UNPSJB

LICENCIATURA EN SISTEMAS OPGCPI

ADMINISTRACIÓN DE REDES Y SEGURIDAD

Bitcoin

Qué elementos de criptografía utiliza la primer criptomoneda que existió

Cátedra
Lic. Bruno Damián Zappellini

Integrantes:
Luciano Serruya Aloisi

17 de diciembre de 2018

Índice

1. Introducción

El presente trabajo de investigación se tratará sobre la *criptomoneda*[1] bitcoin, haciendo foco principlamente sobre los sistemas de criptografía que utiliza.

La primer parte explicará de manera amplia lo que es la criptografía, y hará hincapié en los tipos de criptografía que implementa el protocolo Bitcoin. También incluirá las funciones de *hash* que utiliza, y los algoritmos de codificación. Luego, desarrollará sobre las transacciones en Bitcoin (su ciclo de vida y cómo son validadas), siempre incluyendo los elementos de criptografía utilizados.

Por último, el trabajo explicará brevemente la red de comunicaciones en la que corre el protocolo Bitcoin -*blockchain*- y cómo es la inclusión de nuevos datos a la red (minería de bloques). También otro tema muy importante en el protocolo y con mucha presencia de la criptografía que son las *billeteras* de bitcoin y cómo se generan.

1.1. Qué es Bitcoin

Bitcoin es la primer aplicación de la tecnología blockchain. Comenzó una revolución con la introducción de la primer moneda digital totalmente descentralizada, y demostró ser extremadamente segura y estable [1]

El *paper* titulado *Bitcoin: A Peer-to-Peer Electronic Cash System*, escrito por *Satoshi Nakamoto*, introduce la idea de un *dinero digital* que no necesita un banco intermediario para transferir pagos entre pares.

Bitcoin se puede definir de varias maneras; es un **protocolo**, es una **moneda digital**, y es una **plataforma**. Es una combinación de redes *peer-to-peer*, protocolos de comunicación, y software que facilitan la creación y uso de la moneda digital llamada bitcoin. Nótese que Bitcoin con *B* mayúscula se refiere al protocolo, mientras que bitcoin con *b* minúscula se refiere a la moneda. Los nodos en la red *peer-to-peer* se comunican utilizando el protocolo Bitcoin [1].

Una de las principales ventajas de bitcoin frente a otros proyectos para generar un dinero electrónico, es la forma en la que soluciona el **problema del doble gasto**[2].

[1] Moneda digital o virtual diseñada para funcionar como medio de intercambio. Utiliza la criptografía para asegurar y verificar transacciones [15]

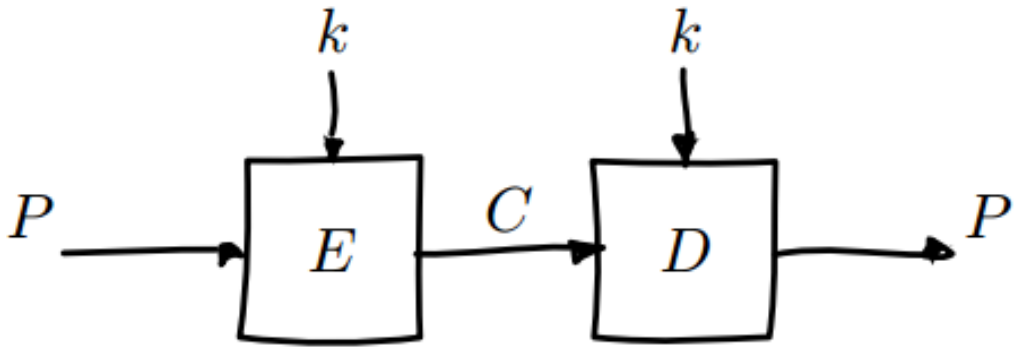
[2] Situación en la que se realizan dos o más transacciones con un mismo dinero

1.2. Criptografía

La criptografía es un área de estudio que consiste de varios esquemas y técnicas para transformar un mensaje en texto plano en un mensaje cifrado; este proceso de transformación se conoce como **encripción**, mientras que el proceso de conseguir el mensaje original a partir del cifrado se llama **desencripción** [22]

1.2.1. Simétrica

La criptograía simétrica (o *encripción de clave secreta*) encripta un mensaje utilizando **una única llave** - la misma llave que encripta el mensaje desencripta el mensaje cifrado para obtener de nuevo el original.



