

# **Blockchain y su Aplicabilidad a una Industria bajo Regulación**

**Agustín Isasa Cuartero**

**Agosto 2017**



Esta obra está sujeta a una licencia de  
Reconocimiento-NoComercial-SinObraDerivada 3.0  
España de Creative Commons



Este trabajo está adaptado directamente del Trabajo Final de Máster de mismo título realizado por el autor para el Máster en Ingeniería Computacional y Matemática, área de Criptografía, de la Universitat Rovira i Virgili y la Universitat Oberta de Catalunya, durante el curso académico 2016-17. El trabajo, redactado entre diciembre 2016 y junio 2017, fue dirigido por el Dr. Oriol Farràs Ventura, y su contenido original es accesible en el repositorio O2 de la UOC en <http://hdl.handle.net/10609/64766>.



UNIVERSITAT ROVIRA I VIRGILI



Puede contactarse con el autor mediante:

[aisasa@uoc.edu](mailto:aisasa@uoc.edu)

[a.isasa.comp@gmail.com](mailto:a.isasa.comp@gmail.com)



## FICHA DEL TRABAJO

<b>Título del trabajo:</b>	<i>Blockchain y su Aplicabilidad a una industria bajo regulación</i>
<b>Nombre del autor:</b>	<i>Agustín Isasa Cuartero a.isasa@uoc.edu</i>
<b>Área del Trabajo:</b>	<i>Criptografía, Computación Distribuida</i>
<b>Idioma del trabajo:</b>	<i>Castellano</i>
<b>Palabras clave:</b>	<i>Blockchain, Bitcoin, Criptografía de Curva Elíptica, Regulación, Privacidad, Consenso, Autoridad Central, Autoridad Descentralizada</i>
<b>Keywords:</b>	<i>Blockchain, Bitcoin, Elliptic Curve Cryptography, Regulation, Privacy, Consensus, Central Authority, Decentralized Authority</i>
<b>Resumen del Trabajo:</b>	

Blockchain es la tecnología soporte de Bitcoin, *criptomoneda* virtual creada en 2008. Bitcoin es un ente inmaterial cuyo valor se establece por su aceptación como medio de pago y depósito de valor entre una multitud creciente de usuarios. Blockchain es novedoso en su tratamiento de la transmisión de valor: rápido, sin intermediarios, descentralizado, criptográficamente seguro, y cuyo estado final es compartido en una única cadena de bloques común, más barata que los actuales esquemas de conciliación, y que disfruta del *consenso* global que emerge de la verificación replicada de todo elemento transaccional del sistema y de la transición a los nuevos estados del mismo tras una demostración de trabajo computacional (*proof-of-work*) difícilmente asumible por un posible atacante.

Dada la desregulación e incumplimiento por Bitcoin de reglas básicas como *Know Your Customer* o privacidad, los operadores y reguladores de la industria financiera clásica conceden mayor relevancia a Blockchain.

Tras años de segura operatividad se plantean multitud de aplicaciones basadas en Blockchain en diferentes mercados, pero su implementación dependerá de la posible adaptación de esta tecnología a la extensa regulación a la que están sometidos. La introducción de regulación en este sistema supone, entre otras intervenciones, (i) aportar un esquema seguro de identidades, (ii) ocultar los elementos de negocio de las transacciones a quienes no conciernen, y (iii) habilitar la cadena de bloques a una posible intervención por la autoridad.

Veremos que las dos primeras son complejas, aunque asumibles con herramientas criptográficas usuales, pero la intervención en Blockchain elimina sus más novedosos fundamentos al sustituir el consenso emergente por el *ordinario* centralizado en autoridad.

<b>Abstract:</b>	
------------------	--

Blockchain is the technological support for Bitcoin, a virtual *cryptocurrency* created in 2008. Bitcoin is an intangible entity whose value is established under its acceptance as a means of payment and as a deposit of value by a growing multitude of users. Blockchain is innovative due to its value transmission processing: fast, with no intermediaries, decentralized, cryptographically secure, and whose final state is shared into a single common block chain cheaper than the current ledger reconciliation schemes, and reaches a broad-based emergent consensus because of the replicated verification of all transactional elements of the system and the transition to the new system states after demonstration of a computational effort (*proof-of-work*) difficult to emulate by a possible attacker.

Given the deregulation and non-compliance by Bitcoin with basic rules such as *Know Your Customer* or privacy, operators and regulators of the classic financial industry confer more value to Blockchain.

After years of secure operation, there are many Blockchain based applications under design in different markets, but their implementation will depend on the possible adaptation of this technology to the extensive regulation to which they are subject. The introduction of an authority in this system means, among other interventions, (i) to provide a secure scheme of identities, (ii) to hide the business elements of the transactions to those who are not its participants, and (iii) to enable the block chain to a possible intervention by the authority.

We will see that the first two aspects are complex though acceptable with common cryptographic tools, but intervention in Blockchain eliminates its most novel and disruptive foundations by substituting the emergent consensus for the *ordinary* one based on a recognized authority.

# Índice

1. Introducción.....	1
1.1 Terminología.....	4
1.2 Objetivos del trabajo.....	6
1.3 De aquí en adelante.....	10
2. Bitcoin como origen de la tecnología Blockchain.....	11
2.1 <i>Bitcoin: A peer-to-Peer Electronic Cash System</i> .....	12
2.2 Transacciones y marcas de tiempo en Bitcoin.....	13
2.3 <i>Proof-of-Work</i> .....	15
2.4 Red e incentivos.....	17
2.5 Simplificando el almacenamiento y la verificación de pagos.....	18
2.6 Combinando y dividiendo valor.....	20
2.7 Privacidad.....	21
2.8 Algunos cálculos.....	22
2.9 Conclusiones.....	26
3. Primitivas criptográficas en Bitcoin/Blockchain.....	28
3.1 Funciones de <i>hash</i> y funciones <i>hash</i> criptográficas.....	28
3.2 Funciones <i>hash</i> y <i>proof-of-work</i> .....	33
3.3 Criptografía de clave pública.....	36
3.4 Criptografía de curva elíptica.....	38
3.5 El estándar <i>secp256K1</i> .....	44
3.6 Firma digital y <i>Elliptic Curve Digital Signature Algorithm</i> - ECDSA.....	45
3.7 Una cadena segura.....	49
4. Blockchain en Bitcoin: conceptos, estructura, funcionalidades.....	50
4.1 Las direcciones como titularidad de la propiedad del valor.....	51
4.2 Las transacciones como mecanismo de intercambio de valor.....	55
4.3 El núcleo de la transferencia de valor: inputs, outputs, scripts.....	59
4.4 La red en Bitcoin/Blockchain.....	64
4.5 Minería de bloques.....	66
4.6 La cadena de bloques.....	69
4.7 El consenso emergente de la red.....	71
5. Aplicabilidad de Blockchain en la industria: un marco de trabajo.....	72
5.1 Características funcionales y estructurales en Bitcoin/Blockchain.....	72
5.2 Un marco de trabajo para la adaptabilidad de Blockchain.....	77
5.3 Transformaciones a considerar en Blockchain.....	80
5.4 Proyectos y tendencias actuales.....	82
6. Un diseño conceptual para la aplicación de Blockchain en la industria.....	85
6.1 Un modelo formal de Bitcoin/Blockchain: actual y transformaciones.....	85
6.2 Encriptación e información pública en las transacciones.....	87
6.3 La participación de una autoridad central.....	91
6.4 La variabilidad en las transacciones cerradas.....	94
6.5 El consenso bajo una autoridad central.....	99
7. Conclusiones.....	101
7.1 ¿Es Blockchain bajo regulación una <i>verdadera</i> Blockchain?.....	101
7.2 Conclusiones: un resumen.....	103
7.3 Futuras líneas de estudio y trabajo.....	105
7.4 Sobre este trabajo.....	106
8. Referencias/Bibliografía.....	R1
Anexo A. Algunos formalismos.....	A1
A.1 Máquina de Turing.....	A1
A.2 Clases de complejidad P, NP y BPP.....	A2
A.3 Algoritmos probabilísticos.....	A3
A.4 Tiempo polinomial no uniforme.....	A3
A.5 Adversario.....	A3
A.6 Funciones unidireccionales y funciones unidireccionales no uniformes.....	A4
Anexo B. Validación de una transacción.....	A5
Anexo C. Valoración de la matriz de correspondencias entre Bitcoin/Blockchain y requisitos de la industria.....	A6

## Lista de figuras

2.1 Transacciones en Bitcoin	14
2.2 Concepto de marcas de tiempo	15
2.3 Estructura básica de los bloques de transacciones en Bitcoin	16
2.4 <i>Hash</i> de transacciones, árbol resultante de Merkle, y poda	19
2.5 La cadena de cabeceras de bloques y rama de transacción no gastada	20
2.6 Transacción construida con varios inputs y varios outputs	21
2.7 Modelo tradicional de privacidad	21
2.8 Modelo de privacidad de Bitcoin	21
2.9 Código en C y cálculo del número de bloques requeridos	26
3.1 Operaciones de una iteración en SHA-256	33
3.2 Estructura básica de los bloques de transacciones en Bitcoin	34
3.3 Parámetros del bloque Génesis de la blockchain de Bitcoin	35
3.4 Variación en los últimos meses de la tasa de <i>hash</i> , en terahashes/seg.	36
3.5 Representación gráfica de $y^2 = x^3 + 7$	39
3.6 Representación gráfica de $A + B$ en la curva $y^2 = x^3 + 7$	40
3.7 Representación gráfica de $y^2 = (x^3 + 7) \bmod 29$	41
4.1 Modelo de privacidad de Bitcoin	51
4.2 Obtención de la dirección Bitcoin desde su clave pública	53
4.3 Obtención de la dirección Bitcoin desde un script de disposición	54
4.4 Relación entre transacciones, inputs, outputs y scripts	60
4.5 Blockchain con cadena principal y cadenas alternativas	69
5.1 Distribución de la potencia de minado de la red Bitcoin	74
5.2 Crecimiento de la cadena de bloques desde 2009	75
6.1 Secuencias del proceso de construcción, validación y registro de una transacción con información cifrada	92



## Lista de tablas

4.1 Principales componentes de una transacción en la clase CTransaction	56
4.2 Elementos de un input de una transacción según la clase CTxIn	57
4.3 Elementos de un output de una transacción según la clase CTxOut	58
4.4 Principales componentes de la estructura completa de una transacción	59
4.5 Ejecución de los scripts en una transacción P2PKH	63
4.6 Principales componentes de un bloque según la clase CBlock	68
5.1 Matriz de transformación de las correspondencias entre características actuales de Blockchain y requisitos de la industria	79
6.1 Necesidades de encriptación de los atributos de una transacción	88
6.2 Incremento de tamaño de una transacción en caso de cifrado de sus elementos no públicos	94
C1 Matriz de correspondencias entre características Blockchain vs. requisitos de la industria	117

# Introducción

La *criptomoneda* Bitcoin nace entre 2008 y 2009 tras la publicación de un diseño descentralizado de *dinero electrónico* por parte de Satoshi Nakamoto [1], nombre que oculta desde entonces a la persona, o grupo de personas, a la/s que debe atribuirse la autoría del trabajo seminal sobre esta criptomoneda. Tras unos pocos años relegada a círculos vinculados a la criptografía, al software y conocimiento libres, y a posiciones alternativas con respecto a los mercados financieros tradicionales, a partir de 2011 se empieza a extender su uso de forma apreciable, de tal manera que a partir de 2013 su alcance y filosofía empiezan a llamar la atención del público, de los desarrolladores, de los operadores en los mercados financieros, e incluso de los reguladores públicos de estos últimos.

Sin intención de prefijar de partida posiciones maximalistas, empezaremos por tratar de convenir que un bitcoin no existe: por un lado, no posee entidad física; pero es que, además, su existencia virtual está soportada por un conjunto de actitudes personales, de creencias, de estados mentales en definitiva, que le acaban otorgando carta de naturaleza entre quienes conforman tal conjunto. Hay que decir, por otra parte, que la cardinalidad de este último es importante: son ya algunos millones los que en mayor o menor medida confían en él y sostienen que Bitcoin (con mayúscula para referirnos genéricamente a la criptomoneda o la tecnología; con minúscula para designar una unidad de cuenta) es un ente real, con funcionalidad propia y concreta, y que trasciende su representación informática (su única encarnación manifiesta) como cadenas de ceros y unos. Pero algo completamente equivalente podríamos decir, como nos recuerdan sus partidarios, sobre la inmensa mayoría del dinero en este planeta, que tras la desaparición del patrón oro en el pasado siglo está respaldado únicamente por la confianza en los respectivos estados u organismos supraestatales emisores.

Esta particular visión compartida del bitcoin tiene como propósito último la extensión de la convención de que existe un valor literal representado por el bitcoin: no es un simple *token* instrumental, no es una etiqueta descriptiva de la representación de algo situado en un nivel existencial físico, sino que es en sí mismo un activo real tal y como lo es, por continuar con la equiparación ya comentada, un billete de 50 euros, del que es obvio que el valor real del papel que lo soporta no tiene vinculación directa con su valor nominal, el cual viene respaldado por la autoridad emisora europea por la que los ciudadanos confiamos en que un billete de 50 euros valga... 50 euros.

Discutamos o no las bondades de la confianza sobre los estados o autoridades centrales en comparación a la confianza sobre la autoridad descentralizada del Bitcoin, o compartamos o no la opinión de los defensores y usuarios de bitcoin, lo cierto es que a la hora de escribir este trabajo hay una evidente y poco ficticia relación de valor entre un bitcoin y cualquier unidad de cualquiera de las monedas tradicionales: si ahora mismo, y tras un magnífico año y medio en la evolución del precio de esta criptomoneda, algún propietario quiere desprenderse de un bitcoin, podrá obtener un

contravalor de más de 2.000 dólares USA.<sup>1</sup> Y una cantidad parecida tendrá que desembolsar quien, a la inversa, desee adquirir una unidad de bitcoin. Esta posibilidad de intercambio está tan desarrollada que desde hace tiempo existen los mecanismos adecuados para fijar su tasa de cambio de acuerdo a la oferta y a la demanda, así como agencias de cambio que automatizan y operan en tiempo real sobre cualquier conversión entre bitcoin y monedas clásicas, aplicando los *spreads* correspondientes y generando negocios sobre opciones o futuros completamente equivalentes a los que los operadores financieros tradicionales vienen realizando históricamente sobre las curvas de los futuros comportamientos previstos en una considerable variedad de activos.

Los que no comparten ese estado mental, ese conjunto de creencias sobre lo que Bitcoin dicen que es los que sí las comparten, asisten un tanto desconcertados a esta asignación de valor *real* originada, quizá como experimento o como prueba de concepto de una nueva tecnología, hace muy pocos años. En poco más de un lustro su *masa monetaria* ha alcanzado un valor de capitalización, a las fechas de redacción de este trabajo (abril 2017), de unos 20 mil millones de dólares USA,<sup>2</sup> lo que para algo de existencia reciente y virtual supone un logro que merece tomarse en consideración.

No obstante lo anterior, hay un aspecto de la cuestión sobre el que todos, *creyentes* y *no creyentes*, estamos de acuerdo: con independencia del valor de Bitcoin y de su futuro devenir, la tecnología subyacente que lo soporta, la Blockchain o Cadena de Bloques (con mayúscula para referirnos a la tecnología genérica; con minúscula para hacerlo sobre una instancia concreta), ha demostrado una confiabilidad extrema en sus funcionalidades básicas, que son las de registrar todas y cada una de las transacciones o transferencias de valor que puedan realizarse con bitcoins; registrar todo el tracto de cada bitcoin generado y de sus posibles fracciones; ofrecer todo el estado de saldos (demostrar, en definitiva, el *valor*) de los propietarios individuales de cada unidad, o conjunto o fracción, de esta moneda en un determinado momento; y hacerlo todo ello en un marco estable y seguro donde no pueda tener lugar ni la falsificación de las transacciones en curso (problema de la prevención del *doblo gasto*) ni una vez registradas cuando ya han sido dadas por válidas de forma consensuada (problema *de los generales bizantinos*).

Para acometer con éxito estas funcionalidades que acabamos de describir, Blockchain utiliza dos herramientas criptográficas de forma intensiva. Por una parte, la firma digital (que permite identificar sin ambigüedades al emisor de un mensaje) se constituye en el ladrillo básico de la construcción de cualquier cadena de bloques puesto que Nakamoto, ya en los primeros compases de su documento inicial, define una moneda electrónica como *una cadena de firmas digitales*. En efecto, una blockchain es una cadena de bloques que contienen, cada uno de ellos, un conjunto de transacciones representativas de transmisiones de valor, y cada transmisión consiste en una firma digital sobre la transmisión anterior y la clave pública del nuevo propietario del valor al que se pretende transferir el derecho. Este último propietario, mediante su clave privada, podrá transferir de la misma manera este valor enlazando la transmisión que acredita su propiedad con la clave pública del nuevo beneficiario, que a partir de ese momento podrá movilizar su valor con su correspondiente clave privada. De esta manera, la posible y sencilla comprobación de la corrección de la cadena de firmas imposibilita en la práctica el

---

<sup>1</sup> <https://blockchain.info/charts/market-price>

<sup>2</sup> <https://blockchain.info/charts/market-cap>

problema del doble gasto, por lo que no cabe la posibilidad de que un propietario transfiera más de una vez su valor. La conformación de la cadena de transacciones o, de forma equivalente, de firmas digitales, es evidente, pero no debe confundirse con la cadena de bloques puesto que éstos constituyen contenedores de las transacciones realizadas en un periodo de tiempo determinado.

La segunda herramienta criptográfica fundamental son las *funciones hash criptográficas*. Estas funciones, en términos generales, mapean un conjunto de caracteres de entrada, de tamaño no definido, en otro conjunto de caracteres de salida de tamaño fijo, usualmente de menor longitud que el de entrada. Esta conversión, para que sea útil en la mayor parte de sus aplicaciones criptográficas, debe ser rápida, que evite en todo lo posible las colisiones (mismo resultado final para diferentes entradas), que ligeros cambios en la entrada resulten en muy importantes cambios en la salida, y que no sea reversible (que dada una salida *hash* resulte computacionalmente intratable obtener la entrada que lo origina). Aunque es habitual el uso de estas funciones en los procedimientos de firma digital, su utilización en Blockchain va más allá de esta mera función instrumental para constituirse en la solución del denominado *problema de los generales bizantinos* para este contexto: acordar entre diferentes partes con posiblemente intereses opuestos un consenso sobre la coherencia de la asignación individual de valor en un instante determinado. Esto se consigue mediante el mecanismo de *Proof-of-Work* (prueba de trabajo), o PoW, que implementa Blockchain, y según el cual cada bloque, y por tanto toda la cadena de bloques, incorpora tal cantidad de trabajo computacional que la falsificación de las transacciones que contienen no es posible a no ser que el conjunto de nodos deshonestos que quisieran hacerlo superara al de nodos honestos que trabajan por una correcta blockchain. Pues bien: este trabajo computacional, esta *proof-of-work*, se implementa como la obtención de manera aleatoria de un *hash* sobre el contenido del bloque en cuestión de tal manera que la representación numérica del resultado de la función *hash* sea menor que un determinado valor prefijado, lo que equivale, tal y como se describe en el trabajo de Nakamoto, a que comience por un determinado número de ceros. Esta obtención aleatoria se consigue añadiendo un campo adicional al contenido del bloque y aplicando la función *hash* a todo el contenido final resultante. Si de la función *hash* no se obtiene un resultado ajustado a la regla convenida (que empiece por la cantidad de ceros prefijada), se modifica ligeramente el contenido del campo adicional del bloque para volver a obtener su *hash*, y se repite así el procedimiento hasta obtener el resultado buscado. Veremos más adelante y con más detalle cómo este resultado lo obtendrá aleatoriamente, pero siguiendo una pauta estadística de acuerdo a su potencia computacional, uno de los miles y miles de *mineros* que trabajan de forma remunerada (en bitcoins, claro) para esta finalidad.

Si a las funcionalidades obtenidas por Blockchain añadimos su naturaleza distribuida (arquitectura de demostrado éxito en innumerables aplicaciones, empezando por la propia Internet) y descentralizada (que no exige una figura central o superior de la que emane la confianza en el sistema), así como el importante ahorro de costes que la implementación de este único registro global y autónomo de conciliación automática puede suponer para los operadores financieros en relación a los costes de los actuales sistemas de registro individualmente administrados (duplicados por definición en cada *contraparte* que opera en un mercado donde nadie se fía de nadie), se justifica plenamente el actual interés en el estudio de la tecnología Blockchain, que se concreta en apoyos de millones de euros o de dólares a *start-ups*, consorcios, o proyectos

internos para estudiar su aplicabilidad práctica y la confirmación de los ahorros vislumbrados.

En este trabajo nos vamos a centrar, precisamente, en la exploración de Blockchain como nueva tecnología, quizá disruptiva (está por ver), de múltiples aplicaciones, y cuyo ámbito natural se encuentra en el mundo financiero y, en particular, y al menos inicialmente, como aplicación de *back-office* para registrar transacciones y saldos de activos financieros entre operadores de estos mercados. Veremos qué se puede (y qué no) aplicar directamente desde la actual Blockchain al muy regulado mundo de las entidades financieras, y en qué habrá que modificarla para que pueda resultar de aplicación en estos mercados. Y no nos preocuparemos tanto de los componentes sociológicos de la confianza sobre Bitcoin, ni de sus connotaciones económicas y financieras, ni de su imprevisible devenir futuro. A partir de ahora Bitcoin, al igual que el resto de criptomonedas basadas en ésta, sólo nos interesará como aplicación exitosa (y de momento única) de la Blockchain. Y, ya entrados en materia, quizá éste sea un buen momento para algunas precisiones terminológicas.

### **1.1 Terminología**

Antes de continuar conviene poner de manifiesto algunas precisiones terminológicas e incluso plantear algunas conceptuales.

Hace pocos años, en los inicios del fenómeno Bitcoin, este término y Blockchain tendían a confundirse. De hecho, el término blockchain ni siquiera llega a aparecer de forma explícita en el *paper* de Nakamoto, refiriéndose a la cadena de bloques simplemente como *chain*. Conforme se analiza la tecnología resulta apreciable que una cosa es la tecnología que conforma la infraestructura *peer-to-peer* (la Blockchain) para soportar de manera segura una criptomoneda, y otra muy diferente la propia criptomoneda (Bitcoin). Aunque es discutible el poder llegar a convenir que Bitcoin es Blockchain, definitivamente es evidente que Blockchain no es Bitcoin.

Algún tiempo después de su aparición empezamos a distinguir entre las *permissionless* o *permissioned blockchains*, (públicas o privadas, respectivamente) según sea su acceso libre a quien quiera (como Bitcoin) o restringido a un conjunto limitado y controlado de partícipes (como la totalidad de los proyectos vinculados a los operadores financieros y que, en general y según veremos, serán las que tratemos en este trabajo). Finalmente, en el tiempo de escribir este trabajo ha tenido éxito la denominación de *Distributed Ledger Technology*, o DLT, en un sentido más amplio que blockchain: esta última es una DLT, pero se pueden implementar DLTs mediante otras estrategias alternativas a la cadena de bloques. Esta última denominación descrita suele estar vinculada a los ámbitos más formales de los usos regulados y comerciales de esta tecnología.

Anticipábamos en el apartado anterior que, a pesar de su aceptación y la correlación *real* de su valor con otros activos clásicos de referencia (monedas como el euro o el dólar), Bitcoin solo existe como representación binaria en una base de datos compartida y replicada en miles de nodos de una extensa red de ámbito planetario. Por tanto, y en toda la estructura Bitcoin/Blockchain, ¿hasta dónde llega la moneda y empieza su infraestructura soporte? En puridad, y como pronto veremos, una cantidad o fracción de bitcoins sólo se instancia en algunos campos numéricos de la estructura de datos que

define una transacción. Quizá resulte conveniente, visto lo anterior, descomponer el sistema Bitcoin/Blockchain en tres entidades conceptualmente diferentes:

- La moneda (o valor) propiamente dicha: hemos avanzado que en términos de representación es un simple campo numérico perteneciente a una estructura mayor, y que en nuestro contexto recoge bitcoins.
- Las transacciones en sí: implementan la cuantía y la transferencia de valor mediante una estructura de datos determinada que es objeto de la acción de diferentes algoritmos y protocolos.
- La cadena de bloques: base de datos que recoge la totalidad de transacciones del sistema y que permite su contabilidad, y ello bajo unos esquemas de seguridad tales que permiten certificar de manera indubitable la titularidad de cada valor en consideración.

Si nos referimos a Bitcoin, el primero de estos tres elementos, y que constituye el valor para cuyo servicio se implementa todo el sistema, ya sabemos que pertenece más al ámbito conceptual que al real, más al ámbito de lo que asimilábamos a *estados mentales* que a activos monetarios en sentido clásico. Se crea de la nada y su única y necesaria conexión con la *realidad* es un conjunto limitado de simples campos numéricos dentro una estructura de datos enrevesada, la transacción, que se lanza a un océano de miles y miles de nodos de diferentes tipos en una compleja red que soporta diferentes arquitecturas de datos y protocolos no triviales diseñados para asegurar la confianza y reafirmar, en definitiva, ese estado mental que da carta de naturaleza a un activo como es Bitcoin.

Los otros dos elementos, las transacciones y la blockchain, representan la totalidad de la infraestructura soporte. Por tanto, podremos convenir que las transacciones, con su propia infraestructura representada como veremos por direcciones, inputs y outputs, *scripts*... quedan fuera de la definición estricta de la criptomoneda. Y si consideramos que, en definitiva, la cadena de bloques no es tan solo un soporte conveniente para el registro de las transacciones, sino que aporta mecanismos para que éstas queden aseguradas, abusando algo del lenguaje podríamos convenir que toda la infraestructura soporte de Bitcoin, esto es, el conjunto formado por las transacciones y la cadena de bloques, es, en definitiva, Blockchain. Por otra parte, no es éste un enfoque original puesto que su semántica es la utilizada por los principales grupos de trabajo sobre las diferentes aproximaciones de la Blockchain a la industria.<sup>3</sup> Conociendo las salvedades y limitaciones de tal definición extendida, y acreditando la practicidad de esta denominación, será la que usemos en este trabajo excepto que se especifique lo contrario.

Además, en lo sucesivo nos referiremos al sistema Bitcoin/Blockchain, o simplemente Blockchain, como la tecnología correspondiente a la implementación original y actual de la cadena de bloques o de la infraestructura soporte de Bitcoin, salvo que se explicite un mayor alcance que recoja implementaciones alternativas.

---

<sup>3</sup> En consultoras (como, por ejemplo, PwC en <http://www.pwc.com/us/en/financialservices/publications/viewpoints/assets/qa-what-is-blockchain.pdf>); bancos centrales (como, por ejemplo, la Reserva Federal en <https://www.federalreserve.gov/newsevents/speech/brainard20161007a.htm>); operadores (como, por ejemplo, BBVA en <http://www.centrodeinnovacionbbva.com/ebook/ebook-tecnologia-blockchain>); o en los propios desarrolladores (como, por ejemplo, Hyperledger en <https://wiki.hyperledger.org/groups/whitepaper/whitepaper-wg>).

## 1.2 Objetivos del trabajo

El presente trabajo tiene como **principal objetivo** el análisis de la aplicabilidad de la concepción original de Blockchain como tecnología soporte de transferencia efectiva de valor entre diferentes partes, a los mercados financieros actuales (aunque extensible a otros muchos campos), fuertemente regulados y centralizados en comparación al objetivo inicialmente contemplado para aplicación de esta tecnología (en particular, el muy descentralizado y libre Bitcoin), así como de las transformaciones que deberían llevarse a cabo en este diseño inicial para acomodarlo, si fuera posible, a las necesidades regulatorias ya comentadas para su uso por los operadores de los mercados financieros *tradicionales*.

Este objetivo principal nos va a permitir reflexionar sobre una serie de **objetivos adicionales**, en parte instrumentales con respecto al principal, pero conceptualmente muy relevantes por sí mismos:

- La comprensión en cierta profundidad de la tecnología Blockchain.
- Entender la necesidad de herramientas criptográficas adecuadas para sostener una infraestructura autónoma y segura que permita la correcta transferencia y administración de valor (estando éste representado por cualquier manera adecuada a la realidad del problema a analizar). En particular, nos ocuparemos intensivamente de la firma digital y de las funciones *hash*.
- La resolución del consenso mediante una *Proof-of-Work* que hace uso de estas herramientas criptográficas.
- Razonar sobre el concepto de *valor* relevante en nuestro contexto, pero sin perder de vista que el trabajo que nos ocupa es eminentemente tecnológico, no financiero, por lo que abstraeremos un *modelo de valor* común a la mayoría de los activos que se *transaccionan* en estos mercados.
- Conocer y analizar las propuestas y soluciones existentes hoy por hoy que se aproximen a, o estén alineadas con, nuestro objetivo principal.

Para acometer estos objetivos, inicialmente tendremos que profundizar y resumir adecuadamente las funcionalidades de la Blockchain. Después revisaremos los mecanismos tecnológicos originales, y en particular los criptográficos, que han hecho de la Cadena de Bloques la robusta tecnología que está dando soporte seguro y, de momento, eficaz a una moneda virtual como el Bitcoin, que registra varios cientos de miles de transacciones al día. Pasaremos a analizar de forma somera las exigencias regulatorias que los operadores financieros soportan desde los diversos reguladores nacionales y transnacionales, lo que nos dará fundamentos para analizar cómo debería adecuarse esta tecnología a los mercados. Finalmente, con estos últimos fundamentos intentaremos formalizar un marco sistemático de adecuación de la tecnología Blockchain a la realidad de los operadores, incluyendo la revisión de cómo esta adecuación se está intentando llevar a cabo desde diferentes proyectos en curso.

Así, prácticamente por definición, éste es un trabajo de análisis y de rediseño, de *refactorización* en un amplio sentido (sobre todo conceptual), de adecuación (si fuera posible) de un producto ya terminado y en producción como es la actual Blockchain que

da soporte al Bitcoin, a estándares de regulación muy alejados de lo contemplado inicialmente por el vigente diseño.

Intentaremos, de la manera más detallada y formal posible, ofrecer un nuevo diseño basado en el original que permita fundamentar opciones que puedan tomar en consideración los operadores en estos mercados, pero siendo conscientes de que tal trabajo no podrá ofrecer un producto final definitivo y aplicable directamente a la industria dadas las limitaciones tanto temporales como de recursos que conlleva un trabajo de estas características sobre un sistema tan extenso y complejo como es Bitcoin/Blockchain. Aunque profundizaremos con cierto detalle en la tecnología de este sistema, nos mantendremos, en general, en ámbitos conceptuales en lo que se refiera a nuestro objetivo de estudiar su aplicabilidad práctica en la industria, si bien intentaremos formalizarlos en la medida que nos sea posible.

Para obtener este diseño intentaremos seguir en lo posible un proceso *clásico* de desarrollo de software [2], para el que de manera más o menos explícita necesitaremos:

1. El conocimiento del dominio del sistema a transformar y su situación actual
2. La enumeración y revisión de los correspondientes requisitos
3. La formalización de un marco de trabajo que incluya, explícita o implícitamente, los casos de uso vinculados a los requisitos identificados
4. El análisis del problema por medio del refinamiento y la estructuración de lo anterior
5. Finalmente, modelado conceptual del sistema resultante mediante la aplicación de los resultados de los pasos anteriores junto con los necesarios condicionantes externos y/o no funcionales que podamos identificar

Todo lo anterior interrelacionado por el proceso iterativo e incremental propio del desarrollo clásico de software. Para ello abordaremos la consecución de nuestros objetivos mediante una planificación del análisis y del diseño cuyas principales tareas pueden resumirse de la forma siguiente:

- Comprensión del **dominio del problema**: fundamentos tecnológicos de la Blockchain
  - Bitcoin como fundamento de Blockchain
  - El diseño de Bitcoin/Blockchain
  - Soporte criptográfico: estudio de funciones hash y firma digital
  - Direcciones Bitcoin
  - Transacciones
  - Red y serialización
  - Scripts
  - Bloques y cadena de bloques
  - Minado y consenso
  - Consenso, *proof-of-work*
  - Software Bitcoin Core
  - Software Wallet
- Comprensión del **dominio del problema**: necesidades regulatorias de los operadores financieros
  - Regulación y autoridades centrales



- Necesidades de la Blockchain en un contexto regulado
- **Análisis** del problema:
  - Carencias de la blockchain para un mercado regulado
- **Análisis** del problema: revisión de propuestas y soluciones en curso
  - Hyperledger
  - Ethereum
  - Corda
  - Ripple
  - Otros
- **Diseño:** una blockchain para operadores regulados
  - Marco de trabajo para relacionar características actuales de la Blockchain y requisitos de la industria
  - Modificaciones
- **Diseño:** conclusiones
  - Conclusiones sobre la Blockchain bajo regulación
  - Conclusiones sobre el desarrollo del proyecto

### ***1.3 De aquí en adelante***

Una vez establecido el marco de realización de este trabajo (objetivos, productos y metodología), avancemos lo que vamos a revisar en los siguientes capítulos.

Inmediatamente después de esta introducción, dedicaremos el segundo capítulo a revisar cómo se diseña y aparece Blockchain como soporte a la criptomoneda Bitcoin.

Dentro los fundamentos tecnológicos de toda la infraestructura Blockchain, tienen gran importancia dos herramientas criptográficas que se usan de forma recurrente en ella: las funciones *hash* y la ECDSA (*Elliptic Curve Digital Signature Algorithm*). Puesto que su comprensión nos permitirá profundizar en tales fundamentos, revisaremos estas herramientas en el capítulo 3.

En el capítulo 4 trataremos más detalladamente el concepto Blockchain, sus características estructurales, sus fundamentos y sus funcionalidades.

Dedicaremos el capítulo 5 a la construcción del marco de trabajo que relacione las características actuales de Blockchain con los requisitos necesarios en caso de regulación, y que nos permita, si fuera posible, la transformación de la actual Blockchain para su adaptación a una industria regulada. Detallaremos en el capítulo 6, en su caso, las transformaciones a ejecutar e introduciremos los efectos de las mismas.

Terminaremos con un capítulo de conclusiones que nos permitirá resumir los principales resultados del trabajo.

# *Bitcoin como origen de la tecnología Blockchain*

El último día de octubre de 2008, desde un servidor público y gratuito de correo electrónico, un tal Satoshi Nakamoto publicó en una lista de correo sobre criptografía un documento bajo el título *Bitcoin: A Peer-to-Peer Electronic Cash System* [1]. Hasta el momento no se ha podido establecer la identidad real de tal nombre.

## **Bitcoin: A Peer-to-Peer Electronic Cash System**

Satoshi Nakamoto  
satoshin@gmx.com  
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Quizá algún día podamos conocer si la persona o personas que están detrás del seudónimo japonés intuía/n el alcance de lo que estaban publicando o si se resignaba/n a anunciar un proyecto más de dinero electrónico probablemente condenado al fracaso, tal y como había sucedido con todos los proyectos anteriores por unas u otras razones. Tampoco sabemos todavía si lo publicado como una *versión peer-to-peer*<sup>4</sup> de dinero electrónico correspondía a una infraestructura de pagos a establecer sobre una moneda real, una prueba de concepto sobre una moneda virtual, o si se pretendía realmente iniciar lo que devino posteriormente, es decir, una moneda alternativa virtual pero de contravalor real con respecto a las monedas *clásicas*.

La publicación del *paper* coincidió con una época convulsa en lo económico, en la que las entidades financieras, tanto americanas como europeas, se encontraban en entredicho por una serie de actuaciones inadecuadas en la comercialización de productos de alto

<sup>4</sup> En contraposición al modelo cliente-servidor, un sistema de intercomunicación entre nodos de una red se denomina *peer-to-peer*, o P2P, o entre pares, cuando los nodos interconectados no están sujetos a una jerarquía formal sino que, en general, todos tienen iguales funcionalidades, por lo que pueden comportarse bien como clientes, bien como servidores, en su interrelación entre todos ellos.

riesgo y/o por situarse en los primeros compases de una de las mayores crisis económicas de la historia. El caso es que, pocas semanas después de su publicación, a principios de 2009 tuvo lugar la primera implementación software del núcleo de Bitcoin y la inicial encarnación de los primeros bitcoins como unidad monetaria. A partir de ese momento, el interés de ciertas comunidades interesadas en la criptografía, en el software libre, o en soluciones financieras alternativas a los tradicionales mercados financieros, fue creciendo de tal manera que Bitcoin se ha convertido en uno de los primeros iconos del siglo XXI tanto en sus aspectos tecnológicos como sociales y financieros.

Pero, ¿qué es lo que ofrecía el *paper* de Nakamoto? Según su propio resumen, una versión *peer-to-peer* de dinero electrónico que permitiría pagos *online* sin contar con la intervención de bancos ni de ninguna autoridad central que se responsabilizara de prevenir el *problema del doble pago*, es decir, evitando que el propietario de un determinado activo representado digitalmente (digamos un bitcoin) pudiera emitir varias transferencias del mismo a diferentes beneficiarios, y que estos no pudieran confirmar esta situación hasta pasado cierto tiempo, con previsibles e indeseables consecuencias.

Este problema del posible doble pago había sido más o menos resuelto en anteriores proyectos de dinero digital (*DigiCash* [3], *Citibank EMS* [4], por nombrar algunos de los más conocidos) contando con la participación de autoridades centrales que garantizaban la seguridad de las transacciones. Ninguno de estos proyectos, ni tampoco unos cuantos más con diferentes fundamentos, pero también con diferentes debilidades, llegó a tener verdadero éxito y continuidad. Hubo que esperar a las estrategias de Nakamoto y su Bitcoin para que la moneda digital haya podido extenderse y generalizarse de la manera en que lo ha hecho.

En lo que queda de capítulo veremos con más detalle el contenido del trabajo original de Nakamoto. Este *diseño* de Bitcoin es denso, carente de adornos, y compacta en pocas páginas toda la lógica y la estructura que subyacen en la complejidad de Bitcoin. A veces da por conocidos pormenores en los que puede costar profundizar, y en otras ocasiones resume en pocas líneas diferentes conceptos o cursos de acción relevantes, no siempre claros por carencia de detalle. Necesitaremos para explicarnos, por lo tanto, algo más de espacio que el utilizado originalmente en el *paper*.

## **2.1 Bitcoin: A Peer-to-Peer Electronic Cash System**

Bajo este título Satoshi Nakamoto elabora un disruptivo estudio en poco más de media docena de páginas, que constituye una descripción general de los mecanismos requeridos por un sistema de moneda o dinero digital cuyo intercambio no necesitaría de terceras partes para aportar la necesaria confianza en el mismo evitando la problemática del posible doble pago.

Desde el resumen, Nakamoto adelanta que la solución a estos problemas viene de la mano de la utilización de firmas digitales que se soportarían en una red *peer-to-peer* diseñada para eliminar figuras de autoridad, certificación o confianza intermedias. En esta red las transacciones son referenciadas en el tiempo mediante su inclusión en un bloque que recoge las transacciones realizadas en el último intervalo de tiempo en consideración, bloque que, a su vez, se añade a una cadena creciente tras haber superado una prueba de trabajo (*proof-of-work*) basada en funciones *hash* (ver detalles sobre estas

funciones en el capítulo 3, apartados 3.1 y 3.2), cadena que constituye un registro imposible de modificar si no se rehace toda la prueba de trabajo y que, por tanto, cuanto mayor sea la cadena tras la transacción que se pretende atacar, mayor será el trabajo computacional para tratar de modificarla con éxito, haciendo este problema intratable (ver anexo A) en última instancia.

Aunque en estos primeros compases del *paper* no se explicitan las causas de una posible bifurcación de la cadena de bloques (la confección del bloque en curso puede incluir diferentes conjuntos de transacciones según la disposición de las mismas por cada nodo minero, por lo que es posible la resolución simultánea de dos o más bloques de contenido diferente cuyo encadenamiento provoca bifurcaciones en la cadena), sí se nos aclara que su resolución mediante la cadena de mayor longitud final supone, en definitiva, la confirmación de que la cadena proviene de, y es consensuada por, un conjunto mayoritario de potencia computacional, es decir, por una mayoría de los nodos partícipes.

De acuerdo a esto último, si se garantiza que la mayoría de nodos mineros son honestos y no cooperan para atacar a la red, se viene a garantizar la legitimidad de la cadena más larga, que en condiciones ordinarias no podrá ser atacada con éxito excepto que se llegue a manejar una mayoría de los recursos computacionales de la misma.

Tras una breve introducción en la que pone de manifiesto la problemática de la reversibilidad de los pagos, su resolución en Bitcoin mediante la sustitución de la confianza en figuras externas por pruebas criptográficas, y algunas cuestiones genéricas adicionales, Nakamoto pasa a describir una de las piezas básicas de cualquier sistema de pagos *online*: las transacciones.

## **2.2 Transacciones y marcas de tiempo en Bitcoin**

*Definimos una moneda electrónica como una cadena de firmas digitales.* Es la sucinta definición con la que Nakamoto abstrae la secuencia de transacciones que una determinada unidad de la criptomoneda, o una fracción de ella, o un conjunto de ellas, puede ir registrando a través del tiempo como transferencias de valor entre sucesivos propietarios. De forma explícita se precisa que cada propietario transfiere a otro este valor firmando digitalmente (es decir, demostrando que gobierna la clave privada correspondiente) un *hash* sobre una entrada compuesta de la transacción anterior (la que dio la propiedad al actual transmitente explicitando su clave pública) y la clave pública del beneficiario de la transmisión. Este último podrá movilizar este valor recibido transfiriéndolo a un tercero de la misma manera. Y así, sucesivamente se va conformando un tracto de transacciones fundamentadas en la firma digital y en el que la situada al final es indicativa de la propiedad última del valor en cuestión.

Tomando prestado el esquema que se aporta en el propio *paper* tenemos:

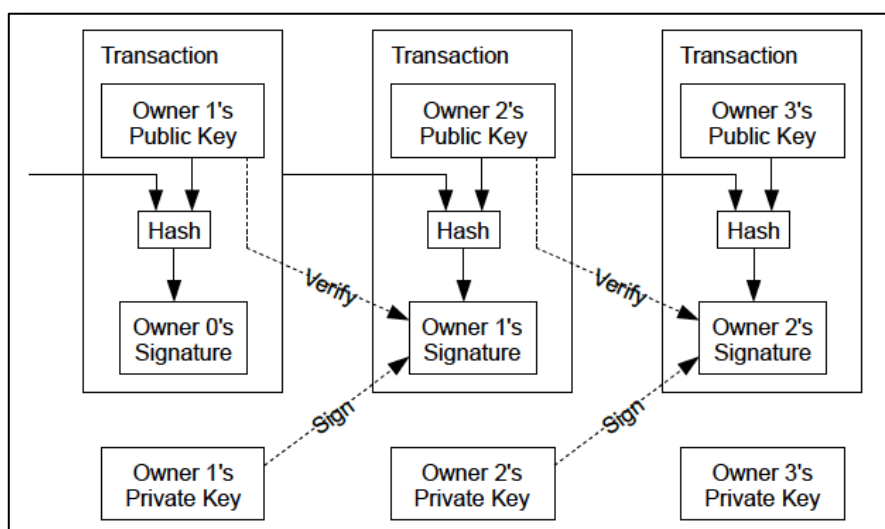


Fig. 2.1: transacciones en Bitcoin (S. Nakamoto)

Vemos de forma esquemática el contenido de cada transacción y su conexión entre sucesivas. Cada transacción incluye la firma digital del *hash* resultante de aplicar la correspondiente función *hash* criptográfica sobre el encadenamiento de la transacción anterior y la clave pública del receptor o beneficiario del valor. Una vez recibido este valor por el beneficiario, éste podrá transferirlo a su vez firmando con su clave privada un nuevo *hash* resultante de una nueva entrada para la función *hash* que incluiría la transacción que certifica su propiedad y la clave pública de aquel a quien desea transferir el valor. En la fig. 1 *Owner 3* es el actual propietario del valor transferido. Resulta evidente que la definición como cadena de firmas digitales no puede ser más concisa ni más innegable (repasaremos los detalles de las firmas digitales, y en particular de ECDSA, la utilizada en Bitcoin/Blockchain, en el capítulo 3, apartado 3.6).

