras implementaciones de un mecanismo para solucionar este problema, se basaba en usar una función de coste para generar una ficha que permitiría disponer de cierto recurso. Esta ficha probaba que el usuario había invertido determinado tiempo de procesamiento en resolver el problema planteado por la función de coste y de esta forma el uso del recurso en cuestión estaba condicionado por la resolución de cierto problema que conllevaba determinado uso de CPU por parte del usuario. Con el Bitcoin este planteamiento basado en un sistema cliente-servidor fue extendido a un contexto descentralizado.

El resultado de ejecutar la función de coste (ficha) debe poder ser verificado fácilmente por cualquiera, mientras que el cálculo de esta función debe ser una tarea moderadamente complicada. La funciones hash cumplen estos requisitos y son usadas como función de coste. Un participante que quiera validar un bloque, es decir, que quiera afirmar que cierto conjunto de transacciones o datos son válidos y que por tanto se pueden añadir al registro toma el bloque tal y como se definió en 3.3 le añade cierto valor en bits cuya posible longitud

y posición dentro del bloque están determinados en el protocolo y analiza el resultado de aplicarle cierta función hash acordada. Si este resultado cumple ciertos requisitos (por ejemplo la cadena hash devuelta empieza por cierto número de *ceros*) entonces se dice que se ha resuelto el problema del algoritmo de prueba de trabajo (llamado puzzle en algunas fuentes) y se comparte con los demás nodos este resultado. Puede encontrarse también en ciertos artículos y especificaciones de criptomonedas que el puzzle que se debe resolver para validar cierto bloque consiste en encontrar una cadena hash menor que cierta longitud. Esto no es más que buscar cadenas hash que comiencen con ciertos número de ceros y considerarlas como de tipo numérico donde esos ceros desaparecen dejando una cadena n posiciones más corta que la longitud hash de la función del algoritmo.

En el algoritmo 4 se presenta una posible especificación (muy simplificada) del método de prueba de trabajo para calcular un *nonce* asociado a cierto bloque. En esta especificación se supone que quien la ejecuta ha verificado antes que todas los datos que se pretenden incluir en la cadena son correctos. Este algoritmo tal y como está planteado puede no terminar, pues es posible que no exista ningún *nonce* de longitud menor o igual que m que resuelva el puzzle. En la práctica si el bloque b está formado por cadenas de la forma $b = s+p+k+t_1+\ldots+t_q$ donde s es el *timestamp*, p la referencia al valor hash del bloque anterior, k es la clave pública del nodo que está calculando el *nonce* y t_i el conjunto de datos que almacena el bloque, se suele proceder reordenando los t_i a la vez que se modifica el *nonce* y comprobando el resultado de estas combinaciones. También se suelen hacer pequeñas modificaciones en el *timestamp* s, pues entre la hora marcada por el bloque anterior y lo que se tarda en generar el siguiente bloque hay un rango de tiempo en el que moverse. Este proceso es lo que se conoce en las criptomonedas como minado. Las implementaciones de este algoritmo deben incluir, una condición de parada extra, pues no tiene sentido seguir operando si alguien ya ha resuelto el puzzle, así, se debe mantener un proceso que en caso de recibir información sobre la resolución (correcta) del problema de prueba de trabajo para el bloque en el que se está trabajando detenga el algoritmo.

Algoritmo 4 Prueba de trabajo (Cálculo)

Entrada:

H función hash del protocolo de longitud de cadena k.
$l < k$ longitud (dificultad) del puzzle.
m longitud máxima del nonce.
b bloque que se quiere validar.

1: Generar *nonce* de forma aleatoria tal que $|nonce| \leq m$
2: **si** $|H(nonce + b)| \leq l$ **devolver** $(nonce, b)$
3: **en otro caso**Volver a 1

Se incluye también el algoritmo de verificación que deben utilizar los nodos para comprobar el par $(nonce, b)$. Si acepta el bloque la forma de comunicarlo al resto de la red es incluir ese bloque en la cadena y empezar a trabajar en uno nuevo que lo tenga como predecesor. Puede ocurrir que diferentes nodos acepten distintos bloques en determinado punto, esto se denomina bifurcación y se trata en la siguiente sección.

Algoritmo 5 Verificación de la prueba de trabajo

Entrada:

H función hash del protocolo de longitud de cadena k.
$l < k$ longitud (dificultad) del puzzle.
m longitud máxima del nonce.
$(nonce, b)$

1: Se comprueba que b está formado por datos válidos (bloque previo, timestamp y transacciones). Si falla cualquiera de estas comprobaciones **rechazar**
2: **si** $|H(nonce + b)| \leq l$ **devolver aceptar el bloque**
3: **en otro caso rechazar**

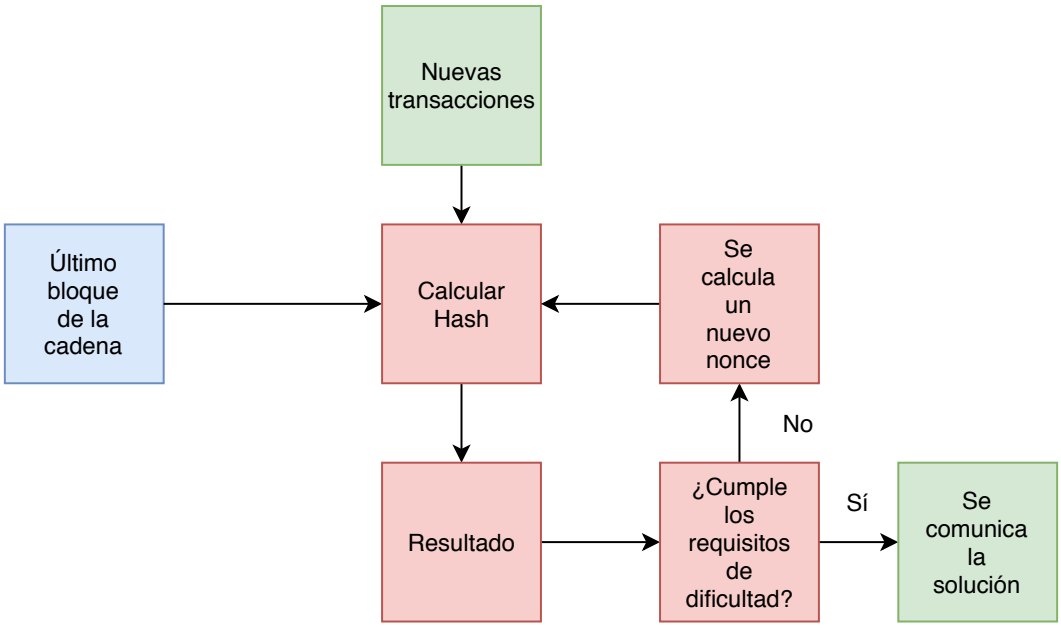


Figura 3.3: Diagrama del proceso de prueba de trabajo

La dificultad del puzzle, es decir, la condición que se le pide al valor hash del bloque para ser aceptado, en general no se mantiene fija. Cada determinado número de bloques (fijado en el protocolo) se vuelve a calcular este valor usando cierta función dependiente de la dificultad actual del puzzle. De esta forma según crece la cadena es necesario invertir más recursos en añadir un nuevo bloque.