Cifrado y descifrado simétrico (`P` representa el texto plano, `C` el mensaje cifrado, `E` y `D` las funciones de cifrado/descifrado respectivamente, y `k` la clave secreta) [18]

Los algoritmos de encriptación simétricos pueden trabajar con *bloques* (encriptando bloques de un mismo tamaño), o con *flujos* (encriptando flujos de datos, pueden ser flujos de 1 bit).

Este tipo de encriptación es muy performante y no incrementa el tamaño del mensaje, pero introduce una vulnerabilidad al tener que compartir la clave secreta entre las partes que se están queriendo comunicar.

1.2.2. Asimétrica

La encriptación asimétrica (o *encripción de clave pública*), a diferencia de la simétrica que utiliza una única clave, necesita de dos llaves para funcionar: una **una privada** y **una pública**. Este sistema de encriptación se basa en que se utiliza una clave para encriptar

el mensaje, y otra clave (diferente de la primera, pero relacionada) para desencriptar el mensaje. Su característica principal es que es computacionalmente inviable determinar la clave de desencripción solamente sabiendo el algoritmo de cifrado y la clave de encripción [23].

Teniendo este par de claves, se puede operar de dos modos distintos:

- Modo encripción: el emisor encripta el mensaje con la clave pública del receptor, de modo que sólo el receptor sea capaz que de desencriptar el mensaje (utilizando su clave privada)
- Modo autenticación: el emisor encripta el mensaje (o un *digesto* del mensaje) con su clave privada y los anexa al mensaje. El receptor desencripta este anexo con la clave pública del emisor, y compara el anexo desencriptado con el mensaje (o el digesto pudo generar el receptor). Si son iguales, entonces se garantiza que el mensaje fue enviado por el emisor, y que no fue alterado en el camino

Claramente este tipo de encripción genera mayor seguridad en la comunicación, debido a que las claves para desencriptar los mensajes no se deben intercambiar entre las partes previamente a comenzar la comunicación (son privadas a cada cliente y no deben ser reveladas). Sin embargo, aumentan el tamaño del mensaje y no son algoritmos tan performantes como los simétricos.

Criptografía de curva elíptica

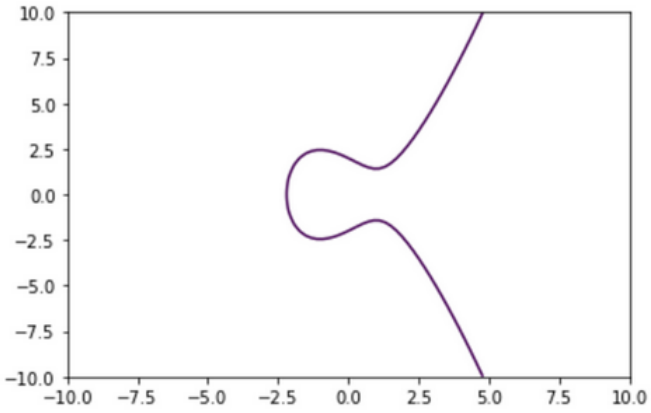
Uno de los primeros algoritmos de clave pública que se diseñaron fue *RSA*[3], que se traba de generar pares de claves privada/pública en base a la teoría de números primos. Para brindar mayor seguridad, el algoritmo precisa números cada vez más grandes; debido a que conseguir números primos resulta una tarea muy costosa para las computadoras, este tipo de algoritmos no es ideal para dispositivos con poco poder de cómputo o poca batería

En 1985, *Neal Koblitz* y Victor Miller plantearon algoritmos de criptografía basados en *curvas elípticas* [17]. Una curva elíptica está definida por el conjunto de valores que genera la siguiente función [16]:

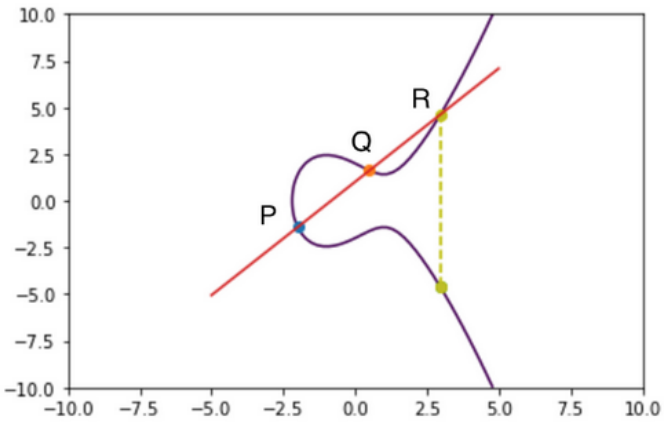
$$y^2 = x^3 + ax + b$$

[3]Nombrado así por las iniciales de sus creadores (*Rivest–Shamir–Adleman*)

Una característica muy interesante que tienen este tipo de curvas (y que es en la que se basa el algoritmo), es que si una recta intersecta dos puntos de la curva, también intersectará un tercero



Curva elíptica [24]



Recta que atraviesa los puntos `(P, Q)`; también atraviesa el punto `(R)` [24]

Nótese también que la curva es simétrica con respecto al eje *Y*.

Sin entrar en más detalles de la criptografía de curva elíptica, el par de claves privada/pública se genera gracias a esta característica de intersección de puntos.

1.3. Funciones de *hash*

Otro elemento de la criptografía muy importante para Bitcoin son las funciones de *hash* (la mayor carga de trabajo del protocolo se trata de calcular *hashes*).

Las funciones *hash* se tratan de funciones que toman una entrada de largo variable y lo convierten en un valor de largo fijo, también conocido como "digesto" [19]

1.3.1. *SHA256*

La familia SHA (*Secure Hash Algorithm*, Algoritmo de *Hash* Seguro) es un sistema de funciones *hash* criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos (NSA) y publicadas por el National Institute of Standards and Technology (NIST) [28]. SHA256 es un *hash* de 64 dígitos hexadecimales *casi único* de un tamaño fijo de 256 bits (32 bytes) [20].

1.3.2. *RIPEMD160*

RIPEMD160 (acrónimo de *RACE Integrity Primitives Evaluation Message Digest*, primitivas de integridad del resumen del mensaje) es un algoritmo del resumen del mensaje de 160 bits desarrollado en Europa por *Hans Dobbertin*, *Antoon Bosselaers* y *Bart Preneel* [27].

RIPEMD160 fue diseñado en la comunidad académica abierta, en contraste con el algoritmo SHA-1, diseñado por la Agencia de Seguridad Nacional estadounidense (*NSA*). Por otra parte, RIPEMD160 es un diseño menos popular y no está tan estudiado com las funciones SHA. Los *hashes* de 160 bits RIPEMD (también llamados resúmenes RIPE del mensaje) se representan típicamente como números en código hexadecimal de 40 dígitos [21].

1.4. Codificaciones

Los algoritmos de codificación *binario-a-texto* transforman una secuencia de datos binarios a una secuencia de caracteres imprimibles. Estas codificaciones son necesarias cuando el canal de transmisión no acepta datos binarios [26]

1.4.1. *Base58Check*

El algoritmo de *Base58Check* es una versión modificada del *Base58*, y sirve para codificar arreglos de bytes en cadenas legibles para un humano [25]

2. Transacciones

Bitcoin utiliza un esquema de clave pública/privada para llevar a cabo la principal función del ecosistema de la moneda, que son las **transacciones**. Cada transacción se compone de al menos **una entrada y una salida**. Las entradas se pueden pensar como monedas siendo gastadas *que fueron creadas en transacciones anteriores*, y las salidas como monedas que están siendo creadas. Existe un caso especial, en la que una transacción no tiene entrada - estas transacciones son las que generan nuevas monedas, y son las que se registran como primer entrada en los bloques [2].

Si un usuario envía monedas a otro usuario, la transacción tiene que ser firmada con la clave privada del emisor, y también se requiere una referencia a las transacciones previas

para demostrar el origen de las monedas, y que todavía no han sido gastadas. De hecho, las monedas son *transacciones sin gastar*, representadas en *Satoshis* [4].