Un resultado directo de lo anterior es que cualquier beneficiario (en realidad, cualquier tercero, aunque no participe en la transacción) puede comprobar la legitimidad de la transacción que le da el derecho de propiedad sobre el valor transferido mediante la verificación de la firma digital puesto que puede acceder a la clave pública del transmitente. En la fig. 1, *Owner 2* puede aplicar el proceso de verificación de la transacción comprobando que la clave pública de *Owner 1* verifica la firma que este último realizó con la clave privada par de la pública conocida. En caso de verificarse la firma, *Owner 1* no podrá por tanto *repudiar* la emisión de la transacción y la transmisión de su derecho o valor.

Ahora bien, en lo que hemos visto no hay nada que impida a *Owner 1* construir después de la transferencia a *Owner 2* una nueva transacción basada de nuevo en la transacción que le aportó la propiedad del valor, pero esta vez utilizando una clave pública de otro beneficiario, que a su vez también podría validar satisfactoriamente la firma digital del transmitente. En resumen, nada impediría que *Owner 1* cometiera un *doble gasto* transfiriendo el mismo valor a dos, o más, beneficiarios. Para solucionar este problema Nakamoto establece lo que podríamos denominar como el principio de prevalencia de la primera transacción registrada, que será la única válida y, por tanto, la única que cuente como transmisión efectiva del valor. Y lo complementa con la estrategia de que la única manera de confirmar la unicidad de la transmisión de un determinado valor es tener constancia de todas las transacciones, que en consecuencia deben ser anunciadas

públicamente según se realicen para poder proceder a la comprobación de la deseada unicidad y evitar de esta manera el doble gasto. En conclusión, el beneficiario de cada transacción podrá sentirse confortable con la seguridad de su derecho siempre que la mayoría de los nodos involucrados en la gestión de la cadena consensúen que la transmisión de ese valor en concreto es la primera registrada. En caso de doble gasto, cualquier posterior beneficiario podrá descubrir de forma prácticamente trivial tal situación problemática al comprobar la existencia de una transmisión previa del valor.

Como herramienta para discriminar esta necesaria temporización entre transacciones, Nakamoto propone inicialmente un *servidor de marcas de tiempo* conceptual que actúe de la siguiente manera:

1. Reunir en un bloque un conjunto de ítems, que corresponderían en nuestro caso a las transacciones realizadas en un determinado periodo de tiempo.
2. Obtener como entrada de una función *hash* la concatenación entre el bloque en cuestión y el *hash* previo obtenido para el bloque anterior de manera similar al resultado de aplicar ahora la función *hash*.
3. Hacer público el resultado *hash* del punto anterior de manera que el momento de su publicación resulte indubitable, lo que probaría que en ese momento los datos que originaron el resumen *hash* existían.

Dado el mecanismo anterior, cada marca de tiempo referenciaría la marca anterior mediante el *hash* de esta última, lo que conformaría una cadena según el siguiente esquema:

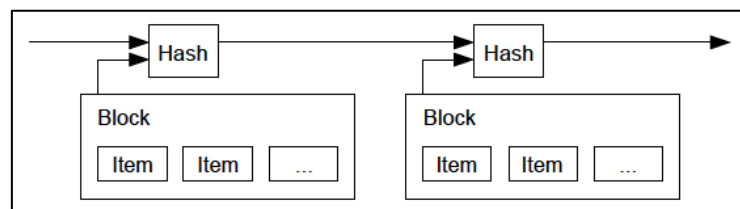


Fig. 2.2: concepto de marcas de tiempo (S. Nakamoto)

Hasta aquí el diseño conceptual, que aparenta ser válido y eficaz. Pero, ¿cómo implementar este servidor de marcas de tiempo en un contexto distribuido, sin autoridad central, y manteniendo su eficacia y eficiencia?

### 2.3 Proof-of-Work

Para resolver la última cuestión planteada en el punto anterior, Nakamoto reutiliza el concepto de *prueba de trabajo* propuesto una década antes por Adam Back para *Hashcash* [5], un mecanismo destinado a dificultar la difusión de *spam* mediante la exigencia de un trabajo computacional previo al envío de un correo electrónico, lo que haría impracticable el envío masivo de correo basura.

Este trabajo computacional consiste en ir modificando la entrada de una función *hash* (tal como SHA-256) hasta obtener un resultado en binario que comience con un determinado número  $n$  de bits a cero (o, lo que es equivalente, a que sea menor que un determinado número dado). La mejor estrategia conocida para obtener un hash de  $n$

ceros bajo SHA-256 requiere un esfuerzo computacional que es exponencial en  $n$  según se vayan añadiendo ceros al número de los iniciales exigidos, ya que por cada bit cero adicional requerido el tiempo medio de trabajo se duplica. Por otra parte, una vez ejecutado el trabajo computacional la comprobación de que se ha obtenido el resultado buscado es trivial. En su aplicación a Bitcoin, en el momento de escribir estas líneas los hashes de cada bloque añadido a la cadena de bloques cuentan con sus primeros 17 dígitos hexadecimales (de un total de 64) a cero, lo que equivale a 68 bits iniciales a cero (de un total de 256 bits).

Para Bitcoin se habilita en cada nuevo bloque de transacciones un campo denominado *nonce* para registrar un contador que irá variando su contenido mientras que el *hash* que acabamos de describir no obtenga un resultado que encaje en lo requerido. De esta manera, el esquema de marcas de tiempo visto en el apartado anterior se modifica ligeramente para incluir este campo, por lo que la función *hash* actuará ahora sobre la concatenación entre el bloque en cuestión, el *hash* previo, y el *nonce*:

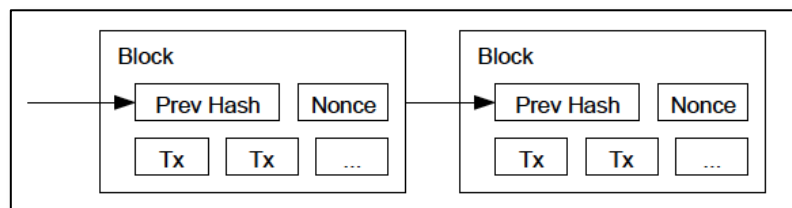


Fig. 2.3: estructura básica de los bloques de transacciones en Bitcoin (S. Nakamoto)

Una vez obtenido el resultado satisfactorio (un valor retornado por la función hash que empiece, para repetir, por un mínimo prefijado de bits a cero), y seguramente tras un importante esfuerzo computacional, la prueba de trabajo se da por satisfecha, el bloque se añade a la cadena, y el contenido del mismo no podrá ser cambiado sin rehacer el esfuerzo. Y como ya habíamos avanzado, conforme se acumulan bloques en la cadena la modificación de uno de ellos supondría no sólo rehacer el trabajo en el bloque en cuestión, sino en todos los siguientes puesto que incluyen los *hashes* de enlace previos que, de forma evidente, también se verían modificados.

La *proof-of-work* también viene a solucionar el problema del consenso (lo que en algunos ámbitos se viene denominando el *problema de los generales bizantinos*): la decisión de la mayoría se refleja en la mayor de las cadenas que pueden darse en el sistema puesto que será en ésta donde se registrará el mayor esfuerzo computacional invertido. Si este esfuerzo proviene de una mayoría de nodos honestos, la cadena honesta crecerá más rápido que cualquiera originada por el ataque de uno o varios nodos deshonestos, garantizando así la legitimidad de las transacciones que almacena.

Por último, el grado de esfuerzo requerido es modificable variando el número de ceros iniciales exigidos. Resulta sencillo, por tanto, modificar la dificultad del trabajo para conseguir un ritmo estable de generación de bloques, ritmo que varía en función de la potencia computacional utilizada en cada momento. El diseño actual supone la generación de un bloque cada diez minutos, aproximadamente. Conforme se van generando más rápido, la dificultad aumenta (o al contrario, pero más inusualmente: la dificultad disminuye si el ritmo de generación es más lento).

## 2.4 Red e incentivos

En el apartado *Network* del documento de Nakamoto se establece un sencillo guión de seis pasos que resume de forma clara los principales flujos de información y de acción de todo el sistema, desde la emisión de una transacción hasta su inclusión en la cadena de bloques:

1. Cada nueva transacción se transmite por la red a todos los nodos *mineros* partícipes.
2. Cada nodo minero registra en un bloque las transacciones recibidas en un determinado periodo de tiempo (unos diez minutos en el diseño actual).
3. Cada nodo trabaja la *proof-of-work* sobre el bloque que ha construido para intentar obtener un resultado ajustado a lo solicitado por el sistema, es decir, variará sucesivamente el *nonce* del bloque hasta obtener un hash que comience, al menos, con tantos bits a cero como los requeridos por la dificultad en curso del problema.
4. Cuando un nodo de los muchos que compiten por tener éxito en la *proof-of-work* obtiene el resultado adecuado, transmite el bloque resuelto a todos los otros nodos.
5. Los nodos que reciben el bloque resuelto comprueban la validez de la resolución de la *proof-of-work* y que todas las transacciones que contiene son correctas y legítimas, esto es, que no existan errores ni que sus emisores hayan incurrido en doble gasto. Una vez comprobada su corrección, cada nodo acepta el bloque en cuestión y lo añade a su cadena de bloques.
6. Inmediatamente después de la aceptación del nuevo bloque, cada nodo empieza a trabajar en la construcción del siguiente, que incluirá, como ya hemos visto, el *hash* del bloque que acaba de añadirse y las transacciones recibidas durante el periodo de producción del bloque que acaba de resolverse y que por tanto han quedado fuera de éste.

Sabemos que puede darse el caso de que dos o más nodos resuelvan versiones diferentes del nodo en curso (porque registren diferentes conjuntos de transacciones), y por este motivo un nodo debe saber responder a una situación en la que reciba varios bloques resueltos y, en consecuencia, tratar adecuadamente esta bifurcación. Para ello, seguirá trabajando sobre el primer bloque recibido, pero guardará registro de la bifurcación. Cuando se reciba el siguiente bloque resuelto por la correspondiente *proof-of-work*, comprobará por dónde se desarrolla la rama de mayor longitud, y abandonará definitivamente la bifurcada que haya resultado más corta.

Para terminar el apartado sobre la red, Nakamoto hace hincapié en dos fortalezas dimanantes de la arquitectura distribuida: por una parte, no es necesario que todas las transacciones alcancen todos los nodos para que puedan ser incluidas en la cadena de bloques; por otra, el sistema es tolerante a fallos, y si un nodo descubre al recibir un nuevo bloque que le falta alguno intermedio, solicitará inmediatamente su reenvío.

Directamente vinculadas al mantenimiento de la red y a su adecuada gestión, nos encontramos con las dos estrategias de incentivación diseñadas por Nakamoto para conseguir estos propósitos. La primera consiste en la generación de nueva criptomoneda en cada bloque validado y añadido a la cadena. Esta nueva moneda es entregada como recompensa al nodo minero que ha resuelto la prueba de trabajo, y su magnitud



disminuye a la mitad (*halving*) cada cuatro años hasta que desaparezca completamente bien entrado el próximo siglo (si es que no se realiza modificación previa en el protocolo original) con el objetivo de limitar la masa monetaria en bitcoins finalmente emitida. Este incentivo supone en la actualidad una cantidad de 12,5 bitcoins para cada nuevo bloque validado, que se transfieren al nodo minero mediante una transacción especial que se habilita en cada bloque y en la que no figura transmitente, y que por tanto origina los bitcoins nuevos.

La segunda estrategia de incentivación al minado de bloques corresponde a los gastos de transacción (*transaction fees*). Si el valor de salida (*outputs*) de una transacción es menor que el valor de entrada (*inputs*), se asume que la diferencia son gastos de la operación que el transmitente transfiere al nodo minero que resuelva el bloque que incluya su transacción. Así, la inclusión de estos *gastos* estimula a los mineros a incluir la transacción lo antes posible en el bloque que estén trabajando.

Esta doble vía de remuneración a los mineros quedará limitada a la segunda estrategia de estímulo vía gastos cuando se llegue al máximo de bitcoins a emitir y no se genere nueva criptomoneda. Previsiblemente esto sucederá mucho antes, cuando el efecto de la reducción a la mitad (*halving*) de la recompensa en nueva moneda haga que ésta sea tan modesta que suponga una cifra marginal en comparación a los ingresos por *fees*.

El trabajo de Nakamoto termina el apartado sobre incentivación con uno de los razonamientos sobre la seguridad de Bitcoin que constituye, probablemente, una de las claves del éxito obtenido: los incentivos vistos estimulan a los nodos a ser honestos, ya que si uno de ellos obtiene mayor potencia computacional que toda la mantenida por el resto de nodos, podrá optar bien por actuar de manera deshonesta incurriendo, por ejemplo, en doble gasto, bien por actuar de manera honesta y utilizar todos sus recursos para crear nueva criptomoneda y mantener la mayor probabilidad de adjudicársela. ¿Y por qué tal nodo encontraría más provecho en actuar honestamente? Porque lo contrario supondría socavar los cimientos mismos del sistema Bitcoin y, en consecuencia, los fundamentos de la propia riqueza del nodo deshonesto...

## ***2.5 Simplificando el almacenamiento y la verificación de pagos***

El contenido de las transacciones antiguas cuyos *outputs* hayan sido ya gastados no es, en general, estrictamente necesario para una gestión ordinaria de la moneda. Por lo tanto, podemos reducir sensiblemente el espacio de almacenamiento necesario para cada bloque construyendo un *árbol hash de Merkle* para cada uno de los bloques antiguos.

Un árbol de Merkle (*Merkle Tree*) [6] es una estructura de datos en forma de árbol cuyos nodos hoja registran el resultado de aplicar una función *hash* sobre el bloque respectivo de datos que le es asignado, mientras que sus nodos internos recogen el resultado de aplicar la función *hash* sobre el encadenamiento de los *hashes* contenidos en sus respectivos nodos hijos.

El esquema aportado en el *paper* ayuda a su comprensión:

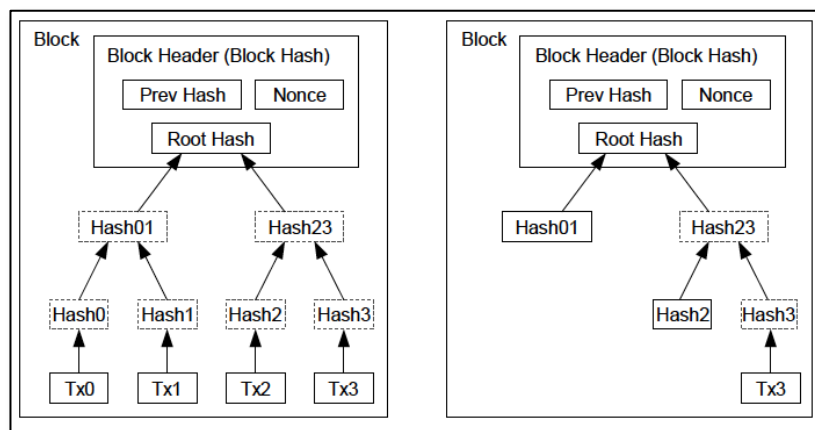


Fig. 2.4: hash de transacciones y árbol resultante de Merkle (i), y el mismo árbol tras la poda hasta transacción Tx2 (d) (S. Nakamoto)

Una vez construido el árbol de Merkle en los términos vistos (fig. 4, izquierda) podremos *podar* los nodos (*hashes*) internos (fig. 4, derecha) en lo necesario para obtener un sustancial ahorro de almacenamiento, aunque manteniendo íntegramente los datos de las transacciones no consumidas y de esta manera pendientes de gasto (la Tx3 del esquema, por ejemplo).

El *hash* del bloque, o cabecera (*block header*), incluye en estos casos la raíz del árbol de Merkle (además del *nonce* y del *hash* previo proveniente del bloque anterior), en lugar de las transacciones propiamente dichas.

La transformación operada en cada bloque antiguo con la eliminación de los datos explícitos de las transacciones de la forma que acabamos de ver reduce el tamaño de la cabecera de cada bloque a unos 80 bytes, lo que para la cadena de cabeceras de bloque completa supone un ahorro de espacio de tal magnitud que podría permitir su almacenamiento completo en la memoria RAM de cualquier máquina estándar.

Con esta estructura reducida de los bloques es posible realizar una verificación simplificada de la legitimidad de los pagos sin requerir la cadena de bloques íntegra, tal y como la necesita un nodo minero completo. El usuario que desee acometer esta verificación simplificada necesita:

- Una copia de la cadena de cabeceras de bloque de la cadena de bloques de mayor longitud.
- La rama del árbol de Merkle donde se encuentre la transacción que se desea comprobar (normalmente, la transacción que da origen a una nueva de la que probablemente el interesado en la verificación es el beneficiario).
- Constatar el enlace de esta rama al bloque que contiene la transacción en cuestión.

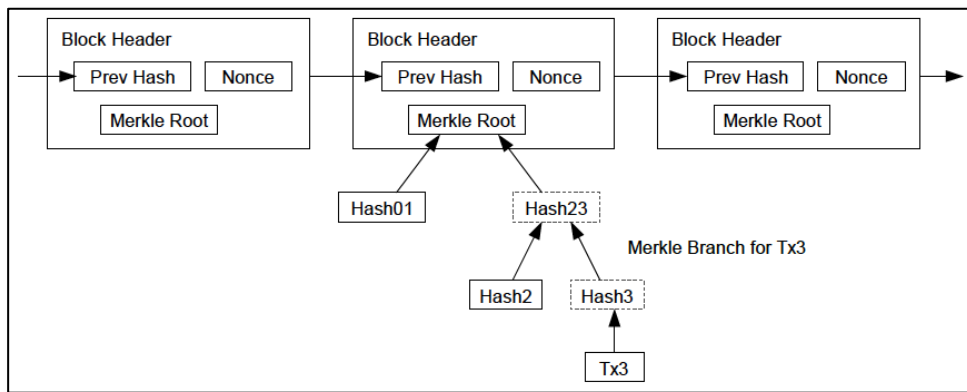


Fig. 2.5: la cadena de cabeceras de bloques y una rama de una transacción no gastada (S. Nakamoto)

En definitiva, no se puede confirmar la transacción por sí misma (como sí podría hacerlo un nodo que almacene toda la cadena), pero se puede confirmar que está incluida en uno de los bloques, y cuantos más bloques posteriores se hayan añadido más fuerte será la confianza en que la transacción está plenamente aceptada por la red.

Este método simplificado de verificación ofrece debilidades si un atacante adquiere la mayoría de la potencia computacional de la red y construye transacciones falsas para engañar a este mecanismo. Para actividades comerciales u otras que reciban frecuentes pagos, se recomienda la verificación con sus propios nodos completos.

## 2.6 Combinando y dividiendo valor

Cualquier transacción debe *gastar* la totalidad del valor de sus *inputs*. Una transacción ideal por compacta correspondería a la transmisión de la totalidad del valor contenido en una única transacción previa que acredita la propiedad del ahora transmitente: un solo *input* y un solo *output* de exactamente igual valor.

Pero esta transacción ideal no es la común en el día a día. Normalmente una transacción se construye con uno o más *inputs* (valores recibidos) para transferir un valor inferior a un tercero. Si tenemos tres *inputs* por, digamos, 3, 4, y 5 bitcoins, y queremos transferir 10 bitcoins a un tercero (importe que no podemos construir con sólo dos de los tres *inputs* disponibles), tendremos que implementar dos *outputs*: uno con los 10 bitcoins a transferir, y otro con los  $3+4+5-10 = 2$  bitcoins de "cambio" que el transmisor querrá que se le retornen a su monedero. Además, probablemente querrá entregar alguna fracción de bitcoin como gasto a favor del nodo minero que introduzca la transacción en el bloque cuya construcción esté en curso en ese momento, por lo que el importe del "cambio" será ligeramente inferior a los 2 bitcoins sobrantes.

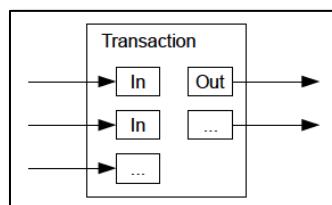


Fig. 2.6: transacción construida con varios inputs y varios outputs (S. Nakamoto)

De esta manera, la transacción tipo contará con uno o varios *inputs* y al menos dos *outputs*. De todas formas, nada impide que puedan darse todas las combinaciones posibles: un *input* y un *output*; varios *inputs* y un *output* (combinación de valores); un *input* y varios *outputs* (división de valor); y varios *inputs* y varios *outputs* (diferentes posibilidades entre combinación y/o división de valor).

## 2.7 Privacidad

El modelo clásico de privacidad de la industria financiera (aunque exportable a buena parte de los sectores económicos), se fundamenta en que las identidades intervinientes en una transacción solo son conocidas por los partícipes de la misma y por la autoridad o tercero que afianza la transacción, manteniendo el conocimiento sobre tales transacciones fuera del alcance público:

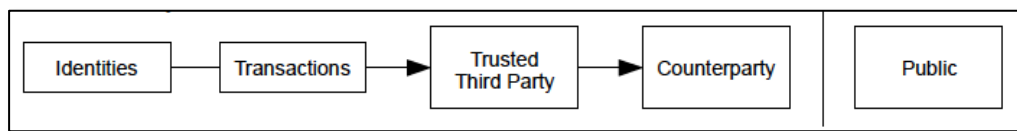


Fig. 2.7: modelo tradicional de privacidad (S. Nakamoto)

Puesto que en Bitcoin, según hemos visto, todas las transacciones deben comunicarse públicamente para su correcta datación y evitar así la posibilidad del doble pago, Nakamoto plantea que el modelo tradicional es inaplicable, y la privacidad de las partes en ausencia de una autoridad central que la proteja viene garantizada por el anonimato de las claves públicas.

De esta manera, las transacciones se mantienen al alcance público, pero no hay información explícita sobre la identidad última de quienes participan en cada una de ellas:

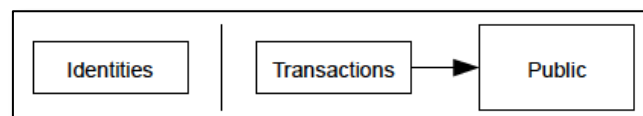


Fig. 2.8: modelo de privacidad de Bitcoin (S. Nakamoto)

No obstante, y como medida adicional de protección de la identidad, puede construirse un nuevo par de claves pública-privada para cada transacción, lo que evitaría atribuir las mismas a su propietario común.

## 2.8 Algunos cálculos

El denominado *problema de la ruina del jugador* (*Gambler's Ruin problem*), del que pueden encontrarse múltiples referencias (aquí seguiremos a [7], referencia aportada por el propio Nakamoto en su trabajo), es un problema clásico en el estudio de probabilidades, y admite un enunciado equivalente en términos de lo que se denomina *camino aleatorio* (*random walk*), también clásico en referencia a física básica de partículas.

El problema genérico intenta calcular la probabilidad de que un jugador llegue a arruinarse (pierda todo su capital inicial  $z$ ) en un juego en el que gana una unidad monetaria con probabilidad  $p$ , o la pierde con probabilidad  $q$ , ante un adversario que también cuenta con un determinado capital,  $a - z$ , para jugar. De lo anterior es inmediato que la suma de los capitales iniciales de ambos jugadores es  $a$ . El juego acaba cuando uno de los jugadores se arruina, lo que implica que el otro ha incrementado su capital hasta  $a$ .

Este enunciado resulta equivalente al camino aleatorio que podría seguir una partícula en el plano bidimensional a lo largo del eje  $x$  desde una posición inicial  $z$ , y que se trasladaría en una unidad hacia la dirección positiva, o hacia la dirección negativa, dependiendo del resultado de la observación del movimiento para un determinado instante (discreto) temporal. De manera arbitraria, pero ciertamente vinculada a la terminología de la distribución binomial, para cada instante en consideración decimos que la observación (o ensayo) es un éxito si la partícula se mueve en dirección positiva, y que es un fracaso si se mueve en dirección negativa. La posición  $x_m$  de la partícula en un determinado momento  $m$  reflejaría la ganancia ( $x_m > z$ ) o pérdida ( $x_m < z$ ) del jugador del enunciado anterior. El experimento con la partícula terminaría cuando ésta alcanzara la posición 0 o bien la posición  $a$  (siendo 0 y  $a$  equivalentes a la ruina o a la victoria total del jugador, respectivamente, en el enunciado relativo al juego), por lo que decimos que ambas posiciones conforman *barreras de absorción*, o fronteras, en el camino aleatorio.

Se plantee como se plantee el problema, admite diferentes juegos de parámetros, y veremos enseguida que nos interesará en particular el caso en el que  $a \rightarrow \infty$ .

En este apartado Nakamoto acota el escenario de la posible actuación de un nodo deshonesto: no puede crear criptomonedas de la nada, ni puede apropiarse de valor ajeno, puesto que los nodos honestos jamás incorporarían tales transacciones, inverificables, al bloque en curso, ni añadirían a la cadena un bloque resuelto con transacciones de estas características; lo máximo que puede intentar es modificar alguna de sus propias transacciones para intentar un doble pago en su beneficio. La competición entre la cadena de bloques honesta y la originada por el nodo deshonesto se caracteriza por seguir un camino aleatorio binomial (*Binomial Random Walk*), donde se considera éxito el que la cadena honesta añada un bloque (y se aumente en +1 su diferencia con la deshonestas), y se considera fracaso cuando el atacante añade un bloque a la cadena deshonestas (y se reduzca en -1 su distancia a la honesta).

En resumen, la probabilidad de que la cadena deshonestas, desde varios bloques atrás, alcance a la honesta puede modelarse mediante el problema de la ruina del jugador: supongamos que el jugador cuenta con crédito ilimitado ( $a = \infty$ ), que empieza desde una posición de déficit (se sitúa varios bloques atrás de la cadena honesta), y que cuenta con un infinito número de ocasiones para tratar de cancelar su déficit (alcanzar la cadena honesta). Denominando

- $p$  = probabilidad de éxito (que un nodo honesto añada el bloque siguiente)
- $q$  = probabilidad de fracaso (que el atacante añada el bloque siguiente); por definición, tendremos  $p = 1 - q$
- $q_z$  = probabilidad de que el atacante alcance a la cadena honesta desde  $z$  bloques detrás

la solución obtenida para el problema en los términos planteados (ver [7], págs. 342-348) es:

$$q_z = \begin{cases} 1, & p \leq q \\ (q/p)^z, & p > q \end{cases} \quad (1)$$

Hagamos una interpretación más informal de este resultado. Si  $p < q$ , es decir, si la probabilidad de que un nodo honesto resuelva la *proof-of-work* es menor que la probabilidad de que la resuelva el atacante, entonces más tarde o más temprano el nodo deshonesto tendrá éxito, alcanzará a la cadena honesta, y podrá suplantar las transacciones correctas por las que le beneficien.

Para el caso  $p = q$  quizá no sea tan evidente el resultado, pero debemos tomar en consideración que  $a = \infty$  y que no hemos puesto límites al déficit (no existen barreras de absorción en el camino). El que la probabilidad matemática  $q_z$  sea 1 nos indica que disponiendo de infinitas pruebas más tarde o más temprano la cadena deshonestas alcanzará a la honesta. En términos más prácticos y con las lógicas limitaciones temporales y físicas, este resultado podría alcanzarse, o no.

Por último, con  $p > q$  la probabilidad  $q_z = (q/p)^z$  de que la cadena deshonestas alcance a la honesta cae exponencialmente según el número de bloques de ventaja  $z$  que lleve la cadena honesta con respecto a la deshonestas.

En conclusión, salvo que el atacante pueda aportar una potencia computacional semejante a la del resto de nodos honestos de la red, será estadísticamente improbable que tenga éxito en sus ataques. Y aunque consiguiera igualar esa potencia, todavía tendría que salvar con dificultades la distancia que separe a ambas cadenas que, para repetir, mayor fortaleza ofrece cuanto mayor sea esa distancia.

Por otra parte, un segundo y último cálculo nos aporta la solución al problema de cuanto tiempo tendrá que esperar el beneficiario de una transacción para estar razonablemente seguro de que el transmitente no ha incurrido en un doble gasto. Por su falta de detalles es quizá el más oscuro de los puntos tratados por Nakamoto en su paper, y seguiremos el desarrollo de Ozisik y Levine dado en su muy reciente trabajo (enero 2017) [8].

Nakamoto parte de que el beneficiario de una transacción se encontrará confortable con su legitimidad tras conocer que se han añadido a la cadena de bloques  $z$  bloques detrás del que registra la transacción en cuestión. Inmediatamente después asevera que, asumiendo que los nodos honestos tardan el tiempo esperado medio en validar un bloque, el progreso potencial de un posible nodo deshonesto seguirá una distribución de Poisson con valor esperado

$$\lambda = z \frac{q}{p}$$

donde  $z$  acaba de ser definida y  $p$  y  $q$ , con  $p = 1 - q$ , son las mismas probabilidades que hemos visto en el cálculo anterior (probabilidad de que sea un nodo honesto o uno

deshonesto el que encuentre, respectivamente, el siguiente bloque). ¿De dónde se obtiene tal expresión?

Sabemos que una distribución de Poisson modela el número de eventos (o éxitos) en estudio que ocurren en un determinado intervalo de tiempo. Viene definida por un único parámetro, su media  $\lambda$  (que también coincide con su varianza), y que representa el número de veces que se *espera* suceda el evento por intervalo de tiempo.

En nuestro contexto, estamos indagando sobre el número de bloques que el atacante puede validar en un intervalo de tiempo. De esta manera, nuestro evento a considerar es la obtención de un nuevo nodo a añadir por el atacante a *su* cadena. En cuanto al intervalo de tiempo a contemplar, tenemos que considerar que con toda la potencia computacional trabajando honestamente se obtiene un nuevo bloque cada intervalo de tiempo  $T = 10$  minutos. Si la red se particiona en un conjunto de nodos honestos y otro de deshonestos, los honestos serán capaces de habilitar  $p$  bloques nuevos cada  $T$  minutos. Por tanto, el intervalo de tiempo que el beneficiario debe esperar para que se añadan  $z$  bloques tras el bloque que contiene su transacción será de  $z \cdot T/p$  minutos. Siguiendo el trabajo de Ozisik y Levine:

$$z \text{ bloques} \cdot \frac{T \text{ minutos}}{p \text{ bloques}} = z \frac{T}{p} \text{ minutos}$$

Por su parte, el atacante descubre  $q$  bloques cada  $T$  minutos,  $q/T$ . Considerando el intervalo de tiempo que para habilitar  $z$  bloques acabamos de determinar, el atacante generará bloques a un ritmo (medio) de

$$\frac{zT}{p} \cdot \frac{q}{T} = z \frac{q}{p}$$

bloques por intervalo. No olvidemos que este ritmo medio corresponde a la media esperada de que suceda (el evento de) la generación de un nuevo bloque por parte del atacante y por intervalo de tiempo, que es nuestra  $\lambda$  de la distribución de Poisson. De ahí la expresión directa de Nakamoto,

$$\lambda = z \frac{q}{p}$$

con la que abre este segundo cálculo.

Por otra parte, la conocida función de probabilidad de la distribución de Poisson es

$$P(X = k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

que nos ofrece la probabilidad de que  $X = k$  ( $k \geq 0$ ) eventos (que sabemos corresponden a que el atacante valide un nuevo bloque) ocurran durante el intervalo de tiempo implícito en  $\lambda$ .

Generalizando la cuestión, y puesto que el beneficiario esperará por  $z$  bloques para sentirse seguro sobre la transacción, podremos preguntarnos sobre cuál es la

probabilidad de que el atacante pueda generar más bloques que los nodos honestos en ese punto o en el futuro. Con la variable aleatoria  $X$  representando el número de bloques que el atacante valida en el intervalo de tiempo en el que los mineros honestos validan  $z$  bloques, y sabiendo que la probabilidad de alcanzar la cadena honesta por el atacante desde  $z - k$  bloques atrás es  $q_{z-k}$  (ver el cálculo anterior), la probabilidad total  $Q$  de que el nodo deshonesto alcance la cadena corresponderá a la suma de todas las posibles  $X$ :

$$\begin{aligned} Q &= P(X = 0; \lambda)q_z + P(X = 1; \lambda)q_{z-1} + \dots \\ &= \sum_{k=0}^{\infty} P(X = k; \lambda)q_{z-k} \\ &= \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} (q_{z-k}) \end{aligned}$$

Puesto que con  $k > z$  (es decir, la cadena del atacante supera a la honesta) tendremos que la probabilidad  $q_{<0} = 1$ , entonces:

$$Q = \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{z-k}, & k \leq z \\ 1, & k > z \end{cases}$$

resultado que Nakamoto consigna directamente y que corresponde a la multiplicación de la función de probabilidad por la probabilidad de que el atacante pueda alcanzar la cadena honesta desde  $z - k$  bloques atrás.

Para terminar este último cálculo, el autor simplifica la expresión anterior utilizando el resultado de que 1 menos la probabilidad de que algo no pase es la probabilidad de que algo pase. De esta manera, si trabajamos con la probabilidad de que el atacante *no* pueda alcanzar la cadena honesta desde  $z - k$  bloques atrás, es decir,  $1 - (q/p)^{z-k}$ , entonces a partir de la última expresión obtenida tendremos ahora:

$$\begin{aligned} Q &= 1 - \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} 1 - (q/p)^{z-k}, & k \leq z \\ 1 - 1, & k > z \end{cases} \\ &= 1 - \sum_{k=0}^z \frac{\lambda^k e^{-\lambda}}{k!} \cdot (1 - (q/p)^{z-k}) + \sum_{k=z+1}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot 0 \\ &= 1 - \sum_{k=0}^z \frac{\lambda^k e^{-\lambda}}{k!} \cdot (1 - (q/p)^{z-k}) \end{aligned}$$

resultado que corresponde a la expresión final que el *paper* registra sobre este tema, y de la que en última instancia ofrece una versión en lenguaje C y algunos resultados numéricos obtenidos que confirman que la probabilidad de que el atacante alcance a la cadena honesta decae exponencialmente conforme  $z$  se incrementa. Con estos resultados es prácticamente directo obtener la respuesta buscada a cuántos bloques deberá esperar un beneficiario que se construyan sobre el que contiene la transacción que le adjudica el valor, fijando para ello un máximo a la probabilidad de que el



atacante tenga éxito (digamos, por ejemplo, que  $Q < 0,001$  para cualquier valor de  $q$  a analizar, lo que nos dará la cantidad de bloques  $z$  que lo cumpliría).

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

$q=0.10$	$z=5$
$q=0.15$	$z=8$
$q=0.20$	$z=11$
$q=0.25$	$z=15$
$q=0.30$	$z=24$
$q=0.35$	$z=41$
$q=0.40$	$z=89$
$q=0.45$	$z=340$

Fig. 2.9: código en lenguaje C y cálculo del número de bloques,  $z$ , requeridos según valores de  $q$ , para que  $Q < 0,001$  (S. Nakamoto)

Llegados a este punto, procede complementar lo anterior con la puntualización del trabajo de Ozisik y Levine sobre el desarrollo de Nakamoto, del que ven erróneas las referencias a la captura de la bifurcación honesta por la deshonesto con la construcción de  $z$  bloques, puesto que el fin último del atacante no sería esta equiparación sino su superación. En opinión de estos autores, que resulta razonable, toda la argumentación debería reconstruirse modificando las referencias a  $z$  por  $z + 1$ , es decir, que la cadena atacante se haga más larga, en definitiva, que la cadena construida por los nodos honestos.

## 2.9 Conclusiones

Nakamoto concluye que su diseño se fundamenta en el marco de trabajo para moneda y transacciones electrónicas basado en firmas digitales, ya usual por aquel entonces, y que proporcionaba una asignación efectiva de la propiedad pero que no resolvía satisfactoriamente y de forma descentralizada el problema del doble gasto. Éste queda resuelto con el análisis que propone y que hemos revisado en este capítulo. Y este diseño dará las claves para que el mecanismo de dinero virtual propuesto bajo Bitcoin se perfeccione y perviva hasta la actualidad, al contrario que el resto de diseños previos disueltos en su propia ineficacia.

En particular, podemos distinguir las siguientes estrategias interrelacionadas como las principales colaboradoras al éxito de esta criptomoneda:

- La inexistencia de una autoridad central que responda del buen fin de las transacciones y de la coherencia del sistema. Esta autoridad central se sustituye por una *autoridad distribuida* entre los diferentes nodos *peer-to-peer* que dan servicio y soportan todo el sistema Bitcoin.
- La infraestructura soporte de las transferencias de valor, o transacciones, y de la coherencia en el estado del sistema resultante tras las mismas, que debe certificar la propiedad actualizada del valor transferido y garantizar su potencial movilización únicamente por su propietario último. Esta infraestructura toma la forma de cadena de bloques ordenados cronológicamente, que almacenan individualmente las

transacciones realizadas en un intervalo de tiempo predeterminado (unos diez minutos en la actual implementación de Bitcoin).

- El mecanismo de recompensas a los nodos que colaboran en la gestión y mantenimiento de la cadena de bloques, que por un lado validan que las transacciones incluidas en cada bloque sean legítimas e impiden la posibilidad del doble gasto; y, por otro, enlazan todo nuevo bloque con la cadena de bloques que le preceden temporalmente tras realizar un importante trabajo computacional (*proof-of-work*), lo que dificulta la falsificación del trazo de la cadena de bloques y de su contenido (y lo hace todavía más conforme se van añadiendo bloques -acumulando trabajo computacional- a la cadena), facilitando al mismo tiempo el necesario consenso sobre el correcto estado del sistema. La remuneración de este trabajo debe sobrepasar la mera compensación por el gasto en energía y equipos de los nodos (*mineros*) que lo realizan, garantizando así la continuidad en el mantenimiento de todo el sistema.

Además, y también explicitado en el trabajo de Nakamoto, se resuelve el problema de la naturaleza reversible de los pagos online que se realizan a través de entidades financieras, que normalmente dejan al vendedor en una cierta posición de debilidad inherente a los mecanismos de solución de disputas en las que con cierta habitualidad deben intermediar las entidades financieras. Sería posible abrir ahora una motivada discusión sobre el traspaso de la posición de debilidad a los compradores ante transacciones que no admiten reversibilidad en caso de errores o fraudes, pero se alejaría de nuestros objetivos ahora y, por tanto, nos conformaremos con considerar ésta como una característica consustancial a la naturaleza de la Blockchain original, que configuraría así lo que se denomina una *hard electronic currency* (en contraposición a una *soft electronic currency* que, como las tarjetas de crédito o PayPal, sí admiten reversión en los pagos). Volveremos necesariamente a la cuestión de la reversibilidad cuando analicemos la aplicación de una cadena de bloques bajo regulación.

Puesto que hemos seguido la publicación de Nakamoto centrándonos en el literal de su trabajo, en este capítulo no hemos hablado de direcciones, apenas de claves, y no hemos profundizado en temas como la construcción de transacciones o la implementación de los circuitos de transferencia de valor. Ni siquiera hemos hecho referencia a Blockchain como tecnología puesto que no se identifica como tal en el texto del autor, si bien está implícita en todo lo que atañe a la gestión de los bloques contenedores de transacciones. Emplazamos todo esto para un poco más adelante, ya que antes necesitaremos una caja de herramientas criptográficas para fundamentar todo lo que vendrá a continuación.

## *Primitivas criptográficas en Bitcoin/Blockchain*

Podría resultar un tanto paradójico considerar que Blockchain está fuertemente fundamentada en algunos elementos y procesos criptográficos y, simultáneamente, que su contenido no está sometido a ningún tipo de encriptación y es por lo tanto de acceso público y directo. No existe tal paradoja:

- Ya hemos revisado la definición de una moneda electrónica como una cadena de firmas digitales, para las que es fundamental el concepto de criptografía de clave pública, pero no el de encriptación puesto que su contenido último debe ser por definición, y en general, texto claro.
- Por otra parte, conocemos la necesidad de la herramienta auxiliar que suponen las funciones *hash* criptográficas para una óptima implementación de las firmas digitales.
- Por último, también hemos visto que estas funciones hash desempeñan el principal papel en el desarrollo de la *proof-of-work* tal y como se implementa en la blockchain original soporte de Bitcoin.

En resumen, el diseño de Bitcoin no incluye encriptación segura de contenido alguno en la cadena de bloques. Ya hemos tratado la versión de privacidad de Nakamoto según la cual se tiene acceso público a las transacciones, cuyo contenido se registra en texto claro, y que deja a salvo la identidad de sus participantes mediante la referencia a sus respectivas claves públicas, de las que no debería existir correlación directa con sus identidades físicas respectivas.

Antes de entrar en profundidad en la tecnología Blockchain, lo que haremos en los capítulos posteriores, revisaremos los elementos o primitivas criptográficas sobre los que está construida buena parte de su infraestructura. Directamente vinculadas a los puntos que acabamos de repasar, las primitivas criptográficas que deberemos tomar en consideración son las funciones *hash*, la criptografía de clave pública o asimétrica (PKC, *Public Key Cryptography*), y la firma digital, así como las implementaciones de alguna de estas primitivas mediante criptografía de curva elíptica, particularmente la ECDSA (*Elliptic Curve Digital Signature Algorithm*) puesto que fue la elegida para Blockchain.

Revisaremos a continuación las funciones *hash* para trabajar luego la criptografía de clave pública y de curva elíptica, hasta llegar a la ECDSA hacia el final del capítulo.

### **3.1 Funciones de hash y funciones hash criptográficas**

Cuando nos encontramos en ocasiones ante una información extensa que hay que tratar, y según la finalidad buscada, puede resultar conveniente hacerlo sobre un resumen o

versión mucho más corta de la misma. Por ejemplo, los mecanismos de firma digital que veremos algo más adelante suponen siempre un cierto coste computacional que es conveniente minimizar reduciendo el tamaño de la entrada, lo que reduce también, como resultado normalmente deseable, el tamaño final de la firma. Como solución a estos inconvenientes se suele trabajar con una versión reducida de la información a firmar, que se consigue normalmente con las denominadas *funciones hash*.

Una función *hash* es una función cuya entrada consiste en una cadena de texto (o, equivalentemente, de bits en última instancia), o mensaje  $m$ , de longitud arbitraria, y que ofrece como resultado otra cadena de texto  $m'$  (bits en última instancia) de longitud determinada y previamente fijada  $n$ . Algo más formalmente, y en términos de cadenas de bits como elementos últimos manejados en computación:

$$h: \{0,1\}^* \rightarrow \{0,1\}^n, m \mapsto h(m) = m'$$

La longitud  $n$  de la cadena  $m'$  resultante, también denominada *hash* o *resumen*, se encuentra habitualmente entre los 128 y los 512 bits. Generalmente, y sobre todo en el contexto de firma digital,  $h(m)$  es de tamaño inferior al de  $m$ , aunque este último es de magnitud arbitraria y por tanto puede ser reducido, como sucede usualmente en el caso de las *passwords*, de las que normalmente se almacenan sus *hashes* como método de confirmación.