3.6. Escalabilidad del blockchain

Aunque a medida que crece la cadena de bloques el algoritmo de prueba de trabajo garantiza una mayor resistencia a posibles ataques, surgen nuevos problemas al aumentar el número de participantes activos en la red.

Por un lado, se tiene que el tamaño máximo de cada bloque (establecido en el protocolo), es una limitación al número de transacciones que se pueden almacenar en cada ejecución del algoritmo de prueba de trabajo. A mayor números de participantes mayor será el número de operaciones y por tanto es más probable que surjan *cuellos de botellas* con transacciones que deben esperar mucho tiempo por ser incluidas en la cadena.

Esto afecta fundamentalmente a la posibilidad de las criptomonedas de convertirse en alternativas serias a los métodos de pago digitales centralizados. Visa, por ejemplo, puede procesar unos 24000 pagos por segundo, mientras que criptomonedas como Ethereum o Bitcoin están limitadas a apenas 20 transacciones por segundo.[13]

Una solución parcial a este problema ha sido aumentar el tamaño límite de los bloques admitidos en la cadena para poder almacenar más operaciones en cada ejecución del algoritmo de prueba de trabajo. Sin embargo, esta clase de cambios en el protocolo deben ser primero aceptadas por todos (o al menos buena parte) de los participantes para ser efectivas. Esto es lo que se conoce como bifurcaciones del protocolo y se analizará en la sección 3.9

3.7. Uso de GPUs

El algoritmo de prueba de trabajo visto en 3.5 se puede implementar usando computación paralela. De hecho esta es en general la forma óptima[15] de implementarlo, pues cada procesador se encarga por separado de analizar cierto grupo de posibles valores para encontrar la solución del puzzle. Las unidades de procesamiento gráfico o GPU por sus siglas en inglés (Graphic Processing Unit) están dotadas de un mayor numero de unidades aritmético-lógicas así como de hilos de proceso, por este motivo resultan mucho mas efectivas en comparación con las CPU para realizar el minado. En los supuestos en los que se han tomado medidas en el algoritmo de consenso para evitar el uso de hardware especifico de minado se convierten en la opción mas efectiva.

Esto, por otro lado contribuyó el aumento de la demanda y por lo tanto del precio de las tarjetas gráficas[10] Una alternativa a las GPUs es el uso de circuitos integrados de aplicación específica o ASIC por sus siglas en inglés (application-specific integrated circuit). Se trata de circuitos integrados diseñados para realizar cierta tarea específica [1], en el caso de las criptomonedas esta tarea es resolver la implementación del algoritmo de prueba de trabajo utilizado. Existen sin embargo variantes del algoritmo de prueba de trabajo que introducen cierta resistencia tanto al minado a través de ASIC como mediante GPU [2].

3.8. Resistencia a ASIC

Con el aumento en la popularidad del Bitcoin, y de otras criptomonedas similares, se generalizó el uso de hardware dedicado exclusivamente al minado, es decir, a resolver el problema planteado por el algoritmo de prueba de trabajo visto en 3.5. El desarrollo de esta clase de hardware ha hecho que resulte prácticamente imposible utilizar ordenadores de uso doméstico durante el proceso de minado, lo que llevado a múltiples usuarios a dejar

de participar en el proceso de creación y validación de la cadena de bloques, delegando en terceros este proceso. Esta delegación de la confianza en una tercera entidad va en contra de los principios fundamentales del Blockchain. Se tienen entonces dos problemas fundamentales: por un lado el desarrollo de hardware especifico que en la práctica excluye a una parte de los usuarios de participar en el algoritmo de prueba de trabajo, por otro lado, este algoritmo hace posible delegar en otra entidad (los llamados mineros) el proceso de creación de nuevos bloques en la cadena, haciendo que parte de los usuarios no participen activamente en el proceso.

Figura 3.4: Ejemplo de dispositivo ASIC (Fuente: amazon.com)

Estos son los motivos del desarrollo de algoritmos resistentes al minado mediante ASIC'S y que a su vez intentan minimizar la creación de nodos dedicados exclusivamente a este proceso, como las llamadas mining pools del Bitcoin. Uno de los principales algoritmos cuyo objetivo es combatir estos problemas es Hashimoto [22]. Se trata de un algoritmo basado en E/S, otros algoritmos creados para este fin se basaban por ejemplo en el mayor uso de memoria, pero para estos ha ido surgiendo hardware específico.

La idea base detrás de Hashimoto no es hacer un algoritmo que sea resistente a ASIC, sino un algoritmo que funcione de forma óptima en ordenadores de uso doméstico, basado en el hecho que los límites de lectura-escritura son un problema estudiado durante años en el campo de la computación y es poco probable que el minado de monedas produzca avances significativos en esta área, y en caso de producirlos su trascendencia seria tal que impactarían en la industria del hardware en general. Tanto en el caso de del algoritmo Hashimoto, como en el algoritmo de prueba de trabajo visto en 3.5 un valor hash correcto es aquel que cumple que su longitud es menor que cierto valor dado, es decir, que el valor hash empieza por n-ceros.

Este valor hash resulta de aplicar cierta función hash (SHA-256 en el caso del Bitcoin) al bloque anterior, raíz Merkle y cierto nonce. La raíz Merkle como se vio en está basada en

las transacciones del bloque que se quiere incluir.

Hasta este punto Hashimoto y el método de prueba de trabajo visto antes coinciden. En lugar de detenerse en el valor hash obtenido hasta este momento en Hashimoto se realiza un segundo procesado de los datos. El conjunto de transacciones almacenados en la cadena de bloques puede considerarse ordenado. Por ejemplo la transacción número 10 del bloque 11 , podría tener el índice numero 560. Lo que hace ahora Hashimoto es acceder a diferentes puntos de la cadena, basándose en el orden anterior y en el valor hash calculado originalmente. El numero de transacciones de la cadena a las que accede es un numero fijado por el algoritmo. Se tiene entonces un conjunto de operaciones extraídas de diferentes puntos de la cadena de bloques para las que se calcula su valor hash.

