CRIPTOMONEDAS Y DEFI

Iker Caballero Bragagnini

Tabla de contenido

La criptografía y las criptomonedas: funciones *hash*

- Todas las divisas necesitan una manera de controlar la oferta y forzar varias propiedades de seguridad para prevenir trampas. Existen diferencias fundamentales entre cómo se lidia con el problema desde las monedas fiat y desde las criptomonedas
    - En las monedas fiat, las organizaciones como los bancos centrales controlan la demanda de dinero y añaden características antifalsificación para lamoneda física
        - Estas características de seguridad hacen que haya más dificultades para el atacante, pero no hacen que el dinero sea imposible de falsificar
        - Además, el cumplimiento de la ley es necesario para hace que las personas no rompan las reglas del sistema
    - Las criptomonedas tienen que tener medidas de seguridad que eviten que las personas hagan alteraciones en el estado del sistema y se equivoquen (que hagan comandos mutuamente inconsistentes a personas diferentes)
        - Si una persona convence a otra de que le paguen en monedas digitales, por ejemplo, esa persona no debería ser capaz de convencer a una tercera persona de que sea pagada con esas mismas monedas
        - A diferencia de las monedas fiat, las reglas de seguridad de las criptomonedas necesitan ser forzadas de manera puramente tecnológica y sin necesitar una autoridad central
    - Las criptomonedas hacen un uso elevado de la criptografía, proporcionando un mecanismo para asegurar la codificación de las normas del sistema de criptomonedas en su propio sistema
        - Se puede utilizar la criptografía para prevenir alteraciones y equivocaciones, y también para codificar, con un protocolo matemático, las reglas para la creación de nuevas unidades de la moneda
        - Por lo tanto, uno puede entender debidamente las criptomonedas a través de los fundamentos criptográficos en los que se basan
        - La criptografía es un área de investigación académica muy profunda que usa muchas técnicas matemáticas avanzadas que

son notoriamente sutiles y complicadas. Afortunadamente, Bitcoin se apoya en solo un conjunto de construcciones criptográficas comunes y simples

- Uno de los conceptos fundamentales en criptografía y en el sistema de las criptomonedas es el de las funciones *hash*

    - Una función *hash* es una función matemática que cumple con las siguientes tres propiedades:

        - Su insumo puede ser un *string* de cualquier tamaño

        - Se produce un resultado de tamaño fijo. En el desarrollo, se asume que este tamaño es de 256 bits, pero puede utilizarse cualquier tamaño lo suficientemente grande

        - La función es computable en un tiempo de ejecución $O(n)$ dado un *string* de tamaño n

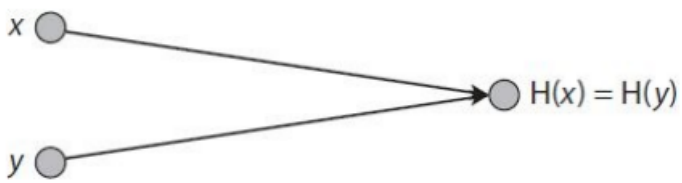
    - Estas propiedades definen una función *hash* general que se puede usar para crear una estructura de datos, tal como una tabla *hash*. El desarrollo teórico se centrará exclusivamente en funciones *hash* criptográficas

        - Para que una función sea criptográficamente segura, se requiere que tenga tres propiedades adicionales: la propiedad de resistencia a la colisión, la propiedad de esconderse y la de *puzzle friendliness*

        - Cada una de estas propiedades se puede detallar por separado para entender por qué es útil que se cumplan estas para la función *hash*

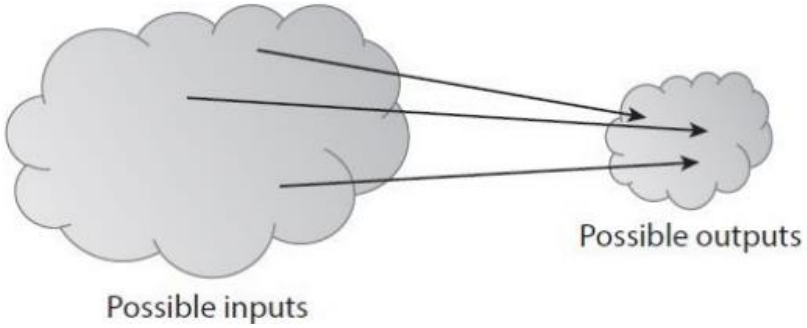
- Una colisión ocurre cuando dos insumos producen el mismo resultado, por lo que una función H es resistente a la colisión si nadie puede encontrar una colisión

    - Se dice que una función *hash* H es resistente a la colisión si es inasequible que dos valores x e y tales que $x \neq y$ pero $H(x) = H(y)$



        - Es especialmente importante ver que no se dice que no existen colisiones, sino que nadie las puede encontrar. Las colisiones existen para cualquier función *hash*

- El espacio de insumos de cualquier función *hash* contiene todos los *strings* de todos los tamaños, pero el espacio de resultados es más pequeño, de modo que la función no puede ser biyectiva (no puede haber un solo insumo para un solo resultado). El espacio de insumo es infinito y el de resultados es finito, y hay resultados para los cuales un número infinito de posibles insumos se mapearán



- Aunque se ha dicho que es imposible que se encontrara una colisión, existen métodos que garantizan el encontrar una colisión. El método más simple sería coger $2^{256} + 1$ valores diferentes, calcular los *hashes* de cada uno y, como el número de insumos con el que se juega es mayor al número de resultados, debe de haber un insumo que haya obtenido el mismo *hash* que otro

  - Si se cogen insumos aleatoriamente y se calculan los *hashes*, entonces se puede encontrar una colisión con una alta probabilidad antes de examinar todos los valores. Si se seleccionaran solo $2^{130} + 1$, entonces hay una probabilidad del 99.8% de que al menos dos insumos colisionen

  - Debido a la paradoja del cumpleaños, se puede encontrar una colisión mirando solo la raíz cuadrada del número de insumos totales

  - Aunque el algoritmo funciona para cualquier función *hash*, este es muy costoso computacionalmente y es inasequible calcular todos los *hashes* del algoritmo en una vida. Por lo tanto, aunque el algoritmo es general, es inasequible

  - Aunque existen funciones para los cuales se pueden encontrar métodos más eficientes de encontrar colisiones (como la función $h(x) = x \ mod \ 2^{256}$), no se sabe si existe un método eficiente para todas las funciones (no se ha demostrado). Las funciones que se utilizan ahora son funciones que mucha gente ha probado y han resultado muy difíciles de encontrar colisiones, y solo se creen que estas colisiones

- Si se sabe que dos insumos x e y para una función resistente a las colisiones H son diferentes, es seguro asumir que $H(x)$ y $H(y)$ son diferentes, pero si alguien supiera que x e y son diferentes y tienen el mismo *hash*, entonces esto violaría la suposición de que H es resistente a la colisión. Este argumento permite usar *hashes* como digestivos de mensajes