Aparte de estas características genéricas, en criptografía nos interesan las funciones *hash* que satisfagan los siguientes requisitos:

- Que sean unidireccionales, es decir, que dado un *hash* o resumen  $m' \in \{0,1\}^n$ , sea computacionalmente *intratable*<sup>5</sup> obtener el mensaje  $m$  tal que  $h(m) = m'$ . Esta característica también se denomina *resistencia a la primera preimagen*.
- Una consecuencia del punto anterior es que, si se cumple tal irreversibilidad, entonces la función hash cumple propiedades de encriptación no reversible, lo que es utilizado para el almacenamiento de *passwords* encriptadas por su *hash* de tal manera que ni los administradores del sistema tengan acceso al texto claro de las mismas, dejando al sistema la comprobación de la corrección de la contraseña computando su hash y comparándolo con el almacenado.
- Que dado  $m$ , sea computacionalmente intratable encontrar otro mensaje  $m_2$ , con  $m \neq m_2$ , y tal que  $h(m) = h(m_2)$ . Esta característica se denomina *resistencia a la segunda preimagen*.
- Generalizando el punto anterior (con la diferencia sutil de que ahora no exista un mensaje previo), que no se puedan encontrar  $m$  y  $m_2$ , también con  $m \neq m_2$  y

---

<sup>5</sup> Diremos que un problema es *computacionalmente intratable*, o simplemente *intratable*, si puede resolverse en tiempo finito pero con un plazo de resolución que no es útil por su excesiva extensión. Algo más formalmente, para un problema intratable no existe un algoritmo que pueda resolverlo con una máquina de Turing determinista (esto es, en la que solo una acción o regla puede ejecutarse una vez se ha llegado a un determinado estado) en tiempo polinómico (es decir, en una potencia del tamaño  $n$  de la entrada del problema, por ejemplo  $n^3$ ), mientras que puede ser resuelto en tiempo exponencial (es decir, utilizando la entrada  $n$  como potencia, como en  $2^n$ , por ejemplo). En este contexto hablamos en general del caso peor, ya que existen algoritmos prácticos de tiempo exponencial ante una magnitud de entrada razonable. En teoría de complejidad computacional, los problemas *tratables*, es decir, problemas para los que existe un algoritmo determinista de complejidad temporal polinómica, se denominan como de *clase P*. Ver Anexo A para definiciones formales de estos conceptos, que pueden ampliarse en, por ejemplo, [20].

tales que  $h(m) = h(m_2)$ , lo que se explicita diciendo que la función *hash*  $h$  es *resistente a la colisión*.

Se puede demostrar que estos requisitos están interrelacionados (ver Delfs y Knebl, *Introduction to Cryptography*, págs. 31-32), de tal manera que podemos decir de una función *hash* que es una *función hash criptográfica* si es resistente a la colisión, puesto que si podemos establecer esto último entonces la propiedad de unidireccionalidad está garantizada como consecuencia de tal resistencia.

Para que una función *hash* sea eficaz exigiremos que sea determinista (una entrada debe producir la misma salida cada vez que se le aplique  $h$ ), que el algoritmo de la función ofrezca un cómputo rápido, que no ofrezca la característica de continuidad (es decir, que pequeños cambios en la entrada no supongan pequeños cambios en la salida, sino sustanciales modificaciones, o *efecto avalancha*), y que asegure una importante uniformidad en los resultados dentro de su posible rango (que cada elemento del conjunto posible de resultados, que será determinado por la longitud del resumen, pueda obtenerse con igual o muy semejante probabilidad), entre otras características deseables.

Algunas de las funciones *hash* más conocidas, y entre las que encontraremos las utilizadas en la actual concepción de Blockchain, corresponden a las familias SHA (*Secure Hash Algorithm*), en particular SHA-256, SHA-512 [9], o la más reciente gama de funciones SHA3 [10]; o RIPEMD (*RACE Integrity Primitives Evaluation Message Digest*), en particular RIPEMD-160 [11].

Veamos algunos ejemplos de cómo actúan en la práctica este tipo de funciones. Para las cadenas de texto encerradas entre comillas simples, y mediante la aplicación de la función *hash* SHA-256, obtenemos los siguientes resúmenes (en hexadecimal):

```
x:~ aisasa$ echo -n 'hola' | openssl sha256
(stdin)= b221d9dbb083a7f33428d7c2a3c3198ae925614d70210e28716ccaa7cd4ddb79

x:~ aisasa$ echo -n 'hola,' | openssl sha256
(stdin)= 76f32065e15369f51bd82b6aee6f6a3785a93647c1b1a9331f5f10980bef27

x:~ aisasa$ echo -n 'hola, mundo' | openssl sha256
(stdin)= cb4c2cde839d658f3b93ed469b60014f0aec42cd1179b7585cf36735135ee010
```

Es apreciable la importante variación de los resultados para mínimas variaciones en el texto (la adición, por ejemplo, de una simple coma entre el primer y el segundo textos), fruto de las características ya comentadas de las funciones *hash* criptográficas. En cuanto al tamaño del *hash* resultante, comprobamos su independencia del texto de entrada, tanto para las reducidas entradas del ejemplo anterior como para cadenas de mayor dimensión:

```
x:~ aisasa$ echo -n 'Con diez cañones por banda, viento en popa, a toda vela,
no corta el mar sino vuela un velero bergantín. Bajel pirata que llaman, por
su bravura, "El Temido", en todo el mar conocido, del uno al otro confín' |
openssl sha256
(stdin)= adace8d8f2416b48bdb7a8c8764eb0b14fcf698f1abc9b4b0fb67cfb61540169
```

Mientras que SHA-256 ofrece resúmenes de 64 caracteres hexadecimales (32 bytes, o 256 bits), otras funciones componen salidas de mayor longitud:

```
x:~ aisasa$ echo -n 'hola, mundo' | openssl sha512
(stdin)= 6d4c6652d27c8431e7d15d1701f410728fb202256c184a831f99eb23d03d2644c3abd850adbb9
8296cca107387097807fff28276b9b80facc7287723e882b9af
```

SHA-512, como vemos en este último ejemplo, resulta en 128 caracteres hexadecimales, lo que hace un total de 64 bytes o 512 bits. RIPEMD-160, por su parte, ofrece una longitud de *hash* de 160 bits (40 caracteres hexadecimales):

```
x:~ aisasa$ echo -n 'hola, mundo' | openssl ripemd160
(stdin)= c3cce70344032a7f0a087a5167ffa3a75a02523d
```

En general, una función *hash* trabaja sobre los bits de la información a resumir, normalmente mediante tres acciones combinadas: la división del input en unidades de trabajo más pequeñas o bloques; el tratamiento del contenido en bits de cada una de los bloques, secuencialmente, mediante su combinación por diferentes operaciones lógicas intra e interbloques; y la compresión de los resultados intermedios o finales. En la mayoría de las funciones *hash* encontramos las siguientes operaciones que pueden realizarse sobre cadenas de bits:

- *AND* lógico (con símbolo usual  $\wedge$ )
- *OR* lógico ( $\vee$ )
- *XOR* lógico ( $\oplus$ )
- *NOT* lógico ( $\neg$ )
- Adición entre palabras (+), usualmente de 32 o 64 bits, *mod*  $2^{32}$  o *mod*  $2^{64}$
- $SHR^n$ ,  $SHL^n$ , o desplazamiento de  $n$  bits a la derecha o a la izquierda, respectivamente
- $ROTR^n$ ,  $ROTL^n$ , o rotación (desplazamiento circular) de  $n$  bits a la derecha o a la izquierda, respectivamente

Como ejemplo, veamos el proceso de obtención del resultado *hash* mediante SHA-256, que puede resumirse en los pasos siguientes:

1. Extensión del mensaje a resumir hasta que su longitud en bits sea múltiplo de 512, lo que permitirá identificar  $N$  bloques de 512 bits cada uno. La extensión a añadir se compone de un 1 inicial, y un entero final de 64 bits que representa la longitud original del mensaje. Entre el 1 inicial y el entero final se coloca la cantidad necesaria de ceros para que la longitud resultante del mensaje final,  $N \cdot 512$ , sea efectivamente múltiplo de 512.
2. Inicialización de las ocho *palabras* de 32 bits cada una,  $H_0$  a  $H_7$ , que posteriormente SHA-256 ofrecerá como resultado final. Este resultado conformará la salida de 256 bits que hemos visto en ejemplo anterior. La inicialización se realiza con determinados valores de origen aleatorio (los primeros 32 bits de la parte decimal de la raíz cuadrada de los primeros ocho números primos). Como veremos, tras su inicialización estas mismas palabras serán modificadas conforme se procese secuencialmente cada uno de los  $N$  bloques. También se inicializa un conjunto de constantes  $K = \{K_0, \dots, K_{63}\}$  con valores correspondientes a los primeros 32 bits de la parte decimal de la raíz

- cúbica de los 64 primeros números primos, y que no se verán modificados a lo largo de todo el algoritmo.
3. A partir de aquí, y tomando secuencialmente cada uno de los  $N$  bloques en los que se ha dividido el mensaje, el algoritmo realiza lo siguiente para cada bloque:
    - I. Inicializa ocho variables  $A, B, C, D, E, F, G$  y  $H$  con los valores de  $H_0$  a  $H_7$ , respectivamente, bien sean los iniciales en caso de tratamiento del primer bloque, bien los valores intermedios que veremos se van obteniendo conforme se desarrolla el procedimiento.
    - II. En lo que se denomina *función de compresión del algoritmo*, éste toma el primero de los bloques de 512 bits identificados en el mensaje a resumir y lo divide en 16 palabras  $W_0, \dots, W_{15}$  de 32 bits cada una, a las que añade otras 48 palabras  $W_{16}, \dots, W_{63}$  cuyo valor se calcula de acuerdo al contenido del bloque, y constituyendo así el conjunto expandido  $W = \{W_0, \dots, W_{63}\}$  del bloque. En un bucle de 64 ejecuciones, va tomando sucesivamente cada uno de los elementos de  $W$ , y en función de su valor; de los valores de las ocho variables  $A, \dots, H$  inicializadas en el punto anterior I.; del conjunto de constantes prefijadas e invariables  $K_0, \dots, K_{63}$ , del que cada uno de sus elementos también va siendo tomado de manera respectiva en cada una de las ejecuciones del bucle; y de una serie de funciones construidas por las operaciones a nivel bit ya comentadas, va obteniendo una serie de nuevos valores para las variables  $A, \dots, H$ . Nótese la conversión de los 512 bits iniciales del bloque a los 256 bits del conjunto de las ocho palabras de 32 bits.
    - III. Una vez terminado el bucle del paso anterior, para cada una de las palabras  $H_0$  a  $H_7$  se obtendrá su nuevo valor sumando a su valor previo el obtenido para su respectiva variable  $A, \dots, H$ . Es decir, y por ejemplo para  $H_0$ , tendremos  $H_0^i = H_0^{i-1} + A$ . El significado de *compresión* en este contexto es más amplio que la reducción de los 512 bits iniciales de cada bloque a los 256 del resultado, ya que puede extenderse a la progresiva acumulación de los bloques del mensaje en un único conjunto limitado de bits.
  4. Tras el procesado del último bloque, se concatena el contenido de las palabras  $H_0$  a  $H_7$  y se devuelve como resultado resumen.

El paso 3., y en particular los procedimientos del punto II. que conforman la denominada función de compresión de esta función *hash*, es el núcleo de SHA-256. Supone aplicar determinadas sumas *mod*  $2^{32}$  y hasta seis funciones que operan a nivel bit definidas de la manera siguiente:

$$\begin{aligned}
 Ch(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
 Ma(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
 \Sigma_0(x) &= ROTR^2 \oplus ROTR^{13} \oplus ROTR^{22} \\
 \Sigma_1(x) &= ROTR^6 \oplus ROTR^{11} \oplus ROTR^{25} \\
 \sigma_0(x) &= ROTR^7 \oplus ROTR^{18} \oplus ROTR^3 \\
 \sigma_1(x) &= ROTR^{17} \oplus ROTR^{19} \oplus ROTR^{10}
 \end{aligned}$$

Las dos últimas funciones  $\sigma$  se utilizan para calcular los elementos  $W_{16}, \dots, W_{63}$  correspondientes a la expansión de cada bloque, mientras que las cuatro primeras se aplican en las iteraciones de la función de compresión propiamente dicha. De esta manera, considerando como entradas la adición *mod*  $2^{32}$  del elemento  $t$  de cada uno de los conjuntos  $W$  y  $K$ , así como los resultados de la iteración previa, tendremos el

siguiente esquema de la función de compresión para cada una de las ejecuciones del bucle descrito en el punto II. del paso 3. del algoritmo descrito, en el que puede apreciarse cómo se calcula el nuevo valor de cada una de las variables  $A, \dots, H$ :

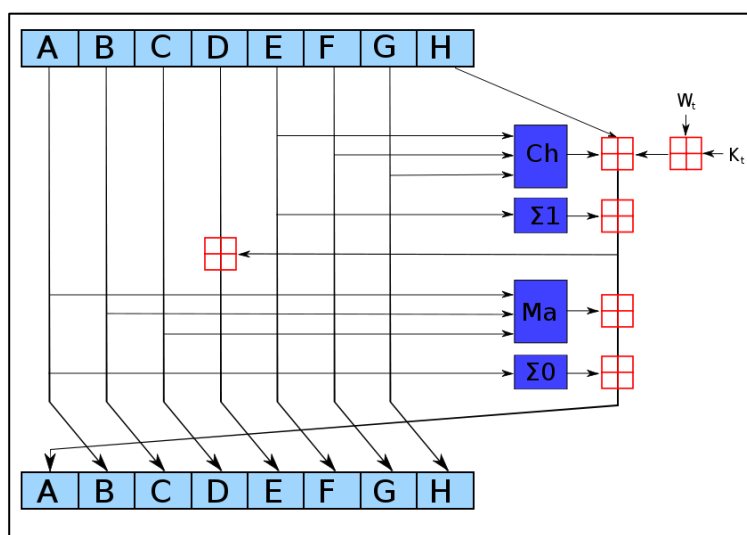


Fig. 3.1: operaciones de la iteración  $t$  en SHA-256 (es.wikipedia.org)

Estas funciones *hash* constituyen herramientas de uso muy extendido en diversas áreas de la computación. Ya hemos comentado su aplicación intensiva en los algoritmos de firma digital, a lo que podríamos añadir su utilización en la verificación de contraseñas (que no serían almacenadas de forma literal en los sistemas que las requieran, ya que solo necesitarían registrar sus respectivos *hashes*), en los resúmenes de archivos para posterior verificación de su integridad, en la identificación e indización de datos por su contenido, y en unas cuantas aplicaciones más. En particular, nos interesa un uso adicional relevante para nuestros propósitos como es el de la medida de un trabajo computacional como prueba de que, en efecto, tal trabajo haya sido realizado.

### 3.2 Funciones hash y proof-of-work

En el capítulo anterior nos referíamos a la *proof-of-work* como el mecanismo que solucionaba el problema del marcaje de tiempo sobre un conjunto de transacciones en un contexto distribuido, sin autoridad central certificadora. Veremos en este apartado cómo se implementa esta prueba de trabajo en Bitcoin/Blockchain mediante una función *hash* criptográfica.

Para la inclusión de cada bloque de transacciones en la cadena final, la estrategia consiste en exigir un importante trabajo computacional *demostrable*, y que si algún nodo deshonesto intente falsificar la cadena para favorecer sus intereses se encuentre con que tenga que repetir ese trabajo no solo para el bloque en cuestión, sino para todo bloque añadido sobre el mismo.

El trabajo computacional se reparte entre los miles y miles de nodos mineros, y uno de ellos descubrirá aleatoriamente la combinación que permitirá enlazar el bloque en construcción a la cadena de bloques, lo que validará las transacciones contenidas. Como sabemos, el descubrimiento de esta combinación y la correspondiente incorporación del



bloque a la cadena se realizan cada diez minutos, aproximadamente, en la implementación actual de Bitcoin. Es fácil apreciar que todo este trabajo realizado por el extenso colectivo de mineros no es fácil de replicar por un solo nodo, a no ser, como también hemos visto al final del capítulo anterior, que disponga de una potencia computacional de magnitud mínima equiparable a la de la mitad del resto de mineros. Pero, ¿de qué trabajo computacional estamos hablando?

Cada bloque necesita un *hash* particular para ser añadido a la cadena de bloques. Este *hash* se obtiene, como también vimos, de una entrada compuesta por la concatenación del *hash* del bloque anterior, del contenido (el conjunto de transacciones) del bloque que se está trabajando, y de un campo denominado *nonce* cuyo contenido se va variando hasta obtener un resultado resumen que debe cumplir determinados requisitos. De forma resumida, Nakamoto nos ofrecía en su trabajo el siguiente esquema, en el que los tres elementos que acabamos de ver conforman la entrada de la función *hash* cuyo resultado, función de los componentes vistos y que servirá a su vez como componente del siguiente bloque, enlazará cada bloque de la cadena:

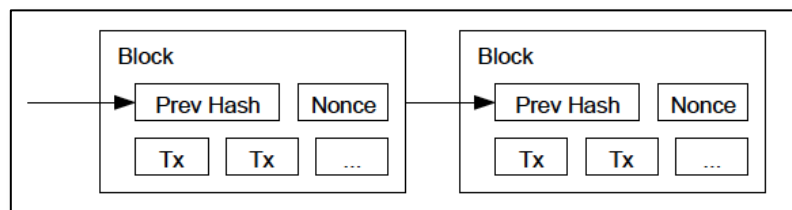


Fig. 3.2: estructura básica de los bloques de transacciones en Bitcoin  
(S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*)

El núcleo de este esfuerzo computacional se encuentra en la particularidad o requisito que debe cumplir el *hash* a obtener por este mecanismo: este resumen, que como todo resultado de una función *hash* debe contener un número predeterminado de bits, debe comenzar por un número determinado de sus bits a cero. Cuanto mayor sea este número, mayor será el trabajo computacional exigido, y para mantener más o menos fijo el periodo de 10 minutos entre nodo y nodo podrá elevarse su complejidad (elevar el número de ceros iniciales requeridos) conforme se incremente la potencia computacional de toda la red. Y a la inversa, si bien esta reducción de la complejidad no es lo usual ante el prácticamente constante incremento en la capacidad computacional de la red Bitcoin. En la Blockchain de esta criptomoneda la complejidad se ajusta cada 2.016 bloques (unos 14 días), de acuerdo al tiempo medio de resolución de los mismos.

El *hash* de cada bloque contiene 256 bits, y la función *hash* utilizada normalmente en Bitcoin para este menester es SHA-256. La complejidad del problema, sobre la que volveremos cuando analicemos los detalles del minado de bloques, supone descubrir un resumen cuyos 68 bits iniciales sean ceros en el momento de redactar estas líneas. Puesto que el esfuerzo computacional requerido es exponencial conforme exigimos más bits iniciales a cero, podemos intuir la evolución de la complejidad desde la exigencia inicial en 2009 de 40 bits iniciales nulos en cada *hash* en comparación con los 68 actuales.

Como ejemplo, veamos las características del primer bloque de la cadena en Bitcoin, el bloque *Génesis* o número 0:



Block #0			
BlockHash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f 			
Summary			
Number Of Transactions	1	Difficulty	1
Height	0 (Mainchain)	Bits	1d00ffff
Block Reward	50 BTC	Size (bytes)	285
Timestamp	Jan 3, 2009 6:15:05 PM	Version	1
Mined by		Nonce	2083236893
Merkle Root	 4a5e1e4baab89f3a32518a8...	Next Block	1

Fig. 3.3: parámetros del bloque *Génesis*, o número 0, de la blockchain de Bitcoin (blockexplorer.com)

Podemos confirmar que el campo *BlockHash* comienza con diez dígitos hexadecimales (5 bytes, o 40 bits), así como el contenido del *nonce* que, buscado de forma aleatoria, conformaba de tal manera la entrada de la función *hash* que su resultado se ajustó al número mínimo de bits a cero buscados.

En términos numéricos, la probabilidad de obtener aleatoriamente un 0 como contenido de un bit es de  $1/2 = 0.5$ . Para calcular la probabilidad de que  $n$  bits consecutivos sean todos cero, tendremos que multiplicar las probabilidades individuales tantas veces como bits. Para los 68 bits que se exigen ahora obtenemos una probabilidad de

```
>> 0.5^68
ans =
    3.3881e-21
```

cuya magnitud es absolutamente marginal como para tener éxito en el minado bajo un ordenador estándar. Si se pretende obtener algún resultado, esta minúscula probabilidad obliga a la utilización de potente hardware dedicado y a la participación en algún *pool de minado* que acumule la potencia computacional de (y divida la posible ganancia en) multitud de partícipes.

Dadas las dimensiones y la actual capacidad de toda la red Bitcoin, a la fecha de redacción de este trabajo (mayo 2017) ésta ha llegado a los 4 millones de billones de operaciones de hash por segundo (o 4 millones de *terahashes* por segundo, TH/s), cantidad inabarcable para un nodo individual pero no excesivamente lejana de la potencia computacional agregada de cada uno de los principales *pools* mineros existentes. Así, el poder computacional de la red ha pasado de unos 1,5 millones de TH/s en junio de 2016 a los 4 millones TH/s en mayo 2017, con picos de hasta casi 4,5 millones TH/s:

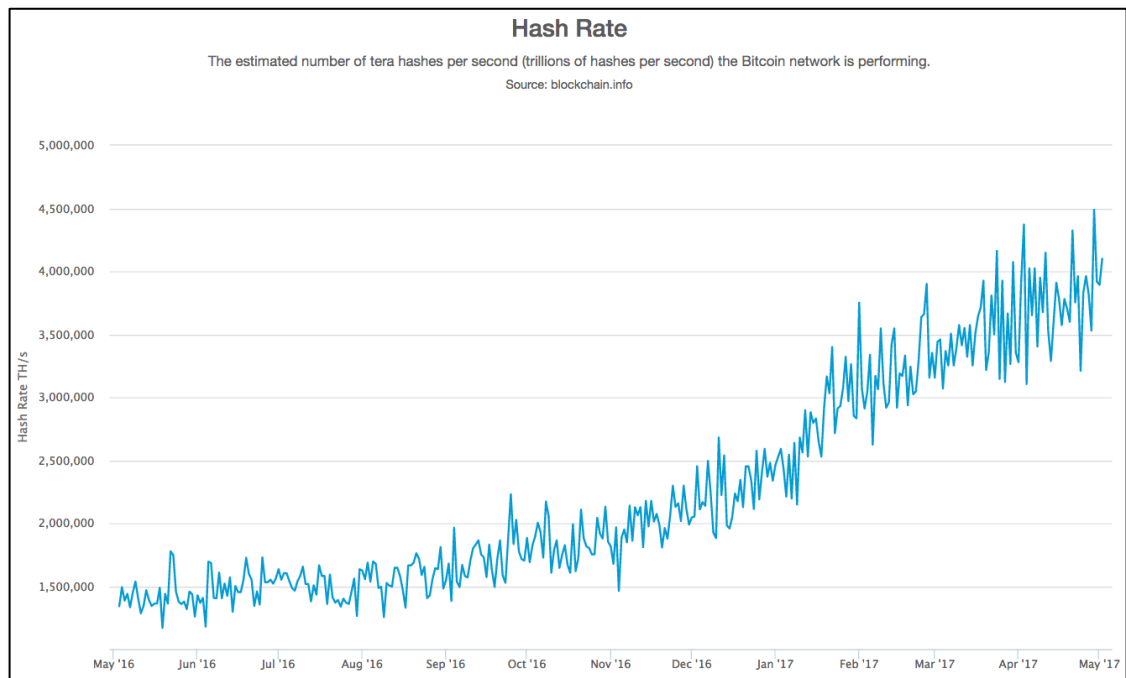


Fig. 3.4: variación en los últimos meses de la tasa de *hash* en *terahashes* por segundo (blockchain.info)

Es de prever que este crecimiento se mantenga mientras también así lo haga el número de transacciones ejecutadas, puesto que este incremento vendrá acompañado por el del número y volumen de *fees* entregados a los mineros en cada bloque validado y añadido a la cadena.

### 3.3 Criptografía de clave pública

Hasta la segunda mitad de la década de los 70 del siglo pasado, la criptografía se fundamentaba en los sistemas de clave simétrica, es decir, que la misma clave con la que se cifraba una información servía para descifrarla. Eso suponía que las claves secretas debían compartirse entre, al menos, dos usuarios de tales sistemas. Además de las ya por todos conocidas recomendaciones de nuestro refranero sobre los secretos compartidos entre dos, o entre tres o más, la criptografía de clave simétrica adolece de diversos problemas:

- Si deseamos mantener al mínimo la magnitud del número de claves compartidas en una red de  $n$  usuarios, es decir, que cada clave se comparta entre tan solo dos usuarios, debe manejarse un número de  $n(n - 1)/2$  claves, lo que origina un resultado cuadrático en función de  $n$ .
- La distribución de claves secretas debe realizarse bien por comunicación directa (que no siempre es posible), bien sobre canales de comunicación generalmente inseguros.

Para eliminar estos problemas, en la década de los 70 del pasado siglo se atisbaron diferentes soluciones que culminaron en 1976 con la publicación del intercambio de claves Diffie-Hellman [12] y el establecimiento por estos autores de cómo debería construirse un protocolo de firma digital mediante funciones unidireccionales (anexo A), aunque dejando abierta la búsqueda de las funciones adecuadas. Un año más tarde

se diseñó el algoritmo RSA de Rivest, Shamir y Adleman [13], el primer criptosistema de clave pública. Posteriormente, algunos años después, pudieron ver la luz otros diseños de criptosistemas como ElGamal [14] o la criptografía sobre curva elíptica [15, 16] (ambos de 1985). Desde entonces, la comunidad criptográfica está trabajando en la búsqueda de nuevas funciones unidireccionales basadas en problemas matemáticos diferentes, así como en el análisis del considerable conjunto ya encontrado de las mismas, para establecer esquemas más eficientes y seguros que perfeccionen los resultados.

De acuerdo a los fundamentos para el intercambio de claves Diffie-Hellman, el uso más extendido de un criptosistema de clave pública completo como RSA se encuentra en crear una infraestructura de comunicaciones seguras mediante el desdoblamiento de las claves clásicas en dos: una privada, solo conocida por su propietario, y otra pública, derivada de la privada y de conocimiento público, asumiendo que de esta clave pública no es posible el cálculo de la privada bajo el estado actual de la tecnología.

Las funcionalidades más comunes de la criptografía de clave pública son las siguientes:

- Por un lado, la encriptación de la información: cualquier conocedor de la clave pública puede codificar cualquier información, y esta información codificada solo podrá decodificarse mediante la clave privada correspondiente y no por la clave pública conocida, por lo que podrá transmitirse sin problemas por un canal de comunicación inseguro.
- Por otro, la autenticación: cualquier información cifrada con una clave privada puede descifrarse con su correspondiente clave pública. Es evidente que todo el mundo tiene acceso a la información en claro y no se cumple en este sentido la función de encriptación, pero supuesto que la clave pública está directamente vinculada a una cierta identidad, ésta no podrá repudiar su origen, es decir, no podrá negar que la información ha sido encriptada con su correspondiente clave privada. Es el fundamento de la firma digital.

Los fundamentos matemáticos de estos criptosistemas los encontramos en funciones unidireccionales cuya imagen es computable fácilmente, pero cuya función inversa es computacionalmente intratable en la actualidad. RSA, por ejemplo, está basado en la facilidad del cálculo de la multiplicación de dos números primos  $p$  y  $q$ , aunque sean de gran tamaño (digamos del orden de  $10^{200}$ ), para obtener un producto  $n = p \cdot q$ , y en la idea básica de que conocido  $n$  no pueden obtenerse fácilmente  $p$  y  $q$ . Formalmente, definimos la función RSA como:

$$RSA_{n,e}: \mathbb{Z}_n \rightarrow \mathbb{Z}_n, x \mapsto x^e$$

en la que  $n$  es el producto de dos primos  $p$  y  $q$  (en la práctica, como hemos anticipado, de gran tamaño), mientras que  $e$  es un entero tal que  $1 \leq e < \phi(n)$  (correspondiendo  $\phi(n)$  a la función  $\phi$  de Euler sobre  $n$ ), y primo con respecto a  $\phi(n)$ . La clave privada será  $d$  tal que

$$e \cdot d \equiv 1 \pmod{\phi(n)}, 1 \leq d < \phi(n) = \phi(p)\phi(q)$$

mientras que la clave pública corresponderá al par  $(n, e)$ . Tanto  $p$ ,  $q$ , como  $\phi(n)$  deben permanecer secretos.

En resumen, la multiplicación puede realizarse eficientemente, pero no es factible computar  $x$  desde  $x^e$  en  $\mathbb{Z}_n$  sin conocer  $p$  y  $q$ , y estos a su vez no pueden computarse hoy por hoy mediante la factorización de  $n$  si son lo suficientemente grandes.

Otro ejemplo de estas funciones unidireccionales, también denominadas funciones trampa, más conocidas por su denominación en inglés *trapdoor functions*, la encontramos en la exponenciación discreta y en su inversa, el logaritmo discreto, en las que está basado el criptosistema de ElGamal, entre otros. Ahora tenemos un primo  $p$  y sea  $g$  un generador del grupo cíclico  $\mathbb{Z}_p^*$ . Definimos esta función como:

$$Exp: \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*, x \mapsto g^x$$

donde  $Exp$  resulta en un homomorfismo desde un grupo aditivo a uno multiplicativo, en el que encontraríamos, por ejemplo, que  $Exp(x + y) = Exp(x) \cdot Exp(y)$ . Pero también es un isomorfismo puesto que admite la función inversa

$$Log: \mathbb{Z}_p^* \rightarrow \mathbb{Z}_{p-1}$$

denominándose esta última función  $Log$  como función del logaritmo discreto. En este caso, mientras que  $Exp$  puede computarse de manera eficiente,  $Log$  se mantiene hoy intratable computacionalmente para una  $p$  lo suficientemente grande.

Existen otras funciones trampa de uso más o menos generalizado en criptografía, pero la búsqueda de utilidad para nuestro contexto en Blockchain/Bitcoin nos lleva a contemplar con algo más de profundidad un esquema análogo al del problema del logaritmo discreto, pero ahora sobre los puntos definidos en una curva elíptica cuyas coordenadas pertenecen a un cuerpo finito.

### 3.4 Criptografía de curva elíptica

La criptografía de curva elíptica fue la elegida desde el diseño original del ecosistema Bitcoin. Como hemos advertido, su funcionalidad para este sistema no se centra en la encriptación de la información sino en su aplicación para firma digital, que a la postre servirá tanto para movilizar los valores que soporta la blockchain como para recibir y ser beneficiario de los mismos. En este último caso, y mediante las claves públicas, resultará fundamental como veremos a la hora de construir la codificación de las direcciones beneficiarias de las transacciones.

En este epígrafe revisaremos brevemente los fundamentos matemáticos de las curvas elípticas y su definición sobre un cuerpo finito, para a continuación ver su aplicación a la criptografía de clave pública. Nos centraremos únicamente en las familias de curvas definidas bajo cuerpos finitos primos, que serán las que nos interesarán en última instancia, obviando otros tipos como las definidas sobre cuerpos binarios.

Definimos una curva elíptica sobre los números reales a la curva plana cuyos puntos satisfacen la ecuación

$$y^2 = x^3 + ax + b$$

Por ejemplo, la curva  $y^2 = x^3 + 7$  (que veremos tiene un especial significado para nuestro contexto Bitcoin/Blockchain) cumple estas características con  $a = 0$  y  $b = 7$ , y su representación gráfica bajo un conjunto de coordenadas muy limitado, pero muy significativo puesto que recoge todas sus principales singularidades, es:

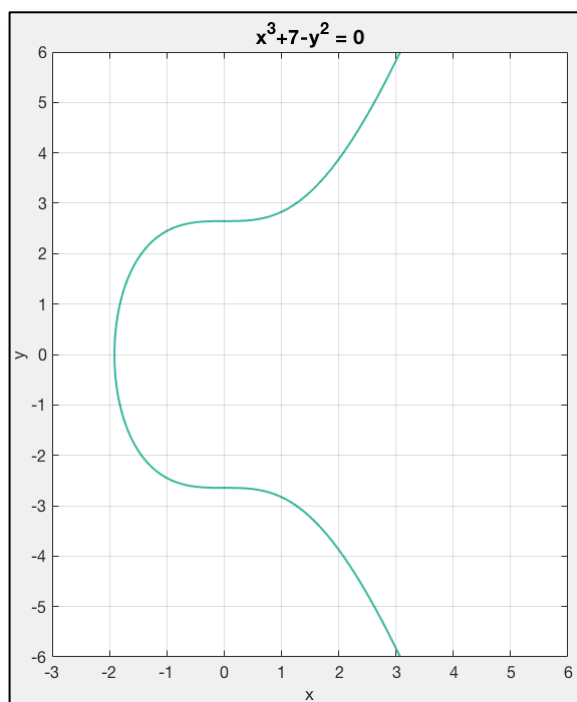


Fig. 3.5: representación gráfica de  $y^2 = x^3 + 7$

En esta representación es apreciable la simetría horizontal de la curva sobre la recta  $y = 0$  (o eje de las  $x$ ). Menos aparente, pero intuitivamente comprensible, es la propiedad de que cualquier recta no vertical que pase por dos puntos de la curva acabará cortando un tercer punto de la misma, lo que puede ratificarse analíticamente considerando  $y = mx + n$  (es decir, girando la curva de acuerdo a la pendiente  $m$ ) y resultando, por tanto, un polinomio en  $x$  de grado 3 que ofrecerá una suma de los órdenes de multiplicidad de cada raíz también de 3.

Esto permitirá definir una operación aditiva sobre la curva según la cual podremos sumar dos puntos para obtener un tercero. Gráficamente, esta operación supone la construcción y el seguimiento de la recta que corta los dos puntos en cuestión hasta localizar el tercer punto de corte, para inmediatamente después, y para que la operación esté bien definida, ubicar la proyección de este punto en el tramo simétrico opuesto de la curva, lo que nos daría el resultado de la adición.

Siguiendo con nuestra curva, la adición de, por ejemplo, los puntos  $A$  y  $B$  define el punto  $A + B$  de la siguiente forma:

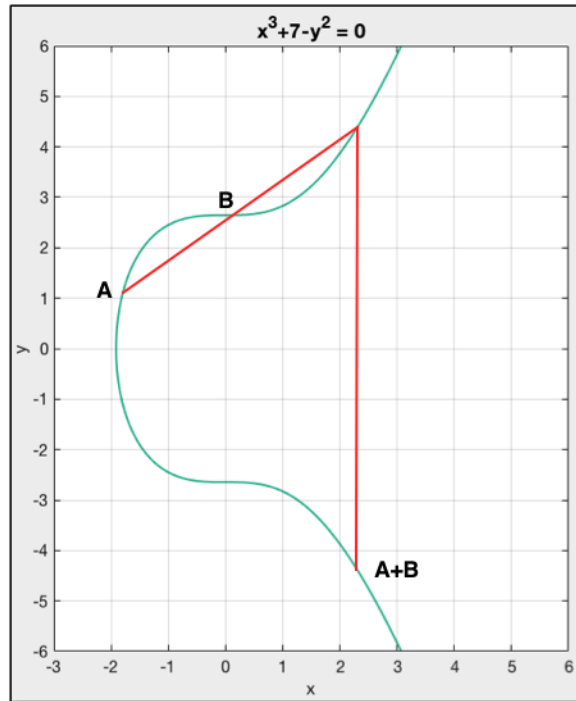


Fig. 3.6: representación gráfica de  $A + B$  en la misma curva de la figura 3.5

Esta operación, conmutativa puesto que  $A + B = B + A$ , y adicionalmente al conjunto de puntos  $G$  que constituye la curva y al punto al infinito  $\mathcal{O}$  (elemento identidad), conforma un grupo abeliano. Veremos algo más adelante una especificación más formal de esta adición derivada de su descripción geométrica.

Para su aplicación en criptografía no resulta adecuada la definición de este tipo de curvas sobre el cuerpo de los números reales puesto que para obtener resultados se necesitaría codificar mensajes de longitud infinita. Haciéndolo sobre un sistema de coordenadas escogidas de un cuerpo finito obtendremos resultados dentro de los enteros que resultarán de aplicación para nuestros propósitos. Por ejemplo, la misma curva, considerada en el ejemplo anterior sobre el conjunto de números reales, nos servirá ahora de aplicación sobre un cuerpo finito  $\mathbb{F}_q = \mathbb{Z}_q$ , con  $q$  necesariamente primo:

$$y^2 = (x^3 + 7) \bmod q$$

Asignando un valor de ejemplo tal como  $q = 29$ , podemos obtener ahora una representación discreta de los puntos que satisfacen la ecuación, representación radicalmente diferente a la que acabamos de ver:

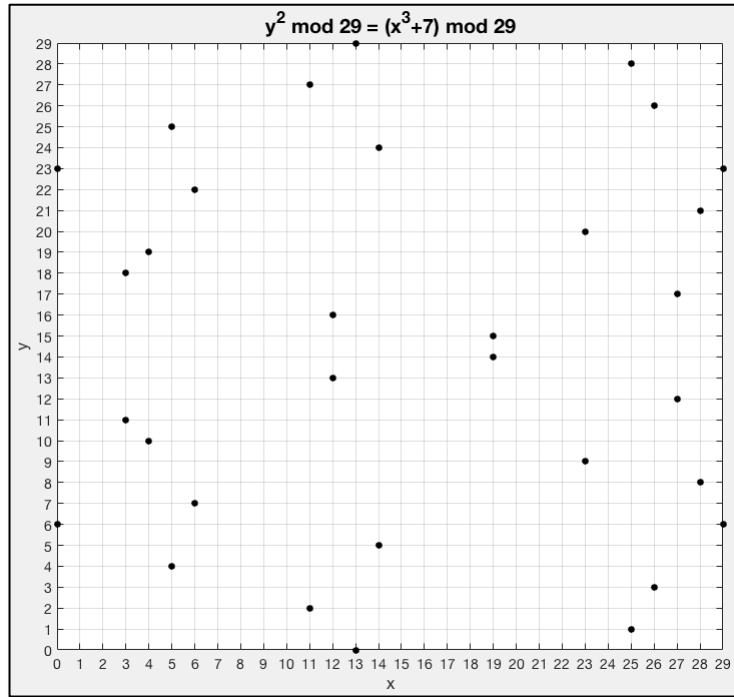


Fig. 3.7: representación gráfica de  $y^2 = (x^3 + 7) \bmod 29$

Vemos que la forma ha variado drásticamente, pero se aprecia el mantenimiento de la simetría horizontal (como no podría ser de otra manera al tener que  $y^2 = (-y)^2$ ), si bien tal simetría ya no se establece sobre el eje de abscisas, sino sobre  $y = q/2$ . Puesto que los elementos de  $\mathbb{F}_{29}$  son  $\{0,1,2,\dots,28\}$ , los valores que toma la función para valores de las rectas  $x = 29$  e  $y = 29$  son los mismos que para los ejes  $x = 0$  e  $y = 0$ , tal y como puede apreciarse en la fig. 3.5, ya que en términos de este cuerpo finito cualquier valor superior a  $q - 1$  se proyecta vía la operación *módulo* en el elemento correspondiente (menor que  $q$ ) de  $\mathbb{F}_q$ : 29 se proyecta en el elemento 0 puesto que  $29 \bmod 29 = 0$ .

También se mantiene la propiedad de que una recta que pasa por dos de los puntos definidos acabará cortando un tercero, puesto que para este caso, definido sobre un cuerpo finito, cabe aplicar el mismo razonamiento visto anteriormente haciendo  $y = mx + n$ , obteniendo así de nuevo un polinomio en  $x$  de grado 3. De igual manera a como hacíamos para la definición sobre números reales, el punto horizontalmente simétrico a este último punto obtenido se considerará el resultado de la operación de adición de los dos puntos iniciales, manteniendo así para este cuerpo finito la caracterización ya vista de grupo abeliano sobre los reales.

La representación de la curva que acabamos de ver para  $\mathbb{F}_{29}$  es muy sencilla puesto que el número primo utilizado es muy pequeño. Representar un gráfico semejante para un primo muy grande, digamos del orden de  $2^{256}$ , no es razonable, si bien el mecanismo es idéntico. Imaginemos por tanto ahora una curva generada para un  $q$  muy grande, de semejante magnitud a la que acabamos de ver, y que, dado un punto  $a$  cualquiera sobre la curva, operamos con la operación de adición, tal y como la hemos definido geoméricamente, sobre sí mismo,  $a + a$ , para obtener el nuevo punto  $a + a = b$ . Continuemos con  $a + b = c$ , con  $a + c = d \dots$  y así sucesivamente hasta haber realizado  $k$  operaciones, con  $k$  suficientemente grande. Conceptualmente podemos obtener de



manera sencilla un resultado final determinado, pero del que si solo conocemos su magnitud, y dada la definición de la operación suma, será computacionalmente intratable conocer el número  $k$  de veces que hemos realizado esta operación de adición de  $a$  sobre sí mismo. En resumen,  $a + a + a + \dots + a = k \cdot a = A$ , análogamente a la ya vista exponenciación, es eficientemente computable. Pero no lo es su operación inversa, esto es, la computación de  $k$  conocido  $A = k \cdot a$ , de manera también análoga al problema del logaritmo discreto ya visto. Nos encontramos de nuevo, por tanto, ante la deseada función trampa como soporte del criptosistema.

Veamos todo lo anterior de manera algo más formal.

Como hemos anticipado al comienzo de este epígrafe, nos interesan en particular las curvas elípticas definidas bajo cuerpos finitos primos puesto que la implementación de la firma digital utilizada bajo Bitcoin/Blockchain, ECDSA, corresponde a este tipo, por lo que obviaremos otras familias como las definidas sobre cuerpos binarios  $\mathbb{F}_q$  donde  $q = 2^n$ , que constituyen el fundamento de otros importantes criptosistemas (ver [17], por ejemplo, para ampliar el análisis a cuerpos binarios). Para nuestros efectos en este trabajo, por tanto, consideraremos únicamente el formalismo relativo a las curvas sobre el cuerpo finito primo  $\mathbb{F}_q = \mathbb{Z}_q$ , con  $q$  evidentemente primo.

Sea  $E$  una curva elíptica definida sobre un cuerpo finito  $\mathbb{F}_q$ , siendo  $q$  primo.  $E$  registrará, en consecuencia, el conjunto de soluciones de:

$$E(\mathbb{F}_q) = E(\mathbb{Z}_q) = \{(x, y) \in \mathbb{Z}_q^2 \mid y^2 = x^3 + ax + b\}$$

con  $a, b \in \mathbb{Z}_q$ , y cumpliendo como condición de regularidad  $4a^3 + 27b^2 \neq 0$ .<sup>6</sup>

$E(\mathbb{F}_q)$ , dado su elemento identidad (o elemento cero de la adición, o punto al infinito),  $\mathcal{O}$ , y la operación conmutativa del grupo, conforma un grupo abeliano. Esta operación conmutativa consiste en la adición de puntos sobre la curva, de tal manera que para dos puntos  $P, Q \in E(\mathbb{F}_q)$  tenemos

$$(P, Q) \mapsto P + Q \in E(\mathbb{F}_q)$$

es decir, que la adición de dos puntos cualesquiera de la curva resultará en otro punto situado también en la misma. Esta operación de adición, y según la descripción geométrica que hemos visto al principio de este apartado, se especifica para cuerpos primos finitos  $\mathbb{F}_q$ , y siempre que  $q > 3$ ,<sup>7</sup> como:

1.  $P + \mathcal{O} = \mathcal{O} + P = P$ , con  $P \in E(\mathbb{F}_q)$ .

---

<sup>6</sup> Sobre esta condición, ver el tratamiento del discriminante de  $E$ ,  $\Delta = -16(4a^3 + 27b^2)$ , para las ecuaciones simplificadas de Weierstrass en [18], pág. 78. En definitiva, la no nulidad del discriminante asegura la *suavidad* de la curva.

<sup>7</sup> En [18] (pág. 79 y siguientes) se introduce la derivación algebraica de la operación del grupo a partir de la descripción geométrica, siempre que la característica del cuerpo subyacente sea diferente de 2 o 3, lo que implica que para un cuerpo finito primo como el que consideramos,  $\mathbb{F}_q$ , necesariamente debe cumplirse  $q > 3$ .