El resultado (solución del algoritmo de prueba de trabajo) sera el nonce que resuelva el algoritmo de prueba de trabajo para esta combinación de valores. Así, se tiene que Hashimoto depende de acceder a casi la totalidad de la cadena de bloques. Para cadenas de bloques pequeñas esto no es un problema, pues pueden tenerse en memoria. Pero cuando estas cadenas deben mantenerse en disco por su longitud, es entonces cuando se aprecia el efecto de los tiempos de acceso de E/S. Además, se obliga a los participantes a tener una copia completa de la cadena de bloques, de otra forma dependen de la respuesta de otros nodos que almacenen el bloque para el cual quieren consultar cierta transacción.

El algoritmo Etash, usado actualmente en la criptomoneda Ethereum está basado en Hashimoto. Este algoritmo busca 4 objetivos fundamentales [9]:

- *Saturar E/S* Debe forzarse el uso de casi toda la memoria RAM, basándose en el hecho que el tamano de RAM en los ordenadores de uso domestico, particularmente en las GPU's se encuentra por encima de los disponibles en dispositivos ASIC
- *Funcionamiento optimo en GPU's* Se busca que el minado mediante GPU's sea lo mas facil posible, dado que optimizar este minado en CPU's
- *Fácilmente verificable* Cualquier participante debe ser capaz de verificar en un tiempo razonable una nuevo bloque.
- *Favorecer a los clientes que almacenan la totalidad de la cadena* Aquellos que dedican espacio de almacenamiento propio para guardar la cadena de bloques tendran tiempos de acceso menores a los diferentes bloques y por tanto cierta ventaja.

3.9. Bifurcaciones

Cuando se envían por la red varias versiones correctas del siguiente bloque los nodos pueden recibirlas en diferente orden. Por protocolo incluirán la primera que han recibido, así que puede ocurrir que la cadena de bloques deje de ser única. Para resolver este problema cada nodo almacena las otras versiones del último bloque que ha recibido, aunque solo considere como buena una de ellas. Con el cálculo de un nuevo bloque alguna versión se hará mas extensa que las otras (tendrá un bloque nuevo) y en este caso todos los nodos que estuvieran en una rama distinta de la cadena cambiarán a la versión más larga.

Hay que señalar que el término bifurcación (*fork*) es usado también en la literatura sobre el blockchain para designar el proceso en el que se produce una modificación del protocolo acordada (o no) por los participantes. Cuando esto ocurre quienes siguen trabajando con una versión anterior dejan de reconocer los mensajes que se producen la nueva. Los que participan en este cambio pueden o bien dejar de reconocer también al protocolo antiguo y se dice entonces que estamos ante una bifurcación dura (*hard fork*) o seguir aceptando estos mensajes y se dice entonces que se ha producido una bifurcación suave (*soft fork*). Con esto se busca corregir errores en el código, prevenir ataques a la cadena de bloque o simplemente introducir mejoras. La resistencia a ASIC vista en 3.8 en caso de no existir originalmente en el protocolo, puede ser introducida posteriormente por esta vía.

3.10. Alternativas a la prueba de trabajo

El método de prueba de trabajo explicado en 3.5 al estar basado en la realización de un gran número de operaciones, ya sea en CPU, GPU o ASIC tiene ciertos inconvenientes. Por un lado, entidades con el poder suficiente para controlar un gran número de máquinas pueden intentar alterar el protocolo de consenso en la red. Como vimos esto se intenta solucionar, al menos parcialmente, al introducir medidas contra el minado usando hardware específico, permitiendo de esta forma que los usuarios domésticos participen en igualdad de condiciones. Pero nada impide que ciertas entidades utilicen gran numero de CPU's o GPU's para intentar hacer modificaciones en la cadena de bloques. En general la filosofía del Blockchain supone que alguien que tiene interés en participar en el protocolo y lo demuestra invirtiendo su tiempo de procesamiento estará también interesado en que el consenso se mantenga y por tanto no validará operaciones falsas. Igualmente, se pueden utilizar las bifurcaciones explicadas en 3.9 para modificar el protocolo si es necesario contrarrestar algún tipo de hardware específico diseñado para hacer más efectivo el proceso de minado.

Otra de las consecuencias negativas del protocolo de prueba de trabajo es de índole ecológica. El coste computacional de una red grande, como la de las criptomonedas Bitcoin y Ethereum por ejemplo, se refleja en un gasto energético extraordinario. Hay que tener en cuenta que en el proceso de validar un conjunto de operaciones solo se tiene en cuenta el resultado del primero que resuelve el puzzle, y por tanto el trabajo de todos los que han estado trabajando en ese mismo bloque ha sido en vano. Se estima que la red del Bitcoin a finales de 2017 ya gastaba más energía eléctrica que 159 países [3]. Teniendo en cuenta que las principales fuentes energéticas continúan siendo los combustibles fósiles es comprensible la preocupación que esto despierta.

A la vista de lo anterior se han propuesto varias alternativas entre las que se destacan las siguientes:

- *Prueba de participación (Proof of Stake)*: Para validar las transacciones se hace necesario disponer de cierta cantidad del recurso en que se basa la red (generalmente criptomonedas). En base a la cantidad de este recurso que posea, cada participante tiene asignado un peso que representa el valor de sus decisiones (a mayor cantidad de

recursos mayor poder de decisión). Para proponer y elegir el siguiente bloque de la cadena se tienen en cuenta los votos y propuestas de cada participante de acuerdo a su peso. La filosofía detrás de esta idea es que quienes posean un mayor número del recurso de la red estarán más comprometidos con ella y por tanto serán los mejores garantes del consenso. Este método se puede combinar con la prueba de trabajo para de esta forma reducir la complejidad del puzzle que hay que resolver, o implementarlo de forma independiente. En ambos casos representa un notable ahorro energético respecto a la prueba de trabajo.

Por otro lado, se tiene el problema de cómo iniciar la cadena cuando ninguno de los participantes posee recurso alguno. Debido a esto, se suele optar por un enfoque mixto entre ambos métodos. Ejemplos del uso de este método están en versiones más recientes de Ethereum y en la criptomoneda Peercoin [12]

- *Proof of burn:* En este caso para obtener el derecho de validar transacciones (crear bloques) hay que destruir cierto recurso como prueba de nuestro compromiso con la red. Generalmente el recurso que se destruye es otra criptomoneda y esto se consigue enviándola a determinada dirección desde donde es irrecuperable. Este método tiene el problema de requerir de la existencia de al menos una moneda con ciertas propiedades similares a la nuestra (si es fácil crearla por ejemplo no tiene valor práctico destruirla). Aunque este mecanismo no requiera de un gran consumo energético la moneda alternativa que estamos destruyendo puede que esté basada en la prueba de trabajo, así que estaríamos en una situación similar a la explicada al inicio de este sección. Por todo esto la aplicación de este método aunque interesante desde el punto de vista económico, resulta contraproducente respecto al problema del gasto energético.