    - Como ejemplo se considera un sistema de almacenamiento *online* de archivos que permite al usuario subir archivos y asegurar su integridad cuando se descarga. Si una persona quisiera subir varios archivos grandes y quiere verificar que el archivo que descarga es el mismo que el que sube, entonces se tiene un problema porque no tiene sentido comparar el archivo subido con una copia local, por ejemplo

    - Los *hashes* que son resistentes a colisiones proporcionan una manera elegante y eficiente, dado que solo es necesario recordar el *hash* del documento original y volver a calcularlo al descargarlo para saber si estos son iguales o no. Esto permite detectar no solo corrupción accidental durante la transmisión, sino también corrupción intencional del archivo por el servidor, que es clave para la criptografía

    - El *hash* sirve como un digestivo de tamaño fijo, o un resumen no ambiguo, de un mensaje, dando una manera eficiente de recordar cosas vistas anteriormente

    - Además, mientras que el archivo puede pesar mucho, el *hash* es de 256 bits (u otro número fijo) de modo que el requerimiento de almacenamiento se reduce mucho. Los archivos tienen un *hash* asociado (un *string*) que los identifica, y hay comandos para poder obtenerlo del dispositivo

- La segunda propiedad es la de esconderse, que se basa en que si se da un resultado de la función *hash* $y = H(x)$, no hay manera asequible de saber cuál fue el insumo x. El problema es que esta propiedad no puede ser verdad expresado de esta manera

    - Considerando el ejemplo simple en donde se tira una moneda y se anuncia el *hash* del *string* “cara” y “cruz” dependiendo del resultado de la tirada, si se preguntara a una persona que no ha visto la tirada, pero si el *hash* cuál era el insumo, esta podría calcular los *hashes* de cada *string* para saberlo

        - Esa persona era capaz de saber el insumo debido a que solo había 2 valores posibles de x, por lo que los podía probar. Para

conseguir la propiedad de esconderse, no debe haber ningún valor de x que sea particularmente probable (se tiene que escoger una x de un conjunto que sea lo suficientemente disperso, de modo que la probabilidad de probar unos pocos valores sea poco probable)

- Es posible conseguir la propiedad aun cuando los valores no provienen de un conjunto que no sea muy disperso. Se dice que una función *hash* se esconde o tiene la propiedad de esconderse si cuando un valor secreto r se escoge de una distribución de probabilidad con alta min-entropía, entonces, dado $H(r \parallel x)$, es imposible encontrar x

    - En la teoría de la información, la min-entropía es una medida de qué tan predecible es un resultado, por lo que un valor alto indica la idea intuitiva de que la distribución está muy dispersa. La expresión $r \parallel x$ significa que se han concatenado r y x

    - Lo que esto significa específicamente es que cuando se muestrea de la distribución, no hay un valor particular que sea probable. Si, por ejemplo, se coge r de una distribución uniforme de todos los *strings* de 256 bits, entonces cualquier *string* particular se escoge con probabilidad $1/2^{256}$ (cercano a cero)

- Un ejemplo de dónde se aplica esta propiedad es en los *commitments* o compromisos. Un *commitment* es el análogo digital de coger un valor, sellarlo en un sobre y poner el sobre a la vista del público, haciendo que uno se comprometa a lo que hay dentro del sobre

    - A parte del usuario, nadie más sabe el valor que hay dentro del sobre, pero después se puede abrir el sobre y revelar el valor al que uno se comprometió anteriormente

    - Un esquema de compromiso consiste de dos algoritmos: la función *commit*, que toma un mensaje y un valor secreto aleatorio (llamado *nonce*) como insumo y devuelve un compromiso, y la función de verificación, que toma un compromiso, un mensaje y un *nonce* como insumos y, si el *commit* es igual al *commit* del mensaje y el *nonce* de insumos, devuelve el valor de verdadero

    $$com \coloneqq commit(msg, nonce)$$

    $$verify(com, msg, nonce) = TRUE$$
    $$if \;\; com == commit(msg, nonce)$$

    - Se requiere que en este algoritmo se cumplan las propiedades de seguridad de esconderse (dado com, es inasequible encontrar

msg) y la de vinculación (no es asequible encontrar dos pares $(msg, nonce)$ y $(msg', nonce')$ tal que $msg \neq msg'$ y, a la vez, $commit(msg, nonce) == commit(msg', nonce')$)

- Para usar el esquema de *commitment* primero se necesita generar un *nonce* aleatorio. Después, se tiene que aplicar la función *commit* al *nonce* y al *msg* (el valor al que uno se compromete) y se publica el compromiso *com*

  - En un punto en el futuro, si se quiere revelar el valor al que uno se ha comprometido antes, se publica el *nonce* aleatorio que se usó a través de la función *verify*

  - Cada vez que uno quiere comprometerse a un valor, se tiene que generar un nuevo valor *nonce* aleatorio. En criptografía, se usa la palabra *nonce* para referirse a un valor el cual solo se utilizará una sola vez

- Las dos propiedades de seguridad dictan que los algoritmos deben comportarse como si se cerrara y se abriera un sobre

  - Dado un *com*, un tercero no puede saber exactamente el valor que está dentro, y una vez se ha hecho un compromiso, uno no puede cambiar el valor *nonce* (no se pueden encontrar dos mensajes diferentes tales que cuando uno se compromete a un valor, puede decir que se ha comprometido a otro diferente)

  - Para saber que estas propiedades se cumplen, se puede usar una función *hash* criptográfica. Definiendo la función *commit* de la siguiente manera, donde el insumo es un *nonce* de 256 bits que se concatena con el mensaje y devuelve el *hash* de este nuevo valor concatenado:

$$commit(msg, nonce) \coloneqq H(nonce \parallel msg)$$

$$where\ nonce\ is\ \ 256 - bit\ value$$

  - Para verificar esto, alguien tendrá que calcular este *hash* del *nonce* que se ha dado concatenado al mensaje, y se verificará si este resultado es igual al compromiso que se ha visto

- Las dos propiedades se pueden expresar en términos de $H(nonce \parallel msg)$: la propiedad de esconderse expresa que, dado $H(nonce \parallel msg)$, es inasequible encontrar msg, mientras que la propiedad de vinculación expresa que es inasequible encontrar dos pares $(msg, nonce)$ y $(msg', nonce')$ tales que $msg \neq msg'$ y $H(nonce \parallel msg) == H(nonce' \parallel msg')$

        - La propiedad de vinculación está implícita en la de resistencia a colisiones de la función de *hash*. Si la función de *hash* es resistente a las colisiones, es imposible encontrar dos valores $msg \neq msg'$ tales que $H(nonce \parallel msg) == H(nonce' \parallel msg')$ porque, si no, habría una colisión (el converso no es verdad)

        - Si una función *hash* H es resistente y se puede esconder, entonces el esquema de *commitments* funcionará porque tendrá las propiedades necesarias de seguridad

- La tercera propiedad de seguridad que se necesita es la de *puzzle friendliness*, la cual es complicada