Las transacciones no están encriptadas y son públicamente visibles.

2.1. Ciclo de vida

El ciclo de vida de una transacción es el siguiente [3]:

1. Un usuario genera una transacción usando algún software de Bitcoin
2. El software firma la transacción con la clave privada del usuario
3. La transacción es *difundida* a toda la red de Bitcoin (blockchain) usando un algoritmo de *inundación* (*flooding*)
4. Los nodos *mineros* incluyen la transacción en el próximo *bloque* a minar
5. El minado comienza una vez que el minero que resolvió la *prueba de trabajo* (*Proof of Work*) difunda el nuevo bloque a la red
6. Los otros nodos verifican el bloque y los siguen propagando
7. Finalmente, la confirmación de la transacción figura en la cuenta (*billetera*) del receptor, y luego de varias confirmaciones (estadísticamente, seis confirmaciones es suficiente), la transacción se considera confirmada y finalizada

2.2. Verificación de una transacción

Cada transacción tiene que ser verificada por los nodos que componen la red de Bitcoin. Los aspectos a verificar son los siguientes [4]:

1. La transacción respeta la sintaxis del lenguaje de *scripting* [5] de Bitcoin
2. Las entradas y las salidas no son vacías
3. El tamaño máximo de la transacción no supera el tamaño máximo del bloque (actualmente 1 MB)

[4] 1/100000000 bitcoin

[5] Lenguaje basado en pilas llamado *script* que describe cómo las monedas pueden ser gastadas y transferidas [5]

4. El valor de las salidas no debe ser menor a 0 ni mayor a 21 millones de bitcoin
5. Todas las entradas deben tener especificada una salida anterior (salvo las transacciones que generan monedas)
6. No debe existir una transacción igual en ningún bloque, ni esperando ser confirmada
7. Las salidas que corresponden a las entradas de la transacción no deben figurar en ningún otra transacción (problema del *double spending*)
8. Para cada entrada, debe existir su correspondiente salida en alguna transacción
9. Para cada entrada, si la salida correspondiente referenciada es la salida que origina las monedas (llamada *coinbase transaction*), esta última debe tener al menos 100 confirmaciones
10. Para cada entrada, si la salida correspondiente referenciada no existe o ya fue gastada, la transacción es rechazada
11. La suma de las entradas debe ser mayor o igual al total de las salidas (si el valor de las entradas es mayor, el resto se considera *tarifa de la transacción* [6])
12. La tarifa de la transacción debe ser mayor o igual a un valor mínimo establecido

3. *Blockchain*

Blockchain, o la *cadena de bloques*, es el "libro de cuentas" público, ordenado en el tiempo, e inmutable de todas las transacciones en la red de bitcoin. Cada bloque está identificado por un *hash* en la cadena y está vinculado con su bloque anterior referenciando su *hash*. Todos los bloques están relacionados con su bloque anterior, a excepción del primer bloque, también llamado bloque *génesis*.

Bloques nuevo son añadidos a la cadena cada 10 minutos, aproximadamente. El protocolo de blockchain maneja una *dificultad*[7] para agregar nuevos bloques que puede ir aumentando o disminuyendo (según la capacidad de cómputo disponible en la red) para que se mantenga la frecuencia de un bloque nuevo cada 10 minutos [6].

[6] Las tarifas de las transacciones definen la prioridad que tendrá la transacción al ser eligida por los mineros para agregarla a un nuevo bloque

[7] La dificultad signifca qué tan difícil es para los mineros generar un bloque nuevo

3.1. Estructura de un bloque

Bytes	Campo	Descripción
80	Cabecera del bloque	Incluye varios campos de metadatos del bloque
variable	Contador de transacciones	Incluye la cantidad de transacciones que incluye el bloque, incluye la *coinbase transaction*
variable	Transacciones	Todas las transacciones del bloque

Estructura de un bloque [7]