2. Si  $P = (x, y) \in E(\mathbb{F}_q)$ , tendremos  $(x, y) + (x, -y) = \mathcal{O}$ . De esta manera, denominaremos negativo de  $P$  a  $-P = (x, -y) \in E(\mathbb{F}_q)$ .
3. Sean  $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_q)$ , con  $P \neq \pm Q$ . Tendremos  $P + Q = (x_3, y_3)$ , donde  $x_3 = \lambda^2 - x_1 - x_2$ ,  $y_3 = \lambda(x_1 - x_3) - y_1$ , y  $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ .
4. Sea  $P = (x_1, y_1) \in E(\mathbb{F}_q)$ . Entonces  $P + P = 2P = (x_2, y_2)$ , donde  $x_2 = \lambda^2 - 2x_1$ ,  $y_2 = \lambda(x_1 - x_3) - y_1$ , y  $\lambda = \frac{3x_1^2 + a}{2y_1}$ , siendo  $a$  el ya conocido parámetro de la curva en la componente  $ay$ .

Como se ve, con  $\mathcal{O}$  podemos tratar casos singulares como que los puntos a sumar se encuentren en la misma vertical, es decir, que tengan la misma componente  $x$ , lo que implicará que su punto suma sea, precisamente,  $\mathcal{O}$ .

En cuanto al cálculo de puntos en la curva, para lo que existen algoritmos apropiados, se exige que la parte derecha de la ecuación sea bien un cuadrado, bien cero, y en general resultará en dos idénticas soluciones, pero de signo contrario, que conforman su simetría.

Dado que su aplicación en criptografía requiere que los correspondientes algoritmos trabajen con un grupo o subgrupo cíclico (y generalmente las curvas elípticas no lo son), deberemos seleccionar un punto  $Q \in E(\mathbb{F}_q)$  con orden  $n$  correspondiente a un número primo muy grande, de tal manera que  $Q$  genere el subgrupo

$$\langle Q \rangle = \{kQ \mid k \in \mathbb{Z}_n\} = \{1Q, 2Q, \dots, (n-1)Q, nQ = 0Q = \mathcal{O}\} \subset E(\mathbb{F}_q)$$

Puesto que  $n$  es primo, el orden de  $\langle Q \rangle$  también lo es, y por tanto cualquier elemento del subgrupo  $P \in \langle Q \rangle$ , con  $P \neq \mathcal{O}$ , será también un generador del subgrupo.

La función unidireccional, o trampa, que definimos en criptografía de curva elíptica

$$\mathbb{Z}_n \rightarrow \langle Q \rangle, k \mapsto kQ = \underbrace{Q + Q + \dots + Q}_k$$

es similar a la definida para el problema de la exponenciación, pero cambiando la operación multiplicativa de la exponenciación por la aditiva en este caso.

La generación de las claves bajo este criptosistema es, visto lo anterior, directa: tras elegir aleatoriamente un entero suficientemente grande,  $k$ , como clave privada, lo multiplicamos por un generador  $Q$  del subgrupo de tal manera que obtendremos otro punto también perteneciente a la curva. Este nuevo punto  $K = kQ$  corresponderá a la clave pública. Y de acuerdo a lo que acabamos de ver, podremos comprobar la validez de cualquier clave pública  $K$  dada mediante las siguientes comprobaciones:

- Comprobar que  $K \neq \mathcal{O}$
- Confirmar que las coordenadas  $x$  e  $y$  del punto  $K$  son elementos de  $\mathbb{F}_q$
- Comprobar que  $K$  corresponde efectivamente a un punto de la curva definida por los parámetros correspondientes

- Comprobar que  $nK = \mathcal{O}$  (siendo  $n$ , como ya hemos visto, el orden del generador)

También nos encontramos con el problema análogo al del logaritmo discreto, ahora denominado problema del logaritmo discreto sobre curva elíptica: dado  $K \in \langle Q \rangle$ , actualmente es computacionalmente intratable (ver anexo A) calcular  $k \in \mathbb{Z}_n$ , con  $K = kQ$ , puesto que todo algoritmo conocido para ello es de tiempo exponencial<sup>8</sup>.

En este último sentido es conveniente reseñar que existen algoritmos que solucionan el problema de la factorización (y por tanto la inversión de la función RSA), o el del logaritmo discreto, en tiempo sub-exponencial<sup>9</sup>, por lo que para igual nivel de seguridad (o de, en otras palabras, intratabilidad de cada uno de los problemas), la longitud de las claves bajo criptografía de curva elíptica es sensiblemente menor que las necesarias para otros esquemas usuales. Así, las recomendaciones del *National Institute of Standards and Technology* (NIST) [21] equiparan la seguridad de las claves de alrededor de 224 bits bajo curva elíptica a la de las claves de 2048 bits de ElGamal o RSA, por nombrar algunos de los criptosistemas más representativos. No es de extrañar, por tanto, que la curva elíptica resultara como criptosistema de elección para Bitcoin y que, en general, se vaya imponiendo en las nuevas aplicaciones.

### 3.5 El estándar secp256k1

La elección última del tipo de curvas elípticas convenientes para su uso en criptografía, así como su parametrización, no es trivial. Ayudan para ello los estándares formulados y aceptados por investigadores, agencias gubernamentales, y por el propio mercado. En este apartado detallaremos la curva utilizada para las necesidades criptográficas de Bitcoin, cuyo estándar ha sido establecido por el NIST, a la vez que nos permitirá repasar la 6-tupla de parámetros que este tipo de curvas necesitan:  $(q, a, b, Q, n, h)$ , con los significados que recordaremos y precisaremos a continuación.

A la hora de elegir la curva elíptica de utilización para el sistema Bitcoin/Blockchain, el diseñador escogió el estándar denominado *secp256k1*, que ofrece claves de 256 bits (que según nos recuerda el propio informe del estándar vienen a ser equivalentes en seguridad a claves de 3072 bits bajo, por ejemplo, RSA o DSA).

A la curva  $y^2 = x^3 + ax + b$ , definida sobre  $\mathbb{F}_q$ , este estándar le asigna como parámetros de configuración los valores de  $a = 0$  y  $b = 7$ , lo que resulta en nuestra ya conocida curva

$$y^2 = (x^3 + 7) \bmod q$$

---

<sup>8</sup> Ver el análisis de la aparente intratabilidad del problema del logaritmo discreto sobre curva elíptica en [19], pág. 15.

<sup>9</sup> Ver para RSA la discusión sobre el tiempo sub-exponencial, normalmente  $O\left(e^{\sqrt{\log n \log \log n}}\right)$ , para factorización de  $n = pq$ , y por tanto la posibilidad de inversión de la misma y la recomendación de un tamaño mínimo de  $n$  de 1024 bits, en [20] (pág. 26); en la misma referencia puede verse un análisis similar para el logaritmo discreto (pág. 23), cuyo coste temporal sub-exponencial se establece entre  $e^{(k \log k)^{\frac{1}{2}}}$  y  $e^{(k \log k)^{\frac{1}{3}}}$ .

Por su parte, el número primo  $q$ , que recordemos debe ser de gran tamaño, queda establecido en el estándar por:

$$q = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Con estos parámetros definimos completamente la forma de la curva. Nos faltaría la elección del generador (también denominado punto base),  $Q$ , cuyo orden  $n$  corresponderá a un número primo grande. El estándar contempla:

$Q = 04\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798\ 483ADA77\ 26A3C465\ 5DA4FBFC\ 0E1108A8\ FD17B448\ A6855419\ 9C47D08F\ FB10D4B8$

con

$n = \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141}$

Por último, el cofactor  $h$ , definido como el multiplicador de  $n$  tal que su producto da como resultado el orden de la curva, es decir,  $h = \frac{1}{n} |E(\mathbb{F}_q)|$ , queda establecido en el estándar como la unidad,  $h = 1$ , por lo que el orden de la curva coincide con el de  $Q$ , lo que equivale a decir que trabajaremos con todos los elementos del grupo, los cuales, con la excepción de  $\mathcal{O}$ , serán todos generadores del propio grupo (puede probarse que esta última propiedad se da siempre que  $h < n$ ). Puesto que  $h = 1$ , entonces  $n$  es el menor entero positivo que cumple  $nQ = \mathcal{O}$ .

Todos los parámetros anteriores son públicos y deben ser comunes entre todos los participantes para el correcto funcionamiento del criptosistema de curva elíptica.

### 3.6 Firma digital y Elliptic Curve Digital Signature Algorithm - ECDSA

La firma digital replica en el mundo de las tecnologías de la información los efectos de una firma ordinaria, manual, en el mundo físico. En términos generales, una firma digital es un número obtenido en función de otro número secreto, conocido únicamente por el firmante, y de un mensaje o información a firmar, que adicionalmente tendrá que poder interpretarse en términos numéricos (lo que es prácticamente trivial dada la constitución última de toda información digitalizada como cadenas de ceros y unos).

Un procedimiento de firma digital debe ser suficientemente seguro, lo que implica que si un atacante consiguiera, sobre un mensaje elegido por él mismo, una determinada firma de un determinado firmante, no será capaz de falsificar tal firma sobre otro mensaje (ver [22], págs. 281-308). De esta manera, una firma digital segura cumple dos importantes funciones interrelacionadas: por un lado, autentifica el origen de un determinado mensaje o información, por lo que el emisor, o firmante, no podrá *repudiar* su vinculación directa; por otro, garantiza que ese mensaje no ha sido alterado en su posible tránsito por canales inseguros.

Dentro de las varias familias de firma digital existentes según el problema matemático subyacente, o función trampa, sobre el que se soportan, las más conocidas son:

- Intratabilidad del problema de la factorización de un número entero (RSA, por ejemplo).
- Intratabilidad del problema del logaritmo discreto en un cuerpo finito (DSA, por ejemplo).
- Intratabilidad del problema del logaritmo discreto sobre una curva elíptica en un cuerpo finito (ECDSA, por ejemplo).

Dada su aplicación en la actual implementación de Blockchain para Bitcoin, en este trabajo, al igual que ya lo hicimos en el apartado sobre critposistemas, nos centraremos en la firma digital basada en curva elíptica, y en la ECDSA en particular.

Al principio del capítulo avanzábamos que las funciones *hash* se aplican de forma intensiva en los procedimientos de firma digital, y revisábamos algunas de las ventajas que aportan al reducir el tamaño del texto a procesar, tales como que los algoritmos de encriptación bajo clave pública actúan de manera más eficiente, o que la firma digital sobre un mensaje extenso tiene un tamaño de magnitud semejante al del mensaje, con los inconvenientes que su manipulación o transmisión conlleva. Pero existen otros motivos menos evidentes para preferir realizar el proceso de firma digital sobre un resumen del mensaje en lugar de sobre el mismo mensaje. Un primer motivo es que se suele requerir por los respectivos mecanismos de firma digital que cualquier mensaje  $m$  a procesar cumpla  $m \in \mathbb{Z}_n$ , es decir, que tenga una longitud determinada, lo que se garantiza por el mapeo de cualquier mensaje a  $\mathbb{Z}_n$  mediante una función *hash*. Además, el resumen obtenido por una de tales funciones que cumpla la propiedad de uniformidad que vimos en el epígrafe correspondiente, aportará mayor seguridad con respecto a una firma sobre texto en claro puesto que reduce la probabilidad de que un atacante pueda descubrir determinadas claves por el análisis estadístico del lenguaje natural.

Cuando hablemos, por tanto, de aplicación de firma digital asumiremos que se realiza sobre un resumen o *hash* del mensaje que se pretende firmar, y junto con lo que ya conocemos sobre los critposistemas de clave pública podemos establecer una abstracción básica de un procedimiento completo de firma digital:

1. El firmante de un mensaje obtiene un resumen del mismo mediante una función *hash* criptográfica.
2. Este resumen lo cifra con su clave privada, y este resultado cifrado lo transmite conjuntamente con el mensaje original, éste sin cifrar, como firma digital de este último.
3. Cualquier receptor del mensaje, y conocedor de la clave pública del emisor, podrá descifrar el contenido de la firma digital y obtener de nuevo el resumen sin cifrar.
4. Utilizando la misma función *hash* que el firmante, el receptor obtendrá el resumen del mensaje original sin cifrar.
5. El receptor comprobará si ambos resúmenes son coincidentes. Si esta comprobación resulta exitosa, significará que ha sido cifrado por el propietario de la clave privada par de la pública, quien no podrá repudiar el mensaje así firmado. Además, también demostrará que el texto en claro no ha sufrido variación en su transmisión.

Habiendo hecho referencia a claves públicas y privadas, y por tanto a criptosistemas de clave pública, parece natural aplicar el muy eficiente criptosistema sobre curva elíptica que hemos visto a la función de firma digital. Es lo que se consigue con ECDSA (*Elliptic Curve Digital Signature Algorithm*), que se ha constituido en un estándar generalizado (ANSI, IEEE, NIST, ISO...) y que además es el utilizado por la infraestructura de Bitcoin cuando se necesita hacer uso de la firma digital.

Para la aplicación de ECDSA recordaremos, en primer lugar, el proceso de generación de claves que debe seguir quien esté interesado en firmar digitalmente un mensaje: dada una determinada curva y los parámetros del criptosistema según la tupla  $(q, a, b, Q, n, h)$ , elegirá aleatoriamente un entero  $k$  perteneciente al intervalo  $[1, n - 1]$ , lo suficientemente grande, como clave privada; este entero se multiplica por el generador  $Q$  del subgrupo de tal manera que obtendremos otro punto,  $K = kQ$ , también perteneciente a la curva y que corresponderá a la clave pública.

Una vez generado el par de claves el firmante estará en posesión de la clave privada y ofrecerá, por los medios que se estimen convenientes, la clave pública a quienes interese hacer uso de ella. A partir de este momento, y considerando el mismo conjunto ya visto de parámetros del criptosistema, públicos y compartidos, para la firma de un mensaje  $m$  el firmante tendrá que seguir el siguiente protocolo de firma:

1. Tomará  $m$  como entrada de una función *hash* (tal como SHA-256) para obtener como resultado un resumen  $m'$ , que asumiremos se trata del número entero  $m' \in \mathbb{Z}_n$  tal y como queda definido por la cadena de bits resultante del *hash*.
2. Elegirá aleatoriamente un entero  $g$  del intervalo  $[1, n - 1]$ .
3. Calculará un nuevo punto  $P$  multiplicando el entero  $g$  por el generador del grupo, obteniendo  $P = gQ = (x_P, y_P)$ .
4. Convertirá a continuación la coordenada  $x_P$  en un entero para calcular  $r_{sig} = x_P \bmod n$ . Si  $r_{sig} = 0$ , reiniciamos volviendo al punto 2.
5. Calculará  $s = g^{-1}(m' + r_{sig}k) \bmod n$ , con  $g^{-1}$  tal que  $1 = (g^{-1}g) \bmod n$ . Si  $s = 0$ , reiniciamos volviendo al punto 2.
6. El par  $(r_{sig}, s)$  es la firma digital de  $m$  obtenida a través de su resumen  $m'$ . Normalmente el firmante enviará el conjunto  $\{m, K, (r_{sig}, s)\}$  a los receptores que necesiten validar el mensaje  $m$  ( $m'$  debe ser calculado desde  $m$  por el receptor con la misma función *hash* utilizada por el firmante, con lo que se evita la posibilidad de su falsificación).

Dada una firma digital ejecutada mediante el procedimiento anterior, cualquier conocedor de la clave pública  $K$  vinculada a la clave privada  $k$  del firmante puede verificar su firma  $(r_{sig}, s)$  mediante los siguientes pasos:

1. Comprobará que  $r_{sig}, s \in [1, n - 1]$ . En caso contrario, la firma no es válida.
2. Puesto que habrá recibido el mensaje  $m$  en claro, computa su resumen  $m'$  con la misma función *hash* que el firmante.
3. Calculará el inverso de  $s$ ,  $s^{-1} \bmod n$ .
4. Calculará  $w_1 = m's^{-1} \bmod n$ , y  $w_2 = r_{sig}s^{-1} \bmod n$ .
5. Calculará el punto  $R = w_1Q + w_2K = (x_R, y_R)$ . Si  $R = \mathcal{O}$ , la firma no es válida.
6. Convertirá la coordenada  $x_R$  en un entero para calcular  $r_{val} = x_R \bmod n$ .
7. Si  $r_{sig} = r_{val}$ , entonces la firma es válida. En caso contrario no es válida.

Si nos detenemos un momento para revisar el significado último de lo que acabamos de ver, observamos que la firma se compone de una coordenada  $x$ ,  $r_{sig}$ , correspondiente a un punto sobre la curva, y de un escalar,  $s$ , calculado tal y como hemos visto en el punto 5 del procedimiento de firma:

$$s = g^{-1}(m' + r_{sig}k) \bmod n$$

Lo importante aquí es que el escalar  $s$  es función del mensaje a firmar  $m'$  y de la clave privada  $k$ , así como del entero  $g$  que nos ha servido para definir el punto  $P$  mediante  $gQ$ , con el generador  $Q$  públicamente acordado y conocido. La estrategia que sigue el algoritmo de firma consiste en verificar el punto implícito en el parámetro  $g$  de  $s$ . El significado último del componente  $r_{sig}$  en la expresión de  $s$  no cumple un papel estratégico en el proceso puesto que podría cambiarse por un tercer parámetro arbitrario de la firma (y por tanto compartido por las partes), pero puesto que debe pasarse en el par que define la firma digital para su definitiva comprobación, se aprovecha en su cálculo para resultar el componente *codificado* de  $s$ . Por su parte, para obviar en la validación este componente codificado por la clave privada  $k$ , en este proceso se tendrá que hacer uso de la conocida clave pública  $K = kQ$ . Si la coordenada  $x$  del punto implícito en  $s$  coincide con el componente  $r_{sig}$  adicional de la firma, ésta es válida.

Al receptor de la firma, como sabemos, le resulta computacionalmente intratable averiguar  $k$  o  $g$  a pesar del conocimiento que tiene sobre  $s$ ,  $r_{sig}$ ,  $m'$ ,  $Q$  y  $K$ . Pero lo que sí puede calcular directamente es  $s^{-1}$ , el inverso de  $s$ . A partir de este inverso, y puesto que todos los elementos de la expresión le son conocidos, puede calcular (puntos 4 y 5 del procedimiento de validación) el punto  $R$  implícito en  $s$  que estamos buscando:

$$R = m's^{-1}Q + r_{sig}s^{-1}K$$

Confirmemos la corrección del procedimiento. Sabiendo que la clave pública  $K = kQ$ , de la expresión de  $R$  podremos convenir:

$$R = m's^{-1}Q + r_{sig}s^{-1}kQ = s^{-1}(m' + r_{sig}k)Q$$

Puesto que de la definición de  $s$ , y operando directamente sobre términos, se sigue que

$$g = s^{-1}(m' + r_{sig}k) \bmod n$$

y sustituyendo en la última expresión del punto  $R$ , recordando que  $P = gQ$  (paso 3 del procedimiento de firma), y explicitando las coordenadas de cada punto, tendremos:

$$R = (x_R, y_R) = s^{-1}(m' + r_{sig}k)Q = gQ = P = (x_P, y_P)$$

y de aquí la exigencia de la igualdad de ambas coordenadas  $x$ ,  $x_P \equiv r_{sig}$  y  $x_R \equiv r_{val}$ , de  $P$  (en la firma) y de  $R$  (en la validación), respectivamente, para ratificar la validez de la firma que refrenda la condición del punto 7 del procedimiento de validación.

Una última referencia sobre coste computacional: el coste de estos procedimientos es moderado puesto que para la firma su grueso se encuentra en la multiplicación de su punto 3, mientras que para su validación está en las multiplicaciones de su punto 5.

### 3.7 Una cadena segura

Hemos visto cómo la elección aleatoria de un gran escalar como clave privada permite obtener de forma sencilla, mediante su multiplicación por un generador, una clave pública en forma de coordenadas de un punto situado sobre una curva elíptica especialmente elegida. Sin embargo, el cálculo inverso del escalar dado el punto de la curva que define la clave pública, aún conociendo el generador por el que se ha multiplicado inicialmente, es hoy un problema computacionalmente intratable cuando nos enfrentamos a valores razonablemente grandes de los parámetros, y ello dada la definición de la operación de adición que hemos analizado. Esto nos permite explotar la virtual unidireccionalidad en la función matemática que soporta el criptosistema de clave pública basado en curva elíptica.

Una vez establecido el criptosistema, la firma digital (que como ya sabemos, y entre otras muchas cosas, es el componente básico de una moneda digital) se abstrae de él como algo casi natural: se firma con la clave privada y se valida tal firma con la clave pública, lo que nos asegura el origen y contenido del mensaje mediante un procedimiento sencillo y seguro.

La otra primitiva criptográfica que hemos revisado con cierto detalle han sido las funciones *hash*, y no solo bajo su función auxiliar, pero clave, como herramienta de preparación de resúmenes para ser firmados, sino en la estratégica implementación del mecanismo de consenso, como *proof-of-work*, que permite añadir bloque tras bloque a la blockchain, también de una manera sencilla y segura.

La virtud de Nakamoto al diseñar Bitcoin radicó en tomar bloques de construcción criptográficos que existían desde muchos años atrás y combinarlos de tal manera que la moneda digital que soportan se ha demostrado tan segura desde su implementación en 2009 que ha sido el primer experimento de moneda electrónica que ha perdurado, como no pudo hacerlo ninguno de los diseños previos realizados hasta este sistema Bitcoin/Blockchain. Por lo demás, una vez demostrado el entorno seguro e inatacable, tiempo habrá para mejoras de diseño que permitan un mayor número de transacciones por segundo, un mayor número de funcionalidades, o una mayor satisfacción de los usuarios. En estos últimos años los únicos incidentes de seguridad han tenido que ver con defectos en la custodia de las claves o por algunas implementaciones incorrectas de los algoritmos criptográficos en algunos clientes Bitcoin, pero ningún atacante, que se conozca, ha podido comprometer la cadena de bloques tal y como está diseñada.

Respecto a esta última, intentaremos detallar en el próximo capítulo cómo se aplican los fundamentos criptográficos vistos y como se interrelacionan entre sí y con el resto de componentes del sistema Blockchain.



## *Blockchain en Bitcoin: conceptos, estructura, funcionalidades*

En los capítulos anteriores nos hemos proveído de las herramientas necesarias para poder comprender y razonar sobre el funcionamiento del *macrosistema* Bitcoin/Blockchain, que es lo que emprendemos a partir de ahora. Y lo vamos a acometer trabajando simultáneamente los dos ejes fundamentales de análisis para un sistema como el que nos ocupa, el estructural y el funcional, bajo un enfoque dirigido por el elemento clave del sistema: la transacción.

La transacción en Bitcoin/Blockchain es conceptualmente clave puesto que Bitcoin nace como un mecanismo de dinero digital, y como todo dinero debe cumplir, entre otras, las funciones clásicas de depósito y de intercambio de valor. Es en esta última función de intercambio en la que la transacción cobra su máxima manifestación literal puesto que toda transacción, excepto algunas excepciones que quedan fueran de nuestro interés, es un intercambio de valor. Pero también la transacción es fundamental para la función de depósito puesto que, como veremos, un determinado *saldo de valor* (de bitcoins en nuestro caso) no es otra cosa que el resultado (output) de una transacción, o el agregado de más de una, no gastado todavía por su beneficiario.

También la transacción es clave en el sentido estructural puesto que resulta transversal y articula todo el mecanismo computacional soporte del sistema: una vez definida la transferencia de valor mediante su correspondiente transacción, ésta se lanza a la red *peer-to-peer* para su validación y registro en un bloque que recogerá todas las transacciones registradas en la red en el último periodo de tiempo tomado en consideración, bloque que una vez validado se anexará a la blockchain, la cual incorpora de esta manera dos dimensiones de la misma información transaccional: la temporal, manifestada en cada uno de los bloques validados (aproximadamente cada 10 minutos en Bitcoin), y la de valor, explícita en el tracto de transacciones desde la creación de cada bitcoin hasta el último propietario actual del mismo.

En lo que queda de capítulo intentaremos esta aproximación al problema introduciendo primero las direcciones como principal soporte transaccional al resultar el mecanismo de asignación del *valor transferido* a su propietario final, y que nos permitirán luego trabajar en profundidad la transacción y sus elementos; continuaremos con la estructura y el recorrido de las transacciones en la red Bitcoin/Blockchain; y terminaremos revisando el minado de bloques, que constituyen las unidades temporales básicas del sistema para el registro de las transacciones, minado que supone la validación de cada nuevo bloque y su consecuente incorporación a la blockchain y que actúa, además, como mecanismo fundamental de consenso mediante la solución para este contexto de lo que se ha dado en llamar el *problema de los generales bizantinos* [23, 24].

Llegados a este punto, y si no se ha hecho previamente, quizá resulte interesante revisar el apartado 1.1 del capítulo de Introducción, en el que repasábamos la semántica actual del término *Blockchain*, así como su uso en este trabajo.

#### 4.1 Las direcciones como titularidad de la propiedad del valor

En el capítulo 2, tratando los conceptos que Nakamoto expuso en el *paper* de referencia de Bitcoin, veíamos que las transacciones se mantienen públicamente, sin encriptación alguna, pero que no existe información explícita sobre la identidad última de quienes participan en ellas puesto que el modelo de privacidad de Bitcoin intenta proteger su identidad.

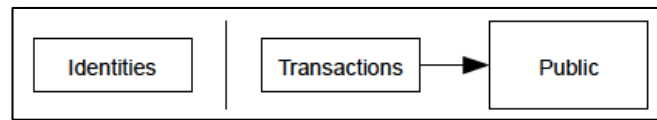


Fig. 4.1: modelo de privacidad de Bitcoin (S. Nakamoto)

Según este diseño, la privacidad de las partes en ausencia de una autoridad central que gestione las transacciones (modelo tradicional de privacidad) viene garantizada por el anonimato de las claves públicas de los que participan. Es, por tanto, la primera clave de interés sobre el soporte no solo de privacidad, sino de propiedad, en Bitcoin/Blockchain: las claves públicas de los intervinientes sirven como referencias para la propiedad del valor en cuestión.

En efecto, y semejante a una cuenta bancaria que almacena y que puede recibir o transferir dinero, una dirección para mantener y a la cual, o de la cual, enviar bitcoins corresponde a una clave pública proveniente de un par de claves privada-pública de un criptosistema de clave asimétrica. Si la clave pública sirve como referencia de la propiedad de los bitcoins recibidos a su favor, la clave privada servirá para la movilización y transferencia de su valor mediante la firma digital. Pero a diferencia de una cuenta bancaria, y en un lúcido resumen de Chris Clark, *en lugar de registrar una cantidad de bitcoins para cada propietario, se registra un propietario* (bajo la forma de dirección, añadimos) *para cada cantidad de bitcoins transferida* ([25], cap. 6).

En Bitcoin/Blockchain el criptosistema de clave pública de referencia para la firma digital es el basado en la criptografía de curva elíptica, y la firma digital se implementa bajo ECDSA, como ya conocemos por el capítulo anterior.

Recordemos que bajo criptografía de curva elíptica una clave pública se obtiene de la multiplicación de un número  $k$  obtenido aleatoriamente (y que corresponderá a la clave privada secreta del par de claves) por el generador  $Q$  del subgrupo cíclico  $\langle Q \rangle \subset E(\mathbb{F}_q)$  (ver su desarrollo en el capítulo anterior), siendo  $Q$  un punto sobre la curva elíptica. La clave pública  $K = \underbrace{Q + Q + \dots + Q}_k = kQ$  así definida será, por tanto, otro punto sobre la curva, y podremos de esta manera representarla bajo dos formatos diferentes:

- Dando de manera explícita sus coordenadas  $x$  e  $y$ , de 256 bits cada una y consumiendo, por tanto, un total de 64 bytes.
- Dando únicamente la coordenada  $x$  puesto que la  $y$ , conocida  $x$ , está implícita en la ecuación de la curva y será fácilmente calculable, requiriendo ahora la mitad de espacio de almacenamiento, 32 bytes, para su representación.

Esta segunda manera de representar las claves públicas se denomina *forma comprimida*, y es actualmente la forma más habitual de representación dado su ahorro de espacio.

Para ambos casos de representación, tanto comprimida como no, se añade un byte adicional como prefijo indicativo de su tipo: 04 corresponderá a la clave no comprimida (utilizando, en consecuencia, un total de 65 bytes en su representación), mientras que 02 y 03 se añadirán como prefijos de las claves públicas comprimidas pares e impares, respectivamente, alcanzando un tamaño final de 33 bytes. Esta distinción entre claves pares e impares, y de acuerdo a la aritmética binaria sobre cuerpos finitos de orden primo y su representación, corresponde directamente al signo de la coordenada  $y$  (recordemos que la ecuación de la curva elíptica  $y^2 = x^3 + ax + b$  implica que, para  $x \neq 0$  y  $b \neq 0$ ,  $y$  toma valor tanto positivo como negativo al resultar  $y^2 = (-y)^2$ ), y cuyo signo es necesario conocer para fijar uno de los dos puntos simétricos resultantes para una determinada coordenada  $x$ .

No obstante, la dirección de referencia para gestionar el valor Bitcoin no corresponde exactamente con el literal de la clave pública, sino que éste es transformado de la siguiente manera para obtener la dirección final:

1. Al literal de la clave pública, tal y como lo hemos contemplado como concatenación de un prefijo indicativo de su tipo y la/s coordenada/s  $x/(x, y)$ , se le aplica la función *hash* SHA-256
2. Al resultado del punto anterior se le aplica la función *hash* RIPEMD-160, obteniendo así un resultado intermedio de 20 bytes, denominado *pubKeyHash*, muy importante a la hora de comprobar las transacciones puesto que constituye la dirección interna del sistema Bitcoin, como veremos
3. Al resultado del punto anterior se le añade un prefijo indicativo del tipo de dirección (para las direcciones ordinarias, para cuya movilización se exige una única clave privada, el prefijo corresponde al byte 0x00; veremos existen otros tipos de direcciones) o del tipo de red sobre la que va a gestionarse, ya que existen redes Bitcoin alternativas, normalmente para pruebas
4. Se elabora un *checksum*, o sufijo para detección de errores, de 4 bytes eligiendo los primeros cuatro del resultado de aplicar una doble SHA-256 sobre el resultado obtenido en el punto anterior
5. Al resultado del punto 3 se le añade el sufijo *checksum* obtenido en el punto 4, obteniendo un resultado intermedio de 25 bytes.
6. Por último, el resultado del punto anterior se codifica bajo representación *Base58Check*,<sup>10</sup> obteniendo así una dirección final de normalmente 34 caracteres alfanuméricos (aunque pueden llegar hasta un mínimo de 26 en algunos casos menos usuales) pertenecientes al conjunto de caracteres de *Base58*

---

<sup>10</sup> *Base58* es un procedimiento de representación de código binario mediante texto, semejante al muy utilizado *Base64* para la transmisión de ficheros binarios por correo electrónico. Está basado en un mapeo específico para Bitcoin de los valores 0 a 57 en un determinado juego de caracteres entre los que no se encuentran los que pueden provocar confusión en un lector humano: I (i mayúscula) y l (ele minúscula), 0 (cero) y O (o mayúscula). La anexión de los 4 bytes del *checksum* que hemos visto complementa la codificación que viene a denominarse *Base58Check*.

Un diagrama de lo anterior:

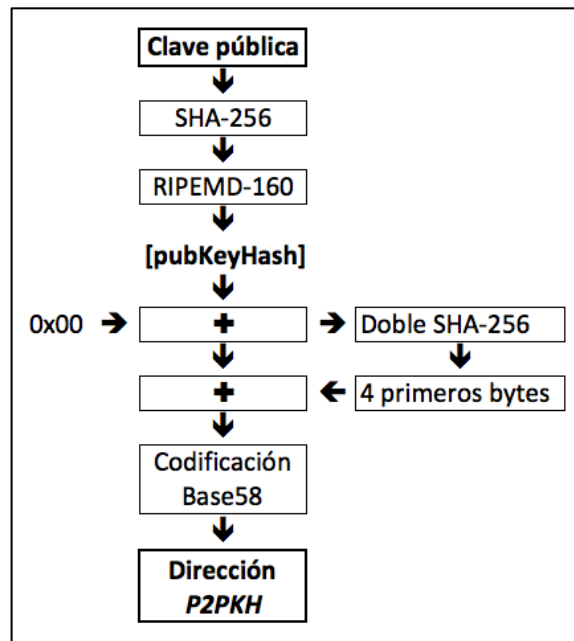


Fig. 4.2: obtención de la dirección Bitcoin desde su clave pública (adaptado de [60])

donde el símbolo '⌢' representa la concatenación, bien por la entrada izquierda (prefijo), bien por la entrada derecha (sufijo), de dos cadenas de bits.

Un ejemplo de dirección correcta puede ser la siguiente:

1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

Este tipo de direcciones *ordinarias* prefijadas por 0x00, y que por efecto de la codificación *Base58* siempre comienzan por el carácter '1' (puesto que el valor 0 se codifica mediante '1' en este procedimiento), se denominan direcciones *P2PKH*, o *Pay-to-PubkeyHash*, por la expresión *Pay to Public Key Hash*, entendible como "pagar a una clave pública representada por su *hash*".

Existe un segundo tipo de direcciones para Bitcoin de uso común: la *P2SH*, o *Pay-to-ScriptHash* ("pagar a un *script*, o guión, representado por su *hash*"). Este tipo de direcciones se estandarizaron a partir de 2012 (BIP<sup>11</sup> 16) para poder aplicar una semántica más compleja a la hora de movilizar sus depósitos que la normalmente requerida por las *P2PKH*: mientras que en estas últimas usualmente basta la presentación de la clave privada para la transferencia del valor asignado a su correspondiente clave pública, bajo *P2SH* pueden requerirse exigencias adicionales (como la necesidad de presentación de varias claves privadas, por ejemplo), pero trasladando la responsabilidad de la complejidad al beneficiario del valor transferido, mientras que el transmitente sólo debe manejar la complejidad básica de *P2PKH* al utilizar una sintaxis de envío semejante y desentenderse de la más extensa de

<sup>11</sup> *Bitcoin Improvement Proposal*, o BIP, son propuestas formales de mejora del sistema Bitcoin/Blockchain, identificadas según su ordinal, y que se someten a la comunidad Bitcoin para su aceptación o rechazo según su grado final de consenso.

movilización diseñada por el beneficiario. Veremos más detalle de esto cuando nos ocupemos de las transacciones.

Al margen de esta complejidad en la disposición de sus valores, formalmente una dirección *P2SH* sólo se diferencia de una *P2PKH* en su carácter prefijo, que en vez de ser un '1' será ahora un '3' como indicativo de este tipo de dirección. Por ejemplo:

3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy

El esquema de obtención de una dirección *P2SH* es idéntico al visto para *P2PKH*, con las salvedades de su input (un *script* ahora) y del valor del prefijo previo a su codificación *Base58* (ahora 0x05):

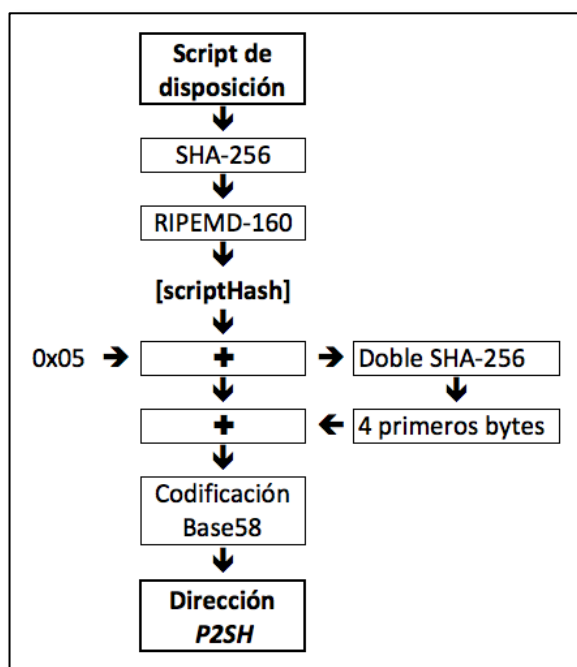


Fig. 4.3: obtención de dirección Bitcoin desde un guión de disposición (*redemption script*) (adaptado de [60])

Puesto que el mantenimiento y utilización frecuente de una o unas pocas direcciones Bitcoin puede dar pistas sobre la identidad última de su propietario, y dado el enorme espacio de posibles direcciones distintas (del orden de los posibles resultados en el procedimiento visto, esto es,  $2^{160}$  para los 20 bytes previos a la adición del *checksum*), es práctica habitual crear y gestionar un conjunto elevado de direcciones. En definitiva, un *wallet* (monedero) de bitcoins es una colección de pares de claves privada-pública vinculados a los posibles saldos de bitcoins pendientes de gastar mediante las direcciones asociadas unívocamente a cada clave pública, saldos que ni siquiera es necesario registrar en el propio *wallet* puesto que pueden recuperarse fácilmente de la blockchain (aunque hacerlo agiliza su gestión).

Existen otros tipos de direcciones, que conllevan aparejados otros prefijos, pero cuya especificidad no nos interesa. También es reseñable que estas direcciones finales tras *Base58*, y como veremos, no son exactamente las internas de Bitcoin/Blockchain, sino las *visibles y manejables* por sus usuarios.

Una vez que conocemos adonde y de dónde transferir bitcoins, podemos analizar la forma de hacerlo mediante las transacciones.

## 4.2 Las transacciones como mecanismo de intercambio de valor

En términos conceptuales una transacción es una transferencia de valor entre direcciones Bitcoin. Puede intercambiar valor de una única dirección a otra, de una a varias, de varias a una, o de varias a varias.

En términos operativos, en Bitcoin/Blockchain una transacción recoge como elementos principales:

- Un conjunto de inputs (en general, uno o más inputs; en especial, uno específico para las transacciones *coinbase*, que veremos más adelante) en el que cada elemento es un output no gastado de una transacción anterior (o UTXO, de *Unspent Transaction Output*) en la que el transmitente actual figuraba como beneficiario
- Y un conjunto de outputs (uno o más outputs) en el que cada elemento representa una dirección o beneficiario, y una cantidad de moneda a asignar a la misma

La suma del valor de los outputs debe ser inferior o igual a la de los inputs, considerándose la posible diferencia de valor como *fees* que se asignarán al minero que enlace el bloque que contenga la transacción en cuestión a la cadena de bloques.

En términos computacionales una transacción se diseña como una estructura de datos implementada por la clase `CTransaction` mediante el lenguaje de programación C++, con el que se escribe buena parte del núcleo del sistema Bitcoin/Blockchain.<sup>12</sup> El literal de los principales componentes de esta clase nos dará referencia de su estructura y función:

```
class CTransaction
{
public:
    // Default transaction version.
    static const int32_t CURRENT_VERSION=2;
    static const int32_t MAX_STANDARD_VERSION=2;

    const int32_t nVersion;
    const std::vector<CTxIn> vin;
    const std::vector<CTxOut> vout;
    const uint32_t nLockTime;

private:
    /** Memory only. */
    const uint256 hash;
```

---

<sup>12</sup> A la fecha de escritura de este trabajo (abril 2017), el código fuente de Bitcoin Core se encuentra en GitHub (<https://github.com/bitcoin/bitcoin/tree/master/src>). En su directorio `primitives` se encuentran sendos ficheros `.h` y `.cpp` para `transaction` y `block`, dos de las estructuras más determinantes del sistema Bitcoin/Blockchain. El código que se transcribe, licenciado bajo Licencia MIT, se ha obtenido de este repositorio en el mes de abril de 2017.

```
...  
};
```

donde se ha eliminado o atenuado alguna parte del código que de momento no nos interesa para nuestros propósitos. De manera análoga, aunque omitimos el código, podremos incidir en los principales componentes de los elementos genéricos de los vectores `vin[]` y `vout[]`, objetos de las clases `CTxIn` y `CTxOut`, respectivamente, y que registran los correspondientes inputs y outputs de cada transacción.

De esta forma podremos esquematizar la estructura de una transacción mediante los elementos más significativos de la misma que se reseñan en la siguiente tabla:

Atributo	Visibilidad	Tipo	Descripción
<b>nVersion</b>	public	int32_t	Formato de la transacción, actualmente 2
<b>vin</b>	public	vector<CTxIn>	Contenedor del tipo vector que recoge los inputs de la transacción; el número de elementos es inmediato con <code>vin.size()</code>
<b>vout</b>	public	vector<CTxOut>	Semejante al atributo anterior, recoge los outputs de la transacción, obteniendo directamente el número de sus elementos con <code>vout.size()</code>
<b>nLockTime</b>	public	uint32_t	Tiempo mínimo antes del cual la transacción no puede ser incluida en un bloque
<b>hash</b>	private	uint256	Registro del hash en 32 bytes del contenido de la transacción como identificativo, o TxID, de la misma

Tabla 4.1: principales componentes de una transacción en la clase `CTransaction`

El atributo `nLockTime` indica el momento a partir del cual una transacción debe ser lanzada a la red para su inclusión en un bloque. Usualmente su valor es cero, lo que indica que puede ser transmitida de manera inmediata. Si su valor es inferior a  $5 \cdot 10^8$  entonces indica el número del futuro bloque en el que debería incluirse, mientras que cualquier otro valor es indicador del tiempo (en formato marca de tiempo UNIX Epoch) a partir del cual debe tramitarse su inclusión en la cadena de bloques.

Por su parte, el atributo `hash` contiene el resultado de la doble aplicación de la función *hash* SHA-256 sobre la totalidad del contenido de la transacción. Este valor servirá como referencia o identificador de la misma ante todo el sistema Bitcoin/Blockchain, y normalmente nos referiremos a él como ID de la transacción o TxID.

En cuanto a los vectores `vin[]` y `vout[]` de objetos respectivos de las clases `CTxIn` y `CTxOut`, pasamos a revisar cada uno de éstas.

Como ya se ha comentado, una transacción se compone de diferentes inputs y outputs, representando los inputs los orígenes de los valores transferidos y los outputs sus aplicaciones. Cada input es representado en el código de Bitcoin con la clase `CTxIn`, mientras que cada output corresponde a un objeto de la clase `CTxOut`.<sup>13</sup>

---

<sup>13</sup> La definición de ambas clases se encuentra en `bitcoin/src/primitives/transaction.h`, en el correspondiente proyecto de GitHub.

Veamos un esquema de los principales elementos de la estructura de un objeto de la clase `CTxIn`:

Atributo	Visibilidad	Tipo	Descripción
<b>hash</b>	public	uint256	Referencia, mediante su atributo <code>hash</code> , o TxID, a una transacción anterior
<b>n</b>	public	uint32_t	Un input de una transacción es un output de otra; este atributo <code>n</code> representa el índice del output (no gastado) en cuestión en el vector de outputs contenido en la transacción identificada por el atributo <code>hash</code> anterior
<b>scriptSig</b>	public	CScript	Script con el que se satisfacen las condiciones de gasto del output no gastado, o UTXO, referenciado por el par ( <code>hash</code> , <code>n</code> ) sobre una transacción anterior (ver los dos atributos anteriores)
<b>nSequence</b>	public	uint32_t	Sin función real actual, con valor usual asignado de <code>0xFFFFFFFF</code> que permite considerar la transacción en la que el input está incluido como registrable en la blockchain

Tabla 4.2: elementos de un input de una transacción según la clase `CTxIn`

En realidad, los atributos `hash` y `n` forman parte actualmente de otra estructura de datos implementada por la clase `COutPoint`,<sup>14</sup> que abstrae la referencia al output de una transacción anterior definido por su par (`hash`, `n`), y que se utiliza como input en la transacción en curso. Esta clase `COutPoint` se instancia en el atributo `prevout` de los objetos de la clase `CTxIn`, lo que no es relevante a la hora de esquematizar esta última.