    - Una función *hash* es *puzzle friendly* si para cada posible valor resultante y de n bits, si k se escoge de una distribución con una min-entropía alta, entonces es inasequible encontrar x tal que $H(k \parallel x) = y$ en un tiempo significantemente menor a 2^n

        - Intuitivamente, si alguien quiere fijarse en la función *hash* para obtener un valor particular y, y si parte del insumo se ha escogido de una manera aleatoria adecuada, entonces es muy difícil encontrar otro valor que obtenga exactamente ese valor objetivo

    - La utilidad de este problema se puede ilustrar a través de un ejemplo en donde se intenta construir un puzle de búsqueda, un problema matemático que requiere buscar en un espacio muy grande una solución. En particular, un problema de búsqueda no tiene atajos, por lo que no hay manera de encontrar una solución válida a no ser que se busque en todo el espacio

        - Un puzle de búsqueda consiste en una función de *hash* H, en un valor id (llamado ID del puzle) que proviene de una distribución con una min-entropía alta y un conjunto objetivo Y. Una solución a este puzle es el valor x, tal que $H(id \parallel x) \in Y$

        - La intuición es que si H tiene un resultado de n-bits, entonces puede tomar cualquier de 2^n valores. Resolver el puzle requiere encontrar un insumo tal que el resultado caiga dentro del conjunto Y, que es típicamente más pequeño que el conjunto de resultados

        - El tamaño de Y determina la dificultad del puzle, si Y es el conjunto de todos los *strings* de n bits, entonces el puzle es trivial (cualquier resultado sería solución), mientras que, si el tamaño es de 1, entonces el puzle sería lo más difícil posible

        - Que el puzle tenga una min-entropía alta asegura que no hay atajos, dado que si un valor particular de ID fuera probable, entonces podría haber un atajo si se calcula el *hash* para ese valor

    - Si una función de *hash* es *puzzle friendly* entonces no hay una estrategia de resolución para este puzle que se mucho mejor que probar valores aleatorios de x

        - Por lo tanto, si se quiere poner un puzle difícil de resolver, esto se puede hacer siempre que el ID del puzle se pueda generar de una manera aleatoria adecuada

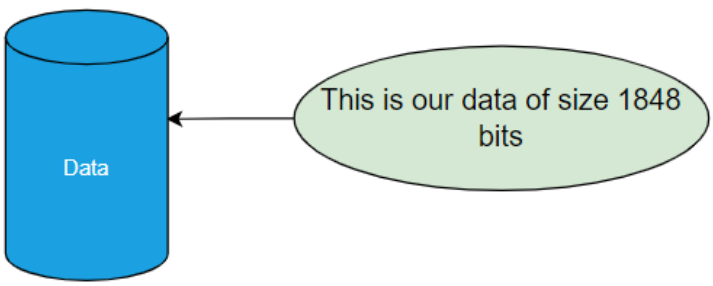
- Ahora que se han discutido las tres propiedades másimportantes de las funciones *hash* y sus aplicaciones, se desarrolla la función de *hash* que utiliza Bitcoin, la SHA-256

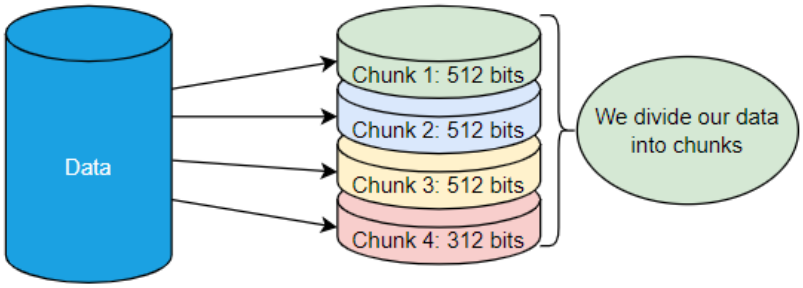
    - Se requiere que las funciones de *hash* trabajen con insumos de una longitud arbitraria. Afortunadamente, se puede construir una función de *hash* que trabaje con insumos de tamaño fijo a través de la transformada de Merkle-Damgard

        - SHA-256 solo es una de las funciones de *hash* que se suelen utilizar para este método. En terminología común, cuando una función de *hash* subyacente de tamaño fijo y resistente a colisiones se denomina función de compresión

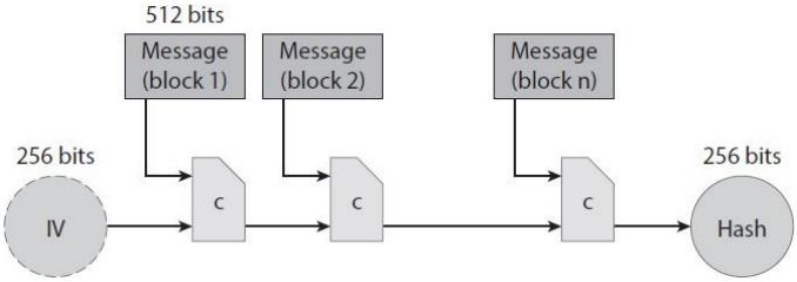
        - Se ha demostrado que, si la función de compresión es resistente a las colisiones, la función de *hash* general también es resistente a las colisiones

        - Suponiendo que existe una función de compresión que toma insumos de tamaño m y produce un resultado de un tamaño menor n, y que el insumo de la función de *hash*, que puede ser de cualquier tamaño, se divide en bloques de tamaño $m - n$. Por lo tanto, un bloque almacena información y tiene una capacidad de almacenamiento específica





- Se pasa cada bloque junto con el resultado del bloque anterior a la función de compresión (el tamaño del insumo será de $(m-n)+n=m$, que es el tamaño del insumo de la función de compresión). Para el primer bloque, para el cual no hay un bloque previo, se usa un vector de inicializaron

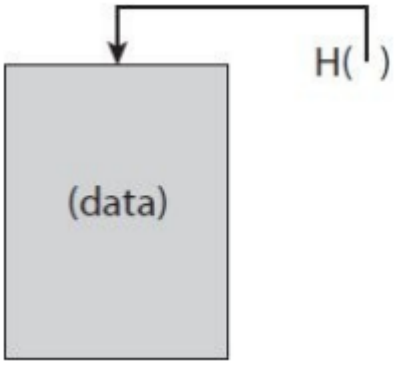


- SHA-26 usa una función de compresión que toma un insumo de 768 bits (concatena insumos del bloque y el resultado previo) y produce un resultado de 256 bits, donde los bloques tienen 512 bits

La criptografía y las criptomonedas: *hash pointers* y firmas digitales

- Ahora que se han visto las funciones *hash*, se discuten los *hash pointers* y sus aplicaciones