Bytes	Campo	Descripción
4	Versión	Número de versión de bloque. Indica las reglas que se siguieron para validar el bloque
32	*Hash* del bloque previo	*Hash* SHA256 del bloque previo
32	*Merkle root hash*	*Hash* del nodo raíz del árbol *Merkle* para garantizar la integridad del bloque
4	Estampa de tiempo	Estampa de tiempo del momento en el que se creó el bloque (formato *Unix epoch*)
4	Dificultad	Valor de la dificultad de la red
4	*Nonce*	Número arbitrario que los mineros van cambiando hasta satisfacer una condición

Estructura de la cabecera de un bloque [8]

3.2. Implementación en Python

A continuación se incluye la implementación de las clases necesarias para modelar una cadena de bloques en el lenguaje de programación Python (implementación tomada de este repositorio).

```
1 from datetime import datetime
2 from hashlib import sha256
3
```

```
4  class Block:
5      """
6      Clase que representa un bloque en blockchain
7      """
8      blockNo = 0
9      data = None
10     next = None
11     hash = None
12     nonce = 0
13     previous_hash = 0x0
14     timestamp = datetime.now()
15
16     def __init__(self, data):
17     """Constructor del bloque - recibe datos para
         almacenar"""
18         self.data = data
19
20     def hash(self):
21     """Devuelve el hash del bloque
22     (hash de todos los datos del bloque)"""
23         h = sha256()
24         h.update(
25             str(self.nonce).encode('utf-8') +
26             str(self.data).encode('utf-8') +
27             str(self.previous_hash).encode('utf-8') +
28             str(self.timestamp).encode('utf-8') +
29             str(self.blockNo).encode('utf-8')
30         )
31         return h.hexdigest()
32
33     def __repr__(self):
34     """Devuelve una cadena que representa el bloque y su
         contenido"""
35         return f"""Block Hash: {str(self.hash())}
36     BlockNo: {str(self.blockNo)}
37     Block Data: {self.data}
```

```
38      Hashes: {self.nonce}
39  --------------------"""
```

Clase para modelar un bloque en blockchain

```
1  class Blockchain:
2      """Clase para modelar la blockchain"""
3
4      # Dificultad de la red
5      diff = 15
6      maxNonce = 2**32
7      target = 2 ** (256-diff)
8
9      # Creación del primer bloque
10     block = Block("Genesis")
11     _ = head = block
12
13     def add(self, block):
14         """Agrega un bloque a la cadena"""
15
16         block.previous_hash = self.block.hash()
17         block.blockNo = self.block.blockNo + 1
18
19         self.block.next = block
20         self.block = self.block.next
21
22     def mine(self, block):
23         """Calcula el hash de un bloque y lo agrega a la
             cadena"""
24         for n in range(self.maxNonce):
25             if int(block.hash(), 16) <= self.target:
26                 self.add(block)
27                 break
28             else:
29                 block.nonce += 1
30
31     def __iter__(self):
```

```
32          return self
33
34      def __next__(self):
35          if self.head == None:
36              raise StopIteration()
37
38          _block = self.head
39          self.head = self.head.next
40          return _block
```

Clase para modelar la blockchain

Teniendo el bloque y la cadena de bloques modelado, para simular el funcionamiento de una blockchain se pueden usar las clases anterior de la siguiente manera:

```
1  blockchain = Blockchain()
2
3  # Agregar diez bloques a la cadena
4  # El tiempo que toma finalizar el bucle depende
5  # del poder de cómputo de la máquina
6  # y de la dificultad configurada en la blockchain
7  for n in range(10):
8      blockchain.mine(Block("Block " + str(n+1)))
9
10 # Imprimir los bloques de la cadena
11 for block in blockchain:
12     print(block)
```

4. Minería de bloques

La minería de un bloques es una tarea que conlleva muchos recursos (de hardware y de electricidad) con el cual se agregan nuevos bloques a la red. Los bloques contienen las transacciones que son validadas mediante el proceso de minado (llevado a cabo por los nodos mineros) y son agregadas a la cadena de bloques. Nuevas monedas son acuñadas por los mineros al gastar los recursos necesarios para realizar la tarea de minado. Esto también asegura el sistema contra fraudes y ataques de *doble gasto* mientras que agregar más monedas al ecosistema.