El atributo `scriptSig`, por su parte, recoge una de las dos partes en las que se fundamenta prácticamente todo el sistema Bitcoin/Blockchain al suponer la certificación de la propiedad del valor transferido por el cumplimiento de las condiciones para su gasto, condiciones que se exigen en su contraparte `scriptPubkey` que encontramos en los objetos de la clase `CTxOut`, es decir, en el correspondiente output de una transacción anterior. Su tipo, `CScript`, corresponde a una extensa clase que debe recoger y gestionar las complejidades de este mecanismo bajo la forma de un script serializado, es decir, convenientemente formateado para su transmisión por la red. Veremos este mecanismo de scripts, o guiones, con algo más de detalle en el apartado siguiente.

Por su parte, el esquema de los principales elementos de la estructura de un objeto de la clase `CTxOut` se resume en la siguiente tabla:

---

<sup>14</sup> También definida en `bitcoin/src/primitives/transaction.h`.



Atributo	Visibilidad	Tipo	Descripción
<b>nValue</b>	public	CAmount = int64_t	Cantidad en <i>satoshis</i> , o cienmillonésimas partes de un bitcoin, del UTXO o output no gastado, y que serán reclamados por algún input en una transacción posterior
<b>scriptPubKey</b>	public	CScript	Script que especifica las condiciones que deberá cumplir el input de una futura transacción, a través de su script <i>scriptSig</i> , para que pueda movilizarse el valor no gastado del output representado en <i>nValue</i>

Tabla 4.3: elementos de un output de una transacción según la clase CTxOut

*nValue* es el único vínculo del mundo inmaterial de Bitcoin como unidad monetaria virtual con la realidad, como hemos avanzado en la introducción a este capítulo.

En cuanto al script *scriptPubKey*, ya nos hemos referido a él cuando lo calificábamos como contraparte de *scriptSig*: mientras que el primero especifica las condiciones que deben cumplirse para movilizar el valor mantenido en el output no gastado, el último debe ratificar el cumplimiento de tales condiciones en el momento de su gasto. Generalmente, esta ratificación pasa por demostrar la posesión de la clave privada par a la pública implícita en una dirección Bitcoin mediante la oportuna comprobación. Veremos este caso general con más detalle en el siguiente apartado.

Existe un segundo tipo de transacción muy importante en Bitcoin, la **transacción *coinbase***, que es la que da entrada al sistema a los bitcoins de nueva creación con los que se remunera al minero que valida un bloque mediante la *proof-of-work* para añadirlo a la cadena de bloques. La estructura de datos de este tipo de transacciones es prácticamente idéntica a la de las transacciones regulares, pero el contenido de alguno de sus campos es diferente.

Dada la naturaleza de las transacciones *coinbase*, no existe referencia a un output no gastado, por lo que solo existirá un input en este tipo de transacciones y los elementos del par (hash, n) toman siempre los siguientes valores:

hash = 0  
n =  $2^{32}-1$

Además, el atributo *scriptSig*, que para estas transacciones se suele denominar como *coinbase*, no necesita registrar la confirmación de las condiciones de gasto puesto que no existe un output previo de referencia. En su lugar, registra en primer lugar el número de bloque al que se asigna la transacción, utilizando el resto del espacio (variable) que puede ocupar en registrar datos arbitrarios. En particular, complementa al campo *nonce* que vimos en el capítulo anterior cuando nos referíamos a cómo se implementaba la *proof-of-work*: se va variando el contenido de *nonce* conforme se aplica la función *hash* para encontrar un valor con el que se satisface que el resultado de la función tiene un determinado número de ceros iniciales. El problema está en que el atributo *nNonce* de la clase CBlockHeader que implementa este *nonce* en la cabecera de un bloque Bitcoin/Blockchain es del tipo uint32\_t, de 4 bytes, y que por tanto solo puede registrar  $2^{32}$  valores diferentes. Puesto que el espacio de soluciones diferentes de la

función *hash* aplicada (una doble SHA-256) alcanza un tamaño de  $2^{256}$  elementos, es evidente que pueden agotarse todos los valores posibles del *nonce* sin llegar a una solución, por lo que a partir de esta potencial situación se utiliza el espacio para información arbitraria en el campo *coinbase* con el objeto de modificar el contenido de la transacción y así expandir los resultados de la aplicación de la función *hash* hasta poder encontrar una solución.

La transacción *coinbase* es la primera transacción de cada bloque. Por norma general, solo registra un output, en el que *nValue* registrará el importe a recibir por el minero que valide la *proof-of-work* (12,5 bitcoins en estos momentos), y *scriptPubKey* planteará, de la manera usual, las condiciones específicas de gasto de este valor vinculándolo a una dirección Bitcoin gestionada por el propio minero. No obstante nada impide que los bitcoins de nueva creación se distribuyan en varios outputs.

Para terminar el apartado resultará conveniente resumir los esquemas vistos en una sola representación de lo que podría ser una **transacción regular**, con *m* inputs y *k* outputs:

Transacción		
nVersion		
vin[]	vin[0]	hash_0
		n_0
		scriptSig_0
		nSequence_0
	...	...
	vin[m-1], m=vin.size()	hash_m-1
		n_m-1
		scriptSig_m-1
nSequence_m-1		
vout[]	vout[0]	nValue_0
		scriptPubKey_0
	...	...
	vout[k-1], k=vout.size()	nValue_k-1
		scriptPubKey_k-1
nLockTime		
hash		

Tabla 4.4: principales componentes de la estructura completa de una transacción

Es interesante observar en este punto que hasta ahora no hemos visto atributos que soporten dirección o clave pública alguna en el sentido de identificación de transmisentes y beneficiarios, tal y como hemos repasado en el apartado 4.1. Es de suponer, por tanto, que estas direcciones o claves tendrán que residir en los únicos atributos que no hemos visto en profundidad, los que registran los *scripts* tanto en los inputs como en los outputs. Esta suposición es correcta.

### 4.3 El núcleo de la transferencia de valor: inputs, outputs, scripts

Hemos visto que tanto inputs como outputs cuentan con un atributo del tipo *CScript* que recoge un script, o programa, con semánticas diferentes pero complementarias:

- **scriptPubKey**, atributo de la clase CTxOut que implementa un output, habilita un guión que representa las condiciones que deben cumplirse para poder disponer del valor correspondiente reflejado en **nValue**, es decir, que el propietario del output pueda movilizar esta partida de moneda.
- **scriptSig**, atributo de la clase CTxIn que implementa un input, habilita un guión que debe demostrar el cumplimiento de las condiciones exigidas por el **scriptPubKey** de una transacción previa, es decir, que el beneficiario de un output no gastado demuestre su derecho a disponer del output con este guión.

Antes de ver cómo se implementan y se ejecutan los scripts, revisemos la interrelación entre transacciones, es decir, entre outputs e inputs, que ya hemos podido anticipar. Consideremos dos transacciones Tx\_1 y Tx\_2, con Tx\_1 registrada en un instante de tiempo anterior al de Tx\_2. El identificador de Tx\_1, es decir, su hash (doble SHA-256) sobre el contenido público de la transacción, es **hash\_Tx\_1**. Esta transacción cuenta con **m1** inputs y **k1** outputs. Por su parte, Tx\_2 cuenta con **m2** inputs y **k2** outputs. En la figura 4.4 representamos la estructura resumida de ambas transacciones.

En Tx\_1 existe un output no gastado hasta ahora (el **vout[k1-1]** para nuestro ejemplo) cuyo propietario quiere transferir su valor a un tercero construyendo la transacción Tx\_2. En esta misma transacción el mismo propietario también consume otros outputs (hasta **m2** outputs en total) de otras transacciones previas para otras transferencias de valor, que darán lugar al conjunto de **k2** outputs de la transacción que está implementando, pero puesto que la mecánica es idéntica, en este ejemplo solo revisaremos la relación entre el output **vout[k1-1]** y el input **vin[0]**.

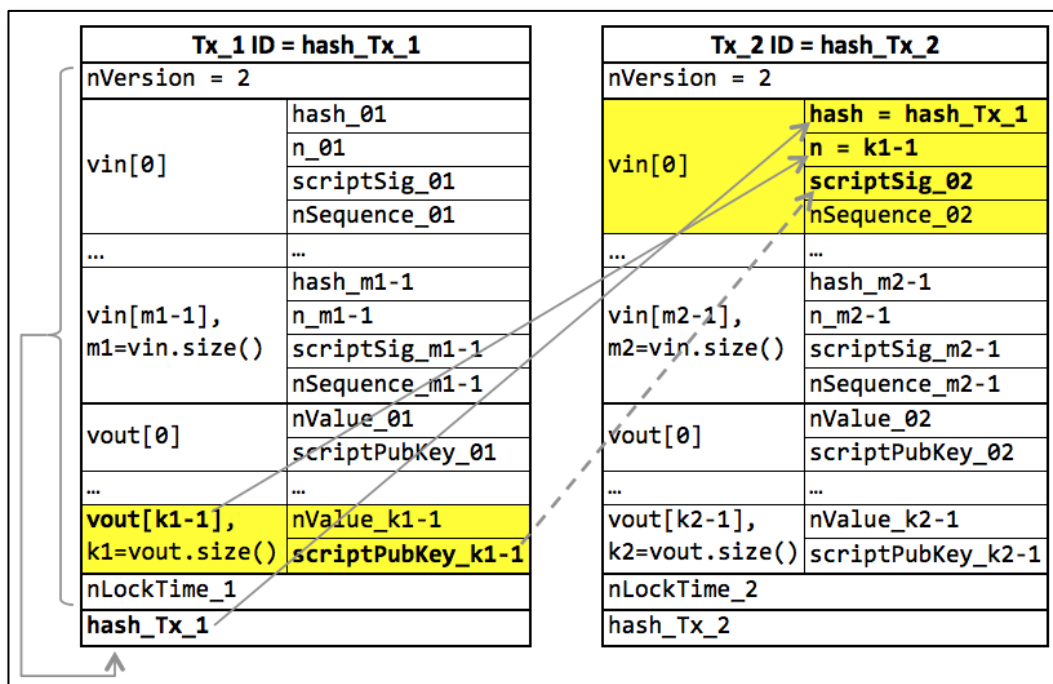


Fig. 4.4: relación entre transacciones, inputs, outputs y scripts

En la figura 4.4 es inmediato apreciar la vinculación entre los outputs no gastados de una transacción (UTXOs) y los inputs que los consumen en otra posterior.

Por una parte, el input `vin[0]` de `Tx_2` consume el output `vout[k1-1]` de `Tx_1` referenciándolo por el identificador de la transacción, `hash_Tx_1`, así como por el índice de tal output en el vector `vout[]` de `Tx_1`, que vemos es `k1-1`. De esta manera, el par (`hash`, `n`) que veíamos en el apartado anterior como referencia inequívoca a un determinado output de una transacción anterior, queda instanciado en este ejemplo como (`hash_Tx_1`, `k1-1`). En la figura estas relaciones quedan explícitas con las flechas sólidas que van de la transacción anterior a la que se está construyendo ahora, la `Tx_2`. Es importante recordar que todo el valor de `vout[k1-1]` representado por `nValue_k1-1` se consume en la transacción `Tx_2`, en la que sus inputs ni siquiera manejan atributo alguno de importe.

De otro lado, y como repasábamos al inicio de este apartado, `scriptPubKey_k1-1` está vinculado (línea discontinua en la figura 4.4) a `scriptSig_02`. Como resumen previo a lo que veremos a continuación, esta vinculación consiste en la ejecución secuencial de ambos scripts en un determinado orden. Si el resultado booleano final de la ejecución es `TRUE`, entonces significará que `scriptSig_02` ha tenido éxito en cumplir el condicionado de disposición o gasto que planteaba `scriptPubKey_k1-1` y que el gasto planteado por `Tx_2` es correcto. Veámoslo con más detalle.

Los scripts que se manejan en las transacciones, tanto en los outputs como en los inputs, se escriben en un lenguaje limitado, con ejecución basada en pila, no *Turing-completo*<sup>15</sup> y que, entre otras limitaciones, no admite bucles para garantizar que todo guión se ejecute en su totalidad y no pueda resultar *colgado*. No se trata, por tanto, de un lenguaje de programación de propósito general sino de uno diseñado específicamente para Bitcoin/Blockchain. Se interpreta de izquierda a derecha y existe un conjunto de algunas decenas de instrucciones, u operaciones, que permiten una alta flexibilidad en la semántica de los guiones.

No obstante esta alta flexibilidad, solo existe un pequeño conjunto de distintas aplicaciones permitidas, entre las que cabe destacar la genérica de permitir una transferencia básica de valor entre dos partes. Para esta última, la más utilizada en las transacciones de bitcoins, se toma en consideración el siguiente procedimiento conceptual:

- El `scriptPubKey` del output no gastado en cuestión, presenta una clave pública bajo su forma *hash* (`RIPEMD-160(SHA-256(pubKey))`), formato también denominado `pubKeyHash` (ver figura 4.2; el ordenante inicial, en su momento, habrá obtenido este hash de la *decodificación Base58* de la dirección *P2PKH*). Esta clave pública habrá sido calculada desde una clave privada solo conocida por el beneficiario del UTXO, y por lo tanto solo podrá ser él quien movilice su valor. El script también contiene algunas operaciones auxiliares que enseguida revisaremos.
- El propietario, para movilizar el valor en una nueva transacción, presenta un `scriptSig` que contiene dos elementos: una firma digital ECDSA de la transacción que acaba de construir y el literal de su clave pública sin ningún tipo de tratamiento.

---

<sup>15</sup> Informalmente, diremos que un lenguaje de programación es Turing-completo si con él puede simularse cualquier máquina de Turing de una única cinta.

- A la vista del contenido del `scriptSig`, cualquiera, y en particular los mineros que validarán la transacción (es decir, la validez de sus inputs y outputs), podrá comprobar que:
  - Un hash de la clave pública, calculado con las mismas funciones que las utilizadas para obtener `pubKeyHash` en el `scriptPubKey`, coincide exactamente con este último. De esta manera se confirma la consistencia de la clave pública en ambos scripts.
  - La firma digital sobre la transacción es correcta, comprobación que puede realizarse puesto que la firma digital se acompaña con su clave pública. De esta manera se manifiesta el conocimiento de la clave privada que habilita la movilización del valor asignado a su correspondiente clave pública.

Esta tipología de scripts que acabamos de ver configura lo que se denomina transacción *Pay-to-PubkeyHash*, o *P2PKH*, puesto que se sirve del formato de clave pública como dirección de igual denominación (ver apartado 4.1).

Veámoslo en términos de programación de los scripts. El tipo de transacciones *P2PKH* registra los siguientes guiones prefijados:

```
scriptPubKey:    OP_DUP    OP_HASH160    <pubKeyHash>    OP_EQUALVERIFY
                  OP_CHECKSIG
scriptSig: <signature> <pubKey>
```

Los operadores, o instrucciones, del lenguaje de script empiezan por el prefijo `OP_`, mientras que la información del exterior aportada al mecanismo se representa entre `<>`. En este último caso, el intérprete del script se limita a introducir la información aportada en la pila, a la espera de que sea consumida por algún operador. Todos los operadores actúan sobre el *top* o sobre los elementos superiores de la pila, dependiendo del número de parámetros sobre los que trabajen (que normalmente son uno o dos).

En primer lugar se ejecuta el script `scriptSig`. En este caso solo se trata de introducir en la pila de ejecución, por el orden en que se encuentran, `<signature>` (la firma digital sobre la transacción) y `<pubKey>` (la clave pública). Si la ejecución de `scriptSig` es correcta, se carga `scriptPubKey` y se continúa con su ejecución. Terminada ésta, se comprueba que el único valor que queda en la pila es el booleano `TRUE`, que corresponderá al resultado final de algún operador. Si así fuera para todos los pares de scripts de la transacción, ésta se dará por correcta y podrá pasarse para su registro definitivo en el bloque. Si alguna de las ejecuciones de algún script de la transacción ofrece un resultado distinto a `TRUE`, la transacción se rechaza. Encontramos un resumen de la ejecución de estos scripts en la siguiente tabla:

Script	Paso	Script pendiente de ejecutar	Pila	Notas
scriptSig	0	<signature> <pubKey>		Situación inicial: cargado scriptSig
	1	<pubKey>	<signature>	Primer paso: introducido elemento en la pila
	2		<pubKey> <signature>	Introducido elemento en la pila. Script terminado sin problemas
scriptPubKey	3	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	<pubKey> <signature>	Se carga scriptPubKey
	4	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	<pubKey> <pubKey> <signature>	La ejecución de OP_DUP duplica el elemento top de la pila
	5	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	<pubKeyHash_n> <pubKey> <signature>	La ejecución de OP_HASH160 realiza HASH160(SHA256(top)), elimina el top, y apila el resultado
	6	OP_EQUALVERIFY OP_CHECKSIG	<pubKeyHash> <pubKeyHash_n> <pubKey> <signature>	Introducido elemento <pubKeyHash> en la pila
	7	OP_CHECKSIG	<pubKey> <signature>	Ejecutando OP_EQUALVERIFY se verifica que el hash obtenido en el paso anterior, que provenía del input, y el que provenía del output no gastado son idénticos
	8		TRUE	Ejecutando OP_CHECKSIG confirmamos que la firma digital es correcta para la clave pública presentada, lo que garantiza que la clave privada corresponde a la clave pública consignada en el UTXO

Tabla 4.5: ejecución de los scripts en una transacción P2PKH

De la misma manera que en el apartado 4.1 identificamos un segundo tipo de dirección Bitcoin, Pay-toScriptHash o P2SH, que recordemos permitía habilitar complejos condicionantes de disposición del valor mediante un script adecuado sin que recayera esta tarea en el transmitente del mismo, identificamos también un segundo tipo de transacción directamente vinculada. En este caso el formato de los scripts queda:

**scriptPubKey:** OP\_HASH160 <scriptHash> OP\_EQUAL  
**scriptSig:** <signature/s> <script>

En este caso <scriptHash> es el resultado de longitud 20 bytes obtenido tras la aplicación de las funciones hash sobre el script de disposición (ver figura 4.3); <signature/s> es un conjunto de una o más firmas digitales, en cuyo caso pueden presentarse si hay necesidad de *multifirma* para confirmar la validez de las correspondientes claves públicas; finalmente, <script> es el guión, en formato serializado, que tendrá que satisfacer que su hash coincide con el aportado por scriptPubKey. Su ejecución requiere un paso adicional específico (es sencillo ver como tras OP\_EQUAL todavía queda el elemento <signature/s> en la pila), que supondrá la carga y ejecución del script de disposición, <script>, que generalmente consistirá en la validación de una serie de claves públicas contra <signature/s>.

Además de las dos vistas, así como de la transacción especial *coinbase*, existen otros tipos de transacciones adicionales admitidas por el sistema<sup>16</sup> (*multisig*, *Pay-to-PubKey*

<sup>16</sup> La implementación de referencia Bitcoin Core recoge varias funciones de comprobación de la adecuación de las transacciones a lo admitido en cada momento por la comunidad en los ficheros policy.cpp y policy.h, radicados ambos en el directorio del código fuente bitcoin/src/policy/.

(*P2PK*), *Nulldata*...), pero no son utilizadas frecuentemente y quedan fuera de nuestro ámbito.

Los scripts vistos, los más numerosos en Bitcoin/Blockchain, apenas recogen una fracción de la complejidad que puede alcanzarse con este lenguaje,<sup>17</sup> cuya referencia completa puede revisarse en [26], pero habrá que esperar, llegado el caso, a que la comunidad Bitcoin acepte futuras composiciones más complejas para contemplar un abanico verdaderamente amplio de posibilidades de tratamiento del valor.

#### **4.4 La red en Bitcoin/Blockchain**

Una vez construida una transacción, y tras su serialización, procede transmitirla a toda la red Bitcoin para su registro final tras su inclusión en un bloque y la adición de éste en la cadena de bloques. Realizado esto último, sabemos que la transacción irá ganando solvencia conforme se añadan más bloques sobre el que la incluye ya que el trabajo computacional que supondría su modificación resultaría ímprobo tras unos pocos bloques adicionales.

Aunque para esta red suele utilizarse de manera generalizada el calificativo *peer-to-peer*, la red completa de Bitcoin está compuesta por una variada tipología de nodos fundamentada en sus diferentes funcionalidades. No obstante, el nodo patrón, el Bitcoin Core diseñado y codificado activamente por la comunidad de desarrolladores de Bitcoin y dotado de todas las principales funcionalidades, es la referencia *peer* en esta tecnología, y cada administrador activa, utiliza, o desactiva sus diferentes funciones o roles según su voluntad. Aparte de este nodo de referencia existen multitud de implementaciones más o menos especializadas que permiten gran flexibilidad a la hora de implementar un nodo para la red Bitcoin de acuerdo a las necesidades de cada usuario.

Las principales funciones que encontramos en los diferentes nodos de esta red son:

- Minado de bloques: los denominados nodos mineros son los encargados de confirmar la corrección de las transacciones, incluirlas en el bloque en construcción con la frecuencia aproximada de un nuevo bloque cada diez minutos, y resolver la prueba de trabajo para la inclusión del bloque en la blockchain. Esta posible resolución (uno de entre los miles de nodos mineros lo logrará) está remunerada con bitcoins de nueva creación y con los *fees* implícitos en las transacciones.
- *Pools* de minado: relacionado con lo anterior, son organizaciones que agregan la potencia computacional de minado de numerosos mineros individuales, incrementando la probabilidad de éxito en la solución de la *proof-of-work* y repartiendo las posibles remuneraciones obtenidas de manera proporcional a la potencia aportada por cada miembro.
- Enrutado: en general, todos los nodos de la red, excepto los más ligeros, implementan la función de enrutado de los mensajes y, sobre todo, las

---

<sup>17</sup> La referencia completa del lenguaje puede encontrarse en [25]. Como podrá observarse, el número de operadores que se utilizan en los scripts que hemos revisado es marginal con respecto al número de los existentes en este lenguaje.

transacciones Bitcoin y bloques resueltos que reciben, es decir, que extienden a nodos topológicamente vecinos las nuevas transacciones y los bloques, siempre que sean correctos, de tal manera que la práctica totalidad de la red comparte, más tarde o más temprano, el mismo estado del sistema global.

- Blockchain completa: un nodo completo, o *full node*, tendrá que registrar toda la cadena de bloques para que sea eficiente y no tenga que realizar búsquedas y descargas en la red para sus operaciones. Puesto que la blockchain completa a abril de 2017 alcanza los 110 GB de tamaño, los nodos ligeros, como los *wallets* en dispositivos móviles, no disponen de la estructura completa. Para la modalidad de verificación simplificada de pagos, tal como la vimos en el apartado 2.5 del capítulo 2 y cuyas implicaciones en la red revisaremos algo más adelante, solo es necesario mantener el registro de las cabeceras de los bloques y no la totalidad de las transacciones en sí, lo que reduce las necesidades de almacenamiento a niveles asequibles para cualquiera.
- *Wallet*: la función de monedero es prácticamente universal. Consiste en el registro y gestión del conjunto de claves, y por tanto direcciones, a las que se asignan los disponibles de bitcoin. En general, todos los que operan sobre la red, sean mineros, prestadores de otros servicios, o usuarios, necesitan como mínimo un *wallet*.

Algunos nodos trabajan con todas las funcionalidades; otros se limitan a funciones específicas como las de monedero para un usuario que tan solo desea gestionar sus depósitos de bitcoins. Por otra parte, existen también nodos especializados o servidores de propósitos menos generalistas, como el soporte de protocolos especializados para *pools* de mineros o la explotación de la información subyacente en la blockchain, y que quedan fuera de nuestro alcance.

El sistema Bitcoin/Blockchain no implementa una base de datos distribuida, sino un registro, la cadena de bloques, inmensamente replicado en cada nodo que lo descarga y lo utiliza. Su protocolo completo, en el que no vamos a profundizar (tampoco lo haremos en cuestiones tales como la serialización de los diferentes mensajes y objetos), puede analizarse por sus principales áreas de aplicación:

- Descubrimiento de la red: al incorporar, o reincorporar, un nodo a la red, debe conectarse con un conjunto de nodos completos para poder desarrollar sus funciones. Estos nodos pueden basarse en los conocidos previamente (si el nodo se reincorpora) o en algunos nodos *semilla* localizables por DNS o por indización directa en el código (si el nodo se incorpora por primera vez), que a su vez podrán informar de otros nodos activos a los que conectarse.
- Actualización: un nodo recién conectado o reconectado necesita actualizar su cadena de bloques, para lo que existen mecanismos de consulta a nodos *peers* y de transmisión de los bloques faltantes (incluida, en su caso, la blockchain completa).
- Transacciones: una vez elaboradas por el nodo correspondiente, son transmitidas al conjunto de nodos de los que se tiene conocimiento para su validación y retransmisión al resto de la red mediante *flooding*,<sup>18</sup> de tal manera que en poco

---

<sup>18</sup> El *flooding*, o inundación, es un mecanismo de retransmisión de determinada información por el que un nodo transmite a todos los nodos con los que está conectado, excepto al remitente, la información recibida de este último.



tiempo prácticamente toda la red tiene conocimiento de la transacción con el objetivo final de su inclusión en el siguiente bloque de la cadena.

- Bloques: una vez resuelta la *proof-of-work* por un determinado minero, éste transmite el bloque a la red, también mediante *flooding*, para que cada nodo que tenga una copia de la cadena de bloques lo incorpore a ésta.
- Verificación de pagos simplificada (*Simplified Payment Verification*, SPV): un nodo ligero que haga uso de SPV necesitará mecanismos para obtener y actualizar la cadena de cabeceras de bloques, con las que podrá confirmar que una determinada transacción existe y que ha sido confirmada por varios bloques posteriores (pero no podrá confirmar directamente que existan los correspondientes UTXOs). También se habilita un mecanismo, denominado *filtrado bloom*, por el que las consultas sobre transacciones que incluyan una determinada dirección quedan enmascaradas para que un potencial observador no pueda sacar conclusiones sobre qué nodo de la red está interesado en qué direcciones, es decir, en vincular determinadas posiciones de bitcoins con determinadas direcciones de red.
- *Pools* de minado: se trata, en realidad, de protocolos adicionales sobre el canónico de Bitcoin para gestionar el minado por agregación de mineros individuales con un interés común. El conjunto de estos protocolos más los ordinarios de explotación de la red Bitcoin se denomina en alguna literatura como la *red extendida Bitcoin*.

Vista la clasificación de los nodos y sus funciones en este apartado, nos interesa particularmente el minado de bloques puesto que supone el objetivo natural de cualquier transacción en Bitcoin/Blockchain.

#### 4.5 Minería de bloques

En los apartados anteriores hemos visto cómo un transmitente de valor construye una transacción consumiendo UTXOs y generando otros nuevos, y cómo comunica tal transmisión de valor al sistema Bitcoin/Blockchain mediante la red *peer-to-peer*. De esta manera, a cualquier nodo minero alcanzará el conocimiento de la transacción en cuestión en un breve periodo de tiempo desde su comunicación a los primeros nodos. En general, los nodos mineros estarán intentando resolver la *proof-of-work*, o PoW, del bloque en curso, que recogerá un conjunto ya cerrado de transacciones previas, probablemente diferente para cada nodo, por lo que a la recepción de cada nueva transacción ésta quedará incluida en un pool o conjunto de transacciones pendientes de considerar, a la espera de ser incluidas en el siguiente bloque a validar. Viendo, por tanto, la trascendencia del bloque en Bitcoin/Blockchain, comenzaremos este último apartado revisando su estructura, lo que nos dará las bases para analizar la actividad de un nodo minero un poco más adelante.

Si estructuralmente veíamos una transacción como un conjunto de inputs y otro de outputs, un bloque es un conjunto de transacciones con algunos campos instrumentales adicionales. Estos últimos quedan recogidos en la denominada cabecera del bloque (*block header*), mientras que las transacciones se registran en un contenedor de tipo vector.

De la misma manera que hacíamos con las transacciones, el código de un bloque<sup>19</sup> nos aclara sus principales componentes. En este caso encontramos en primer lugar la definición de la clase CBlockHeader que implementa la cabecera de un bloque:

```
class CBlockHeader
{
public:
    // header
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;
    ...
};
```

donde se ha eliminado parte del código no relevante para nuestros fines. Heredada de la clase anterior se define la clase CBlock, que implementa un bloque completo:

```
class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransactionRef> vtx;
    ...
    CBlockHeader GetBlockHeader() const
    {
        CBlockHeader block;
        block.nVersion      = nVersion;
        block.hashPrevBlock = hashPrevBlock;
        block.hashMerkleRoot = hashMerkleRoot;
        block.nTime         = nTime;
        block.nBits         = nBits;
        block.nNonce        = nNonce;
        return block;
    }

    std::string ToString() const;
};
```

donde también se ha eliminado o atenuado código no relevante ahora. Como vemos, CBlock recoge, adicionalmente a los atributos heredados de CBlockHeader, el vector de transacciones (o más precisamente, de referencias o punteros a transacciones) vtx[ ].

Podemos establecer, por tanto, el siguiente esquema de los componentes de un bloque:

---

<sup>19</sup> Directorio bitcoin/src/primitives, ficheros block.h y block.cpp, del código fuente de Bitcoin Core.

	Atributo	Visibil.	Tipo	Descripción
Header/Cabecera	nVersion	public	int32_t	Formato del bloque, habitualmente del tipo 2, aunque pueden contemplarse otras versiones adicionales
	hashPrevBlock	public	uint256	Hash (doble SHA-256) de la cabecera del bloque previo, que debe cumplir con las restricciones exigidas por la PoW
	hashMerkleRoot	public	uint256	Raíz del árbol de Merkle (mediante doble SHA-256) de todas las transacciones recogidas en el bloque en vtx[]
	nTime	public	uint32_t	Marca de tiempo en formato UNIX Epoch que implementa la certificación de tiempo diseñada para Bitcoin (ver apartado 2.2, capítulo 2), y que recoge el tiempo (aproximado) de creación del bloque
	nBits	public	uint32_t	Representación del nivel de dificultad de la PoW (la representación numérica del hash del bloque debe ser inferior a este nivel); su formato $0xh_0...h_7$ se interpreta como el nivel de dificultad $0xh_2...h_7 \cdot 2^{8 \cdot (h_0 h_1 - 3)}$
	nNonce	public	uint32_t	32 bits que se varían aleatoriamente para encontrar el resultado hash de la PoW que sea inferior al nivel de dificultad (ver limitaciones sobre su tamaño en apdo. 4.2)
	vtx[]	public	vector	Vector de genéricos <CTransactionRef> que referencian las transacciones del bloque; es inmediato el número de transacciones mediante vtx.size()

Tabla 4.6: principales componentes de un bloque implementado por la clase CBlock

Una vez revisada la estructura y las funciones de cada elemento del bloque podemos considerar el siguiente bucle de trabajo de un nodo minero tipo:

1. Recoge en un pool de transacciones pendientes de tratar todas las recibidas antes de la verificación, mediante la *proof-of-work*, del actual bloque en construcción, hasta que sea resuelta
2. Una vez resuelta la PoW:
  - i. Si lo ha resuelto otro nodo, recibirá el bloque por la red, confirmará el resultado *hash* de la PoW, comprobará sus transacciones, y si todo es correcto cesará su minado inconcluso del bloque y añadirá el recibido a su cadena más larga de bloques
  - ii. Si la ha resuelto el mismo nodo considerado, añadirá el bloque a su propia cadena y lo transmitirá a la red para que cada nodo proceda a su inclusión
  - iii. Si posteriormente recibe otro bloque válido de igual *peso* (con igual hashPrevBlock), significará que se ha producido una bifurcación en la cadena, y guardará registro de este bloque alternativo por si la mayor cadena futura tomara este bloque como base o, en caso contrario, para incluir sus transacciones no procesadas, si existieran, en siguientes bloques

3. Recogerá desde el pool de transacciones pendientes de registrar, y según criterios particulares de cada nodo (por los que normalmente se priman las transacciones con mayores *fees*), el conjunto de transacciones a incluir en el siguiente bloque a construir
4. Valida cada transacción (lo que implica, entre otras muchas acciones (ver anexo B), validar los scripts de cada input y output) y empieza a construir el nuevo bloque particular a someter a la PoW
5. A las transacciones anteriores les añade, como primera transacción, la *coinbase* que, en caso de éxito con la PoW, le permitirá recibir la remuneración establecida por la adición del bloque a la blockchain
6. Del conjunto de transacciones así conformado obtiene la raíz de su *Merkle tree*
7. Recupera el hash del último bloque recién minado y lo registra en `hashPrevBlock`; registra `nTime`
8. Con todos los elementos necesarios ya conocidos, y asignando un valor inicial aleatorio a `nNonce`, construye la cabecera del nuevo bloque
9. Comienza a aplicar la doble SHA-256 sobre la cabecera obtenida con el objetivo de encontrar un resultado ajustado a la PoW, es decir, inferior al nivel de dificultad corriente aportado por los cálculos efectuados con los tiempos medios cada 2016 bloques y representado por `nBits`, modificando progresivamente el contenido de `nNonce` y, en caso necesario, el del campo *coinbase* de la transacción del mismo nombre, y ello mientras no se obtenga el resultado buscado
10. Recomienza el bucle de minado

Para finalizar este recorrido por el sistema Bitcoin/Blockchain nos falta por revisar la blockchain en sí.

#### 4.6 La cadena de bloques

Cuando un nodo tiene éxito en su PoW, o recibe el bloque de otro nodo minero con mayor éxito, el bloque está listo para ser incluido tras el último bloque extremo de la cadena preexistente.

Una representación habitual de la cadena de bloques puede ser la siguiente:

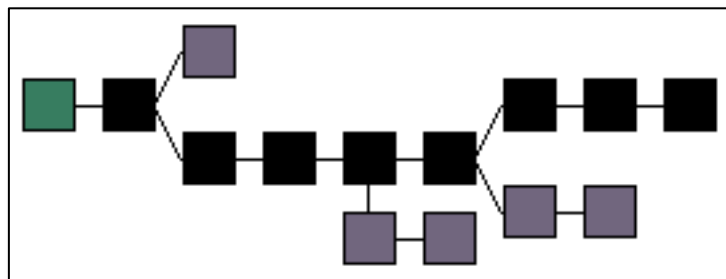


Fig. 4.5: blockchain con cadena principal (bloques negros) y cadenas alternativas (grises) (en.bitcoin.it)

En la figura 4.5 podemos apreciar el primer bloque (bloque cero, o *Génesis*) en color verde, y la cadena principal, la más larga que representa el consenso de la red en la corrección de las transacciones que contiene, con bloques negros. Los grises

corresponderían a bifurcaciones originadas por la solución prácticamente simultánea de la *proof-of-work* en dos nodos mineros distintos, que en ocasiones los veremos denominados como bloques huérfanos una vez conocido que la cadena más larga se desarrolla por otro lado, si bien sus transacciones no son olvidadas y, si no figuran en otro bloque, son rescatadas y tratadas. Como hemos visto anteriormente, los nodos toman en consideración para su blockchain el primer bloque que les llega resuelto, pero guardan el alternativo por si la cadena de consenso finalmente se desarrolla a partir de éste.

Los enlaces entre bloques son implementados por el atributo `hashPrevBlock` de cada bloque, que ya sabemos registra el *hash* resultante de la PoW del bloque anterior.

En términos de almacenamiento, la cadena de bloques de cada nodo está dividida en ficheros denominados `blk*.dat`, cada uno de ellos conteniendo varios bloques en formato serializado para la red, ficheros en los que se van almacenando los nuevos bloques hasta alcanzar un determinado tamaño<sup>20</sup> y habilitar entonces un nuevo `.dat`. Pero un almacenamiento de datos de estas características y magnitud resultaría ineficiente en tareas de búsqueda de información (de un determinado bloque, una determinada transacción, un determinado input o un output no gastado...) si no viniera acompañado de una estructuración adecuada (aunque redundante) de la información que contiene para que resulte fácil y rápidamente obtenible cualquier información necesaria. Aunque esta estructuración difiere según las implementaciones, lo usual es mantener como mínimo las siguientes estructuras de datos adicionales a la blockchain:

- Metadatos sobre los bloques: normalmente una base de datos *key-value*<sup>21</sup> (LevelDB [27] en Bitcoin Core) que permite búsquedas rápidas sobre bloques y la información que contienen.
- Registro de UTXOs: también usualmente bajo la forma de base de datos (y también LevelDB en Bitcoin Core) para facilitar la gestión eficiente del conjunto de outputs no gastados de toda la cadena de bloques, y que permite, entre otras acciones, una ágil comprobación de la corrección de cualquier output presuntamente no gastado o contabilizar los saldos de cualquier *wallet*.
- Registro inverso de elementos de transacciones: bajo la forma de ficheros `rev*.dat`, permite seguir fácilmente la pista hacia atrás por las conexiones entre los outputs no gastados en dirección a sus correspondientes inputs.

No merece la pena extenderse mucho más en detalles puesto que, computacionalmente, el procedimiento de adición de un bloque a la cadena, una vez validado, es razonablemente sencillo. Sin embargo, y siguiendo con términos computacionales, ya conocemos la complejidad previa que cualquier transacción requiere hasta su validación completa. Pero lo realmente importante, lo que trasciende toda la complejidad tecnológica hasta llegar al plano inmaterial de la confianza en la transferencia en el valor, es el consenso que estos mecanismos brindan a terceros, con intereses contrapuestos, sobre el correcto estado global del sistema Bitcoin/Blockchain.

---

<sup>20</sup> El tamaño máximo de estos ficheros se representa en la constante `MAX_BLOCKFILE_SIZE` de `bitcoin/src/validation.h`, que en la actualidad es de 128 MB.

<sup>21</sup> Una base de datos *key-value* indiza sus registros (*value*) de acuerdo a una determinada clave (*key*), relación que se implementa, usualmente, mediante una función *hash*.

#### 4.7 El consenso emergente de la red

Lo que hemos venido contemplando tanto explícita como implícitamente a lo largo de todo este trabajo, es conveniente resumirlo en el apartado final de este capítulo puesto que el *consenso emergente* ([28], cap. 8) y descentralizado ha sido necesario para dar valor y hacer exitoso a Bitcoin.

En palabras de Antonopoulos, *el consenso es un artefacto emergente de la interacción asíncrona de miles de nodos independientes, todos siguiendo reglas simples*, y cuya emergencia, según el mismo autor, proviene de la interacción de cuatro procesos independientes:

- Verificación independiente de cada transacción
- Agrupación independiente de las transacciones en bloques tras un minado basado en *proof-of-work*
- Verificación independiente de cada nuevo bloque y agregación a la blockchain por todos los nodos
- Selección independiente por cada nodo de la cadena con mayor trabajo computacional acumulado y demostrado por la *proof-of-work*

Estos procesos y todo lo que sobre ellos hemos visto aquí, constituyen el detalle de lo que Nakamoto apuntaba en las conclusiones finales de su conocido *paper*: el consenso para Bitcoin (y la solución al *problema de los generales bizantinos* en este contexto, añadimos) se resuelve mediante el voto de una red *peer-to-peer*, proporcional a su potencia de computación, y complementado por (i) la aceptación de la cadena de *proof-of-work* que certifica la corrección de las transacciones que constituyen el devenir de cada unidad monetaria y (ii) el rechazo de cualquier objeto representativo de valor o de su transferencia (transacciones, bloques) que no cumpla las reglas establecidas en cada momento para el mismo o para sus componentes.

# *Aplicabilidad de Blockchain en la industria: un marco de trabajo*

Tras haber revisado los fundamentos tecnológicos del sistema Bitcoin/Blockchain en diferentes niveles de abstracción, en particular su diseño (cap. 2), los componentes criptográficos subyacentes (cap. 3), y su implementación (cap. 4), y tras años de explotación exitosa de Blockchain en su implementación de referencia como soporte de la criptomoneda Bitcoin, completaremos ahora nuestro programa analizando su aplicabilidad conceptual a la industria financiera y las modificaciones que habría que realizar, en su caso, para implementar este caso de éxito en el ecosistema financiero que parece natural (aunque desde luego no único) para esta tecnología. No obstante este éxito, existen inconvenientes en esta tecnología que todavía no hemos revisado.

El análisis que pretendemos lo efectuaremos resumiendo en primer lugar las características básicas que hacen tan especial a esta tecnología, así como las debilidades que se han ido manifestando, sobre todo los que puedan concernir a una extensión de la tecnología a la industria. Esto facilitará la delimitación de un posible marco de actuación que deberá permitirnos poner sus elementos en relación con la realidad *hiperregulada* de cualquier operador financiero, y sacar así las correspondientes conclusiones sobre su aplicabilidad. Terminaremos este capítulo con un breve resumen de los principales proyectos en desarrollo que ya hoy suponen una ebullición de ideas, diseños y prototipos para diferentes industrias, y en particular la financiera. Fruto de todo ello, en el siguiente capítulo veremos las transformaciones que deben tener lugar en Blockchain para su aplicación a cualquier industria regulada.

## *5.1 Características funcionales y estructurales en Bitcoin/Blockchain*

A lo largo de este trabajo hemos podido identificar las siguientes características funcionales relevantes:

- El producto obtenido de la explotación de Blockchain es un **registro histórico de todas las transacciones, público y no encriptado, descentralizado a través una red peer-to-peer, no sujeto a autoridad central** alguna, pública o privada. Se encuentra **replicado** (no distribuido) en miles de nodos repartidos globalmente por medio de Internet en una **red abierta** (*permissionless blockchain*) en la que cualquiera puede participar, y en la que es **impracticable la intervención** por terceros de todo el sistema (y por lo tanto del activo que soporta) por mucha autoridad que éstos representen.
- El registro está diseñado de tal manera que guarda referencia del devenir de cada unidad de **valor** introducida en el sistema, que podría resultar bien dividida en fracciones, bien acumulada a otros valores para obtener uno mayor. Estos contenedores de valor tienen identidad propia y son así mismo objeto de registro y seguimiento. El valor queda representado en cada transacción por cualquier salida, o

**output, que no se haya gastado**, y se vincula a una **dirección** (que podríamos aproximar por una *cuenta* en el sentido financiero) vinculada a una clave pública de un criptosistema de criptografía asimétrica.

- Las **transacciones**, o transferencias de valor, se realizan directamente y **sin intermediarios** entre ordenante y beneficiario, lo que es coherente con la ausencia de autoridades centrales. Se les asigna una **marca de tiempo para evitar un posible doble gasto** por una transacción posterior. Para ello se cuenta con la ayuda de **bloques** contruidos en un determinado periodo (10 minutos), que recogen las transacciones cursadas en ese periodo, y cuyo enlace desde el original hasta el último bloque incorporado conforma el registro o blockchain.
- La **seguridad** del registro queda garantizada por un **consenso** que es resultado de la participación de toda una red *peer-to-peer* en la verificación de cada transacción (imposibilidad de doble gasto), de cada bloque, y de la cadena que incorpore mayor trabajo computacional vía ***proof-of-work***.
- El sistema está orientado a la protección del **anonimato** puesto que la representación mediante clave pública de la identidad de los participantes en las transacciones no se acompaña de su vinculación a la identidad física correspondiente. Ésta, manteniendo su anonimato, certificará su **propiedad** mediante la utilización de su clave privada pareja de la pública en cualquier transacción que movilice el valor.

Blockchain es, como vemos, una tecnología segura que implementa una moneda digital (pero que podría servir de soporte a otros tipos de activos tanto financieros no virtuales como físicos), y que ofrece coherencia en el estado final del sistema, considerado éste como el resultado obtenido tras el tratamiento e inclusión en la cadena de bloques de la última transacción considerada. Desde 2009 se encuentra implementada como soporte de Bitcoin y sus principales incidencias de seguridad han tenido que ver con temas como la custodia de las claves privadas, externas a la propia Blockchain, o por errores en el código, y no por el diseño o la estructura del sistema en sí.