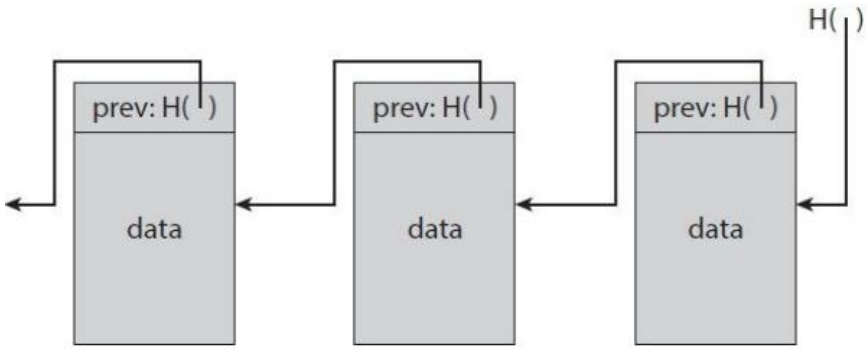
    - Un *hash pointer* es simplemente un puntero que indica la dirección de memoria en donde una información se almacena junto con un *hash* criptográfico de la información



- Mientras que un puntero normal da una manera de recolectar la información de una localización de la memoria, un *hash pointer* también permitirá verificar que esta información no se ha cambiado (en algún momento fijo del tiempo). Por eso, tiene dos elementos: la localización de los datos a donde apunta y el *hash* de los datos a donde apunta

- Se pueden usar *hash pointers* para crear todo tipo de estructuras de datos. Intuitivamente, se puede crear una estructura familiar que utilice punteros, tales como una lista vinculada o un árbol de búsqueda binario e implementarla con *hash pointers*

- Una lista vinculada que usa *hash pointers* se conoce como *blockchain* o cadena de bloques. En una lista vinculada regular, se tendría una serie de bloques y el puntero del bloque anterior en la lista, mientras que en la *blockchain*, el puntero se reemplaza por un *hash pointer*



- Cada bloque, por tanto, no solo indica dónde estaba el valor del bloque anterior, sino también una digestión o resumen de ese valor, que permite que se verifique que el valor no haya cambiado. Se guarda la cabeza de la lista (el último elemento del esquema), que es el *hash pointer* regular que apunta al bloque más reciente de datos

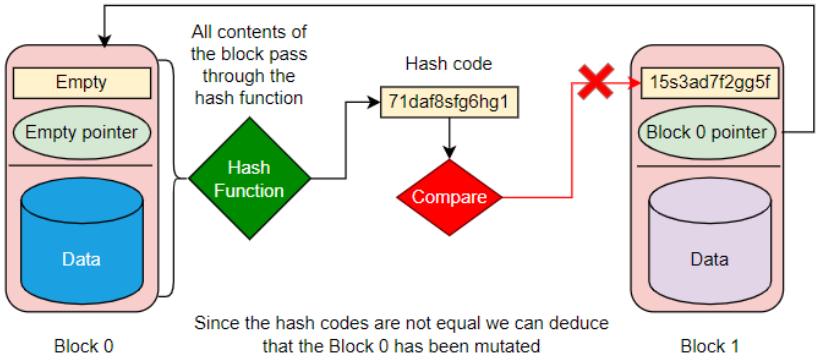
- En cada bloque se almacena el *hash* que se obtiene para los datos previos, el *hash pointer* que apunta al bloque previo y los datos de ese respectivo bloque

- Un caso de uso para la *blockchain* es el *tamper-evident log*, una estructura de datos que permita guardar datos y permita adjuntar datos al final del *log*, pero que, si alguien altera los datos antes de ese *log*, que se pueda detectar

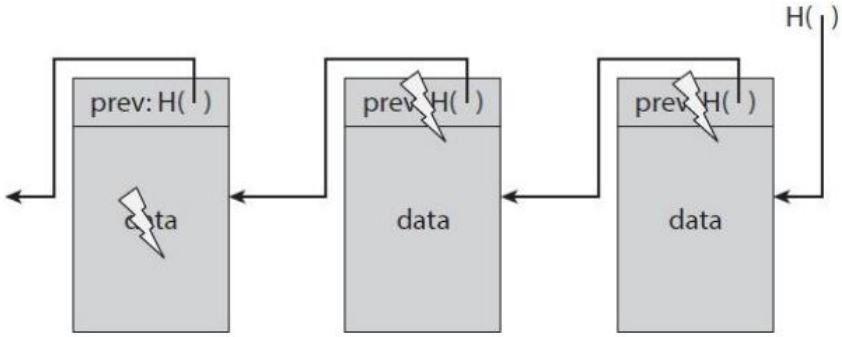
- Si un tercero intentara alterar los datos en el medio de la cadena de modo que alguien que solo recuerde el *hash* no pueda detectar

el cambio. Para ello, este tercero cambia los datos de algún bloque k

- Como los datos se han cambiado, el *hash* en el bloque $k+1$, que es un *hash* del bloque entero k, no va a coincidir y se detectará la inconsistencia entre el *hash* del bloque k con los datos alterados y el *hash* almacenado en el bloque $k+1$



- Aunque el tercero podría intentar alterar esto y alterar también el *hash* del siguiente bloque para corregir este desequilibrio, y así, esta estrategia fallaría al final cuando llegara al *hash* final

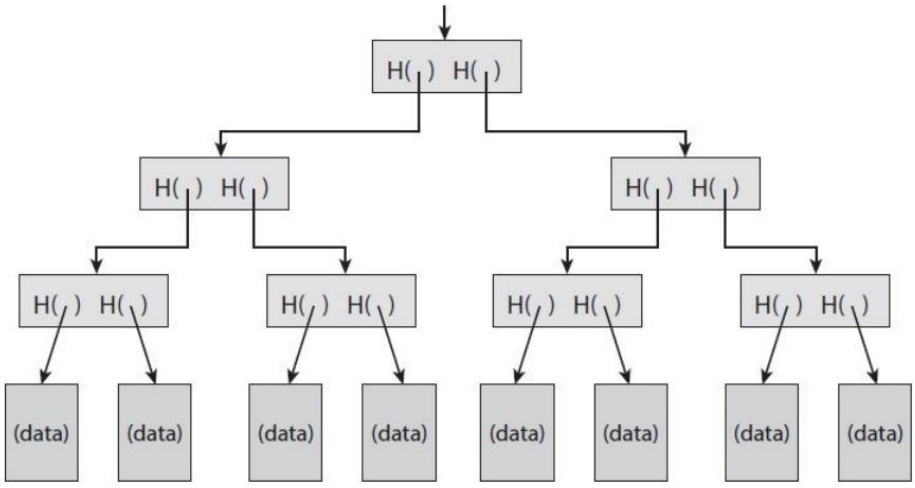


- Específicamente, si se almacena el *hash pointer* de la cabeza de la lista (aquel suelto en el esquema) donde el tercero no pueda alterarlo, este nunca será capaz de alterar ningún bloque sin ser detectado. Por lo tanto, si se necesitara cambiar cualquier bloque, tendría que cambiar todos los *hash pointers* hasta el final y, al final, vería que no coinciden los datos ni los *hashes*