  La criptomoneda Slimcoin es la primera implementación de este algoritmo de consenso [14]

3.11. Privacidad y seguridad

Al inicio de este capítulo se explicaron los objetivos del Bitcoin, sin embargo, no se hizo referencia a la privacidad. En los sistemas bancarios tradicionales se mantiene cierto nivel de confidencialidad, y por tanto la información sobre el balance y las operaciones de determinada cuenta no se comparten libremente. En un sistema basado en el blockchain surge el problema de como garantizar la privacidad en un entorno público donde las transacciones por definición deben ser vistas por todos los participantes. Esto se resuelve manteniendo en secreto la identidad de los poseedores de las claves públicas. Además si se usa una clave pública distinta en cada transacción incluso aunque se llegue a comprometer la privacidad de alguna de nuestras claves se siguen manteniendo ocultas las otras y en consecuencia es imposible que alguien llegue a conocer todas las operaciones que hayamos realizado y nuestro balance total.

En este sentido los sistemas monetarios basados en blockchain guardan cierta semejanza con el dinero físico y se hace necesario disponer de algún sitio seguro donde guardar el con-

junto de claves privadas que hayamos usado ya sea para recibir transferencias o recompensas por haber participado en el algoritmo de consenso. Estos son los llamados monederos (wallets), que existen tanto en forma de software como de hardware específico. Estas cuestiones sobre privacidad aunque en las criptomonedas se consideran necesarias no son imprescindibles y en otras aplicaciones del blockchain se pueden obviar.

Otra cuestión de gran relevancia es como de segura es una red basada en blockchain. Por un lado, hay que distinguir aquellos ataques basados en intentar encontrar la clave (o claves) privada de un usuario y por otro los ataques al protocolo de consenso de la red. En el primer caso, la seguridad de la red depende en gran medida de la implementación del algoritmo visto en 2. Sobre el problema del logaritmo discreto para curvas elípticas (en el que se basa el ECDSA) no se conoce ningún algoritmo que lo resuelva en tiempo subexponencial [27]. Por tanto, se puede considerar que encontrar la clave privada a partir de la pública es infactible siempre que se haya implementado correctamente el algoritmo de cifrado. Evidentemente es igual de importante almacenar las claves privadas en un sitio seguro. Los ataques al protocolo de consenso se basan en intentar que sean incluidos en la cadena bloques con información falsa, en el caso de las criptomonedas esto es el problema del doble gasto mencionado al inicio de este capítulo. Por ser este un problema de gran interés dentro de la programación distribuida y que va más allá de la idea de blockchain se tratará en el siguiente capítulo con más detalle.

4. El problema del consenso

Uno de los problemas fundamentales dentro de la computación distribuida es el de mantener o alcanzar alguna clase de acuerdo entre los miembros de determinada red. Una de las definiciones posibles de un sistema distribuido nos dice que son colecciones de ordenadores que ante sus usuarios funcionan como una única máquina [34], así, el hecho de disponer de algún mecanismo de consenso es fundamental para que el sistema se comporte de forma adecuada. Algo más cerca de la idea de consenso que nos interesa de cara al blockchain están los acuerdos que se alcanzan en una base de datos entre todos los procesos antes de hacer un *commit*, donde deben decidir si abortar o no la transacción con la certeza de que la acción tomada será la misma para todos.

En un sistema distribuido genérico suponiendo que todos los miembros actúan siempre de acuerdo a ciertas reglas comunes (no hay comportamientos maliciosos ni fallos), las máquinas de los participantes no se desconectan de la red de forma imprevista y que los canales de comunicación son estables, es decir, todo mensaje llega a su destinatario de forma íntegra y a lo sumo en tiempo t, el consenso se puede alcanzar siempre de forma sencilla con algún protocolo como el commit de 2 fases [24]. En la práctica no se suelen cumplir alguna o varias de estas condiciones, así que el interés está en crear algoritmos robustos que puedan funcionar bajo múltiples condiciones adversas.

El protocolo blockchain descrito en 3 es de hecho un algoritmo de consenso. Queda por ver las limitaciones de esta clase de consenso y como se podría aplicar a un contexto más general.

4.1. El problema de los generales bizantinos

El problema de los generales bizantinos [26] plantea un experimento mental en el que un grupo de divisiones de un ejército (del Imperio de Bizancio) se encuentran asediando una ciudad. Los generales al frente de cada división deben acordar, en una versión simplificada del problema, si atacan la ciudad o se retiran. Las líneas de comunicación no son seguras, así que algunos mensajes pueden no llegar a su destino. Igualmente, entre los generales existen traidores que pueden no tomar la mejor decisión o comunicar una acción y realizar otra diferente. El problema es por tanto alcanzar un acuerdo entre los generales leales sobre que decisión tomar. Para garantizar el éxito la decisión tomada debe ser la misma para todos ellos.

Este es claramente un problema de consenso como los definidos al inicio de este capítulo. En general, para alcanzar un acuerdo se necesitan al menos $3m + 1$ generales en total si tenemos m generales traidores [26], así por ejemplo con 3 generales si uno de ellos es traidor no existiría solución.

El algoritmo *Paxos*, creado por Leslie Lamport resuelve el problema anterior [25]. El algoritmo *Raft* se considera una simplificación de *Paxos* usando métodos e ideas similares [30]. Tanto *Paxos* como *Raft* para alcanzar el consenso requieren de la elección de un líder.

4.2. Solución desde el Blockchain

El problema anterior puede plantearse desde la perspectiva del protocolo blockchain. Los nodos o participantes de la red serán los generales y el acuerdo que buscan estará reflejado en la cadena de bloques. Algunos nodos toman una decisión (atacar o retirarse) y se la comunican a los otros participantes. Cada nodo *trabaja* en el primer mensaje que reciba y con el que esté de acuerdo: si por ejemplo considera que lo mejor es retirarse *trabajará* con el primer *retirarse* que reciba. Aquí trabajar equivale a buscar un nonce del bloque que contiene la palabra *retirarse* o *atacar*, junto con información adicional como un índice y un timestamp por ejemplo, de longitud fijada por el protocolo. Cuando un nodo resuelve este problema comunica a los demás su solución y todos los que estén de acuerdo con esta solución la incorporan a la cadena y comienzan a trabajar en el siguiente bloque que tenga como predecesor esta respuesta. Pasado cierto tiempo una de las dos cadenas (la de *atacar* o la de *retirarse*) habrá alcanzado cierta longitud fijada en el protocolo y por tanto se puede considerar que son mayoría quienes apoyan realizar la acción definida en esa cadena.