Pero es también conveniente introducir los problemas asociados, que los hay, a las características del sistema, principalmente estructurales, y que procederá evitar en cualquier implementación alternativa. Junto con las propiedades funcionales que acabamos de ver, estos inconvenientes nos aportarán una sólida base de partida para trabajar las posibles transformaciones necesarias en orden a obtener una Blockchain aplicable a la industria:

- **Costes de la minería:** si bien en principio parecería sencillo hacer recuento del número de nodos conectados a la red Bitcoin, existen importantes discrepancias entre las cifras obtenidas.<sup>22</sup> Tampoco parecen existir cifras definitivas sobre los nodos completos o del número de mineros pero, al menos, sí existen sobre los pools mineros, puesto que la minería individual tiene ya poco que hacer actualmente: en el momento de escribir estas líneas los 22 principales pools se reparten el 99,5% de la potencia computacional de la red minera para resolver la ***proof-of-work***, y los 3 o 4

---

<sup>22</sup> Ya desde los primeros años de éxito y extensión de Bitcoin las cifras aportadas experimentan importantes variaciones según las fuentes; por ejemplo, en [29] se pone de manifiesto cuando se refiere a otros trabajos relacionados.



más importantes alcanzan conjuntamente una potencia que podría empezar a comprometer la seguridad de la criptomoneda. El más importante a día de hoy, AntPool, con el 17% de la potencia computacional de la red, declara 97 mil mineros, pero quedaría por definir qué se entiende exactamente por un minero (¿un único elemento hardware que cumple su función *hash*?, ¿una determinada IP?), y en cualquier caso no estarán todos activos seguramente; pero no parece descabellado asumir para toda la red varias decenas de miles, quizá algunos pocos centenares de miles de nodos mineros constituidos por hardware especializado,<sup>23</sup> aunque deberemos insistir en que no es lo mismo que los nodos completos, que parece son sólo unos cuantos miles pero que algunos de ellos pueden incluir, como vemos, decenas de miles de elementos.

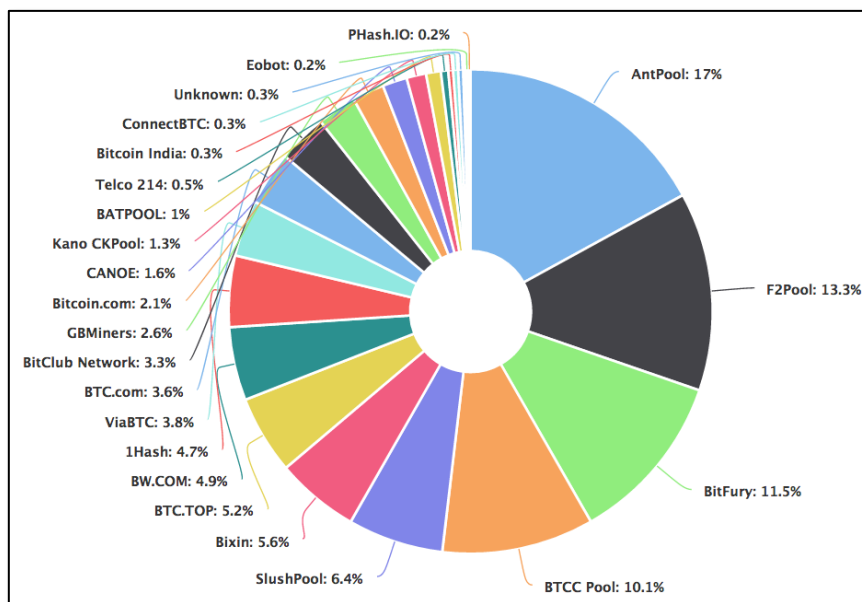


Fig. 5.1: distribución de la potencia de minado de la red Bitcoin  
(<https://blockchain.info/pools>, consultada a mediados de abril 2017)

El consumo energético de todos estos nodos es enorme, y se calculó a mediados de 2016 en unos 400 millones de dólares USA al año [31]. Como es conocido, uno de los principales objetivos de cualquier industria es la minimización de sus costes, y los que aquí nos ocupan, vinculados a la *proof-of-work*, y por lo tanto al consenso, no podrían quedar al margen.

- **Escalabilidad:** de notable actualidad en la comunidad Bitcoin puesto que ya se empiezan a manifestar problemas de rendimiento ante el incremento en el número de las transacciones generadas en el periodo de minado de cada bloque, de unos 10 minutos aproximadamente. El tamaño máximo de un bloque es de 1 MB, lo que establece un máximo en el número de transacciones que puede recoger. Asumiendo que el tamaño mínimo de una transacción (con un solo input y un único output) es de unos pocos centenares de bytes, y conociendo que la media actual de transacciones por bloque alcanza prácticamente las 2.000, todo parece indicar que en 2017 se está alcanzando el máximo de transacciones por día que pueden ser confirmadas mediante

<sup>23</sup> El interesante análisis de 2015 [30] sobre la cuestión del número de mineros establece entre 80 mil y 125 mil el número de nodos mineros hacia mediados de ese año.

su inclusión en uno de los 144 bloques de como mucho 1 MB que se generan, aproximadamente, cada 24 horas.

En efecto, este número de transacciones confirmadas por día ronda las 300.000 desde diciembre de 2016,<sup>24</sup> mientras que el tamaño de las transacciones en el *pool* de pendientes ha alcanzado picos recientes que rozan los 70 MB. Se están debatiendo soluciones que, en general, pasan bien por la optimización de la información contenida en el bloque para multiplicar el número de transacciones recogidas (*Segregated Witness*) [32, 33, 34, 35], bien por la ampliación del tamaño de los bloques (*Bitcoin Unlimited*, para el que [36] puede considerarse su *primer*), bien por estrategias mixtas, pero en estos momentos la comunidad no ha tomado una decisión definitiva puesto que su ampliación otorga evidentes ventajas pero también algunos, quizá no tan evidentes, inconvenientes.<sup>25</sup> No obstante, desde una visión tecnológica y para nuestros propósitos es oportuno aseverar que este sistema es escalable, y así lo consideraremos en adelante.

También es conveniente incluir en este apartado la problemática del **tamaño de la cadena de bloques**, que la implementación original y actual de Bitcoin/Blockchain hace creciente, sin límite. Desde 2012 es apreciable un crecimiento cuasi-exponencial que ha llevado su tamaño en estos momentos a más de 110 GB, cuando un año antes rondaba los 65 GB (figura 5.2).

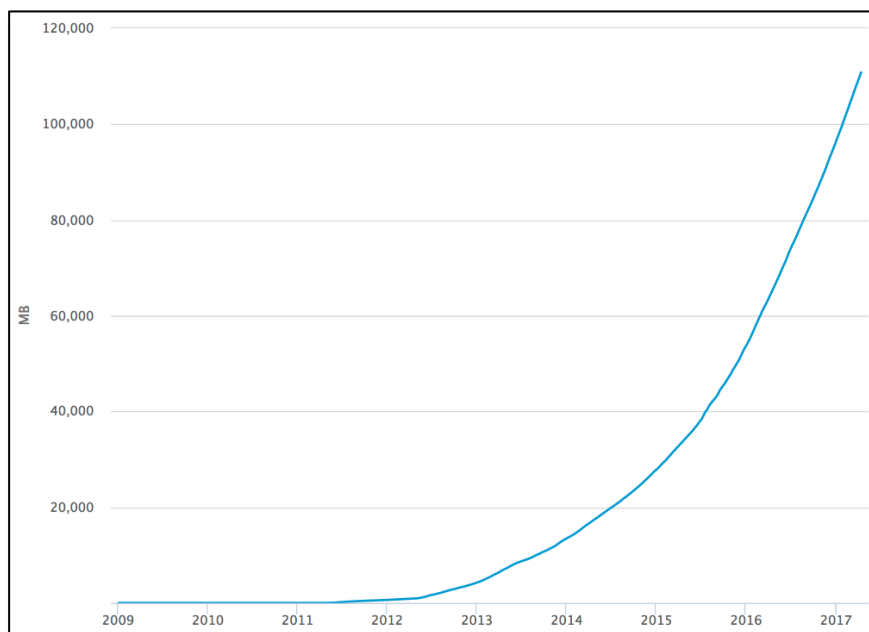


Fig. 5.2: crecimiento de la cadena de bloques desde 2009 (blockchain.info)

La cadena completa es necesaria ya no solo para las actividades de minería, sino para una verificación independiente de las transacciones y los bloques, y existen opiniones que anticipan futuros problemas de independencia si la blockchain alcanza

<sup>24</sup> <https://blockchain.info/charts/n-transactions>

<sup>25</sup> En [37] se revisan en detalle estas cuestiones, tales como el máximo de TPS (*Transactions per Second*), consideraciones sobre los tamaños del bloque y conveniencias de su modificación, tipos de *fork* (*hard*, *soft*), diferentes propuestas en consideración, y otros temas relacionados.

un tamaño tal que sea difícilmente manejable por nodos usuarios compuestos de hardware ordinario, y que solo pueda ser gestionada por los mayores operadores de la red. Esto último, de manera un tanto paradójica, podría resultar en una mayor facilidad de intervención de la red por parte de terceros, lo que choca frontalmente con la filosofía Bitcoin (ver [24], cap.8).

- **Irreversibilidad:** en el capítulo 2, en el que analizábamos el trabajo de S. Nakamoto sobre Bitcoin, veíamos que el autor consideraba como una ventaja del diseño la *solución* al problema de la reversibilidad de los pagos *online*, que aparentemente sitúan a los vendedores en una cierta posición de debilidad con respecto a los consumidores de acuerdo a los protocolos de actuación, en caso de disputa comercial, de los gestores de estos medios de pago y de las entidades intermediarias. Esta solución estriba en la irreversibilidad de los pagos realizados: ni el ordenante ni ninguna autoridad (puesto que no existe) pueden retroceder una transferencia de valor, y si hay disputa dependerá de la voluntad o de la buena fe del beneficiario tal retrocesión mediante una nueva transferencia a la inversa. Digamos, en resumen, que el transmitente tendrá que asegurarse previamente de la procedencia de la transacción, asumiendo pocas posibilidades de recuperar el valor transferido llegado el caso. Bitcoin se configura así como una *hard electronic currency*, a diferencia de las *soft electronic currencies* (PayPal, tarjetas de crédito u otros medios de pago) que sí habilitan mecanismos para la reversión en los pagos.

Pero en un ámbito en el que sus participantes suelen tener intereses contrapuestos, las disputas (sean comerciales, o por errores, o por fraudes, o por cualquier otro motivo) están a la orden del día, y dejar en posición de debilidad al ordenante puede resultar tan arbitrario como la posición opuesta llevada al extremo, por lo que podría considerarse esta situación como una debilidad del diseño si pretendemos extenderlo a una industria ordenada y que se pretenda equitativa y racional.

De un rápido análisis de las características expuestas, podemos identificar cuatro grandes áreas de agrupación de las mismas:

- **Operatividad:** que agrupa características como la de registro histórico, su creciente tamaño, su replicación, descentralización, ausencia de intermediarios, apertura, escalabilidad, irreversibilidad, la representación del valor, o el alto coste de la *proof-of-work*, que solapa con el área de seguridad
- **Regulación:** características como descentralización (solapada en cierto sentido en el área de operatividad), inexistencia de autoridades centrales o regulación o intervención, y anonimidad (esta última solapada dentro del área de privacidad)
- **Privacidad:** no encriptación (registro público), anonimato
- **Seguridad:** eliminación del doble gasto, inmutabilidad, consenso distribuido, y costosa *proof-of-work* como mecanismo de consenso, que solapa con el área de operatividad

Expuestas todas estas características, debemos trabajarlas para intentar articular una estructura que nos permita razonar sobre el enlace, si es que resultara posible, entre Blockchain y la industria.

## 5.2 Un marco de trabajo para la adaptabilidad de Blockchain

Tras revisar las características actuales de la Blockchain de Bitcoin, continuaremos con los principales requisitos, sin ningún orden particular, que cualquier industria exigiría a un diseño de estas características para recoger sus transacciones:<sup>26</sup>

- **Seguridad:** las transacciones y su registro deben ser confiables, es decir, que no resulten manipulables o falsificables; que se controle el doble gasto y resulte imposible de consolidar; y que el estado del sistema debe estar soportado por el consenso entre todos los partícipes en todo momento.
- **Identidad:** todos los operadores deben estar perfectamente identificados y vinculados fehacientemente a personalidades (normalmente jurídicas en este contexto) reales. Deben satisfacerse, en particular, los requerimientos KYC (*Know Your Customer*, conoce a tu cliente) y AML (*Anti-Money Laundering*, contra el lavado de dinero), vigentes históricamente en mayor o menor grado en la industria, pero que a partir de la *Patriot Act* de los EE.UU. de 2001 han tomado una relevancia excepcional para la lucha contra el terrorismo y sus estructuras financieras.
- **Privacidad:** las transacciones solo pueden ser accesibles por sus partícipes.
- **Confidencialidad:** de los partícipes y de las transacciones, en su caso, solo podrán ser accesibles sus datos públicos para la generalidad de los partícipes de la blockchain.
- **Auditabilidad:** debe quedar registrado todo el tracto de transmisión de cada valor identificado en el sistema, entendido éste por cada partida pendiente de gasto, así como de cada partícipe involucrado en tales transferencias.
- **Regulación:** el registro completo debe ser observable por una o varias autoridades, o una jerarquía de ellas, así como controlable en un sentido amplio que incluiría la reversibilidad e, incluso, su intervención, total o parcial.
- **Escalabilidad:** que se contemplen posibles incrementos sustanciales en su operación: número de transacciones, número de operadores, picos de actividad...
- **Flexibilidad:** deben contemplarse posibles adaptaciones a nuevos requisitos, que deberían poder ser implementadas con relativa facilidad.
- **Eficiencia en costes:** la incorporación de Blockchain a las operaciones de una industria debe constituir un ahorro de costes (administrativos, temporales, operativos...) importante con respecto a las estructuras actuales, lo que implica que

---

<sup>26</sup> La literatura sobre este tema (informes, *white papers*, presentaciones... de consultoras, entidades financieras, bancos centrales y desarrolladores) es muy amplia. Hacemos selección de los requisitos más relevantes para nuestros propósitos, que en una medida u otra se encuentran en la mayor parte de las fuentes. Una muestra cualitativamente significativa por sus orígenes, pero marginal en su número, puede encontrarse en los trabajos [38, 39, 40].

los ratios productividad/coste, eficiencia energética, facilidad de operación y mantenimiento... deben ser adecuados.

Adicionalmente, pueden identificarse otros requisitos deseables tales como estandarización, modularidad, analítica de datos, interoperabilidad (sistemas *legacy*,<sup>27</sup> *clouds*, otras blockchains y sistemas), soporte a *smart contracts*<sup>28</sup>... que quedan alejados de nuestros propósitos.

De la misma manera que hemos hecho cuando hemos tratados las características actuales de Blockchain, analizando estos requisitos expuestos podemos identificar las mismas cuatro grandes áreas de agrupación:

- **Operatividad:** que agrupa requisitos como escalabilidad, flexibilidad o costes.
- **Regulación:** requisitos como *auditabilidad*, regulación o identidad (este último solapado con el área de privacidad)
- **Privacidad:** confidencialidad, privacidad, cierto solapamiento con regulación por identidad
- **Seguridad:** el propio atributo de seguridad

Con todo lo anterior, partiendo de una *matriz de correspondencias* (ver anexo C sobre la elaboración de esta matriz) podremos derivar una *matriz de transformación de correspondencias* como la de la tabla 5.1. En esta matriz encontraremos la siguiente información:

- Características actuales de Bitcoin/Blockchain (filas), individualizadas (primera columna) y clasificadas por áreas (*pestañas* a la derecha)
- Requisitos de una Blockchain genérica para la industria (columnas), individualizadas (primera fila) y agregadas por áreas (*pestañas* en última fila)
- El color de cada celda codifica cómo se relaciona cada característica con cada requisito: blanco si no existe relación, verde si la característica favorece o implementa el requisito, rojo si lo dificulta en extremo o lo impide, y amarillo si la característica actual es un inconveniente no invalidante para implementar el requisito
- En cada celda amarilla o roja se especifica la acción o acciones que deberían tomarse para corregir cada una de las limitaciones identificadas

En resumen, cualquier fila en la que encontremos una casilla en rojo ya nos indica la no idoneidad de la característica actual para soportar una nueva Blockchain genérica en la industria de acuerdo a su colisión o colisiones con algún/os requisito/s determinado/s, por lo que salvo en algunas excepciones (el control del doble gasto, la inmutabilidad del registro, completo registro histórico, la amplia replicación, la escalabilidad... y aún así con ciertos reparos en la mayor parte de estas características puesto que encontramos

---

<sup>27</sup> Particularmente importante en la industria financiera, la conexión de nuevas plataformas con muchos sistemas heredados, que conduce a lo que se suele denominar *spaghetti systems*, es actualmente un problema de calado por la falta de flexibilidad y compleja operatividad.

<sup>28</sup> Un *smart contract*, o contrato inteligente, es un protocolo o procedimiento informático por el cual se ejecutan automática y obligatoriamente una o varias transacciones cuando se cumplen ciertos requisitos controlables computacionalmente.

algunas celdas amarillas) es inmediato ver que debe actuarse sobre buena parte de las características que definen Bitcoin/Blockchain.

		Requisitos industria									
		Privacidad	Confidenc.	Identidad	Auditab.	Regulación	Escalabl.	Flexibild.	Costes		Seguridad
Blockchain actual	Imposibilidad de doble gasto								·Otros consensos	Seguridad	
	Registro inmutable							·Reversibil., variabilidad	·Otros consensos		
	Consenso/PoW costoso							·Otros consensos	·Otros consensos		
	Consenso/Pow distribuido							·Otros consensos	·Otros consensos		
	Tamaño						·Podar registro	·Podar registro	·Podar registro	Operatividad	
	Registro histórico						·Podar registro	·Podar registro	·Podar registro		
	Registro replicado					·Autoridad	·Limitar réplica	·Limitar réplica	·Limitar réplica		
	Escalable						·Activar		·Controlar		
	Valor = output no gastado										Privacidad
	Sin intermediación					·Jerarquizar					
Abierta o permissionless	·Login ·Encriptar	·Encriptar	·Identificar ·Login	·Identificar ·Login	·Identificar				·Login ·Encriptar		
Irreversibilidad					·Autoridad ·Jerarquizar ·Reversibil., variabilidad		·Jerarquizar				
Descentralizada				·Jerarquizar	·Autoridad ·Jerarquizar					Regulación	
Sin autoridad			·Identificar ·Login	·Jerarquizar ·Identificar ·Login	·Autoridad ·Identificar						
No regulada			·Identificar ·Login	·Jerarquizar ·Identificar ·Login	·Autoridad ·Jerarquizar						
Sin intervención			·Identificar		·Autoridad ·Jerarquizar ·Reversibil., variabilidad						
Direcciones anónimas		·Limitar info. pública	·Identificar	·Identificar	·Identificar						
Registro no encriptado	·Encriptar	·Encriptar ·Limitar info. púb.							·Encriptar		
		Privacidad	Regulación			Operatividad			Seguridad		

Tabla 5.1: matriz de transformación de las correspondencias entre características actuales de Blockchain y requisitos de la industria.<sup>29</sup>

Por otra parte, y como ya hemos establecido en el anexo C, son apreciables dos núcleos de actuación sobre correspondencias problemáticas:

- En el área de regulación para los requisitos, con cierta extensión al área de privacidad, se observa qué características actuales de Bitcoin/Blockchain tienen que ser transformadas: inexistencia de autoridad y regulación, falta de intervención, anonimato... Están señaladas por el óvalo inclinado en centro-inferior de la tabla.

<sup>29</sup> Bitcoin/Blockchain es técnicamente escalable, y así la representamos, pero en el momento de redactar este trabajo existen discrepancias en la comunidad sobre cómo implementar tal escalabilidad.

- En las áreas de operatividad y seguridad, esquina superior derecha, se apunta que serán necesarias transformaciones en el tamaño del registro y en la *proof-of-work* dados sus costes. El indicador rojo de la característica de seguridad con respecto al requisito de coste (esquina inferior derecha) está motivado también por el coste de la PoW.

Pueden observarse dos transformaciones (en *cursiva*) vinculadas que deben gestionarse fuera de la Blockchain: la escalabilidad debe activarse puesto que, como hemos anticipado, Bitcoin/Blockchain es escalable, pero debe elegirse qué estrategia utilizar; sobre los costes de escalabilidad asumimos que la estrategia elegida incluirá un adecuado control de los mismos. En adelante no precisaremos más sobre estas acciones.

En lo que queda de capítulo precisaremos qué acciones o transformaciones deberemos tomar para intentar acomodar la actual Blockchain a una conceptualmente genérica que pueda resultar de utilidad para la industria.

### 5.3 Transformaciones a considerar en Blockchain

Los elementos del conjunto de las transformaciones del *núcleo de regulación* son:

- **Identificar:** cada dirección debe relacionarse de forma indubitable con una identidad física, normalmente mediante un nombre o identificador del operador. Esta identificación no será necesariamente pública, pero al menos deberá ser posible de establecer, aunque sea privadamente, por una autoridad central.
- **Login:** cada operador deberá acceder al sistema o red debidamente identificado, y con la seguridad suficiente como para no poder repudiar sus operaciones.
- **Autoridad:** recursos a explotar por nodos operados por quienes tengan las facultades de regular (en el más amplio sentido, incluyendo la intervención) el sistema.
- **Jerarquizar:** habilitar una jerarquía de operadores, que puede resumirse en operador básico, administrador, auditor, y otras figuras semejantes, y en la que se habilitan diferentes niveles de acceso y operación. En la cima de la jerarquía se encontraría la autoridad o autoridades de regulación.
- **Variabilidad, reversibilidad:** considerando fundamental la inmutabilidad del registro, o cadena de bloques, una modificación de una transacción dictaminada por quien tenga facultades sólo podrá implementarse en general por una nueva (o varias) transacción, y la transacción modificada debe poderse marcar como tal a los efectos de auditoría, regulación, o simplemente informativos.
- **Encriptar:** para el cumplimiento de los requisitos industriales, no basta con mantener la privacidad, sino que cualquier detalle de las transacciones no relevante para terceros no debe ser público.
- **Limitar información pública:** vinculado a lo anterior, en las transacciones deben mantenerse como públicos solo los detalles administrativos no vinculados a la transferencia de valor.

Sobre alguna de estas transformaciones no vamos a profundizar en este trabajo. La identificación y autenticación de usuarios está perfectamente definida y aplicada hoy en múltiples ámbitos mediante variadas infraestructuras de clave pública (PKI, *Public Key Infrastructure*),<sup>30</sup> entre otros diseños. Se trata, en resumen, de mantener un mecanismo de certificación externo a la propia Blockchain que gestione las identidades y los certificados, y que proporcione acceso seguro e identificado al sistema a los operadores. En lo sucesivo asumiremos que cualquier clave pública situada dentro del sistema estará vinculada unívocamente a una determinada identidad, vinculación que puede ser pública o tan solo conocida por la autoridad central y por los partícipes de las transacciones en las que tome parte, según la estrategia escogida para el sistema.

Tampoco discutiremos más allá de estas líneas la necesaria disponibilidad de una jerarquía de partícipes en cualquier Blockchain industrial. Un modelo que contemple, por ejemplo, una única autoridad es un modelo limitado, pero abstraemos un posible modelo de jerarquía (autoridad/es central/es, operadores, administradores, auditores, técnicos...) considerando que con un adecuado diseño de ciertas limitaciones sobre las facultades genéricas de la autoridad central que iremos viendo, podremos implementar todas las funcionalidades de los diferentes partícipes.

Por su parte, en el conjunto del *núcleo de operatividad/seguridad* tenemos:

- **Otros consensos:** si bien la minería en Bitcoin/Blockchain, a través de la *proof-of-work*, es una de las claves del consenso en un sistema descentralizado y sin autoridad como éste, es necesario contar con otros mecanismos de consenso menos gravosos, que deben encontrarse en un contexto jerárquico y con autoridades centrales. Estos otros consensos, aunque deben suponer un menor coste que la PoW actual, deben proporcionar el mantenimiento de semejantes niveles de confianza.
- **Poda del registro:** el tamaño de la cadena de bloques no puede crecer indefinidamente sin llegar a generar ineficiencias en algunos nodos que deban gestionarla. Debe preverse su reducción mediante la eliminación de transacciones consumidas, quizá con una antigüedad determinada u otras características adecuadas.
- **Limitar réplica:** una réplica global de una cadena de bloques de una determinada industria, gestionada por un conjunto limitado de operadores (en cualquier caso muy inferior al conjunto de, por ejemplo, usuarios actuales de Bitcoin), no parece tener sentido en un entorno restringido y controlado.

Obviaremos profundizar en las dos últimas transformaciones: la reducción del tamaño del registro puede considerarse trivial si se cuenta con un mecanismo auxiliar que, bajo criterios y parámetros predefinidos, vaya eliminando periódicamente del sistema *online* transacciones consumidas y no necesarias para su operatividad habitual, usualmente antiguas, almacenándolas en sistemas auxiliares que contarían con sus propias reglas de acceso; la limitación de la replicación, por su parte, vendría dada prácticamente por

---

<sup>30</sup> En particular, el estándar ITU-T X.509 [41] es el ejemplo habitual de PKI para el formato de los certificados de claves públicas.



definición en una *permissioned* Blockchain que soporte un conjunto limitado (digamos del orden de centenares o unos pocos miles) de operadores.

Así, y en resumen, nuestro programa de transformaciones, que trabajaremos en el siguiente capítulo, contará con:

- La revisión conjunta de la necesidad de la limitación de información pública en las transacciones y del cifrado de la información no pública
- Veremos a continuación que la solución al problema anterior, en un contexto de obligada supervisión por parte de una autoridad central, es precisamente la intervención de esta última en el contenido y gestión de las transacciones
- Revisaremos luego la necesaria implementación de reversibilidad, o variabilidad, sobre una estructura, la transacción, que por definición debe ser inmutable a partir de su registro en la blockchain
- Terminaremos discutiendo el mecanismo de consenso bajo una autoridad central

Recordemos que estas intervenciones lo serán conceptualmente, de manera que nuestro principal objetivo será reflexionar sobre las cualidades de la actual Blockchain y su posible adaptabilidad a otros contextos. Pero antes de acometerlas será conveniente revisar el panorama actual de los proyectos relacionados en curso.

#### **5.4 Proyectos y tendencias actuales**

Consenso y autoridad distribuidos, así como eficiencia y productividad en costes, han sido lo suficientemente novedosos, disruptivos y prometedores como para que los operadores, reguladores y desarrolladores se hayan fijado en la tecnología que nos ocupa y estén tratando de aplicarla a esquemas más prácticos y comerciales. Veamos qué queda de la Blockchain original y qué ha debido sacrificarse en los proyectos que a estas fechas están despuntando por la afiliación de cada vez más operadores. En un pequeño, pero representativo, abanico de desarrolladores, revisaremos de forma resumida, y sin ningún orden en particular, el trabajo de Corda, Hyperledger, Ethereum y Ripple.

##### **5.4.1 R3 Corda [42]**

Desde el ámbito financiero, la compañía R3, a la que están afiliados casi un centenar de operadores, está desarrollando la iniciativa Corda como implementación bajo código abierto de lo que denominan un *registro lógico global* distribuido, destinado a la industria financiera, pero no sobre una cadena de bloques. No obstante, la finalidad de este registro es equivalente en cuanto a que lo que registra es definitivo e inmutable, por lo que los errores o retrocesiones deben ser tratados como nuevas transacciones.

Más que un simple registro de transacciones, se diseña para abarcar estructuras más complejas tales como *smart contracts*, que dan lugar a *objetos de estado* que representan un determinado acuerdo entre las partes, y cuyo estado se modifica de acuerdo a las transacciones que les conciernen. Éstas cuentan con inputs y outputs, al igual que en Bitcoin/Blockchain.

Como es natural, los registros del sistema solo podrán ser accedidos por sus partícipes, y serán éstos los que fundamenten el consenso, en lugar de conseguirse a nivel global. También serán los únicos validadores de las transacciones. No existe el concepto de minado ni de *proof-of-work*.

Existe una jerarquía de partícipes que incluye observadores, como reguladores y supervisores, y pueden confeccionarse subgrupos de operadores con intereses informacionales concretos. Operadores y operaciones están debidamente identificados.

La lógica de cualquier negocio a soportar se implementa en los contratos, ejecutados en una *Java Virtual Machine* (cerrada) Turing-completa, lo que permite utilizar para la implementación de los contratos cualquiera de los lenguajes que pueda ser compilado a *bytecodes* (la mayoría de los más extendidos lenguajes de programación).

#### 5.4.2 Hyperledger [38]

Gestionado por la *Linux Foundation*, y bajo sus mismos principios, este proyecto está recibiendo también importante atención y apoyo de la industria. Se basa en cuatro componentes arquitectónicos conceptualmente relevantes: los servicios de identidad; los de políticas (acceso, configuración, privacidad); los servicios para *smart contracts* (contenedores, registro, ciclo de vida); y los servicios de Blockchain, que incluyen tanto los del registro distribuido como las transacciones (consenso, almacenamiento, protocolo de red, registro). Soporta, por tanto, privacidad y confidencialidad.

Mantiene la abstracción de bloques y cadena de bloques como principal sistema de registro, necesitando de almacenamiento *off-chain* para información de elevado tamaño, cuyos hashes sí se incluyen en la blockchain para certificar su integridad.

Está orientado a la modularidad en sus diferentes servicios, permitiendo así aplicar diferentes esquemas de consenso, de almacenamiento, de encriptación... También pone el foco en la *interoperabilidad*, asumiendo un futuro de blockchains, probablemente soportando diferentes tipos de activos, interconectadas.

Aunque admite diferentes algoritmos de consenso, el aportado por defecto, Scribe, está basado en PBFT (*Practical Byzantine Fault Tolerance*) [43], que pretende controlar los posibles resultados erróneos, como peor alternativa a su parada total, que puede ofrecer un nodo participante en un contexto del ya visto problema de los generales bizantinos. El consenso, que al igual que Bitcoin/Blockchain se establece sobre el orden y la corrección de las transacciones de un bloque, es global entre todos los nodos de la red.

#### 5.4.3 Ethereum [44]

En este proyecto se mantiene una blockchain y una criptodivisa (*ether*) semejantes a las de Bitcoin/Blockchain, pero se propone una forma alternativa de minería, o consenso, más eficiente que la PoW que hemos visto y que evitaría el problema del actual oligopolio y la centralización de la seguridad en unos pocos *pools*: exigiendo a todo hardware minero el acceso a la cadena de bloques (el algoritmo de minado necesita

datos aleatorios de la cadena) se evita la necesidad de la concentración en *pools* de minado.

La principal diferencia se encuentra en su orientación a *smart contracts* y, aún más allá, al concepto de *Decentralized Autonomous Organization*, DAO. Mientras un *smart contract* ejecuta una transacción en el momento en que se hayan cumplido una serie de requisitos previamente fijados por las partes, una DAO implementa una posible selección de contratos alternativos con diferentes flujos de código de acuerdo a una mayoría equivalente a la mayoría social de una sociedad mercantil clásica.

Lo anterior se consigue mediante un sistema Turing-completo que actúa sobre una *Ethereum Virtual Machine*, EVM, cuyo principal mecanismo para evitar ejecuciones infinitas se encuentra en la limitación del número de pasos computacionales.

En su implementación estándar, las transacciones y los contratos se registran sin cifrar, por lo que su contenido es accesible a toda la red (si bien en el caso de los contratos se accede al código compilado, no al original, difícil de identificar salvo en los casos más sencillos). En su caso, se necesitan herramientas adicionales de privacidad.

Como consecuencia del diseño, no se conciben mecanismos regulatorios externos o autoridad central, puesto que un buen diseño de los contratos evitarían situaciones de disputa. Sin embargo, quedaría por especificar qué sucedería ante un contrato o una DAO erróneamente contruidos, de los que ya hay experiencia negativa en Ethereum que incluye un histórico y forzado *fork* en el sistema.

#### 5.4.4 Ripple [45]

Un protocolo distribuido, basado en un registro consensuado, y la existencia de una moneda virtual (*ripple*), son probablemente los únicos elementos comunes con el sistema Bitcoin/Blockchain. A partir de estos componentes, Ripple se constituye más en un sistema de pagos interbancario que en una criptomoneda al uso, siendo uno de los sistemas más avanzados en implementación y ejecución de pruebas exitosas por operadores financieros y en aparentes condiciones de mercado.

La cadena de bloques desaparece en favor de una base de datos común compartida e inmutable. El consenso se *externaliza* en nodos independientes que validan las transacciones y el registro, aunque cada operador puede correr su propio nodo validador. Avanzan la futura posibilidad de conseguir un consenso distribuido entre la red de nodos validadores mediante un algoritmo BFT (*Byzantine Fault Tolerant*) [46].

Como sistema de pagos no parece incluir mecanismos de regulación o intervención explícitos, pero se fundamenta en la aportación de liquidez por parte de varios *market makers* que, a cambio de ciertos beneficios, responden del flujo monetario entre partícipes sin confianza mutua.

Las transacciones son privadas entre los partícipes, y no requiere minado dadas sus características y la propia estructura del protocolo.

# Un diseño conceptual para la aplicación de Blockchain en la industria

## 6.1 Un modelo formal de Bitcoin/Blockchain: esquema actual y transformaciones para la industria

Presentamos a continuación un modelo que recoge el esquema actual de funcionamiento de Bitcoin/Blockchain según su concepción original. Nos basaremos aquí en la notación y en el esquema de introducción a las transacciones Bitcoin (ambos ampliados y adaptados a nuestros propósitos) que se encuentran en [24]. Este modelo nos servirá para hacer explícitas, también de una manera más formal, pero preliminar, las transformaciones que sobre las actuales transacciones de la Blockchain soporte de Bitcoin propondremos inmediatamente después para acomodarla a una industria regulada, y en particular por los efectos de la introducción de privacidad y confidencialidad en las transacciones, así como de una autoridad reguladora y con facultades de intervención. El resto del capítulo lo emplearemos en discutir y precisar con más detalle tanto las transformaciones aquí presentadas como otras aproximaciones al problema.

Denotaremos  $\{KPub, KPriv\}$  como un par de claves, pública y privada, de un criptosistema de clave pública. Sea  $Addr(KPub)$  una función que retorna la dirección Bitcoin/Blockchain vinculada a la clave pública  $KPub$ . Consideraremos  $Sig_{KPriv}(m)$  como la firma digital del mensaje  $m$  con la clave privada  $KPriv$ . Sea  $V$  un entero representativo de un valor.

Definimos un output  $T_t output$ , incluido en una transacción  $T_t$  (que definiremos a continuación), como un par que asigna un valor  $V$  a una determinada dirección,  $\{Addr(KPub), V\}$ . Definimos un input  $T_t input$ , incluido en una transacción  $T_t$ , como una 3-tupla que recoge un output no consumido (que se define a continuación) hasta ese momento, una firma digital sobre la transacción que lo contiene, y la clave pública vinculada a tal firma digital,  $\{\{Addr(KPub), V\}, Sig_{KPriv_A}(T_t), KPub_A\}$ . Definimos una transacción  $T_t$ , donde  $t$  representa el momento de su construcción, como un par de conjuntos ordenados, uno de inputs y otro de outputs,

$$T_t = \{\{T_t input_0, \dots, T_t input_{m-1}\}, \{T_t output_0, \dots, T_t output_{k-1}\}\}$$

en la que se cumple la igualdad de la suma de sus respectivos valores implícitos:<sup>31</sup>

$$\sum_{i=0}^{m-1} V_{T_t input_i} = \sum_{i=0}^{k-1} V_{T_t output_i}$$

---

<sup>31</sup> Para considerar esta igualdad asumimos los *fees* como un output adicional; también podemos considerar  $\sum_{i=0}^{m-1} V_{T_t input_i} \geq \sum_{i=0}^{k-1} V_{T_t output_i}$ , que sería la condición literal a cumplir en el sistema Bitcoin/Blockchain, y en la que debe considerarse un output implícito tal que se cumpla  $\sum_{i=0}^{m-1} V_{T_t input_i} = \sum_{i=0}^{k-1} V_{T_t output_i} + V_{fees}$ .

Las siguientes definiciones de output no consumido, o no gastado, son equivalentes:

- Es aquel que tras su instanciación en una transacción no ha participado como input en otra transacción posterior
- Aquel que es un elemento del conjunto  $UTXO$ , definido este último como el conjunto de outputs que no han participado como inputs en ninguna transacción

Consideraremos  $dH$  como la función *hash* (doble SHA-256) que se aplica sobre el literal de una transacción para obtener su identificador,  $Tx_tID = dH(T_t)$ . Dadas las definiciones anteriores, cualquier output podrá referenciarse a la transacción en la que se instanció y, dentro del conjunto de outputs de ésta, al ordinal de su situación en tal conjunto (denotamos equivalencia con el símbolo ' $\equiv$ '):

$$T_t output_g = \{Addr(KPub), V\}_{(T_t, g)} \equiv (dH(T_t), g) \equiv (Tx_tID, g)$$

donde el par  $(dH(T_t), g)$  denota la referencia al output  $g$ -ésimo de la transacción  $T_t$ .<sup>32</sup>

Para nuestros efectos no es necesario completar el modelo con las definiciones de bloque y cadena de bloques.

Sean ahora  $\{KPub_A, KPriv_A\}, \{KPub_B, KPriv_B\}$ , dos pares de claves, pública y privada, de dos usuarios  $A$  y  $B$ . Supongamos que el usuario  $A$  es propietario de un valor  $V$  según un output no gastado,  $T_{t0} output_g$ , de una transacción genérica  $T_{t0}$  con  $m$  inputs y  $k$  outputs:

$$\begin{aligned} T_{t0} &= \{\{T_{t0} input_0, \dots, T_{t0} input_{m-1}\}, \{T_{t0} output_0, \dots, T_{t0} output_{k-1}\}\} \\ T_{t0} output_g &= \{Addr(KPub_A), V\} \end{aligned}$$

y donde  $g \in [0, k - 1]$ . Por definición, este output no gastado cumplirá

$$T_{t0} output_g = (dH(T_{t0}), g) \in UTXO$$

donde el par  $(dH(T_{t0}), g)$  denota, como sabemos, la referencia al output  $g$ -ésimo de la transacción  $T_{t0}$ .

Si  $A$  desea transferir este valor  $V$  a  $B$ , construirá una nueva transacción  $T_{t1}$ , correspondiendo  $t1$  necesariamente a un instante de tiempo posterior a  $t0$ :

$$\begin{aligned} T_{t1} &= \{\{T_{t1} input_0\}, \{T_{t1} output_0\}\} \\ T_{t1} input_0 &= \{(dH(T_{t0}), g), Sig_{KPriv_A}(T_{t1}'), KPub_A\} \\ T_{t1} output_0 &= \{Addr(KPub_B), V\} \end{aligned}$$

---

<sup>32</sup> De esta manera, el par  $(dH(T_t), g)$  tiene vinculación directa al objeto computacional  $COutPoint$ , componente del objeto input  $CTxIn$ , a su vez componente (bajo el contenedor  $vin$ ) de  $CTransaction$ , y que veamos en el apartado 4.2 del capítulo 4.

donde el output  $(dH(T_{t0}), g)$  se consume en la nueva transacción  $T_{t1}$ , mientras que  $T_{t1}'$  es el contenido (parcial y serializado de manera no relevante aquí) de la transacción  $T_{t1}$ . Tras su validación, y dado el consumo (o gasto) del output en cuestión, así como la creación de un nuevo output todavía no gastado, el conjunto de outputs no gastados se actualizará a:

$$UTXO' = \{UTXO \setminus \{T_{t0}output_g\}\} \cup \{T_{t1}output_0\}$$

Analicemos ahora las implicaciones de una posible adaptación de Blockchain, tal y como la hemos visto, a una industria sujeta a regulación. Las principales novedades serán la necesidad de privacidad y confidencialidad sobre la información de negocio (partícipes y valor) de las transacciones, así como la inclusión de una autoridad central, que entre otras facultades debe contar con la de poder acceder, precisamente, a esa información de negocio oculta, así como a la posible intervención del sistema.

Elegimos una estrategia (que detallaremos más adelante) de compartición de claves simétricas: sean  $KSim_A$  ( $KSim_B$ ) una clave de criptografía simétrica compartidas entre el usuario  $A$  ( $B$ ) con la autoridad central  $AC$ ; y sea  $CSim_K(m)$  el resultado de aplicar sobre el mensaje  $m$  una función de cifrado simétrico mediante una clave compartida  $K$ . Ahora, bajo la existencia de autoridad, la transmisión del valor  $V$  de  $A$  a  $B$  quedará:

$$\begin{aligned} T_{t1}^1 &= \{\{T_{t1}^1input_0\}, \{T_{t1}^1output_0\}\} \\ T_{t1}^1input_0 &= \{CSim_{KSim_A}(dH(T_{t0}), g), Sig_{KPriv_A}(T_{t1}^1'), CSim_{KSim_A}(KPub_A)\} \\ T_{t1}^1output_0 &= \{CSim_{KSim_B}(Addr(KPub_B)), CSim_{KSim_B}(V)\} \end{aligned}$$

donde es evidente que ningún tercero que desconozca las claves simétricas correspondientes podrá acceder a la información de negocio.

Pero también tenemos la necesidad de que la autoridad central introducida pueda intervenir el sistema, es decir, que entre otras cosas pueda realizar transacciones en nombre de los operadores. De esta manera, y considerando  $\{KPub_{AC}, KPriv_{AC}\}$  un par de claves pública-privada de la autoridad  $AC$ , la transferencia del valor  $V$  desde  $A$  a  $B$  podrá realizarse alternativamente en caso de necesidad como

$$\begin{aligned} T_{t1}^2 &= \{\{T_{t1}^2input_0\}, \{T_{t1}^2output_0\}\} \\ T_{t1}^2input_0 &= \{CSim_{KSim_A}(dH(T_{t0}), g), Sig_{KPriv_{AC}}(T_{t1}^2'), KPub_{AC}\} \\ T_{t1}^2output_0 &= \{CSim_{KSim_B}(Addr(KPub_B)), CSim_{KSim_B}(V)\} \end{aligned}$$

Bajo cualquiera de estas dos alternativas se actualiza el conjunto de outputs no gastados. Para la última, por ejemplo, tendremos:

$$UTXO' = \{UTXO \setminus \{T_{t0}output_g\}\} \cup \{T_{t1}^2output_0\}$$

Los cambios no aparentan ser extensos, pero veremos en el resto del capítulo que sus implicaciones sobre el sistema, incluido su mecanismo de consenso, son de gran calado.

## 6.2 Encriptación e información pública en las transacciones:

Definimos una *transacción confidencial* como aquella en la que su información *comercial* no puede ser interpretada: su ordenante, beneficiarios y valores deben ser ininteligibles para cualquiera que no participe en las mismas.