- El primer bloque de datos no tiene ningún *hash* y tiene un *hash pointer* vacío dentro, el cual se denomina bloque de génesis (desde donde se origina toda la cadena)

- Otra estructura de datos muy útil es un árbol binario, que cuando usan *hash pointers* se convierte en un árbol de Merkle. Suponiendo que los bloques contienen datos, estos bloques conforman las hojas del árbol y

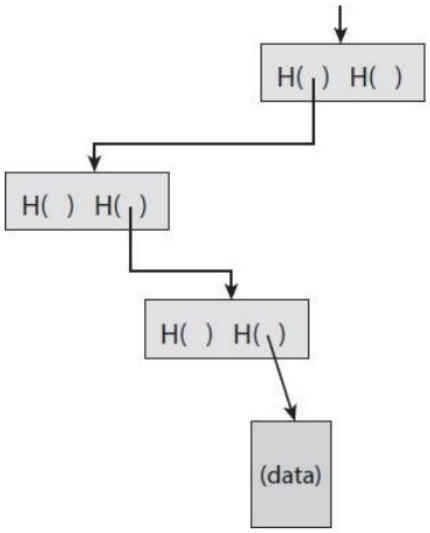
se juntan estos bloques en pares con tal de construir una estructura de datos con los *hashes* de ambos bloques (se concatenan)



- Igual que antes, solo hace falta acordarse o guardar un *hash*, que es el de la raíz principal del árbol

- Entonces, a partir de este árbol, es posible cruzar los punteros a cualquier punto de la lista. Esto permite que uno se asegure de que los datos no han sido alterados (los niveles de arriba no coincidirán con los de abajo después de ser alterados)

- Otra característica interesante de estos árboles es que, a diferencia de la *blockchain* que se ha formado anteriormente, estos permiten una demostración de membresía o *proof of membership*



- Suponiendo que alguien quiere demostrar que un bloque de datos concreto es un miembro del árbol de Merkle, se tiene que mostrar este bloque y los bloques anteriores hasta la raíz

        - Si hay n nodos en el árbol, solo se necesita mostrar $\log(n)$ elementos, y como cada paso requiere calcular el *hash* del bloque posterior, entonces toma $\log(n)$ pasos para verificarse. Por lo tanto, aunque el árbol tenga un número de bloques grande, se puede probar la membresía en un tiempo relativamente corto

        - De este modo, la verificación se ejecuta en el tiempo y espacio que es logarítmico en el número de nodos

    - Un árbol ordenado de Merkle es un árbol de Merkle en donde se toman los bloques del fondo y se organizan usando una función de orden concreta. Usando un árbol de Merkle ordenado, se vuelve posible verificar también la no membresía en un tiempo y espacio logarítmico

        - Se puede demostrar la no membresía simplemente mostrando un camino hacia el elemento justo antes del elemento y uno justo después de este elemento

        - Si ambos elementos son consecutivos en el árbol, entonces esto sirve como prueba de que el elemento no está incluido, porque de no ser así habría habido un espacio entre ellos

    - Uno puede usar *hash pointers* para construir cualquier estructura basada en punteros mientras que no hayan ciclos. Si hay ciclos en la estructura, entonces uno no puede hacer que coincidan todos los *hashes*

        - En una estructura de datos acíclica, se puede comenzar a partir de cualquier elemento que no tenga ningún puntero saliendo de ellos, calcular los *hashes* de estos e ir hasta el inicio del árbol

- El segundo elemento criptográfico más importantes es la firma digital, que es fundamental para las criptomonedas

    - Se supone que una firma digital es un análogo a una firma hecha en papel. Se desea que haya dos propiedades de las firmas digitales que corresponden a esta analogía:

        - La primera es que solo uno mismo pueda hacer su firma, pero que cualquiera que la vea pueda reconocer que es válida

        - La segunda es que se quiere una firma para que esté vinculada a un documento particular, de modo que la firma no se pueda usar para otros documentos

    - Un esquema de firma digital se puede plantear de la siguiente manera:

- Se crea un método $generateKeys$ que toma como insumo la longitud de esta llave y genera un par de llaves: la llave secreta sk es privada, mientras que la llave publica pk se puede usar por todo el mundo para verificar la firma

$$(sk, pk) \coloneqq generateKeys(keysize)$$

- Se crea un método $sign$, el cual toma un mensaje y una llave secreta sk como insumo y resulta en una firma para el mensaje bajo sk

$$sig \coloneqq sign(sk, message)$$

- Se crea un método $verify$, el cual toma como insumo el mensaje, la firma y la llave pública y devuelve un valor booleano dependiendo de si la firma es válida

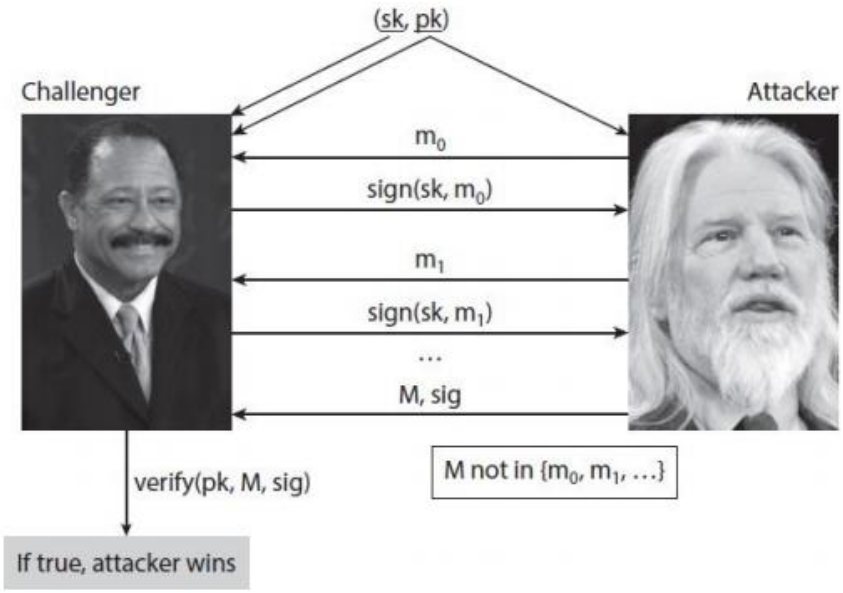
$$isValid \coloneqq verify(pk, message, sig)$$

- Los dos primeros métodos deben ser generados aleatoriamente porque debería generar diferentes llaves para diferentes personas, y verificar siempre será una acción determinista

- Se requiere que las firmas válidas se deban verificar (de modo que $verify(pk, message, ig) == TRUE$) y que las firmas digitales sean existencialmente infalsificables

    - El segundo requerimiento es que sea computacionalmente inasequible falsificar, de modo que un tercero que conozca una llave pública y vea las firmas en otros mensajes no puede falsificar la firma en otros mensajes que no la hayan visto previamente