Como se decía en la sección anterior, aproximadamente cada 10 minutos se añade un bloque nuevo a la red. Los mineros son recompensados con monedas si son los que crearon el bloque nuevo. La tasa de creación de bitcoin decrementa el 50 % cada 210 mil bloques, aproximadamente 4 años. Cuando bitcoin comenzó, la recompensa por un bloque nuevo eran 50 bitcoin; en 2012 se redujo a 25, y en Julio de 2016 llegó a 12,5.

El concepto de "minería" se debe a que las monedas se crean o se "emiten" al crear un nuevo bloque; es una analogía a la minería de oro, en donde en base a un trabajo se obtiene algo de valor.

El sistema de bitcoin está diseñado para la creación de monedas tenga un límite. Aproximadamente en 2140, cuando se hayan creado 21 millones de bitcoins, los mineros de bloques no serán recompensados por crear un nuevo bloque, sin embargo podrán tener una ganacia por las tarifas de las transacciones que incluyan en el bloque [9]

4.1. Prueba de trabajo

La prueba de trabajo (*Proof of Work*) es una demostración de que suficiente poder computacional fue empleado para construir un bloque válido. En este modelo, los nodos compiten para ser seleccionados en proporción a su capacidad computacional.

La prueba de trabajo consiste en calcular el *hash* del bloque. Éste se compone de la suma de todos los datos del bloque, y debe comenzar con *n* número de ceros (o que sea menor a cierto valor). La cantidad de ceros o el valor *objetivo* está definido por la *dificultad* de la red.

Debido a que es fácil calcular el *hash* de un valor, pero imposible conseguir el valor original a partir de un *hash*, la única forma de calcular el *hash* del bloque es **con fuerza bruta** (probar valores hasta encontrar un resultado). El campo `nonce` de la cabecera del bloque es el que tienen que ir incrementando los mineros hasta encontrar el *hash* indicado.

$$H(N||P\_hash||Tx_1||Tx_2||...||Tx_n) < Objetivo$$

Cálculo de la cabecera del bloque, donde `H` es la función de *hash*, `N` es el *nonce*, `P_hash` es el *hash* del bloque anterior, `Tx` son las transacciones que incluye el bloque, y `Objetivo` la dificultad

Una vez conseguido el *hash*, el bloque es inmediatamente difundido por la red. Debe ser aceptado por los otros mineros para ser agregado a la red [10].

Algoritmo de minado

La secuencia de pasos que llevan a cabo los mineros que crear un nuevo bloque es la siguiente [11]

- El bloque anterior se recupera de la red
- Se obtiene un conjunto de posibles transacciones que podrían conformar el bloque
- Computar el *hash* de la cabecera del bloque con un *nonce* = 0
- Si el *hash* obtenido es menor al objetivo, detener el proceso
- Si no es menor, repetir el proceso incrementando el *nonce*

5. Billeteras

Las billeteras en Bitcoin es el software encargado de almacenar las claves privadas o públicas y las direcciones bitcoin. Las claves privadas pueden ser generadas de distintas maneras y son usadas por diferentes tipos de billeteras. Cabo destacar que las billeteras **no almacenan monedas**; los balances de las cuentas de los usuarios se calculan en base a las transacciones (más especficimante, a las *salidas de las transacciones no gastadas*) almacenadas en blockchain [12].

Para generar el par de claves pública/privada, Bitcoin utiliza **criptografía de curva elíptica**. Las direcciones de bitcoin son creadas aplicando el siguiente algoritmo a la clave pública correspondiente a una clave privada [13]:

1. Tomar la clave pública y aplicarle el algoritmo de SHA256
2. A ese *hash*, aplicarle de nuevo SHA256
3. Al segundo resultado, aplicarle el algoritmo RIPEMD160
4. Al *hash* resultante de 160 bits, se le agrega un prefijo con un número de versión
5. Por último, se codifica el valor con el algoritmo Base58Check, resultando en una cadena de entre 26 y 35 caracteres que comienza con 1 o 3