Puede darse el caso que la decisión a tomar no esté clara y por tanto los generales leales se encontrarán divididos en dos grupos de tamaño similar. En ese caso la decisión dependerá de los mensajes de los traidores, o incluso puede existir un empate en la longitud de ambas cadenas de bloques. En ese caso, como se plantea en [26] las posibles acciones se pueden considerar como igual de buenas así que el empate en la cadena de bloques se puede romper con algún mecanismo fijado en el protocolo, por ejemplo tomando la decisión dada por la cadena cuyo bloque inicial se haya creado antes (suponiendo que se incluye un timestamp)

Así, hemos conseguido una posible solución para el problema de los generales bizantinos donde no se requiere la elección de un líder. Más aún, el posible problema de suplantación de identidad (de un general traidor intentando modificar el mensaje de un general leal) se resuelve mediante el uso del algoritmo ECDSA como se dijo en 3.11.

4.3. Limitaciones del consenso alcanzado mediante el Blockchain

Como se vió en 3.5 dada una función hash y cierta cadena de caracteres el algoritmo de prueba de trabajo está basado en encontrar cierto nonce que unido a la cadena de entrada y al aplicarle la función hash devuelva un resultado con determinadas propiedades fijadas en

el protocolo. Esto es una medida del esfuerzo computacional puesto en resolver ese problema y no es nada democrático, pues un único participante con una CPU avanzada podría igualar o superar al esfuerzo de varios nodos independientes. La idea detrás de usar este método en el blockchain, como se mencionó en 3 es suponer que el *valor intrínseco* de la red en su conjunto está relacionado directamente con el esfuerzo puesto en construir la cadena de bloques. Así, un ataque a una red, como la del Bitcoin por ejemplo, con un gran número de participantes sería muy difícil pues la potencia de cálculo combinada de sus nodos superaría a la de casi cualquier atacante. Y es justamente esta inversión en tiempo de CPU expresada en la cadena de bloques la que mide la importancia de la red.

Todo lo anterior supone que el bien en el que se basa la implementación del blockchain o el objetivo que persigue carece de valor por sí mismo. En el caso de las criptomonedas esto es cierto en general, pero otras situaciones, como la que plantea el problema de los generales bizantinos, puede no cumplirse. En estos casos, dado que puede ser interesante para alguna entidad interferir en el consenso desde un principio, la solución pasa por recurrir a un blockchain de consorcio o privado. Los participantes en este tipo de redes solo tienen en cuenta los bloques generados por ciertos nodos que definidos en una lista (consorcio) o llegan incluso a ignorar los mensajes que no provengan de esos nodos previamente (privado). El carácter público

El consenso que se alcanza en el el blockchain es probabilístico: con cada bloque que se añade aumenta la probabilidad de que los nodos lleguen a un acuerdo [23]. Como para los algoritmos *Paxos* y *Raft* hay que poner ciertas exigencias sobre el número de nodos maliciosos respecto al total de participantes para que sea probable llegar al consenso. Para el Bitcoin se suele aceptar que siempre que un atacante no posea más del 50 % del poder cálculo de la red será prácticamente imposible que pueda afectar al consenso. Aunque se llegue a un acuerdo sí puede ocurrir que un atacante consiga bloquear transacciones legítimas y alterar negativamente el funcionamiento de la red. Por ello en las diferentes implementaciones de criptomonedas se suelen adoptar medidas de seguridad adicionales en este sentido.

5. Implementación del protocolo blockchain

Siguiendo las ideas planteadas en el capítulo 3 se ha realizado una implementación del protocolo Blockchain, donde se han puesto en práctica los conceptos teóricos que se han tratado en este trabajo.

Esta implementación, realizada en el lenguaje de programación Python funciona como un gestor de documentos donde se aprovecha el carácter inmutable de la estructura de cadena de bloques. Los participantes enviarán información en formato de cadena de caracteres, que bien puedan ser artículos, trabajos o cualquier otra clase de texto que desean almacenar. Esta información va debidamente firmada usando cierta clave privada, incluyéndose también la clave pública que permite verificar la firma. Utilizando el algoritmo de prueba de trabajo visto en 3.5 los nodos transformarán esta información en bloques de forma que se pueda tener un registro que sirva para probar que un texto fue verificado y existía desde un momento de tiempo concreto.

Este código puede ser utilizado en un entorno real haciendo ciertas modificaciones. Por ejemplo, la estructura enlazada de la cadena de bloques puede ser utilizada a manera de índices consecutivos en tablas SQL (*structured query language*) de forma que cada nuevo dato que se añada (*inserts*) deba seguir la lógica de la base de datos.

La implementación se encuentra disponible en el repositorio de Github https://github.com/lglaria/TFG

5.1. Librerías utilizadas

En la realización de este código se han utilizado las siguientes librerías de Python:

- *multiprocessing*[7] Este paquete permite generar procesos que ofrecen concurrencia tanto local (en la misma máquina aprovechando los múltiples hilos (*threads*) de los procesadores actuales), así como concurrencia remota, en caso de máquinas conectadas a través de una red. También nos permite disponer de una cola FIFO (*first in first out*) para que los diferentes procesos puedan compartir información, sin necesidad de hacer uso de ninguna primitiva de concurrencia. Este paquete funciona tanto en sistemas Unix como Windows y está disponible tanto para Python 2 como para Python 3.

Generación de nuevos procesos

```
from multiprocessing import Process

def func(argument):
    print('hello', argument)

if __name__ == '__main__':
    p = Process(target=func, args=('world',))
    p.start()
    p.join()
```

- *hashlib* [8] Esta librería implementa diversas funciones hash, entre las que se encuentran: SHA1, SHA224, SHA256, SHA384, y SHA512 entre otras. Para cada función hash invocada se crea un objeto con el mismo tipo de interfaz, por ejemplo si se invoca a *sha256()* se creará un objeto de la función hash SHA256 para el que podrán por ejemplo calculares los valores hash que devuelve al aplicarle esta función a cierta cadena de caracteres.