    - Para poder entender esto mejor, se ejemplifica el juego de la falsificación, en donde una persona crea las llaves y firma en diversos archivos y la otra persona es el adversario, el cual tiene la llave pública y puede pedir que firme todos los archivos necesarios hasta que él desee. Este juego acaba si al final el adversario adivina cuál es el mensaje



- El adversario no puede conseguir hacer infinitas adivinaciones, solo puede hacer unas cuantas, por ejemplo, un millón (pero no 2^{80}). Cuando el adversario tiene suficientes firmas, entonces escoge un mensaje M en donde intentará falsificar la firma (y este mensaje no pertenece a los mensajes previos $\{m_0, m_1, \ldots\}$ correspondiente a los archivos que pidió ver) y el otro jugador verificará con la llave pública si ha adivinado la firma

- Se dice que un esquema de firma es infalsificable si, y solo si, sin importar el algoritmo que el adversario utilice, la probabilidad de que falsifique correctamente es extremadamente pequeña (en la práctica, se asume que no pasará)

- Existen varias preocupaciones prácticas para hacer que la idea algorítmica del mecanismo de firma digital se pueda implementar

  - Muchos algoritmos de firmas son aleatorizados, por lo que se necesita una buena fuente de aleatoriedad. Este requerimiento es importante porque la fuente de aleatoriedad puede hacer que un algoritmo pase de ser seguro a ser inseguro

  - Otra preocupación práctica es el tamaño del mensaje: en la práctica existe un límite en el mensaje que uno puede firmar, dado que los esquemas reales operan en *strings* de bits limitados. Una solución eficaz a este problema suele ser firmar el *hash* del mensaje más que el mensaje mismo, de modo que siempre se firmen mensajes de un número de bits concretos (y es seguro debido a que las funciones de *hash* son resistentes a colisiones)

  - Una manera alternativa de superar esta problemática es firmar un *hash pointer*: si uno firma este, entonces la firma cubre o

protege toda la estructura (no solo el puntero, sino toda la cadena de *hash pointers* a los que apunta). Por ejemplo, si se firmara el último *hash pointer* al final de una *blockchain*, entonces efectivamente se está firmando toda la *blockchain*

- Bitcoin utiliza un sistema de firma digital particular llamado *Elliptic Curve Digital Signature System* (ECDSA), el cual es usado por el gobierno de los Estados unidos (una adaptación del algoritmo DSA para usar curvar elípticas) y se cree que es generalmente seguro

    - Específicamente, Bitcoin utiliza ECDSA sobre la curva elíptica estándar $secp256k1$, que se estima que proporciona 128 bits de seguridad (es tan difícil romper este algoritmo como realizar 2^{128} operaciones criptográficas de llave simétricas, tales como invocar una función de *hash*)

    - Aunque esta curva es un estándar público, no se suele ysar fyera de Bitcoin, dado que en otras aplicaciones como intercambios de llaves para la navegación segura por la red se utiliza la curva $secp256r1$

- Aunque los detalles matemáticos no son importantes, es interesante tener una idea de las varias cantidades que se usan en el sistema de firma digital:

    $Private\ key$: $256\ bits$

    $Public\ key, uncompressed$: $512\ bits$

    $Public\ key, compressed$: $257\ bits$

    $Message\ to\ be\ signed$: $256\ bits$

    $Signature$: $512\ bits$

    - Aunque ECDSA solo puede asignar mensajes de 256 bits, siempre se calcula el *hash* para estos antes de estar firmados, de modo que cualquier tamaño de mensaje puede ser firmado eficientemente

    - Con ECDSA, una buena fuente de aleatoriedad es esencial porque una mala fuente hará que se filtre la llave. Si se usa ECDSA, si se usa una mala fuente solo cuando se firma, pero se tiene una llave generada con una buena fuente, la mala firma también hará que se filtre la llave privada (típico en algoritmos DSA) y se de la posibilidad de que se falsifique la firma

- Las llaves públicas vistas anteriormente para sistemas de firmas digitales se pueden interpretar como identidades de los actores o participantes en este sistema, y esto tiene relación directa con la gestión de identidades descentralizada

    - Un truco muy útil con las firmas digitales es tomar la llave pública e igualarla a la identidad de una persona o a un actor del sistema, de modo que si se ve un mensaje que se verifica correctamente bajo la llave pk, entonces es como si fuera pk el emisor del mensaje

        - Por lo tanto, las llaves públicas pueden actuar como identidades de los participantes o actores (y para hablar en su nombre, se debería saber la llave sk

        - En Bitcoin no se encripta ninguna información: la encriptación solo es un conjunto de técnicas modernas de criptografía, pero Bitcoin utiliza técnicas que requieren esconder información de alguna manera, pero sin encriptar nada

    - Una consecuencia de tratar las llaves públicas como identidades es que se puede crear una nueva identidad cuando uno quiera, dado que solo es necesario crear un par de llaves a través de la operación $generateKeys$ en el sistema de firmas digitales

        - Esta pk es la nueva identidad pública que se puede usar, y sk es la llave secreta correspondiente que solo uno mismo sabe para poder actuar con su propia identidad pk

        - En la práctica, uno puede usar el *hash* de pk como su identidad, dado que las llaves públicas son largas. Si se quiere hacer eso, para verificar que el mensaje proviene de la identidad deseada, uno necesita comprobar que pk tiene un *hash* igual a la identidad de uno y que el mensaje se verifica bajo la llave pública pk

        - Además, la llave pública pk parecerá aleatoria por defecto, y nadie será capaz de descubrir la identidad real de alguien al examinar pk

    - Esto lleva a la idea de la gestión de identidades descentralizada: más que tener una sola autoridad registrando a los usuarios en un sistema, uno puede registrarse como usuario uno mismo

        - Uno no necesita que se le asigne un nombre de usuario ni informar a nadie, y se puede crear una nueva identidad (una cada vez) tantas veces como uno quiera. Esto permite que se puedan tener varias identidades o que uno pueda estar en el anonimato

        - Esta es la manera en la que Bitcoin gestiona las identidadesm y se suelen llamar direcciones o *addresses* a los *hashes* de las llaves públicas (las identidades)

        - Debido a que se pueden generar identidades sin una autoridad central, podría haber el riesgo que se generen dos identidades iguales. Este riesgo es probabilísticamente muy pequeño (no se tiene en cuenta en la práctica), pero uno tiene que asegurarse de que las fuentes de aleatoriedad que se utilizan para la generación son buenas, dado que si no la teoría no aplica

    - Aunque uno pueda pensar que esto lleva a mucho anonimato y privacidad, no es tan simple, dado que, con el tiempo, la identidad realiza una serie de comandos y acciones que la gente puede ver para identificar que una serie de acciones se han hecho por esa identidad concreta

        - Se pueden realizar inferencias de la identidad real de una identidad a través de identificar y relacionar series de acciones cometidas por esa identidad

        - Por lo tanto, el patrón de comportamiento permite mostrar información que puede permitir identificar realmente a un usuario o identidad del sistema