Si recordamos el esquema de una transacción genérica que presentamos al final del apartado 4.2, en el que especificábamos todos los campos de las estructuras de datos (clases) CTransaction (que implementa la transacción en sí), CTxIn (input) y CTxOut (output), podremos analizar sus elementos uno a uno para ver su necesidad de ocultación de acuerdo a nuestros intereses:

Transacción			Informac. sensible	Elemento a ocultar
nVersion			No	
vin[]	vin[0]	hash_0	Parcialmente	hash
		n_0	Parcialmente	n
		scriptSig_0	Sí	<pubKey>
		nSequence_0	No	
	...	...		
	vin[m-1], m=vin.size()	hash_m-1	Parcialmente	hash
		n_m-1	Parcialmente	n
		scriptSig_m-1	Sí	<pubKey>
nSequence_m-1		No		
vout[]	vout[0]	nValue_0	Sí	nValue
		scriptPubKey_0	Sí	<pubKeyHash>
	...	...		
	vout[k-1], k=vout.size()	nValue_k-1	Sí	nValue
		scriptPubKey_k-1	Sí	<pubKeyHash>
nLockTime			No	
hash			No	

Tabla 6.1: necesidades de encriptación de los atributos de una transacción

De los campos *administrativos* de la transacción (nVersion, nLockTime, hash) no es necesaria su encriptación. En cuanto a los inputs, su referencia a una determinada transacción (hash) y, dentro de ésta, su índice en el vector de outputs vout[], deben considerarse cifradas para que la vinculación entre el output de una transacción y su utilización posterior como input de otra quede oculta, ya que aunque no ofrezcan en claro información comprometedor para la privacidad de los intervinientes, sí permitirían seguir el tracto de las transmisiones, lo que no es apropiado. Por su parte, el campo nSequence de los inputs no tiene una función definida actualmente, por lo que su encriptación solo añadiría carga de trabajo adicional. Sí requerirá ocultación el contenido de scriptSig que haga referencia al transmitente puesto que este script presenta una firma digital de la transacción y una clave pública como demostración de su facultad de movilizar el output de una transacción anterior definido por el par (hash, n). En consecuencia, la referencia a la clave pública, que recordemos es la referencia a la propiedad del valor, y que representamos por el elemento <pubKey> debe ser cifrado. Para los outputs, debe encriptarse tanto el campo del valor transferido, nValue, como las referencias al beneficiario de la transferencia en forma de su clave pública que deben aparecer en scriptPubKey, y que representamos por el elemento <pubKeyHash>.

Para nuestro propósito, es decir, eliminar el acceso de terceros a elementos de la transacción que tengan que ver con el negocio subyacente, resulta clara la necesidad del

cifrado del importe almacenado en `nValue` en los outputs de la transacción. Veamos ahora la menos obvia necesidad de cifrar `<pubKeyHash>` y `<pubKey>`:

- Si se elige para el sistema una estrategia según la cual las claves públicas (direcciones) están vinculadas públicamente a una determinada identidad, entonces está clara la necesidad de cifrarlas.
- En el caso de la estrategia anterior, ¿para qué cifrar `<pubKeyHash>`, que por definición ya lo está puesto que resulta de aplicar una función hash (compuesta) sobre una clave pública, mediante `RIPEMD-160(SHA-256(pubKey))`? Porque en un entorno limitado de usuarios, y por tanto de claves públicas (vinculadas públicamente, para recordar, a identidades reales), resultaría posible recorrer todas las claves públicas, ir aplicando la conocida función *hash*, y así averiguar la identidad final del propietario de tal output, que no es lo que se pretende.
- Aunque se elija la estrategia de restringir el conocimiento de la clave pública solo a los partícipes de la transferencia de valor (porque, por ejemplo, es enviada de forma segura y bajo demanda al transmitente por parte del beneficiario), estrategia que es la que asumiremos en adelante, la vinculación entre `<pubKeyHash>` y `<pubKey>` debe mantenerse oculta a terceros por los mismos motivos argumentados para cifrar el par (*hash*, *n*).
- El cifrado de las direcciones en todos los casos permitirá modificar la estrategia de publicidad de las claves sin tener que ocuparse de los efectos de tales cambios en el acceso por terceros a información sensible de las transacciones.

Por tanto, en transacciones del tipo P2PKH (apartado 4.3 del capítulo 4) tendremos que aplicar encriptación a todas las referencias tanto a identidades (en `scriptSig` y `scriptPubKey`) como a importes (`nValue`). Si recordamos el contenido de ambos *scripts*, según vimos en el capítulo 4:

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY  
               OP_CHECKSIG  
scriptSig: <signature> <pubKey>
```

resulta evidente que el cifrado deberá hacerse sobre `<pubKeyHash>` y `<pubKey>`, mientras que no es necesario realizarla sobre `<signature>` puesto que como firma digital que es, computada por el transmitente una vez construida la transacción, ya se encuentra encriptada.

Por su parte, recordemos que los scripts para las transacciones P2SH eran:

```
scriptPubKey: OP_HASH160 <scriptHash> OP_EQUAL  
scriptSig: <signature/s> <script>
```

En este caso es evidente que `<scriptHash>` puede mantenerse sin cifrar, pero no así `<script>` ni `<signature/s>` dada la información que soportan (capítulo 4, apdo. 4.3).

Queda por definir cómo realizamos la encriptación. Existen, en general, tres aproximaciones notables a la resolución de la confidencialidad en Bitcoin/Blockchain:



- *Confidential Transactions* [47] (CT) es un trabajo basado en el esquema de compromiso<sup>33</sup> (*commitment scheme*) de Pedersen<sup>34</sup> [49], orientado a la ocultación del valor de los inputs y los outputs de las transacciones utilizando tales valores (ocultos) como parámetros para la obtención de unos *compromisos* públicos bajo la forma de otros valores, con los que puede demostrarse la invariabilidad de la información oculta, cumpliendo además la propiedad de que puede definirse una operación sobre estos compromisos de tal manera que su aplicación es igual al compromiso de la aplicación de tal operación sobre los valores ocultos,<sup>35</sup> resultando así un esquema aditivo *homomórfico* que permitirá la confirmación de las transacciones por terceros operando sobre los compromisos en lugar de sobre los valores reales. En CT el esquema original de Pedersen se modifica para construir los compromisos utilizando puntos sobre una curva elíptica criptográfica y la operación suma definida según vimos en el apartado 3.4 del cap. 3, para lo que se cumple la propiedad homomórfica de que la suma de las claves públicas ( $K$ ) es igual a la suma (*mod*  $q$ ) de las claves privadas ( $k$ , con  $K = kQ$ ) multiplicada por el generador del grupo:  $K1 + K2 = (k1 + k2 \text{ (mod } q))Q$ .
- Protocolos de mezcla y ofuscación de pares inputs-outputs (*coin mixing*), en particular *CoinJoin* [50], orientados en general a preservar el anonimato de los transmitentes y que se fundamentan en la reunión de diferentes ordenantes que mezclan sus valores a transferir en una única transacción de tal manera que sea imposible vincular un determinado output con un determinado input, obteniéndose así una verdadera fungibilidad del valor sujeto a intercambio. Suele obligar a que los diferentes pagos sean de la misma cuantía para evitar enlazar importes singulares entre ordenante y beneficiario, y conlleva una estructura adicional de mensajes para la mezcla que permite, adicionalmente, la identificación de posibles atacantes dada la necesidad de firmar por parte de todos y de cada uno de los intervinientes la transacción última resultante.
- El anonimato de los beneficiarios suele obtenerse mediante la utilización de direcciones de un solo uso que, en general, solo permiten al ordenante, y a nadie más, vincular tales direcciones con los propietarios últimos. *Stealth Addresses* [51] es un esquema automatizado de ocultación de direcciones por el cual se obtienen diferentes

---

<sup>33</sup> Un esquema de compromiso (*commitment scheme*) es una primitiva criptográfica que permite *comprometer* un valor oculto de tal manera que siempre podrá demostrarse su invariabilidad. El esquema más simple es una función *hash* sobre el valor a ocultar. Un resultado *hash* invariante en el tiempo (compromiso) garantiza la invariabilidad del valor oculto, que siempre podrá demostrarse aplicando públicamente la función *hash* convenida sobre él.

<sup>34</sup> El esquema de Pedersen es uno de los variados esquemas de compartición de secretos que podemos encontrar. Bajo cualquiera de estos esquemas, y para su aplicación en múltiples necesidades en contextos donde en general no existe una autoridad de confianza, se distribuye un secreto entre un conjunto de participantes, a cada uno de los cuales se entrega una parte con la que podrá reconstruirse el secreto completo con las partes de un subconjunto de los partícipes (quizá todos). Puede verse una introducción a estos esquemas en [48]. El de Pedersen es un *verifiable secret sharing scheme*, o VSS, familia de esquemas diseñados para rechazar partícipes deshonestos, incluyendo la corrupción de un posible agente centralizador del secreto compartido.

<sup>35</sup> Consideremos una operación suma sobre un esquema de compromiso  $C$  con dos valores ocultos,  $val1$  y  $val2$ ; se cumplirá  $C(val1) + C(val2) = C(val1 + val2)$ ,  $C(val1) - C(val1) = 0$ . En la práctica, para no poder identificar con facilidad resultados de compromiso sobre valores usuales o pequeños, se utilizan factores de ocultación,  $FO$ , de tal manera que se cumple  $C(FO1, val1) + C(FO2, val2) = C(FO1 + FO2, val1 + val2)$ .

direcciones de un solo uso partiendo de una única, pública y permanente dirección fijada por el beneficiario. De esta manera, los pagos ni siquiera necesitan de comunicación previa entre transmitente y beneficiario, y a la vista de la dirección publicada, o *stealth address*, nadie podrá dilucidar qué valores le han sido transferidos, mientras que las creadas a través de esta dirección conocida solo podrán ser vinculadas al beneficiario por el propio ordenante de la transacción. El mecanismo se basa en el protocolo de establecimiento de claves ECDH (*Elliptic Curve Diffie-Hellman*):<sup>36</sup> sea una función *hash*  $H$  y dos pares  $(K_t, k_t)$  y  $(K_b, k_b)$  de claves públicas ( $K$ ) y privadas ( $k$ ) de un transmitente  $t$  y un beneficiario  $b$ , respectivamente, correspondientes a un criptosistema de curva elíptica y cumpliéndose, por tanto,  $K_t = k_t Q$  y  $K_b = k_b Q$ .<sup>37</sup> La *stealth address* del beneficiario será  $K_b$ , que se hace pública por cualquier medio; el transmitente obtendrá un secreto compartido  $S = H(k_t K_b) = H(k_t k_b Q)$ , que simultáneamente puede ser obtenido por el beneficiario mediante  $S = H(k_b K_t) = H(k_b k_t Q)$ ; el transmitente obtendrá la dirección, o clave pública, a la que transferir el valor mediante  $K_b + SQ$  (desconociendo la clave privada que lo podrá movilizar), mientras que el beneficiario también podrá calcular la clave privada vinculada a  $K_b + SQ$  mediante  $K_b + SQ = k_b Q + SQ = (k_b + S)Q$ , resultando por tanto  $k_b + S$  la clave privada.

Como puede apreciarse, cada uno de las herramientas vistas está orientada a una ocultación parcial del contenido de negocio de una transacción: bien sobre el transmitente, bien sobre los beneficiarios, bien sobre la cantidad de valor transferido. Un esquema de integración de estos tres mecanismos puede encontrarse en el reciente trabajo sobre *ValueShuffle* [54], un nuevo protocolo construido sobre *CoinJoin*, *Confidential Transactions* y *Stealth Addresses* con el que se consigue privacidad total sobre toda la información sensible de una transacción, manteniendo ésta bajo la actual estructura de Bitcoin/Blockchain y haciendo innecesaria una ruptura del protocolo Bitcoin. Además, proporciona resistencia DoS e identificación de posibles atacantes por heredar semejantes mecanismos de *CoinJoin*. El principal coste, aparte de la adición de una importante capa de complejidad adicional, estriba en la necesidad de uno o varios puntos centrales que actuarían como *tableros de anuncios* activos para coordinar todo el tráfico de mensajes necesarios entre los diferentes ordenantes, y cuya ejecución no admite situaciones de caída de nodos o partición de la red. *ValueShuffle* establece varias fases consecutivas en su explotación: la generación de los outputs por cada transmitente, su transmisión P2P a través del mecanismo de intercambio de mensajes, su mezcla y comprobación de la suma resultante para evitar creación deshonestas de valor, la comprobación por cada transmitente de la transacción construida y, finalmente, la firma de esta última por cada transmitente (si alguno no firma se considera deshonesto y el protocolo se reinicia sin su participación).

En cualquier caso, este esquema de privacidad total, orientado a su utilización por Bitcoin sin necesidad de ruptura, o *hard fork*, de su actual protocolo, no es aplicable a una Blockchain regulada puesto que no permite bajo su actual diseño la participación de una autoridad en el sistema que tenga el necesario acceso a la información. Precisamente la participación en el sistema de una autoridad central con facultades prácticamente ilimitadas, nos conducirá a otras estrategias.

<sup>36</sup> El esquema ECDH puede revisarse en [51], apdo. 6.1 *Elliptic Curve Diffie-Hellman Scheme*. La conveniencia de la utilización de una función *hash* auxiliar puede analizarse en [53].

<sup>37</sup> Ver cap. 3, apdo. 3.4.

### 6.3 La participación de una autoridad central:

Según hemos visto en el apartado anterior tenemos varios campos por ocultar mediante cifrado en cada transacción: para una transacción genérica P2PKH, uno en cada input y dos en cada output. Una solución *ingenua* pasaría por replicar cada uno de estos campos con un diferente cifrado para cada partícipe y autoridad, pero con un mayor coste tanto de almacenamiento en cada transacción y en los registros de cada partícipe para almacenar las correspondientes claves, como de procesamiento al tener que contemplar diferentes procesos de tratamiento. Intentaremos optimizar estos costes en lo posible.

El enfoque que escogemos es un procedimiento orientado a mantener la transacción en un tamaño semejante al que tiene en Bitcoin/Blockchain, centrado en cada beneficiario de cada transacción como principal interesado, y dirigido por una autoridad central AC (por simplicidad asumimos una única autoridad como elemento aislado de jerarquía) que aporta a los diferentes beneficiarios las claves (simétricas) necesarias para el cifrado de la información a ocultar. Así, cada beneficiario podrá acceder a los outputs que le conciernen, que también podrán ser revisados por la autoridad al compartir las claves. Además, cada transmitente conoce obviamente cada una de sus transacciones. Ningún operador distinto a los indicados será capaz de acceder a esta información.

En la figura 6.1 representamos un diagrama de secuencias simplificado con los mensajes cruzados entre los intervinientes:

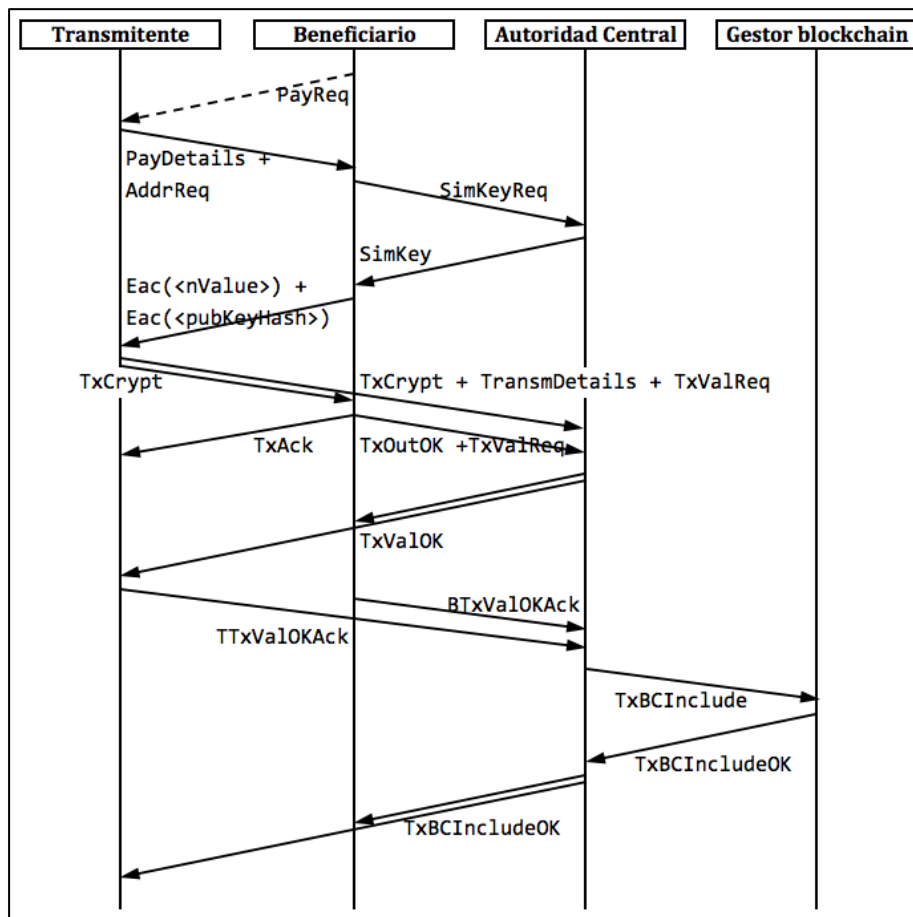


Fig. 6.1: secuencias del proceso de construcción, validación y registro de una transacción con información cifrada (línea discontinua para mensajes opcionales)

Para hacer más explícita la acción de inclusión de la transacción en la cadena de bloques, asumimos la existencia de un gestor de la misma, figura que puede considerarse como una extensión de la autoridad central. Es posible mayor eficiencia en el flujo de mensajes, incluyendo posibles eliminaciones de confirmaciones duplicadas, pero que dejamos explícitas para mayor claridad:

El proceso se resume:

1. El transmitente, quizá bajo requerimiento previo (mensaje `PayReq`), comunica detalles de la transmisión de valor que quiere realizar a un determinado beneficiario, y solicita dirección de envío (mensajes `PayDetails` y `AddrReq`)
2. El beneficiario solicita clave de cifrado simétrico a la AC (`SimKeyReq`), que se vinculará, quizá con una referencia provisional, a la transacción en construcción
3. El beneficiario cifra con la clave obtenida la dirección en la que desea recibir el valor transferido ( $E_{AC}(\text{pubKeyHash})$ ), así como el importe que previamente le ha detallado el transmitente ( $E_{AC}(nValue)$ ), y los envía al transmitente
4. Con la información cifrada recibida de cada beneficiario, el transmitente construye la transacción
  - i. cifrando para cada input `<pubKey>` en `scriptSig`, así como el correspondientes par (`hash`, `n`), con la clave simétrica que en su día tuvo que recibir de la autoridad central para cifrar `<pubKeyHash>` en el output que en su momento le otorgó la propiedad (paso anterior),
  - ii. computando la firma digital para cada input a incluir en `scriptSig`,
  - iii. calculando el `hash` de la transacción, que servirá de referencia definitiva, como usualmente,
  - iv. y enviando la transacción definitiva tanto a cada uno de los beneficiarios como a la AC (`TxCrypt`), a la cual deberá aportar simultáneamente el detalle de cada transmisión (`TransmDetails`), como mínimo el par (identidad, valor), para que con cada clave simétrica compartida con cada beneficiario compruebe la corrección de la información cifrada; además, explícita o implícitamente solicitará a la AC la validación de la transacción (`TxValReq`)
5. Cada beneficiario comprobará que tanto su dirección como el importe (cifrados) son los correctos, y confirmará al transmitente (`TxAck`); también dará su visto bueno del output a la autoridad central (`TxOutOK`), y solicitará de ésta la comunicación de la validación final de la transacción (`TxValReq`)
6. La AC, tras comprobar que la transacción satisface los detalles de cada transmisión identidad-valor, y tras haber recibido el visto bueno de cada uno de los beneficiarios, emitirá un mensaje de validación de la transacción (`TxValOK`) a cada interviniente, que devolverán acuse de recibo (`BTxValOKAck` o `TTxValOKAck`, según se actúe como beneficiario o transmitente)
7. Una vez confirmada la transacción por todas las partes, la autoridad encomienda su registro al gestor de la cadena de bloques (`TxBCTxInclude`), donde se implementa el consenso práctico de todos los operadores con respecto al registro bajo una cadena de bloques. Este gestor confirmará tal inclusión cuando el bloque que la incluya se registre en la cadena.

Los scripts que hemos visto para una transacción P2PKH quedarían ahora como:

**scriptPubKey:** `OP_DUP OP_HASH160 <EAct2(pubKeyHash)> OP_EQUALVERIFY`

OP\_CHECKSIG  
**scriptSig:** <signature> < $E_{Act_1}(\text{pubKey})$ >

además de tener  $E_{Act_2}(\text{nValue})$  como valor del campo nValue de la transacción cifrada, y donde  $E_{Act_1}$  es la función de encriptación con clave simétrica utilizada en una transacción previa,  $t_1 < t_2$ , para el output que dio la propiedad al actual transmitente, y  $E_{Act_2}$  es la función de encriptación con clave simétrica, obtenida esta última de la autoridad central por el beneficiario de la transmisión de valor que se está realizando ahora.

Veamos ahora en términos de tamaño de almacenamiento cómo influye esta solución para una transacción regular P2PKH con un solo input y un único output. Asumiremos la utilización del esquema AES (*Advanced Encryption Standard*) [55] como mecanismo de cifrado simétrico entre beneficiarios y autoridad. La longitud de las claves bajo AES no influye en el tamaño de la información encriptada resultante puesto que para todos los casos se trabaja con un tamaño de bloques de 128 bits, lo que implica que el output mínimo del algoritmo AES tendrá un tamaño de 16 bytes. Podremos considerar, por tanto, que trabajamos con tamaños de clave AES de 192 o 256 bits, ambos ya extraordinariamente seguros.

El efecto del cifrado de los tres elementos en consideración se resume en la siguiente tabla:

Elemento	Tamaño (Bytes)	A cifrar (Bytes)	% s/ Tx	Tras cifrado (Bytes)	% s/ valor inicial
<b>CTxIn</b>	<b>145</b>		<b>77%</b>	<b>144</b>	<b>99%</b>
(hash, n)	36			36	
ScriptSig	105			104	
<pubKey>		33	17%	32	97%
nSequence	4			4	
<b>CTxOut</b>	<b>32</b>		<b>17%</b>	<b>36</b>	<b>112%</b>
nValue	8	8	4%	16	200%
ScriptPubKey	24			20	
<pubKeyHash>		20	11%	16	80%
<b>Resto Tx</b>	<b>12</b>		<b>6%</b>	<b>12</b>	<b>100%</b>
<b>Ctransaction</b>	<b>189</b>	<b>61</b>	<b>100%</b>	<b>192</b>	<b>102%</b>

Tabla 6.2: incremento de tamaño de una transacción (uno solo input y output) en caso de cifrado de sus elementos no públicos

En estos resultados observamos un mínimo crecimiento del total del tamaño de la transacción tipo, que pasa de unos 189 bytes a 192. Aunque el elemento nValue doblará su tamaño al pasar de 8 a 16 bytes (mínimo tamaño de salida de AES), las reducciones en <pubKey> y <pubKeyHash> por el tratamiento a nivel de bytes compensan parcialmente el incremento total. Los peores resultados (mayor tamaño) se obtendrán en transacciones con numerosos outputs, pero aún así el incremento sobre una transacción Bitcoin/Blockchain rondará el 110%. No obstante, debe reseñarse que este análisis se efectúa sobre la transacción que va a incluirse en la cadena de bloques, y que adicionalmente deben registrarse por cada partícipe, incluida la AC, las claves simétricas utilizadas, además de, en su caso, los posibles parámetros adicionales si resultaran necesarios (como un vector de inicialización).

Es evidente que el procedimiento de resolución de una transacción se complica sobremanera en comparación con la transacción bajo Bitcoin/Blockchain. En este

sistema, el transmitente construye y lanza la transacción sin más requisitos, mientras que bajo regulación deberá someterse, como mínimo, al procedimiento descrito en la figura 6.1, mucho más complejo y sujeto a múltiples acciones por varias partes. No obstante, y como veremos en el apartado sobre las transformaciones de operatividad/seguridad, esta mayor complejidad se aprovecha para una necesariamente nueva estrategia de consenso.

#### **6.4 La variabilidad en las transacciones cerradas:**

La irreversibilidad de las transacciones, que hemos tratado en el apartado 5.1 del capítulo anterior, es una de las características de la Blockchain a cuestionar en un nuevo modelo destinado a la industria. La mayoría de las aproximaciones actuales al problema de la regulación no toman en consideración esta posibilidad de reversión, aunque en el contexto de entrega de mercancías o prestación de servicios<sup>38</sup> existen diseños de servicios de *escrow* que aportan, hasta cierto grado, un nivel adicional de seguridad introduciendo una figura de autoridad que decide, en caso de disputa, a cuál de las partes debe corresponder la asignación definitiva del valor.

La implementación más sencilla de *escrow* consiste en una transacción *multifirma*, tal como la apuntamos cuando nos referimos a las transacciones P2SH (apdos. 4.1 y 4.3, cap. 4), en la que el valor se transfiere inicialmente a una dirección provisional y resultará movilizable bajo un esquema de firmas<sup>39</sup> 2-de-3 en el que participan el transmitente, el beneficiario y el ente *escrow*: si transmitente y beneficiario alcanzan conformidad en la transacción, podrán movilizar el valor con sus dos firmas para la entrega al beneficiario; en caso contrario, la autoridad del *escrow* deberá participar en la resolución de la transacción decidiendo mediante una lógica seguramente externa a la Blockchain a quién se asigna en última instancia el valor, movilizándolo así con la firma del *escrow* y del destinatario último.

En lo que probablemente es el primer (y muy reciente) trabajo de formalización del problema *escrow* [56], sus autores aproximan la estructura del protocolo correspondiente bajo tres dimensiones o ejes básicos:

- **Privacidad:**
  - *Ocultación de disputa (dispute-hiding)*: un protocolo *escrow* la cumple si un observador externo (que no participa ni media en la transacción) no puede identificar un caso de disputa
  - *Ocultación externa (externally-hiding)*: si un observador externo no puede determinar qué transacciones en la cadena de bloques son componentes del protocolo *escrow*

---

<sup>38</sup> En estos contextos se manifiesta una dependencia circular: ¿qué se realiza antes, el pago o la entrega de las mercancías? O, de forma equivalente, ¿quién debe manifestar primero confianza en la contraparte?

<sup>39</sup> Vinculado a los esquemas de compartición de secretos que hemos apuntado anteriormente, y para lo que nos referíamos a, por ejemplo, [48], encontramos los esquemas de *umbral de firmas*, en los que para decodificar un mensaje o ejecutar una determinada acción se exige un mínimo número de firmas  $t$  de entre un conjunto de  $n$  partícipes. Diremos en este caso que estamos ante un  $(t, n)$ -umbral, o simplemente bajo un esquema de firmas  $t$ -de- $n$ .

- *Ocultación interna (internally-hiding)*: si el propio mediador, o autoridad *escrow*, no puede discernir si se ha ejecutado el protocolo en ausencia de disputa
- **Actividad:**
  - *Activo en depósito (active on deposit)*: un protocolo *escrow* es activo en depósito si debe participar activamente en cuanto registra un depósito
  - *Activo en disposición (active on withdrawal)*: un protocolo *escrow* es activo en disposición si debe participar activamente en la retirada de un depósito aún cuando no exista disputa
  - *Pasivo (optimistic)*: un protocolo *escrow* es pasivo, u optimista, si no es activo en depósito ni en disposición
- **Seguridad:** un protocolo *escrow* es seguro si no puede enviar el valor depositado a una dirección arbitraria sin la colaboración de alguno de los partícipes de la transacción; mención aparte se realiza sobre posibles ataques DoS (*Denial of Service*), cuya resolución tiene implicaciones en la estrategia elegida por la autoridad *escrow* para diseñar su estructura, pero cuyo tratamiento explícito y efectivo no se suele considerar en este contexto

A partir de estas definiciones, repasan los diferentes modelos de *escrow* más usuales y que cumplan el requisito de ser seguros, desde los más sencillos (como el visto de *multifirma 2-de-3*) hasta la introducción de grupos que, en lugar de hacer recaer la responsabilidad de la resolución de una disputa sobre una única autoridad, utilizan un mecanismo de mayoría entre un grupo (de cardinalidad impar) de terceros elegibles por los partícipes en una transacción. Finalmente, cada modelo queda caracterizado por el conjunto de características que cumplen, o no, de las vistas en las definiciones.

Como resumen del trabajo, los autores recomiendan dos soluciones según se busque transparencia o privacidad completa. Como soluciones transparentes, sin privacidad absoluta, eligen el protocolo básico de *multifirma 2-de-3*, ya presentado; o su derivado directo para un grupo *escrow* en el que  $A$  y  $B$  representan las firmas de las partes de una transacción y  $M_1, \dots, M_{2n+1}$  las de los  $2n + 1$  mediadores, y en el que el protocolo se limita a comprobar si se satisface:

$$(A \wedge B) \vee (A \wedge n + 1 \text{ de } \{M_1, \dots, M_{2n+1}\}) \vee (B \wedge n + 1 \text{ de } \{M_1, \dots, M_{2n+1}\})$$

es decir, o bien no hay disputa o bien una mayoría de mediadores da la razón bien a  $A$  o bien a  $B$ . Ambos protocolos son pasivos, con ocultación interna, y seguros pero con cierta debilidad ante ataques DoS (que debe tratarse aparte, como en la mayoría de estos protocolos *escrow*).

En cuanto a una solución con privacidad completa, eligen el grupo *escrow encrypt-and-swap*, que es una generalización directa a decisión por mayoría dentro de un grupo del protocolo *escrow* (simple) *encrypt-and-swap*. Para este último (no veremos aquí su generalización a grupo), resumimos la actuación del protocolo considerando un transmitente  $t$ , un beneficiario  $b$  y un mediador  $m$ :

- Las partes de la transacción (transmitente y beneficiario) generan sendos pares de claves ECDSA, resultando  $k_t$  y  $k_b$  las claves privadas respectivas, y

construyen una clave compartida<sup>40</sup> cuya componente pública es  $K = Q^{k_t+k_b}$ , siendo  $Q$  el generador del grupo (ver cap. 3, apdo. 3.4)

- El transmitente envía al beneficiario  $C_t = E_M(k_t)$ , siendo  $E_M$  la función de encriptación bajo una clave pública que en nuestro caso,  $M$ , corresponde a la clave pública del mediador  $m$ ; simétricamente, el beneficiario envía al transmitente  $C_b = E_M(k_b)$
- Si no existe disputa, el transmitente enviará la clave privada  $k_t$  al beneficiario, que podrá construir la clave necesaria (secreto compartido) para movilizar el valor; de manera inversa, el beneficiario puede renunciar al valor enviando la clave privada  $k_b$  al transmitente, que podrá recuperar el valor
- En caso de disputa, el mediador  $m$  dará la razón bien al transmitente, bien al beneficiario, que representamos con  $x \in \{t, b\}$ ; éste enviará  $C_{\{t,b\} \setminus x}$  al mediador, que podrá descifrarlo puesto que está encriptado con su clave pública, y lo devolverá a  $x$ , que contará desde ese momento con todo lo necesario para construir la clave con la que movilizar el valor.

Con la excepción (compartida, como sabemos, con casi todos los otros procedimientos) de una limitada resistencia a DoS en su diseño original, *encrypt-and-swap* en grupo cumple todas las características deseadas para una solución *escrow* adecuada que cuente, además, con privacidad completa: es un protocolo pasivo, seguro, con ocultación externa, interna y de disputa, permitiendo adicionalmente trabajar con grupos predefinidos o creados *ad-hoc*.

Conociendo las opciones de *escrow* vistas, y considerando que aún tras la confirmación de las partes a la transacción pueden darse todavía situaciones de disputa por diferentes incumplimientos (de cantidad, de calidad, de requerimientos técnicos o legales...), en este trabajo asumiremos, y exploraremos, la necesidad de la reversibilidad de las transacciones, a dilucidar y ejecutar por una autoridad cuya lógica y protocolos de actuación ante una disputa quedan fuera de nuestro ámbito por quedar, seguramente, fuera del propio ámbito de la Blockchain. Conformamos con esta posibilidad de reversión lo que podríamos denominar una *hard-regulation*, a diferencia de la mayor parte de las estrategias, que suelen introducir al regulador como un mero observador de las transacciones.

Se trata, en definitiva, de poner en manos de la correspondiente autoridad la posibilidad de modificar o revertir cualquier transmisión de valor que haya incumplido alguna/s de las reglas del marco normativo en el que pueda estar inmersa una Blockchain explotada por cualquier industria regulada, incluyendo la posibilidad de embargo dictaminado, incluso, por una autoridad externa.

Pero lo anterior se contrapone a una rígida estructura que por definición está orientada, y además con éxito, al mantenimiento inmutable de cualquier transacción, de cualquier bloque que las reúna, o de la propia cadena de bloques, de tal manera que nadie con una potencia computacional *normal* pueda modificar su contenido arbitrariamente. Esta

---

<sup>40</sup> Consideramos éste como un caso particular con un esquema 2-de-2 firmas del caso general *m-de-n* firmas en un sistema de compartición de secreto mediante umbral de firmas. Los autores se remiten a un trabajo previo de buena parte de ellos mismos [57] donde el protocolo genérico es de aplicación a este caso particular.



inmutabilidad es deseable por motivos obvios, y nuestra estrategia aquí pasará por respetarla. Si consideramos inmutable la cadena de bloques adaptada a la industria una vez incorporadas las últimas transacciones, la solución a la posible variabilidad pasa necesariamente por la externalización de las posibles modificaciones que haya que realizar, es decir, respetar el registro del estado erróneo del sistema al que se ha llegado y que siga siendo *traceable* hasta ese punto, pero con un estado final coherente de acuerdo a las modificaciones exigidas llegado el caso.

De nuevo se ofrecen diferentes estrategia para conseguir lo anterior, pero de nuevo la introducción de una autoridad central facilita una solución. Basta analizar la estructura de una transacción para convenir que las posibles modificaciones a las que podemos enfrentarnos son las vinculadas al negocio o transmisión de valor propiamente dicha, y que son precisamente las que eran objeto de ocultación o cifrado en los apartados anteriores: los intervinientes y el valor transferido. En resumen, nos podemos encontrar con que un transmitente puede cometer los siguientes errores:

- Transacción no procedente: tras su emisión acaecen hechos que obligan a su anulación o retrocesión
- Error en beneficiario: el valor se transfiere a un nuevo propietario cuando correspondía transferirlo a otro
- Error en importe: se transfiere un valor erróneo, superior al que debía haberse cursado
- Error/es en la transacción: se ejecuta una transacción con uno o varios outputs que no deberían haberse incluido

La solución obvia en todos los casos es la retrocesión mediante una nueva transacción en la que el beneficiario se convierte en transmitente, y es en general la solución usual en cualquier industria en la que no se tolere la mala fe. Pero también es frecuente la disputa comercial que puede llevar a una situación de bloqueo en la que el receptor del valor no lo retrocede a origen. Es en estas situaciones en las que, bien por actuaciones deshonestas, bien por disputa comercial, es necesario un mecanismo de reversibilidad que permita su resolución.

De manera natural, la introducción de una autoridad central, AC, deriva hacia ella la encomienda de la gestión del mecanismo de reversibilidad en caso de necesidad. Y también de manera natural surge la solución: la AC estará facultada para realizar las transacciones adicionales necesarias que permitan dejar al sistema en un estado final equivalente al existente previamente a la comisión del error a corregir. Esto significa que la AC, en caso de necesidad, debe ser capaz de ejecutar transmisiones de valor de los operadores incluidos en el sistema, *en representación* de éstos.

Si recordamos de nuevo los scripts básicos de una transacción estándar P2PKH:

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY  
OP_CHECKSIG  
scriptSig: <signature> <pubKey>
```

vemos que el problema que tenemos en este tipo de situaciones es que ni la AC puede (ni debe) llevar registro y seguimiento de todas las claves privadas del sistema, por lo que en caso de retrocesión existirá una situación singular en `scriptSig`: si bien la clave

pública <pubKey> del actual propietario del valor podría ser conocida por la AC, no lo será su clave privada con la cual computar <signature>, por lo que actuar en representación de este propietario (quizá contra su voluntad) acarrea particularidades:

1. La AC que haya tomado la decisión de retroceder un determinado output de una transacción en una situación genérica de disputa o actuaciones no correctas, bloqueará el output que otorga el derecho al valor al actual (y declarado erróneo) propietario. Este bloqueo puede realizarse mediante su eliminación del registro de UTXOs (outputs no gastados) y su inclusión en una *blacklist* de transacciones anuladas por la autoridad, registros a consultar previamente a la movilización de cualquier valor. Aunque no es complicado hacer también explícito este bloqueo en alguna estructura de datos anexa al bloque que contiene la transacción, recordemos que la permanencia en la cadena de este output no gastado no tiene efecto ante terceros puesto que no tienen acceso a su información.
2. La AC empezará a construir la nueva transacción como ya hemos visto, otorgando una clave simétrica compartida con el beneficiario (probablemente el transmitente original, que recupera ahora el valor transferido inicialmente), clave con la que se cifra <pubKeyHash> completando así `scriptPubKey` de la manera usual.
3. La AC construye una nueva transacción, especial, probablemente con un solo output, y donde `scriptSig` recogerá una firma de la propia AC y su clave pública. Enviará esta transacción al beneficiario como parte del circuito habitual.
4. El beneficiario del output actúa como ya hemos visto: comprueba su posición en el output y la corrección del valor a recibir, y confirma estos extremos a la AC.
5. Tras el paso anterior, la AC incluye esta nueva transacción en el bloque en curso para que, en su momento, sea a su vez incluido en la cadena de bloques). El estado final del sistema equivale al estado que habría resultado de no haberse ejecutado la transmisión errónea del valor.

### **6.5 El consenso bajo una autoridad central:**

Hemos visto en el apartado 6.3, cuando repasábamos los efectos de la introducción de una autoridad central en Blockchain, que el procedimiento de construcción y gestión de una transacción se complica mucho en comparación con la relativa sencillez del lanzamiento de una transacción en el sistema Bitcoin/Blockchain.

En el procedimiento que presentábamos quedaba implícita la posposición de la cuestión del consenso global sobre las transacciones, abstrayéndolo en algún mecanismo oculto previo al registro definitivo en la cadena de bloques. No obstante, y como ya anticipábamos al terminar ese apartado, era fácil intuir que en ese mecanismo no presentado en esos momentos tendría un papel determinante la propia autoridad central.

Al final del capítulo 4 hacíamos referencia al consenso como artefacto emergente de la red, que se originaba por la interacción de cuatro procesos independientes ([28], cap. 8):

- Verificación independiente de cada transacción
- Agrupación independiente de las transacciones en bloques tras un minado basado en *proof-of-work*

- Verificación independiente de cada nuevo bloque y agregación a la blockchain por todos los nodos
- Selección independiente por cada nodo de la cadena con mayor trabajo computacional acumulado y demostrado por la PoW

A la luz de la vinculación con estos procesos, conviene repasar las características de una Blockchain industrial tal y como la hemos abstraído:

- La validación de las transacción se encomienda a una autoridad que centraliza un previo proceso de conformidad por parte de todos los operadores (transmitente y beneficiario/s)
- Esta misma autoridad tiene facultades tan amplias sobre el sistema que le permitirían, llegado el caso y asumiendo la toma de todas las cautelas necesarias, la modificación o reversión de las transacciones y, por tanto, la intervención del sistema y su estado final
- Implícito en los puntos anteriores, si la autoridad es responsable de la validación de las transacciones no existen motivos para que no lo sea, también, de la construcción y verificación de cada bloque, que ya sabemos es un almacén con dimensión temporal de las transacciones

En cuanto a la *proof-of-work*, a la vista de la autoridad omnímoda que se hace cargo de toda verificación en el sistema, hay que relativizar su trascendencia en el modelo regulado de Blockchain: si la autoridad central se ocupa de la validación de las transacciones y los bloques, tampoco parece existir motivo para que no lo haga de la inclusión de los bloques en la cadena. Si a una utilidad relativa como la que podría tener la PoW en la nueva situación le añadimos el prohibitivo coste de la misma que hemos analizado en el apartado 5.1, y que ha marcado como necesaria su transformación según hemos visto en la *matriz de transformación de correspondencias* (tabla 5.1, cap. 5) que nos ha servido de marco de trabajo, es evidente que podemos dejar de contar con ella.

Por resumir, el consenso bajo autoridad central, tal y como hemos diseñado el mecanismo de transacciones en este trabajo, viene dado por lo siguiente:

1. Comprobación por parte de cada beneficiario de su posición en el correspondiente output de una transacción: la dirección que lo vincula a su identidad y el valor que recibe son cifrados, compartidos con la AC, y confirmados a la misma por cada beneficiario, que *sella* así su corrección e imposibilita su acceso por terceros no concernidos
2. Recolección por parte de la AC de todas las confirmaciones de todos los beneficiarios de todos los outputs de una transacción
3. Revisión por parte de la AC de todos los inputs de cada transacción, cifrados en su información sensible por claves simétricas compartidas entre el propietario que ahora moviliza su valor y la AC
4. Recolección e inclusión en un bloque por la AC de todas las transacciones validadas durante el último periodo de tiempo de referencia
5. Inclusión del bloque construido en el punto anterior en la cadena de bloques, que seguirá manteniendo las dos dimensiones conocidas de la cadena de bloques de Bitcoin/Blockchain: la temporal (vía la confección periódica de bloques) y la del tracto de cada valor transferido. La AC asegura la integridad de la cadena.

De la misma manera que en Bitcoin/Blockchain se necesitan registros adicionales a la cadena de bloques para su operatividad (apdo. 4.6, cap. 4), también serán necesarios ahora, entre otros:

- Registro de UTXOs: base de datos de los outputs no gastados, gestionada por la AC y compartida con todos los operadores, pero cifrada de forma parecida a la los outputs. La información solo debe ser accesible según su propiedad
- *Blacklist* de UTXOs revertidos por la AC: ya descrita en el apartado 6.4
- Metadatos sobre los bloques: base de datos que permite búsquedas eficientes en el contenido de los bloques
- Registro inverso: indización desde los outputs a sus correspondientes inputs
- Registros individuales: de forma semejante a los *wallets*, es conveniente que cada partícipe registre sus outputs (gastados o no), las transacciones emitidas, o las claves privadas y compartidas, entre otra posible información útil
- Cadena de bloques en claro: para la administración de la autoridad central puede resultar conveniente que ésta vaya construyendo una copia privada de la cadena de bloques con toda la información en claro, sin cifrar

## Conclusiones

Plantearemos inmediatamente las conclusiones finales del trabajo, para concluir a continuación el estudio con posibles líneas de trabajo adicionales, que ya hoy suponen una ebullición de ideas, diseños y prototipos para diferentes industrias, y en particular la financiera.

### 7.1 ¿Es Blockchain bajo regulación una verdadera Blockchain?

Con todo lo que hemos podido ver a lo largo de este trabajo, y en particular en los capítulos 5 y 6, deberíamos ser capaces de reconocer inmediatamente una de las principales conclusiones a la que hemos llegado: que bajo una autoridad central ya no estamos ante un consenso emergente desde lo que venimos identificando, al menos conceptualmente, como una *autoridad distribuida*, sino ante un consenso emanado e impuesto desde la propia autoridad central, que es exactamente el mismo fundamento actual (e histórico) de toda industria sometida a regulación. Y normalmente aceptado por todos sus partícipes, cabría añadir. ¿Estamos, entonces, en el mismo punto de partida existente antes de la aparición de Blockchain?

En términos de los propósitos últimos de Blockchain, con la introducción de la autoridad eliminamos la principal premisa que Nakamoto introducía en su *paper* [1] y que veíamos en nuestra introducción: los principales beneficios se pierden si una tercera parte de confianza, una autoridad, se requiere para evitar el doble pago.<sup>41</sup> Lo ratificamos con este trabajo.

En efecto, la autoridad que conceptualmente se ha hecho cargo de nuestro sistema acomete tres funciones fundamentales:

- Regula y vigila el cumplimiento de las normas de este nuevo sistema: debe ser capaz de acceder a toda la información.
- Es capaz de intervenir el sistema en caso necesario: debe poder emitir transacciones válidas de forma delegada, en nombre de otros operadores, llegado el caso.
- Aporta el mecanismo de consenso necesario para que las partes confíen en el sistema: debe ser capaz de validar las transacciones de forma transparente a sus partícipes y de recoger el consenso de todos ellos para, una vez alcanzado este consenso de los partícipes de una transacción, registrar la misma en su correspondiente bloque temporal, y éste en la cadena de bloques.