Hashlib

```
import hashlib
"""
Hex output of sha-224 for message m
"""
m = "Message"
hashlib.sha224(m).hexdigest()
```

- *fastecdsa* [6] Este paquete implementa protocolos criptográficos basados en curvas elípticas tal y como se vieron en el Capítulo 2

  Para ser utilizada esta librería debe ser instalada previamente, pues no viene por defecto en las diferentes distribuciones de Python. Puede instalarse por ejemplo mediante el comando textitpip install fastecdsa

Generación de claves

```
from fastecdsa import keys, curve

"""The reason there are two ways to generate a
keypair is that generating the public key requires
a point multiplication, which can be expensive.
That means sometimes you may want to delay
generating the public key until it is actually needed."""

# generate a keypair (both keys) for curve P256
priv_key, pub_key = keys.gen_keypair(curve.P256)
```

```
# generate a private key for curve P256
priv_key = keys.gen_private_key(curve.P256)

# get the public key corresponding to the private key we just generated
pub_key = keys.get_public_key(priv_key, curve.P256)
```

Firma y verificación

```
from fastecdsa import curve, ecdsa, keys
from hashlib import sha384

m = "a_message_to_sign"  # some message

''' use default curve and hash function (P256 and SHA2) '''
private_key = keys.gen_private_key(curve.P256)
public_key = keys.get_public_key(private_key, curve.P256)
# standard signature, returns two integers
r, s = ecdsa.sign(m, private_key)
# should return True as the signature we just generated is valid.
valid = ecdsa.verify((r, s), m, public_key)

''' specify a different hash function to use with ECDSA '''
r, s = ecdsa.sign(m, private_key, hashfunc=sha384)
valid = ecdsa.verify((r, s), m, public_key, hashfunc=sha384)

''' specify a different curve to use with ECDSA '''
private_key = keys.gen_private_key(curve.P224)
public_key = keys.get_public_key(private_key, curve.P224)
r, s = ecdsa.sign(m, private_key, curve=curve.P224)
valid = ecdsa.verify((r, s), m, public_key, curve=curve.P224)
```

5.2. Características de la implementación

El algoritmo de consenso elegido ha sido el de prueba de trabajo, combinado con la elección de la cadena de bloques más larga en caso de tener varias alternativas disponibles.

Los bloques de la cadena son arrays de Python y almacenan (en ese orden):

- El valor hash del bloque anterior en formato hexadecimal. Excepto para el bloque inicial al que se le asigna por defecto el valor cero en este campo.
- La información que se desea guardar así como la firma y la clave pública de quien ha enviado (o validado) este texto. En el bloque inicial se pone el valor cero.
- Una marca temporal, se ha usado el sistema UNIX timestamp
- Un índice incremental. Empieza en cero en el bloque inicial.

- El nonce del bloque. Cadena de 32 bits a partir del algoritmo de prueba de trabajo. Este algoritmo ha sido implementado de forma similar a lo visto en 3.5

5.3. Aspectos técnicos.

En esta implementación se ha utilizado la versión 3 del lenguaje de programación Python. Para el proceso de firma basado en el algoritmo ECDSA se ha usado la curva elíptica P-256 del Instituto Nacional de Estándares y Tecnología (NIST) dependiente del Departamento de Comercio de los Estados Unidos. El algoritmo de firma basado en la curva anterior, y obtenido a partir de la librería *fastecdsa* cumple con los requisitos de seguridad mencionados en el capítulo 2.

En la implementación del algoritmo de prueba de trabajo se ha utilizado la función hash SHA-256 que ya fue mencionada en la sección 3.5. Esta función, ampliamente utilizada en diversas implementaciones del Blockchain se ha obtenido a través de la librería *hashlib* [8]. Los nodos calculan el nonce generando valores aleatorios de 32 bits. Por defecto se ha puesto una dificultad fija, pero este valor puede ser modificado y calculado a partir de una función tal y como ocurre en el Bitcoin. En el cálculo del nonce no se modifica el timestamp.

Algoritmo de prueba de trabajo

```
def pow(self,bloque): #algoritmo de prueba de trabajo
  bloque_str = ''.join(str(e) for e in bloque)
  while True:
    var = format(random.getrandbits(32), '08b') #buscamos nonces de 32 bits
    hash_value = sha256((bloque_str+str(var)).encode('utf-8')).hexdigest()
    if hash_value[:self.longitud_nonce] == '0'*self.longitud_nonce:
      break
  bloque.append(var)
  return bloque
```

En esta implementación cada nodo almacena el conjunto de las operaciones realizadas y no la raíz Merkle, pero este punto puede ser modificado en aras de la compresión de la información.

Como ya se mencionó la red p2p sobre la que funciona el protocolo se ha construido usando la librería *multiprocessing* [7]. Quedando así solucionados los problemas de concurrencia que podrían aparecer al mantener activos al mismo tiempo dos procesos que trabajan sobre la misma estructura de datos: uno para recibir los mensajes que le llegan a un usuario (nodo) y otro para ejecutar el algoritmo de consenso, actualizar la cadena y transmitir información.

Cada nodo es capaz de calcular un nuevo bloque, validar un bloque recibido, validar una cadena completa, incluir nuevos bloques previamente verificados en la cadena existente y transmitir la cadena a otros nodos. Igualmente existe un mecanismo para generar el bloque inicial, en caso que se quiera construir la estructura desde cero. Todas estas funcionalidades se encuentran implementadas como métodos de la clase Blockchain que es invocada por cada nodo.

Clase Blockchain

```
class Blockchain(object):
        """
        formato bloques [hash_bloque_anterior, (data,data), timestamp, indice, nonce]
        """
        def __init__(self):
                self.cadena = []
                self.longitud_nonce = 4 #dificultad algoritmo pow (modificable)

        def primer_bloque(self):
        """
        Generacion primer bloque
        """
                primer_bloque = [0,0,int(round(time.time())),0]
                self.cadena.append(self.pow(primer_bloque))
```

5.4. Seguridad de la implementación y posibles mejoras.

Queda por hacer un análisis similar al de la sección 3.11 para esta implementación en concreto, ignorando los supuestos de privacidad que carecen de sentido en este caso. La fortaleza del algoritmo de cifrado y de la función hash dependen de las librerías elegidas para usar estos métodos y de acuerdo con la documentación no tienen problemas de seguridad conocidos. Hay que ver por tanto si el consenso que busca alcanzar este implementación es vulnerable a algún tipo de ataque.

Un posible agresor podría intentar realizar dos tipos de acciones que alteren el consenso:

1. Incluir en la cadena información incorrecta.

2. Modificar la cadena para incluir nuevos bloques en determinada posición (distinta de la última) o modificar bloques existentes.