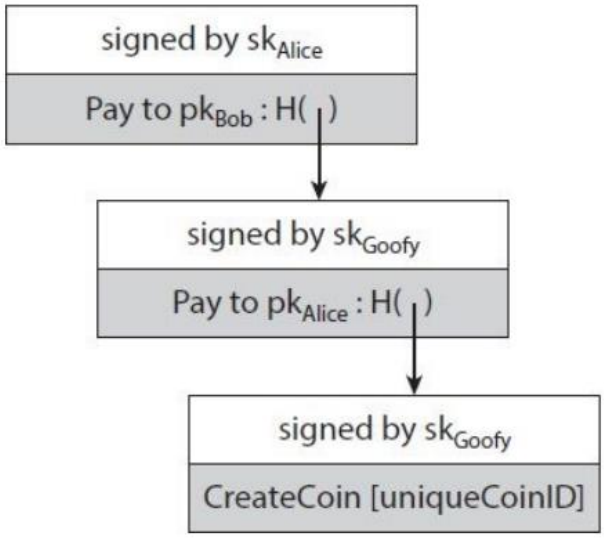
La criptografía y las criptomonedas: ejemplos de monedas

- Se plantea una primera moneda llamada Goofycoin, que es una moneda simple con dos reglas: una entidad designada, Goofy, puede crear nuevas monedas cuando quiera y estas monedas creadas pertenecen a Goofy, y quien sea que posea la moneda puede transferirla a otro

    - Para crear la moneda, Goofy genera una identidad de moneda única $uniqueID$ que no se ha generado nunca antes y construye el *string* $CreateCoin\ [uniqueCoinID]$, calculando después la firma digital del *string* con su llave secreta

        - El *string*, junto con la firma de Goofy, es una moneda. Cualquiera puede verificar que la moneda contiene la firma válida de un comando $CreateCoin$ y, por tanto, es una moneda válida

    - La segunda regla de Goofycoin no es una simple cuestión de enviar la estructura de datos de la moneda a otro recipiente, sino que se tienen que usar operaciones criptográficas



- Si Goofy quiere transferir su moneda a Alicia, tiene que crear un nuevo comando que sea “Pagar esto a Alicia”, donde “esto” es el *hash pointer* que referencia a la moneda en cuestión (para resumir). Como las identidades son llaves públicas, entonces Alicia se refiere a su llave pública

- Finalmente, Goofy firma el *string* que representa el comando, dado que Goofy es el dueño de la moneda y debe firmar cualquier transacción que gaste esa moneda. Una vez que esa estructura de datos representando la transacción hecha por Goofy se ha firmado, entonces Alicia es propietaria de la moneda

- Esta puede demostrar a todo el mundo que posee la moneda, dado que puede presentar la estructura de datos con la firma válida de Goofy. Además, esta estructura apunta a una moneda válida poseída por Goofy, de modo que la validez y la posesión de las monedas es evidente en este sistema

- Una vez que Alicia posee la moneda, ella puede gastarla otra vez creando el comando “Pagar esto a la llave pública de Bob” y firmando este comando. Cuando cualquiera vea la moneda, se puede verificar que Bob es el propietario siguiendo la cadena de *hash pointers* hasta la creación de la moneda y verificar que en cada paso, el poseedor legal firmó un comando de los descritos

- Para resumir, las reglas de Goofycoin son las siguientes:

  - Goofy puede crear nuevas monedas simplemente firmando un comando de que está haciendo una nueva moneda con un ID de moneda único

        - Cualquiera que posea la moneda puede pasarla a otro a través de firmar un comando que diga “Pasar la moneda a X”, donde X es una llave pública

        - Cualquiera puede verificar la validez de una moneda a través de seguir la cadena de *hash pointers* hasta la creación por Goofy, verificando todas las firmas por el camino

    - Hay un problema de seguridad fundamental con Goofycoin, el cual se conoce como *double-spending attack*: la capacidad de que alguien pueda gastar su moneda más de una vez

        - Si Alicia pasara su moneda a Bob con el proceso descrito, pero este no se lo dijera a nadie, entonces podría realizar el mismo proceso de pago a Chuck, haciendo que ambas transacciones parezcan perfectamente válidas. No obstante, ambas provienen de la misma moneda original

        - Este tipo de problemas es uno de los más importantes a solucionar por parte de las criptomonedas

        - Goofycoin es una moneda no segura. No obstante, es simple, y su mecanismo de transferencia se parece al de Bitcoin

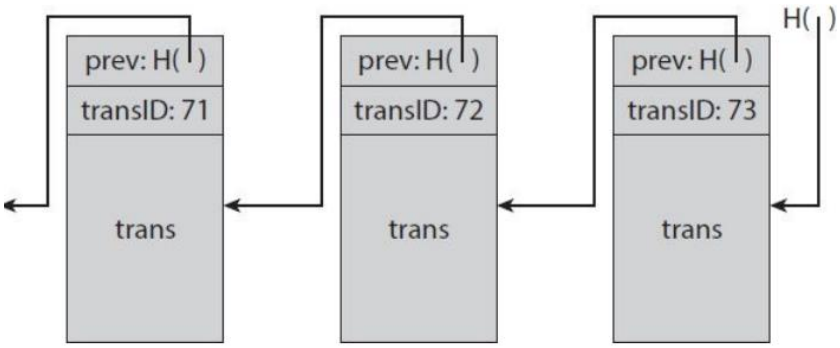
- Para resolver el problema de doble gasto, se diseña otra moneda llamada Scroogecoin, que se construye a partir de Goofycoin pero es más complicada en términos de estructuras de datos

    - La primera idea clave es que hay una entidad designada llamada Scrooge que publica un *append-only ledger* (libro mayor de solo anexar) que contiene la historia de todas las transacciones

        - Esta propiedad de *append-only* asegura que cualesquiera datos escritos en este libro mayor se queden para siempre en este (no se pueden borrar)

        - Si este libro mayor de verdad respeta la propiedad, se puede usar para defender cualquier problema de doble gasto requiriendo que las transacciones se escriban en el libro antes de que sean aceptadas. De este modo, se documentará públicamente si las monedas se han mandado previamente a un destinatario diferente

    - Para implementar esta funcionalidad de solo anexar, Scrooge puede crear una *blockchain* que firmará digitalmente



- Esta consiste en una serie de bloques de datos, cada uno con una transacción en el (en la práctica, se suelen poner muchas transacciones en un mismo bloque para optimizar, igual que Bitcoin). Cada bloque tiene un ID de la transacción, los contenidos de la transacción y un *hash pointer* al bloque previo

- Scrooge firma digitalmente el último *hash pointer*, el cual vincula todos los datos en esta estructura y publica la firma junto a la *blockchain*

- En Scroogecoin, una transacción solo cuenta si está en la *blockchain* firmada por Scrooge. Cualquier puede verificar que la transacción se ha patrocinado por Scrooge al revisar la firma de Scrooge en el bloque que registra la transacción

- Scrooge se asegura de que no patrocina una transacción que intente hacer un doble gasto en una misma moneda ya gastada