¿Es posible en este contexto regulado evitar este último punto de manera que mantengamos la filosofía inicial de Nakamoto en cuanto a proporcionar un consenso

---

<sup>41</sup> Ya desde el resumen de su *paper*, Nakamoto indica que "...Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending..."

global emanado de una red distribuida en lugar de por una autoridad central? Quizá técnicamente sea posible, aunque con serios inconvenientes:

- La mayor complejidad del sistema resultante, puesto que manteniendo la capacidad de regulación e intervención (que son innegociables en una industria regulada) habría que habilitar un mecanismo de consenso externo, tipo *proof-of-work*, para el que tendrían que habilitarse recursos de confirmación de las transacciones, que además están cifradas como sabemos.
- El coste del consenso, del que ya hemos avanzado que en su implementación actual como *proof-of-work* para Bitcoin/Blockchain es prohibitivo para cualquier industria, pero del que cualquier diseño que evite la participación de una autoridad está condenado a resultar más gravoso que si ésta, o un representante, lo asume directamente de forma natural.
- Finalmente, y más allá de la tecnología, quedaría por resolver cualquier posible colisión entre una decisión de la autoridad y otra del consenso distribuido: ¿cómo se arbitra y cuál prevalece?

Pero no es solo el consenso emergente de la red distribuida lo que se pierde con una Blockchain bajo regulación. En este trabajo apenas lo hemos referido, pero el concepto de *autorregulación* (nada que ver con la regulación en los mercados, sino en su sentido de sistema autónomo), que seguramente merecería otro trabajo adicional, y que está cimentado por la distribución de la computación, la replicación de los resultados, y el consenso global, está implícito en toda la estructura de Bitcoin/Blockchain. Hagamos abstracción de la necesidad de mantenimiento y actualización de todo software complejo, y supongamos que el código de Bitcoin/Blockchain hubiera alcanzado su máxima corrección y eficiencia: una vez distribuido de forma suficientemente global en los nodos de la red, el sistema no requerirá de intervención externa para mantenerse operacional y cumplir sus funcionalidades sin verse comprometido por caídas de nodos, actuaciones deshonestas (hasta los límites razonables ya conocidos), intentos de intervención... Estaríamos ante un *robot* autónomo y virtual capaz de instanciar consensos y cuya supervivencia estaría ligada, únicamente, a la de la propia red y a la de sus usuarios. Por su parte, y por definición, la regulación es cambiante, a veces caprichosa, y siempre está reñida con la autorregulación.

Acabamos este apartado: la respuesta es no, al menos hasta donde sabemos. Una Blockchain regulada, sin consenso emergente ni autorregulación, tal y como la hemos descrito, no es la Blockchain original, disruptiva y novedosa que Nakamoto nos apunta en su trabajo. No obstante, la aparición de Blockchain ha marcado un antes y un después en el diseño de la futura interrelación entre operadores financieros, y la búsqueda de mejores, más eficientes y más baratos circuitos para las transmisiones de valor está siguiendo la huella de la Blockchain original, que actúa como referencia, todavía administrativa pero referencia al fin y al cabo. Recordemos los ejemplos de proyectos en esta línea que hemos podido ver en el apartado 5.4 del capítulo 5.

## **7.2 Conclusiones: un resumen**

Enumeramos a continuación las principales conclusiones obtenidas en la realización de este trabajo. Las conclusiones C4 a C8, y C10, responden, sin ningún orden en

particular, a los objetivos principales perseguidos, a las propuestas de este trabajo, y a las lecciones aprendidas. En particular, C7 resume nuestra propuesta de diseño, mientras que C8 es el resultado de la probablemente imposible convivencia entre una autoridad central y otra distribuida, que dejaría sin resolver aspectos como la posible colisión entre decisiones dispares de una y otra. El resto de conclusiones (algunas, como C1 o C2, muy resumidas) corresponden a los apartados descriptivos del trabajo:

- C1. La criptomoneda Bitcoin y su estructura soporte, Blockchain, nacen con una clara orientación descentralizada y carente de autoridad central o intermediarios, que hace innecesaria la presencia de terceros de los que emane la confianza en el sistema monetario virtual. Al contrario, de esa autoridad descentralizada emerge un consenso basado en la verificación independiente de las transacciones, de los bloques y de la cadena de bloques, así como del sellado de cada bloque mediante una *proof-of-work* basada en una función *hash* criptográfica, computacionalmente costosa, que garantiza la confiabilidad de la blockchain resultante siempre que un posible atacante no obtenga una potencia computacional semejante a la de los nodos honestos de la red.
- C2. Una criptomoneda es una *cadena de firmas digitales*: un determinado valor queda vinculado a una clave pública de un sistema de criptografía asimétrica, y para su transmisión (a una clave pública del nuevo beneficiario de la transacción) debe manifestarse la propiedad del valor transferido haciendo uso de la clave privada par de la pública mediante una firma electrónica sobre la propia transacción. En cada transacción existirán outputs (beneficiarios bajo un alias conformado por su clave pública) e inputs (orígenes de los valores a transferir que son outputs de transacciones anteriores, y que son consumidos -gastados- en la transacción). En este sistema Bitcoin/Blockchain un valor se representa por un output no gastado.
- C3. La aparición de Blockchain está marcando un antes y un después en el diseño de los próximos modelos de interrelación entre operadores de diferentes mercados, pero en particular del financiero. Los desarrolladores buscan mejores y más eficientes circuitos de transmisión y gestión del valor, sobre todo por la promesa de menores costes que Blockchain (y los mecanismos derivados de ésta recogidos genéricamente bajo la denominación de *Distributed Ledger Technology*) apunta al permitir el mantenimiento de un único registro compartido y conciliado globalmente, en lugar de los registros individuales actuales que requieren costosos recursos de gestión a multiplicar por cada organización.
- C4. La estructura del sistema actual Bitcoin/Blockchain puede adaptarse a los requisitos de una industria regulada. En particular, cuestiones como la privacidad en las transacciones y la habilitación de la supervisión e intervención de una autoridad central pueden resolverse con mecanismos criptográficos habituales y ciertos cambios en la estructura de las transacciones y el sistema. No se mantiene compatibilidad con la Blockchain original.
- C5. La privacidad en las transacciones sin una figura de regulación puede lograrse mediante mecanismos adicionales no triviales (*ValueShuffle*, *Confidential Transactions*, *Stealth Addresses*...), pero la necesaria introducción de una autoridad reguladora dificulta su aplicación puesto que debe habilitarse el acceso de esta autoridad a tales transacciones. Es, precisamente, la introducción de una autoridad

la que puede facilitar una nueva mecánica de la privacidad en partícipes y valores, con el coste de la introducción y gestión de nuevas claves (simétricas en nuestra propuesta) en el sistema, así como un muy ligero incremento en el tamaño de las transacciones.

- C6. Por medio de un servicio *escrow* puede añadirse una capa de confirmación de las transacciones y resolución de disputas entre las partes de una transacción. Pero no se resuelve la irreversibilidad de las transacciones, para la que, de nuevo, la introducción de una autoridad simplifica su resolución.
- C7. Hemos propuesto, en definitiva, una posible solución mediante el cifrado simétrico sobre los elementos de negocio de una transacción (partícipes y valor), compartido entre beneficiarios de las transacciones y una autoridad central, y que habilita el acceso de los partícipes solo a la información que les concierne, mientras que la autoridad puede acceder a la totalidad de la cadena (apdo. 6.3). Además, solucionamos la intervención de tal autoridad mediante la reversibilidad de las transacciones de acuerdo a lo que proponemos en el apdo. 6.4 para ciertas transacciones especiales solo ejecutables por una autoridad con suficientes facultades.
- C8. La introducción de esta autoridad con facultades absolutas distorsiona la filosofía original de Bitcoin/Blockchain: el consenso descentralizado da paso al consenso por autoridad (es decir, a la situación ordinaria de los mercados *clásicos*), y la autorregulación del sistema desaparece para habilitar la mutabilidad asíncrona sobre el mismo que caracteriza la actuación de cualquier autoridad, que por definición suele demandar alteraciones por motivos económicos, fiscales, políticos, sociales... Tras finalizar el trabajo, se ratifica la dificultad de la convivencia de una estructura o autoridad descentralizada con otra centralizada y con facultades de intervención. En resumen, una Blockchain bajo regulación no es, excepto en sus aspectos más de forma y más superficiales, la Blockchain original.
- C9. Desde la industria se están proponiendo actualmente diferentes proyectos (cuyos más significativos ejemplos hemos revisado en el apdo. 5.4), y en todos se contemplan las necesidades de los mercados regulados a los que van dirigidos. No obstante, y aunque el análisis que hemos realizado sobre ellos en este trabajo es necesariamente limitado, podemos concluir que mientras el cifrado de la información sensible está generalmente contemplado (aunque haya que contar con mecanismos adicionales, como en Ethereum), no está tan generalizada la solución de la posible participación de autoridades. No obstante, Corda sí tiene en consideración una jerarquía de operadores, y Ripple propone una traslación de responsabilidades económicas a unos suministradores de liquidez que supondrían, en nuestra opinión, una cierta clase de regulación e intervención del sistema *de facto*.
- C10. Las posibles soluciones aportadas desde los diferentes ámbitos para introducir Blockchain a la regulación, incluidas las apuntadas desde este trabajo, suelen adolecer de características no deseables. De manera muy resumida y general:
- Las que se adhieren a la actual estructura Bitcoin/Blockchain intentando evitar, en la medida de lo posible, un *hard-fork* en el sistema (propuestas normalmente provenientes de la comunidad investigadora y de



desarrolladores del ámbito de las criptomonedas), no acaban de introducir un concepto formal de autoridad; y *ni falta que hace*, podemos pensar que argumentarán desde esos ámbitos, puesto que Bitcoin/Blockchain nace en su momento como paradigma de estructura desregulada y descentralizada.

- Las propuestas que provienen del ámbito industrial, aunque todavía inacabadas, suelen contemplar la autoridad como mera espectadora, y cuyas posibles vías de actuación son únicamente por fuera de la tecnología que nos ocupa (*off-chain*), esto es, que la autoridad actúe como una especie de *deus ex machina* que soluciona las incidencias extremas de la Blockchain de una manera mágica y ajena al propio sistema. Quizá sea el enfoque definitivo (¿único?) que la industria requiere, pero aún así quedaría por analizar como se homogeneizan y se hacen coherentes los estados del sistema y de la realidad: si un operador se niega, por ejemplo, a retroceder una transacción en un contexto de irreversibilidad de las mismas, y aunque se tomaran medidas en el mundo físico, ¿cómo se ajusta el estado de la blockchain?
- Las que proponemos en este trabajo, en particular la posibilidad de construcción de transacciones por la autoridad bajo delegación, y que podemos denominar como *hard-regulation*, modifican tanto los fundamentos de la Blockchain que se generan dudas sobre si podemos seguir denominándola así.

### 7.3 Líneas adicionales de estudio y trabajo

Concluimos los aspectos conceptuales del trabajo con materias destacadas que no correspondía tratar en profundidad aquí, o que podrían ser tomadas en consideración en otros estudios vinculados:

- Hasta donde hemos podido ver, no hay trabajos relevantes sobre un posible equilibrio entre autoridad central y autoridad descentralizada que permitiera lo mejor de ambos mundos en un sistema fundamentado en Blockchain u otros DLTs. La conclusión más directa, y a la que en parte colaboramos con este trabajo, es que no es posible, o al menos no lo es fácilmente, tal equilibrio. Pero eso supondría resignarnos a perder la posibilidad y los beneficios del consenso emergente y la autorregulación. Estos beneficios son, en nuestra opinión, tan disruptivos y de una importancia tan extraordinaria que merecen un análisis en profundidad, no solo de sus fundamentos (que ya los hay), sino en su aplicabilidad práctica a un mundo tan regulado como es el nuestro.
- Por extensión, tras la impaciencia actual sobre Blockchain en cuanto a sus aplicaciones industriales y su posible resolución, queda por analizar su aplicabilidad directa a consumidores u otros partícipes minoristas. Si la regulación dentro de circuitos establecidos para operadores mayoristas es elevada, la introducción de consumidores u otras unidades microeconómicas (pequeñas empresas o negocios) lleva el problema a dimensiones hoy, hasta donde sabemos, no examinadas.
- Los *smart contracts*, o contratos inteligentes, son en esencia *triggers* de transacciones que aseguran su ejecución si se han cumplido una serie de condiciones que se han pactado previamente entre las partes, y que han sido codificados convenientemente para recoger ese cumplimiento en cuanto se produzca. Por su

naturaleza, son extensión directa de sistemas tales como Blockchain puesto que su ámbito natural se encuentra en servir de puente entre, por ejemplo, una Blockchain monetaria y otra sobre cualquier activo: si se cumple la entrega de la propiedad de un activo, se garantiza la ejecución de una transferencia monetaria, o a la inversa.

- Como vimos en el apdo. 5.4 al repasar algunos de los proyectos en curso, las DAO (*Decentralized Autonomous Organizations*) serían una especie de *más allá* de los contratos inteligentes: corresponderían a toda una organización, quizá buena parte de la estructura societaria y administrativa de una empresa, resumida en una infraestructura distribuida, basada en Blockchain y *smart contracts*, que habilitaría diferentes cursos de actuación (la ejecución alternativa de diferentes tipos de contratos) según una cualificada mayoría de partícipes con derecho a voto; o incluso la gestión automatizada de medios de producción autónomos, para lo que se suele poner como ejemplo la no muy lejana explotación de un vehículo de alquiler de conducción autónoma [58], que conectado a las correspondientes blockchains podrá cobrar por sus servicios o satisfacer sus correspondientes gastos de suministros, de mantenimiento, de dividendos a sus propietarios...

## Referencias/Bibliografía

- [1] Nakamoto, S.: *A Peer-to-Peer Electronic Cash System*. <https://bitcoin.org/bitcoin.pdf> (2008)
- [2] Jacobson, I. et al.: *The Unified Software Development Process*. Addison Wesley (1999)
- [3] Chaum, D.: *Blind Signatures for Untraceable Payments*. <http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF> (1990)
- [4] Rosen, S.: *Electronic-monetary System*. <http://www.freepatentsonline.com/5963648.html> (1997)
- [5] Back, A.: *Hascash - A Denial of Service Counter-Measure*. <http://www.hashcash.org/papers/hascash.pdf> (2002)
- [6] Merkle, R.C.: *A Digital Signature Based on a Conventional Encryption Function*. [http://link.springer.com/chapter/10.1007/3-540-48184-2\\_32](http://link.springer.com/chapter/10.1007/3-540-48184-2_32) (1988)
- [7] Feller, W.: *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons (1968)
- [8] Ozisik, A.P., Levine, B.N.: *An Explanation of Nakamoto's Analysis of Double-spend Attacks*. <https://arxiv.org/pdf/1701.03977> (2017)
- [9] National Institute of Standards and Technology - NIST: *FIPS 180-4 - Secure Hash Standard*. <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf> (2015)
- [10] National Institute of Standards and Technology - NIST: *FIPS 202 - SHA-3 Standard*. <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf> (2015)
- [11] Dobbertin, H. et al.: *RIPEMD-160: A Strengthened Version of RIPEMD*. <http://www.esat.kuleuven.be/~bosselae/ripemd160/pdf/AB-9601/AB-9601.pdf> (1996)
- [12] Diffie, W., Hellman, M.E.: *New Directions in Cryptography*. IEEE Transactions on Information Theory, vol. IT-22, pág. 644. <https://ee.stanford.edu/%7Ehellman/publications/24.pdf> (1976)
- [13] Rivest, R.L., Shamir, A., Adleman, L.M.: *Cryptographic communications system and method*. <https://www.google.com/patents/US4405829> (1977)
- [14] ElGamal, T.: *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. [http://link.springer.com/content/pdf/10.1007%2F3-540-39568-7\\_2.pdf](http://link.springer.com/content/pdf/10.1007%2F3-540-39568-7_2.pdf) (1985)
- [15] Koblitz, N.: *Elliptic Curve Cryptosystems*. Mathematics of Computation, vol. 48, n. 177, pág. 203. <http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf> (1985)
- [16] Miller, V.S.: *Use of Elliptic Curves in Cryptography*. CRYPTO '85, LNCS 218, pág. 417. [http://link.springer.com/content/pdf/10.1007%2F3-540-39799-X\\_31.pdf](http://link.springer.com/content/pdf/10.1007%2F3-540-39799-X_31.pdf) (1985)
- [17] Delfs, H., Knebl, H.: *Introduction to Cryptography*, 3<sup>a</sup> ed. Springer (2015)
- [18] Hankerson, D., et al.: *Guide to Elliptic Curve Cryptography*. Springer (2004)
- [19] López, J., Dahab, R.: *An Overview of Elliptic Curve Cryptography*. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.2771> (2000)

- [20] Goldwasser, S., Bellare, M.: *Lecture Notes on Cryptography*. [diamond.boisestate.edu/~liljanab/MATH308/CryptoBook.pdf](http://diamond.boisestate.edu/~liljanab/MATH308/CryptoBook.pdf) (2008)
- [21] Barker, E., et al. - National Institute of Standards and Technology - NIST: *SP 800-57 - Recommendations for Key Management - Part 1: General (Revised)*. <https://dx.doi.org/10.6028/NIST.SP.800-57p1r2007> (2007)
- [22] Goldwasser, S., et al.: *A digital signature scheme secure against adaptive chosen message attacks*. SIAM Journal on Computing (1988)
- [23] Lamport, L., et al.: *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, vol. 4, núm. 3, pág. 382. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/The-Byzantine-Generals-Problem.pdf> (1982)
- [24] Pérez-Solà, C., Herrera-Joancomartí, J.: *Bitcoins y el problema de los generales bizantinos*. <https://web.ua.es/en/recsi2014/documentos/papers/bitcoins-y-el-problema-de-los-generales-bizantinos.pdf> (2014)
- [25] Clark, C: *Bitcoin Internals - A Technical Guide to Bitcoin*. (2013)
- [26] VV.AA.: *Script*. <https://en.bitcoin.it/wiki/Script>. Bitcoin Core Documentation - Developer Documentation (2017)
- [27] LevelDB: *LevelDB documentation*. <https://github.com/google/leveldb/blob/master/doc/index.md>. (2017)
- [28] Antonopoulos, A.M.: *Mastering Bitcoin - Unlocking Digital Cryptocurrencies*. O'Reilly (2014)
- [29] Donet, J.A., et al.: *The Bitcoin P2P network*. [http://fc14.ifca.ai/bitcoin/papers/bitcoin14\\_submission\\_3.pdf](http://fc14.ifca.ai/bitcoin/papers/bitcoin14_submission_3.pdf) (2014)
- [30] Organ Ofcorti (seudónimo): *May 10th 2015 Network Statistics*. Neighbourhood Pool Watch blog. <http://organofcorti.blogspot.com.es/2015/05/may-10th-2015-network-statistics.html> (2015)
- [31] Aste, T.: *The fair cost of Bitcoin proof of work*. <http://ssrn.com/abstract=2801048> (2016)
- [32] VV.AA.: *Segregated Witness Benefits*. BitcoinCore.org. <https://bitcoincore.org/en/2016/01/26/segwit-benefits/> (2016)
- [33] VV.AA.: *Segregated witness: the next steps*. BitcoinCore.org. <https://bitcoincore.org/en/2016/06/24/segwit-next-steps/> (2016)
- [34] VV.AA.: *Segregated Witness Costs and Risks*. BitcoinCore.org. <https://bitcoincore.org/en/2016/10/28/segwit-costs/> (2016)
- [35] VV.AA.: *Segregated Witness Upgrade Guide*. BitcoinCore.org. <https://bitcoincore.org/en/2016/10/27/segwit-upgrade-guide/> (2016)
- [36] Rizun, P.R.: *A Transaction Fee Market Exists Without a Block Size Limit*. <https://www.bitcoinunlimited.info/resources/feemarket.pdf> (2015)
- [37] VV.AA.: *Scalability FAQ*. bitcoinwiki. [https://en.bitcoin.it/wiki/Scalability\\_FAQ](https://en.bitcoin.it/wiki/Scalability_FAQ) (2016)
- [38] VV.AA.: *Hyperledger Whitepaper*. hyperledger.org. <https://wiki.hyperledger.org/groups/whitepaper/whitepaper-wg> (2016)
- [39] Brown, R.G., et al.: *Corda: An Introduction*. <https://www.r3cev.com/s/corda-introductory-whitepaper-final.pdf> (2016)
- [40] Mills, D., et al.: *Distributed ledger technology in payments, clearing and settlement*. Finance and Economics Discussion Series 2016-095. Board of Governors of the Federal Reserve System. <https://doi.org/10.17016/FEDS.2016.095> (2016)

- [41] VV.AA.: *Recommendation X.509*. International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). <http://www.itu.int/rec/T-REC-X.509-201610-I/en> (2016)
- [42] Hearn, M.: *Corda: A distributed ledger (Corda Technical White Paper)*. [https://docs.corda.net/\\_static/corda-technical-whitepaper.pdf](https://docs.corda.net/_static/corda-technical-whitepaper.pdf) (2016)
- [43] Castro, M., Liskov, B.: *Practical Byzantine fault tolerance*. ACM DL. <http://dl.acm.org/citation.cfm?id=296824> (1999)
- [44] VV.AA.: *Ethereum White Paper*. ethereum.org. <https://github.com/ethereum/wiki/wiki/White-Paper> (2017)
- [45] VV.AA.: *Ripple: Product Overview*. Ripple. [https://ripple.com/files/ripple\\_product\\_overview.pdf](https://ripple.com/files/ripple_product_overview.pdf) (2017)
- [46] Castro, M., Liskov, B.: *Byzantine fault tolerance*. <https://www.google.com/patents/US6671821> (2000)
- [47] Maxwell, G.: *Confidential Transactions*. [https://people.xiph.org/~greg/confidential\\_values.txt](https://people.xiph.org/~greg/confidential_values.txt) (2015)
- [48] Martin, K.M.: *Challenging the adversary model in secret sharing schemes*. [www.isg.rhul.ac.uk/~martin/files/Brusselsfinal.pdf](http://www.isg.rhul.ac.uk/~martin/files/Brusselsfinal.pdf) (2008)
- [49] Pedersen, T.P.: *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. CRYPTO '91. [http://link.springer.com/chapter/10.1007/3-540-46766-1\\_9](http://link.springer.com/chapter/10.1007/3-540-46766-1_9) (1991)
- [50] Maxwell, G.: *CoinJoin: Bitcoin privacy for the real world*. <https://bitcointalk.org/index.php?topic=279249> (2013)
- [51] Todd, P.: *Stealth Addresses*. <https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html> (2014)
- [52] Certicom Research: *SEC 1: Elliptic Curve Cryptography*. <http://www.secg.org/sec1-v2.pdf> (2009)
- [53] Menezes, A., et al.: *An Efficient Protocol for Authenticated Key Agreement*. <http://www.cacr.math.uwaterloo.ca/techreports/1998/corr98-05.pdf> (1998)
- [54] Ruffing, T., Moreno-Sanchez, P.: *Mixing Confidential Transactions: Comprehensive Transaction Privacy for Bitcoin*. <https://people.mmci.uni-saarland.de/~truffing/papers/valueshuffle.pdf> (2016)
- [55] National Institute of Standards and Technology - NIST: *FIPS 97 - Advanced Encryption Standard (AES)*. <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf> (2001)
- [56] Goldfeder, S., et al.: *Escrow protocols for cryptocurrencies: How to buy physical goods using Bitcoin*. Financial Cryptography and Data Security (abril 2017). [www.jbonneau.com/doc/GBGN17-FC-physical\\_escrow.pdf](http://www.jbonneau.com/doc/GBGN17-FC-physical_escrow.pdf) (2017)
- [57] Gennaro, R., et al.: *Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security*. <https://eprint.iacr.org/2016/013.pdf> (2016)
- [58] Boucher, P.: *How Blockchain technology could change our lives*. European Parliamentary Research Service. [http://www.europarl.europa.eu/RegData/etudes/IDAN/2017/581948/EPRS\\_IDA\(2017\)581948\\_EN.pdf](http://www.europarl.europa.eu/RegData/etudes/IDAN/2017/581948/EPRS_IDA(2017)581948_EN.pdf) (2017)
- [59] VV.AA.: *Protocol rules - "tx" messages*. bitcoinwiki. [https://en.bitcoin.it/wiki/Protocol\\_rules#22tx.22\\_messages](https://en.bitcoin.it/wiki/Protocol_rules#22tx.22_messages) (2016)
- [60] Okupski, K.: *Bitcoin Developer Reference*. enetium.com/resources/Bitcoin.pdf (2016)

## Algunos formalismos

Aunque conceptos como intratabilidad, complejidad, adversario u otros vinculados al campo que nos ocupa tienen interpretaciones intuitivas aparentemente directas, éstas no son válidas, o no lo son suficientemente, si queremos evitar inexactitudes o sesgos en un contexto formal. A continuación definiremos formalmente una serie de conceptos utilizados en el trabajo y cuya introducción informal en el mismo ha resultado más conveniente para una menos densa y más fluida presentación. Algunos se corresponden directamente a lo referenciado, mientras que otros los necesitaremos para definir correctamente a los anteriores.

En general, seguiremos a [20].

### A.1 Máquina de Turing

Conceptualmente, una *máquina de Turing* es una máquina de estados finitos construida sobre una cinta de longitud no necesariamente finita y en la que se representan símbolos de un determinado alfabeto finito  $K$ . Siguiendo una determinada función de transición, dependiendo del estado actual y del símbolo leído, la máquina escribe un nuevo símbolo en el lugar del leído (quizá el mismo), se mueve hacia delante o hacia atrás o se queda inmóvil, y entra en un nuevo estado. Opcionalmente, si la máquina se para puede devolver sí o no.

Formalmente, una máquina de Turing queda definida por un alfabeto finito  $\Sigma$ , un conjunto finito de estados  $K$  que incluye el estado inicial  $s$ , y una función de transición

$$\delta: K \times \Sigma \rightarrow (K \cup \{\text{alto}, \text{sí}, \text{no}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$$

donde los conjuntos  $\Sigma$ ,  $K$ ,  $\{\text{alto}, \text{sí}, \text{no}\}$  y  $\{\leftarrow, \rightarrow, -\}$  son disjuntos. Además,  $\Sigma$  contiene dos elementos especiales,  $\triangleright$  y  $\sqcup$ , que representan el comienzo y el fin, respectivamente, de la *cinta* de la máquina de Turing, requiriéndose

$$\forall q \in K, \delta(q, \triangleright) = (p, \sigma, d) \Rightarrow (\sigma = \triangleright \wedge d \neq \leftarrow)$$

con  $p \in K$ ,  $\sigma \in \Sigma$  y  $d \in \{\leftarrow, \rightarrow, -\}$ . Es decir, que la máquina nunca sobrescribe el símbolo de comienzo de la cinta ni se sitúa más a la izquierda de él.

El conjunto  $\Sigma^*$  es el conjunto de secuencias finitas de elementos (símbolos) de  $\Sigma$ . Podremos representar mediante  $x$  un elemento de  $\Sigma^*$ , y por  $|x|$  la longitud (número de símbolos) de  $x$ .

Una configuración de una máquina de Turing es una *3-tupla*  $(x, q, k) \in \Sigma^* \times K \times \mathbb{N}$ , donde  $x$  denota la cadena representada en la cinta de la máquina,  $q$  el estado actual de la misma, y  $k$  la posición de la máquina. Un cómputo de una máquina de Turing es una secuencia de configuraciones  $(x_i, q_i, k_i)$ , donde  $i$  recorre desde 0 hasta  $T$ , con  $T$  no

necesariamente finito, y donde cada par consecutivo representa una transición válida. Asumiendo que la máquina comienza su cómputo con una configuración válida ( $q_0 = s$  y  $k_0 = 0$ ), entonces si  $T = \infty$  diremos que el cómputo no se para, mientras que si  $T < \infty$ , decimos que la computación se detiene y se cumplirá que  $q_T \in \{alto, sí, no\}$ .

Para una máquina de Turing  $M$ , el resultado de un cómputo lo denotaremos como  $M(x)$ , donde  $x$  es la cadena de entrada (sin incluir los símbolos  $\triangleright$  y  $\sqcup$ ).  $M(x)$  podrá resultar en los valores *sí*, *no*, o la cadena final en la cinta (sin los símbolos  $\triangleright$  y  $\sqcup$ ) si  $q_T = alto$ . Si la computación no se detiene lo representaremos como  $M(x) = \nearrow$ .

## A.2 Clases de complejidad P, NP y BPP

Un lenguaje  $L$  se encuentra en la clase de complejidad P (*Polynomial*) si existe una máquina de Turing  $M$  y un polinomio  $Q(y)$  tales que la entrada  $x \in L$  si y solo si  $M$  computa bajo  $x$  (que denotamos  $M(x)$ ), y  $M(x)$  se obtiene como mucho en  $Q(|x|)$  pasos.

Esta clase P corresponde a los lenguajes que denominamos '*fácilmente computables*' o '*factibles*'. Utilizaremos '*algoritmo eficiente*' como la denominación de una máquina de Turing que actúa en tiempo polinómico.

Un lenguaje  $L$  se encuentra en la clase NP (*Non-deterministic Polynomial*) si existe una máquina de Turing  $M$ , una entrada  $x$ , unos polinomios  $p$  y  $l$ , y una cadena  $y$  de símbolos tales que  $x \in L \Rightarrow \exists y, |y| \leq l(|x|)$ , tal que  $M(x, y)$  computa y se obtiene en como mucho  $p(|x|)$  pasos. Por otra parte,  $x \notin L \Rightarrow \forall y, |y| \leq l(|x|)$ ,  $M(x, y)$  se rechaza y no es posible su computación.

En otras definiciones solemos encontrar que  $L$  es NP si existe una máquina de Turing no determinista de tiempo polinómico que acepta  $x$  si y solo si  $x \in L$ . En nuestro caso, y correspondería al soporte del no determinismo de  $M$ .

Un lenguaje  $L$  se encuentra en la clase BPP (*Bounded-error Probabilistic Polynomial*) si con las mismos elementos de premisa del caso NP tenemos:

$$x \in L \Rightarrow \text{Prob}_{|y| \leq l(|x|)}(M(x, y) \text{ compute}) \geq \frac{2}{3}$$

$$x \notin L \Rightarrow \text{Prob}_{|y| \leq l(|x|)}(M(x, y) \text{ compute}) \leq \frac{1}{3}$$

y  $M(x, y)$  se computa como mucho en  $p(|x|)$  pasos. Los lenguajes de esta clase también se consideran '*fácilmente computables*' o '*factibles*'.

Está demostrado que  $P \subseteq NP$  y que  $P \subseteq BPP$ , pero no se han demostrado otras relaciones entre ellas.

Por extensión, un problema se considera '*computacionalmente intratable*', o simplemente '*intratable*', si no existe un algoritmo que lo resuelva en tiempo polinomial.

### A.3 Algoritmos probabilísticos

Definimos una *máquina de Turing probabilística de tiempo polinomial* o, de forma equivalente, un *algoritmo probabilístico de tiempo polinomial*, a una máquina de Turing extendida que implementa una nueva instrucción equivalente a lanzar una moneda. La clase de complejidad BPP se podría haber definido con este tipo de máquinas de Turing.

Un algoritmo probabilístico de tiempo polinomial, o algoritmo PPT (*Probabilistic Polynomial Time*), se considera un '*algoritmo eficiente*', y también diremos, en igual sentido que el utilizado en A.2, que si un cómputo puede llevarse a cabo con un algoritmo PPT, entonces es una '*computación fácil*', o '*factible*'.

Siendo  $M$  una máquina de Turing probabilística,  $M(x)$  será ahora un espacio de probabilidades sobre el resultado de computar la entrada  $x$ . Podremos, por tanto, indicar con  $z \in M(x)$  que  $z$  es una de los posibles resultados de  $M$  cuando se le aplica una entrada  $x$ . Además,  $\Pr[M(x) = z]$  es la probabilidad de obtener  $z$  como resultado de  $M$  cuando se le aplica una entrada  $x$ . Finalmente,  $M(x, y)$  corresponderá al resultado de  $M$  cuando se le aplica una entrada  $x$  y se considera una sucesión interna de tiradas de monedas  $y$ .

### A.4 Tiempo polinomial no uniforme

Un algoritmo no uniforme  $A$  es una secuencia infinita de algoritmos  $\{M_i\}$ , uno para cada tamaño de entrada  $i$ , tal que con una entrada  $x$  se ejecuta  $M_{|x|}(x)$ . Diremos que  $A(x)$  se valida, o computa, si y solo si se valida  $M_{|x|}(x)$ . Y decimos que  $A$  es un *algoritmo no uniforme de tiempo polinomial* si existen polinomios  $P$  y  $Q$  tales que  $M_{|x|}(x)$  termina en no más de  $P(|x|)$  pasos, y el tamaño de la extensión de  $M_i$  está acotado por  $Q(|i|)$ .

### A.5 Adversario

Un *adversario*, o *atacante*, o *nodo deshonesto*, o cualquier sinónimo adicional, se modela en nuestro contexto mediante su potencia computacional.

Un *adversario uniforme*, o simplemente *adversario*, es un algoritmo probabilístico de tiempo polinomial.

Un *adversario no uniforme* es un algoritmo no uniforme de tiempo polinomial, por lo que podrá usar diferentes algoritmos para diferentes tamaños de entrada.

De lo anterior, es inmediato que un adversario no uniforme es computacionalmente más poderoso que uno uniforme.

De acuerdo a las clases de complejidad vistas en A1.2, para un adversario uniforme, y aún asumiendo  $P \neq NP$  o, más allá,  $BPP \neq NP$ , no es suficiente para evitar que, por ejemplo, pueda dar con la clave de nuestro sistema en tiempo polinomial. Lo que buscamos es generar, para cada entrada de tamaño  $n$  lo suficientemente grande, un input  $x$  de ese tamaño, tal que cualquier adversario uniforme tome con una alta probabilidad una decisión errónea sobre  $x$ , es decir, que se equivoque al tomar en consideración



inputs  $x$  de tamaño  $n$  para decidir si  $x \in L$ , con  $L \in NP$ . Esto lo conseguiremos mediante *funciones unidireccionales uniformes*.

En presencia de un adversario no uniforme, lo conseguiremos mediante *funciones unidireccionales no uniformes*.

#### A.6 Funciones unidireccionales y funciones unidireccionales no uniformes

Una función  $v$  se define como *negligible*, o *despreciable*, si para toda constante  $c \geq 0$  existe un entero  $k_c$  tal que para todo  $k \geq k_c$  se cumple que  $v(k) < k^{-c}$ . En resumen,  $v$  es despreciable si sus resultados decaen más rápidamente con respecto al inverso de cualquier polinomio.

Una función  $f: \{0,1\}^* \rightarrow \{0,1\}^*$  es *unidireccional* si existe un algoritmo PPT (ver A.3) que tomando un input  $x$  computa  $f(x)$ , y que para cada algoritmo PPT  $A$ , existe una función negligible  $v_A$  tal que para cualquier  $k$  suficientemente grande se cumple

$$\Pr[f(z) = y : x \leftarrow \{0,1\}^k ; y \leftarrow f(x) ; z \leftarrow A(1^k, y)] \leq v_A(k)$$

donde  $1^k$  denota la longitud (en sistema unario) de  $x$  (del que ya sabemos  $x \in \{0,1\}^*$ ), y actúa como un mecanismo de seguridad para entradas suficientemente grandes puesto que tiende a  $\infty$  con el tamaño de la entrada. En resumen, el adversario, que busca algún inverso de  $y$  trabajando en tiempo polinómico sobre  $|x|$ , tiene una probabilidad despreciable de invertir la función.

Ante un adversario no uniforme dotado de un algoritmo no uniforme de tiempo polinomial, necesitaremos una función  $f$  unidireccional *no uniforme* como una versión más fuerte de la que acabamos de ver. Ahora debe cumplirse que exista un algoritmo PPT que la compute, y que para cada familia de algoritmos (posiblemente no uniformes) de tiempo polinomial  $A = \{M_k\}$  (con  $k \in \mathbb{N}$ ) exista una función negligible  $v_A$  tal que para cualquier  $k$  suficientemente grande se cumple

$$\Pr[f(z) \neq y : x \leftarrow \{0,1\}^k ; y \leftarrow f(x) ; z \leftarrow M_k(y)] \leq v_A(k)$$

## Validación de una transacción

Las siguientes reglas de comprobación de una transacción corresponden a la traducción de [59], , adaptada en lo necesario (contenido accedido en abril 2017):

1. Comprobar su corrección sintáctica
2. Asegurar que no está vacía ninguna de las listas de inputs o outputs
3. Tamaño en bytes  $\leq$  MAX\_BLOCK\_SIZE
4. Cada valor de cada output, así como su total, debe estar en el rango permitido
5. Asegurar que ninguno de los inputs cumple  $\text{hash}=0$ ,  $n=-1$ , que corresponderían a una transacción *coinbase*
6. Comprobar que  $n\text{LockTime} \leq \text{INT\_MAX}$ , tamaño en bytes  $\geq 100$ , y  $\text{signature opcount}$  (operandos en la firma de transacciones estándar)  $\leq 2$
7. Rechazar transacciones no estándar: que *scriptSig* haga otras cosas que no sean apilar elementos, o que *scriptPubKey* no corresponda a las formas usuales
8. Rechazar si ya tenemos registrada la transacción en el *pool* de pendientes, o en un bloque de la cadena más larga
9. Para cada input, rechazar si el output referenciado existe en otra transacción en el *pool* de pendientes
10. Para cada input, buscar en la cadena principal o en el pool de transacciones pendientes la transacción con el output referenciado. Si esta transacción no se encuentra para cualquier input, entonces será una transacción huérfana, por lo que se añadirá al *pool* de huérfanas si no estuviera ya allí
11. Para cada input, si la transacción referenciada como output es *coinbase*, debe tener al menos COINBASE\_MATURITY (actualmente 100) confirmaciones; si no, rechazar
12. Para cada input, si el output referenciado no existe (bien por que nunca ha existido, bien por que ha sido consumido), rechazar
13. Comprobar que el valor de cada input (por medio del output referenciado), así como su suma, se encuentra en el rango permitido
14. Rechazar si la suma de los valores de los inputs es menor que la suma de los valores de los outputs
15. Rechazar si los *fees* (suma de los valores de los inputs menos la suma de los valores de los outputs) es demasiado pequeña para ser incluida en un bloque vacío
16. Verificar que *scriptPubKey* es correcto para cada input, y rechazar en caso contrario

Una vez validados todos estos extremos, la transacción se añade al *pool* de transacciones pendientes (de incluir en un bloque) y se retransmite a los *peers*. Como casos particulares, encontramos:

- Para cada transacción huérfana que use la transacción bajo análisis como uno de sus inputs, ejecutar todas estas comprobaciones recursivamente sobre tal huérfana
- Si la transacción es a nuestro favor, la añadimos a nuestro *wallet*

## *Valoración de la matriz de correspondencias entre Bitcoin/Blockchain y requisitos de la industria*

El análisis de la correspondencia entre las características de la actual implementación de Blockchain para Bitcoin y los requisitos de la industria para una Blockchain genérica puede representarse mediante una matriz que ponga de manifiesto la interrelación entre aquéllos, pudiendo resultar:

- Que una característica actual sea neutra (ni facilite ni dificulte) con respecto a un determinado requisito industrial (celdas de intersección, o de correspondencia, blancas)
- Que una característica actual favorezca o implemente un determinado requisito industrial (celdas verdes)
- Que una característica actual dificulte en gran medida o imposibilite un determinado requisito industrial (celdas rojas)
- Que una característica actual pueda resultar un inconveniente (no invalidante) para un determinado requisito industrial (celdas amarillas)

De acuerdo a lo anterior, los motivos tanto de facilitación o habilitación (celdas verdes), como de obstrucción o invalidación (celdas rojas), como de neutralidad (celdas blancas) requieren, en general, poca explicación. Por ejemplo, la irreversibilidad favorece, por definición, la seguridad en el sistema al limitar posibles acciones futuras sobre una transacción ya realizada, pero dificulta sensiblemente la regulación si se requiere una retrocesión en caso de disputa; un registro no encriptado colisiona con la confidencialidad, mientras que facilita cualquier proceso de auditoría o regulación...

Mayor explicación cabría aportar para las situaciones dudosas (celdas amarillas), en las que o bien expresamos argumentos de causa-efecto (con símbolos de mayor y menor: por ejemplo, cuanto mayor descentralización, menos se favorece la regulación), o bien expresamos dudas (por ejemplo, el anonimato de la blockchain no impide un seguimiento del uso frecuente de una dirección Bitcoin que podría ser relacionada con una determinada IP; pero podría aproximarse por la generación de nuevos pares de claves en cada transacción), o bien explicaciones someras (por ejemplo, el modelo Bitcoin aporta privacidad aún con un registro público no encriptado, pero no parece resultar un modelo adecuado para una industria).

Así, tomando como características y requisitos los vistos en los apartados 5.1 y 5.2, respectivamente, del capítulo 5, y agrupándolos y ordenándolos convenientemente, obtenemos la siguiente matriz:

		Requisitos industria									
		Privacidad	Confidenc.	Identidad	Auditab.	Regulación	Escalabl.	Flexibilidad	Costes	Seguridad	
Blockchain actual	Imposibilidad de doble gasto								>Seguridad >Coste		Seguridad
	Registro inmutable							<Mutabilidad <Flexibilidad	<Mutabilidad >Coste		
	Consenso/PoW costoso										
	Consenso/PoW distribuido										
	Tamaño										Operatividad
	Registro histórico						>Registro >Escalabl.	>Registro <Flexibl.	>Registro >Coste		
	Registro replicado					>Replicación <Regulación	>Replicación <Escalabl.	>Replicación <Flexibilidad	>Replicación >Coste		
	Escalable								>Escalabl. >Coste		
	Valor = output no gastado										Regulación
	Sin intermediación					<Intermediac. <Regulación					
	Abierta o <i>permissionless</i>	>Apertura <Privacidad	>Apertura <Confidenc.	>Apertura <Identificación	>Apertura <Auditabl.	>Apertura <Regulación				>Apertura <Seguridad	
	Irreversibilidad							>Irreversibl. <Flexibilidad			
	Descentralizada					>Descentr. <Regulación					Privacidad
	Sin autoridad										
	No regulada										
	Sin intervención			<Intervención <Identidad?							
	Direcciones anónimas		>Anonimato <Confidenc.?								
	Registro no encriptado	Modelo Bitcoin: público con privacidad								>Acceso <Seguridad	
		Privacidad		Regulación		Operatividad		Seguridad			

Tabla C.1: matriz de correspondencias entre características Blockchain vs. requisitos industria

Tanto las características actuales como los requisitos pueden incluirse en cuatro áreas de agrupación: seguridad (inmutabilidad, PoW...), privacidad (con temas como privacidad, confidencialidad, encriptación o identidad), regulación (descentralización, intervención, autoridad, *auditabilidad*, regulación e identidad/anonimato, que quedan solapadas entre privacidad y regulación), y operatividad (tamaño, replicación, *proof-of-work*, escalabilidad, flexibilidad, costes...). La traslación de estas áreas así identificadas a la matriz (línea inferior y columna derecha), y una posterior y conveniente ordenación de los elementos de cada una de ellas, nos permite identificar dos núcleos o conjuntos de correspondencias problemáticas:

- En el área de regulación para los requisitos, con cierta extensión al área de privacidad, se observa qué características de Bitcoin/Blockchain tienen que ser transformadas: inexistencia de autoridad y regulación, imposibilidad de intervención, anonimato, no encriptación... Están señaladas por el óvalo inclinado en centro-inferior de la tabla.
- En el área de operatividad, esquina superior derecha, se apunta que, dados sus costes, serán necesarias limitaciones o transformaciones en el tamaño (siempre creciente) del registro y en la *proof-of-work*.

Esta *matriz de correspondencias* (características/requisitos, en este caso) es la base de la *matriz de transformación de correspondencias* del capítulo 5, en la que en cada celda se indican las transformaciones necesarias para adaptar cada característica a los requisitos.