Dado que este protocolo no dispone de un mecanismo para distinguir cuando un texto es correcto o no, y como en general tales mecanismos automáticos resultan difíciles de modelar una posible solución es hacer que esta implementación funcione como blockchain de consorcio o privado. Esto tiene sentido pues en una situación real quienes se encargan de verificar que un documento o trabajo cumple con ciertos criterios son exclusivamente las personas autorizadas a ello. De esta forma, aunque se pueda incluir información incorrecta al final de la cadena esto será responsabilidad de quien la haya firmado. La lista de claves públicas autorizadas a realizar validaciones se podría añadir como un campo más en la cadena de bloque o mantener una estructura de datos alternativa donde se almacene esta lista. Además de verificar que la firma se corresponde con la clave pública habría también que comprobar también que esta clave está autorizada a verificar documentos.

El segundo tipo de ataque como se vio en el capítulo 3 depende de la dificultad de la función hash y de la longitud de la cadena. Un atacante que quiera modificar algún bloque sabemos debería rehacer toda la cadena, pues los valores hash que señalan al bloque anterior y por tanto el nonce del algoritmo de prueba de trabajo habría que volverlos a calcular. Pero ahora no dependemos solamente de la dificultad de esta operación (que por sí sola garantizaría la seguridad) sino que se puede aprovechar el hecho de disponer de un conjunto definido de claves públicas autorizadas a validar. Si además de firmar el texto hacemos que haya que firmar el valor hash del bloque anterior, un posible atacante debería disponer de alguna clave privada y además de eso la cadena falsa que genere se caracterizará por incluir a partir de la posición donde ha realizado la modificación una única firma, lo que hace muy fácil identificar esta clase de ataques.

5.5. Cálculos sobre el algoritmo de prueba de trabajo

Cuando se explicó el funcionamiento del algoritmo de prueba de trabajo se hizo referencia a la relación entre la longitud del puzzle (la condición que se le impone al valor hash que se considere válido) y al tiempo de procesamiento necesario para obtener este valor. Dado que ahora se dispone de una implementación de este algoritmo es posible comprobar este hecho en la práctica.

Se incluye una tabla con los tiempos medios que se tardado en obtener el nonce según la dificultad para ciertas cadenas generadas de forma aleatoria.

Dificultad (ceros del hash)	Tiempo medio (segundos)	Número de iteraciones
1	9.87052917e-05	100
2	1.49291515e-03	100
3	3.24990416e-02	100
4	4.12933254e-01	100
5	6.35731197e+00	100
6	9.97811596e+01	10

Cuadro 5.1: Resultados algoritmo prueba de trabajo.

Aunque estos valores dependen de la máquina en la que se efectúen los cálculos, y en este caso por limitaciones técnicas no ha sido posible realizar más comprobaciones, puede apreciarse que hay relación directa entre la longitud de la cadena pedida como solución del problema y el tiempo necesario para encontrarla.

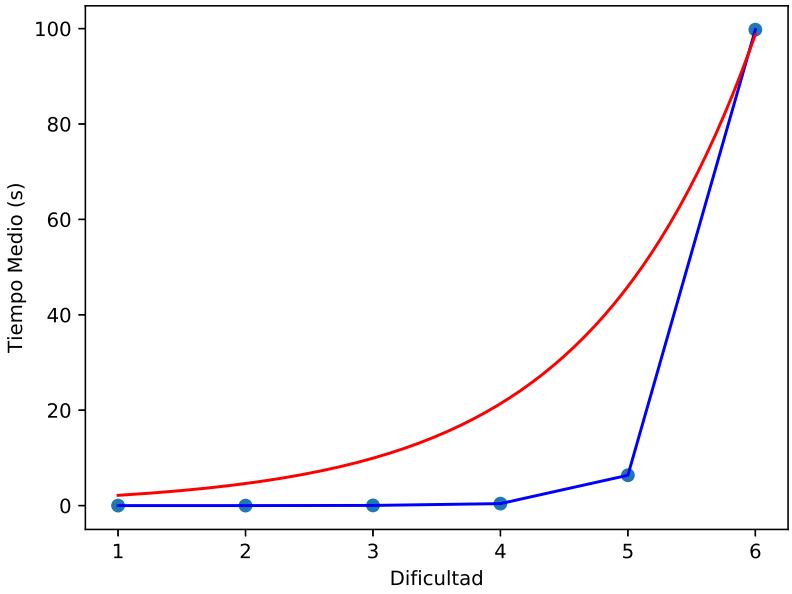


Figura 5.1: Comparación entre los resultados y la función $y = 2^x$

En la figura 5.1 se aprecia que los resultados obtenidos tienen un comportamiento similar a la función $y(x) = 2^x$. Aunque esto no puede considerarse como una aproximación buena del algoritmo de prueba de trabajo con la función hash SHA-256, pues ni el tamaño de la muestra es lo suficientemente grande, ni se han hecho comprobaciones para dificultades superiores a 6, sí es un indicador de que este comportamiento es cuasi-exponencial.

6. Conclusiones

6.1. Discusión plan de estudios

En la realización de este trabajo han sido importantes los conocimientos adquiridos en las asignaturas de Álgebra Básica y Ecuaciones Algebraicas del plan de estudios de Matemáticas. El contenido de estas materias resultó esencial para entender las diversas cuestiones referentes a la criptografía tratadas en el capítulo 2. Sin embargo, todo lo relativo a curvas elípticas se estudia en la asignatura de Curvas Algebraicas, optativa del último curso del itinerario de Matemática Pura, que por tanto no es común a todos los estudiantes. Igualmente, no existe en la carrera ninguna asignatura donde se traten en profundidad la criptografía, que tal y como se ha visto no solo es un área de conocimiento fundamental en el desarrollo del Blockchain, sino que resulta fundamental en multitud de áreas de la economía, la ciencia o la industria y a su vez está conectada directamente con el álgebra.

Al problema del consenso tratado en 4 se hace mención en la asignatura de Programación Paralela, optativa del itinerario de Ciencias de la Computación, pero no es este tema uno de los contenidos centrales de esta materia.

En lo referente a la implementación del protocolo blockchain han sido de gran importancia los conocimientos adquiridos en la asignatura de Informática de primer curso, sobre el lenguaje de programación Python. Han sido también de gran importancia la optativa de cuarto curso Programación Paralela, mencionada anteriormente, donde se estudian casos prácticos de sistemas distribuidos. Esto ha hecho posible construir la red p2p sobre la que se ha establecido la implementación del protocolo Blockchain realizada en este trabajo. Al igual que no existe una asignatura donde se traten cuestiones teóricas referentes a la criptografía, tampoco existe una donde estas cuestiones se analicen desde un punto de vista más práctico, algo que también se ha echado en falta en el proceso de implementación, por lo que se ha tenido que recurrir a diferentes manuales y documentos mencionados en la bibliografía.