5.1. Tipo de billeteras

- No-determinísticas: contienen claves privadas generadas aleatoriamente (son conocidas también como *Just a Bunch on Key wallets*). El cliente de bitcoin genera las claves cuando es iniciado por primera vez y genera claves cuando es requerido
- Determinísticas: las claves son derivadas de una *semilla* mediante una función de *hash*. Esta semilla es generada aleatoriamente y es comúnmente representada por una **palabra clave**. Esta palabra clave puede servir para recuperar todas las claves en caso de que se pierdan
- Determinísticas jerarquícas: almacenan claves en una estructura de árbol derivadas de una semilla. La semilla genera la **clave padre**, que es utilizada para generar **claves hijas**, y subsecuentemente **claves nietas**. La jerarquía completa se puede recuperar si se conoce la clave padre.
- *Brain wallets*: similar al tipo anterior, pero la clave padre es una contraseña definida por el usuario
- Papel: billetera basada en papel; se imprime y tiene la clave privada codificada en QR
- Hardware: dispositivo electrónico que almacen la clave privada
- Online: las claves privadas son almacenadas por un servicio online
- Móviles: billetera implementada en dispositivos móviles [14]

Referencias

[1] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin*.

[2] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Transactions*.

[3] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - The transaction life cycle*.

[4] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Transaction verification*.

[5] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - The script language*.

[6] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Blockchain*.

[7] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - The structure of a block*.

[8] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - The structure of a block header*.

[9] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Mining*.

[10] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Proof of Work*.

[11] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - The mining algorithm*.

[12] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Wallets*.

[13] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Keys and addresses*.

[14] Imran Bashir. «Mastering Blockchain». En: primer edición. Packt Publishing, mar. de 2017. Cap. 4 - *Bitcoin - Wallet types*.

[15] Cointelegraph. *¿Qué es criptomoneda? Guía para principiantes*. URL: https://es.cointelegraph.com/bitcoin-for-beginners/what-are-cryptocurrencies#futuro-de-las-criptomonedas.

[16] Andrea Corbellini. *Elliptic Curve Cryptography: a gentle introduction*. Mayo de 2015. URL: http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/.

[17] Grayblock. *Elliptic-Curve Cryptography*. Jun. de 2018. URL: https://medium.com/coinmonks/elliptic-curve-cryptography-6de8fc748b8b.

[18] Laurens Van Houtven. «Crypto 101». En: Creative Commons, 2013. Cap. 6 - *Block ciphers*.

[19] Laurens Van Houtven. «Crypto 101». En: Creative Commons, 2013. Cap. 10.1 - *Hash functions - Description*.

[20] Alex Preukschat. *¿Qué es y de qué sirve el algoritmo SHA-256 en el protocolo Bitcoin? – Secure Hash Algorithm (VII)*. Ene. de 2014. URL: https://www.oroyfinanzas.com/2014/01/algoritmo-sha-256-protocolo-bitcoin-secure-hash-algorithm/.

[21] Alex Preukschat. *¿Qué es y por qué se utiliza el algoritmo RIPEMD-160 en la creación de claves públicas Bitcoin? (VIII)*. Ene. de 2014. URL: https://www.oroyfinanzas.com/2014/01/ripemd-160-bitcoin/.

[22] Williams Stallings. «Cryptography and Network Security». En: septima edición. Pearson, 2017. Cap. 3 - *Classical Encryption Techniques*.

[23] Williams Stallings. «Cryptography and Network Security». En: septima edición. Pearson, 2017. Cap. 9.1 - *Principles of Public-Key Cryptosystems*.

[24] Hackernoon - Short Tech Stories. *Elliptic Curve Crypto, The Basics*. Jun. de 2017. URL: https://hackernoon.com/eliptic-curve-crypto-the-basics-e8eb1e934dc5.

[25] Wikipedia. *Base58Check Encoding*. Nov. de 2017. URL: https://en.bitcoin.it/wiki/Base58Check_encoding.

[26] Wikipedia. *Binary-to-text encoding*. Dic. de 2018. URL: https://en.wikipedia.org/wiki/Binary-to-text_encoding.

[27] Wikipedia. *RIPEMD160*. Sep. de 2015. URL: https://es.wikipedia.org/wiki/RIPEMD-160.

[28] Wikipedia. *Secure Hash Algorithm*. Nov. de 2018. URL: https://es.wikipedia.org/wiki/Secure_Hash_Algorithm.