6.2. Ideas finales y posible trabajo futuro

Este trabajo ha permitido desarrollar los conceptos fundamentales en los que se sustenta el protocolo Blockchain. Ha sido posible justificar su funcionamiento y corrección a partir de la teoría de curvas algebraicas, en particular del algoritmo ECDSA para la firma y verificación de mensajes, así como de las funciones hash. El interés despertado por esta

tecnología en el último lustro se ha debido fundamentalmente a las oportunidades de negocio que han generado las criptomonedas. Por tanto una parte importante de los trabajos y documentos sobre el blockchain se han centrado en como generar valor a partir de este protocolo pasando por alto los aspectos teóricos que sustentan su funcionamiento. También suele ser común tratar los términos Blockchain y Bitcoin como sinónimos, esto además de no ser exacto, pues como hemos visto el Blockchain trasciende a esta criptomoneda, enturbia a este protocolo al vincularlo con ciertas actividades ilegales (o al menos alegales) que se han aprovechado del anonimato y seguridad que ofrece el Bitcoin.

Por otra parte se han conseguido explicar cuestiones generales sobre la especificación del Blockchain que resultan de utilidad en el estudio de aplicaciones de esta tecnología. Entender este protocolo como un mecanismo de consenso permite utilizarlo en la resolución de diversos problemas de la programación distribuida. Por último, la implementación del protocolo además de mostrar uno de los posibles usos que se le puede dar a esta tecnología prueba que llevar a la práctica las ideas básicas del Blockchainn es una tarea directa y sencilla.

Las reservas o temores de diversos organismos entre los que se encuentran los estados respecto a las criptomonedas están bien fundadas en la medida en que la estructura descentralizada de cadena de bloques anula el control de las entidades centrales sobre el sistema monetario. Pero más allá de las criptomonedas el Blockchain tiene el potencial para introducir transformaciones en diversos aspectos de la sociedad. Esta tecnología cambiará la forma de entender los contratos, las transacciones y en general todas las relaciones que requieren de cierta confianza entre las partes y que tradicionalmente han necesitado de la intervención de intermediarios que actúen como garantes del buen funcionamiento de tales actividades. En cualquier caso, hablar del Blockchain como una (o la) tecnología del futuro tampoco es exacto, porque ya es una tecnología del presente.

Bibliografía

[1] Application-specific integrated circuit (asic). https://www.techopedia.com/definition/2357/application-specific-integrated-circuit-asic. Accessed: 2019-08-20.

[2] Asic, gpu and cpu mining. https://coincentral.com/asic-gpu-cpu-mining/. Accessed: 2019-08-20.

[3] Bitcoin mining now consuming more electricity than 159 countries. https://powercompare.co.uk/bitcoin/. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uNlJOrn).

[4] A brief history of p2p content distribution, in 10 major steps. https://medium.com/paratii/a-brief-history-of-p2p-content-distribution-in-10-major-steps-6d6733d25122. Accessed: 2019-08-20.

[5] Descriptions of sha-256, sha-384, and sha-512. http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf. Accessed: 2019-08-21.).

[6] Documentación librería fastecdsa. https://pypi.org/project/fastecdsa/. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uOA1nmI).

[7] Documentación librería hashlib de python. https://docs.python.org/2/library/multiprocessing.html. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uNs8AP7).

[8] Documentación librería hashlib de python. https://docs.python.org/2/library/hashlib.html. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uO4Hti7).

[9] Ethash design rationale. https://github.com/ethereum/wiki/wiki/Ethash-Design-Rationale. Accessed: 2019-08-20.

[10] The gpu industry is booming thanks to blockchain. https://www.investopedia.com/tech/gpu-industry-booming-thanks-blockchain/l. Accessed: 2019-08-20.

[11] Private, public, and consortium blockchains. https://errna.com/private-public-blockchain.htm. Accessed: 2019-08-20.

[12] Proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ#what-is-proof-of-stake. Accessed: 2019-02-19. (Archived by WebCite at http://www.webcitation.org/76JNGcsPi).

[13] Scalability is blockchain's biggest problem but it can be resolved. https://cryptoslate.com/scalability-is-blockchains-biggest-problem-but-it-can-be-resolved/. Accessed: 2019-08-20.

[14] Slimcoin a peer-to-peer crypto-currency with proof-of-burn. http://www.doc.ic.ac.uk/~ids/realdotdot/crypto_papers_etc_worth_reading/proof_of_burn/slimcoin_whitepaper.pdf. Accessed: 2019-02-19. (Archived by WebCite at http://www.webcitation.org/76JMlsixP).

[15] Why a gpu mines faster than a cpu. https://en.bitcoin.it/wiki/Why_a_GPU_mines_faster_than_a_CPU. Accessed: 2019-08-20.

[16] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[17] John William Scott Cassels and John William Scott Cassels. *LMSST: 24 Lectures on Elliptic Curves*, volume 24. Cambridge University Press, 1991.

[18] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[19] S. Vanstone D. Hankerson, A. Menezes. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[20] Wei Dai. B-money. http://www.weidai.com/bmoney.txt. Accessed: 2019-08-20.

[21] Peter J. Denning. The science of computing: The arpanet after twenty years. *American Scientist*, 77(6):530–534, 1989.

[22] Thaddeus Dryja. Hashimoto: I/o bound proof of work. 2014. *URL {https://mirrorx.com/files/hashimoto. pdf}*.

[23] Vincent Gramoli. From blockchain consensus back to byzantine consensus. *Future Generation Computer Systems*, 2017.

[24] Jim Gray and Leslie Lamport. Consensus on transaction commit. *ACM Transactions on Database Systems (TODS)*, 31(1):133–160, 2006.

[25] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[26] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[27] Kristin E Lauter and Katherine E Stange. The elliptic curve discrete logarithm problem and equivalent hard problems for elliptic divisibility sequences. In *International Workshop on Selected Areas in Cryptography*, pages 309–327. Springer, 2008.

[28] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[29] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uMoT8YE).

[30] Diego Ongaro and John Ousterhout. Raft consensus algorithm. 2015.

[31] FIPS PUB. Secure hash standard (shs). *FIPS PUB*, 180(4), 2012.

[32] William Stallings. *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 2017.

[33] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[34] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[35] H Verrill. Group law for elliptic curves. http://www.math.ku.dk/~kiming/lecture_notes/2000-2001-elliptic_curves/grouplaw.pdf. Accessed: 2018-11-13. (Archived by WebCite at http://www.webcitation.org/73uN8g2Qm).