- Se necesita que haya una *blockchain* con *hash pointers* además de firmar cada bloque para que se asegure la propiedad de solo anexar

  - Si Scrooge intenta añadir o quitar una transacción, o cambiar una transacción existente, esto afectaría a todos los bloques siguientes por los *hash pointers*. Mientras haya alguien que monitoree el último *hash pointer* publicado por Scrooge, el cambio será obvio y será fácil de evidenciar

  - En un sistema en donde Scrooge firmara los bloques individualmente, uno debería seguir cada una de las firmas que Scrooge ha hecho. Una *blockchain* hace que sea más fácil para dos individuos verificar que han observado la misma historia de transacciones firmada por Scrooge

- En Scroogecoin hay dos tipos de transacciones: la primera es la de tipo $CreateCoins$ y la segunda es la de tipo $PayCoins$. Ahora se desarrolla la primera

- La primera es como la operación que Goofy hacía en Goofycoin para hacer una nueva moneda. Con Scroogecoin, la transacción crea múltiples monedas, en donde cada moenda tiene un número serial de transacción, un valor (número de Scroogecoins) y un recipiente (una llave pública que permite tener la moneda cuando se crea)

transID: 73		type:CreateCoins
coins created		
num	*value*	*recipient*
0	3.2	0x…
1	1.4	0x…
2	7.1	0x…

coinID 73(0)
coinID 73(1)
coinID 73(2)

- De este modo, $CreateCoins$ crea múltiples monedas nuevas con diferentes valores y las asigna a identidades como propietarios iniciales. Uno se refiere a las monedas con $CoinID$s, que es una combinación del ID de la transacción con el número serial de la moneda en esa transacción

- Por definición, una transacción $CreateCoins$ siempre es válida si se firma por Scrooge (independientemente de cuantas pueda crear)

o Con el segundo tipo se consumen monedas (destruirlas) y crea nuevas monedas del mismo valor total

transID: 73		type:PayCoins
consumed coinIDs: 68(1), 42(0), 72(3)		
coins created		
num	*value*	*recipient*
0	3.2	0x…
1	1.4	0x…
2	7.1	0x…
signatures		

- Estas nuevas monedas pueden pertenecer a diferentes identidades. Esta transacción se tiene que firmar por todos los que pagan la moneda

- Las reglas de Scroogecoin dicen que una transacción de $PayCoins$: las monedas consumidas tienen que ser válidas (se

crearon antes), no tienen que estar consumidas previamente, el valor total de las monedas que se consumen es igual al de las monedas que se metieron en la transacción (no se crea valor de la nada) y la transacción tiene que estar firmada por todos los propietarios de las monedas que intervienen

- Scrooge escribirá en el libro mayor esta transacción a través de anexarla a la *blockchain*, y todo el mundo puede ver que esta transacción ha ocurrido. Hasta que se publique, esta transacción podría ser reemplazada por una de doble gasto, aunque se haya validado por las primeras tres condiciones

- Las monedas en este sistema son inmutables: no cambian nunca, no se pueden subdividir ni combinar. Cada moneda se crea una vez en una transacción y se consume después en otra transacción, pero uno puede tener el efecto de subdividir y combinar monedas usando transacciones

  - Por ejemplo, para subdividir una moneda, Alicia puede crear una nueva transacción que consume esa moneda y produce dos nuevas monedas con el mismo valor total. Estas nuevas monedas se pueden asignar otra vez a Alicia

  - Por lo tanto, aunque las monedas son inmutables en este sistema, tiene toda la flexibilidad de un sistema sin monedas inmutables

- El problema de Scroogecoin es que, aunque funcione en el sentido de que la gente pueda ver qué monedas son válidas, Scrooge tiene demasiada influencia en el sistema

  - Aunque no puede crear transacciones falsas porque no puede falsificar las firmas, pero puede dejar de patrocinar transacciones de algunos usuarios. Si Scrooge fuera avaricioso, podría negarse a publicar transacciones a no ser que se pagara una comisión, o crear cuantas monedas quiera para si mismo, o incluso bloquear todo el sistema

  - El problema principal es la centralización, y aunque parece una propuesta no muy realista, esta se contemplaba en la investigación de criptosistemas en forma de “bancos centrales”. En la vida real, la mayoría de divisas tienen un emisor confiable que crea la moneda y determina la validez

  - Las criptomonedas con una autoridad central han fallado en la práctica, posiblemente por la dificultad de los usuarios de aceptar una moneda virtual con una autoridad central

La descentralización en criptomonedas: descentralización y consenso distribuido

- Como se ha visto anteriormente, el problema de la centralización para las criptomonedas es relevante, de modo que se tiene que discutir y comparar sobre la centralización y la descentralización

    - El conflicto entre los paradigmas de centralización y descentralización nace en variedad de tecnologías digitales

        - Está internet, un sistema descentralizado que compite contra otras alternativas de servicios de información centralizada como AOL o CompuServe; el e-mail, un sistema descentralizado basado en SMTP y el estándar para comunicarse por la red; la mensajería instantánea, que es un modelo mixto entre centralizado y descentralizado; y finalmente las redes sociales, en donde el modelo centralizado es dominante

        - La descentralización no es blanco o negro, dado que ningún sistema es puramente descentralizado. Por ejemplo, aunque el e-mail sea un sistema descentralizado, varios proveedores centralizados de e-mails son dominantes en la red, e incluso el protocolo Bitcoin, que es descentralizado, las bolsas en donde se comercia con Bitcoin pueden ser centralizadas o descentralizados en diferentes grados

    - Por lo tanto, se puede subdividir la pregunta de cómo el protocolo Bitcoin consigue la descentralización en 5 preguntas diferentes:

        - Las primeras tres serían saber quién mantiene el libro mayor de transacciones, quién tiene autoridad para validar las transacciones y quien crea los Bitcoins. Estas tres reflejan los detalles técnicos del protocolo Bitcoin

        - Las otras dos restantes serían saber quién determina cómo cambian las reglas del sistema y cómo los Bitcoins adquieren valor de intercambio. Estas dos preguntas no son tan técnicas, sino que atienden más a cuestiones político-económicas

    - Diferentes aspectos del Bitcoin caen en diferentes puntos en el espectro de centralización/descentralización

        - Primero, la red *peer-to-peer* (P2P) está cerca a ser puramente descentralizada, dado que cualquiera puede ejecutar un nodo de Bitcoin, y la barrera de entrada es bastante baja. Uno puedo ir a la red y descargar y cliente Bitcoin para ejecutar un nodo en la computadora, habiendo varios miles de estos nodos

        - La minería de Bitcoin también está abierta a todo el mundo técnicamente, pero requiere un coste de capital alto, por lo que hay un cierto grado de centralización o concentración de poder en el ecosistema

        - Los nodos de Bitcoin ejecutan actualizaciones de *software*, lo que influye en cómo y cuándo cambian las reglas del sistema. Se puede imaginar que existen numerosas implementaciones interoperables del protocolo, como ocurre con el correo electrónico, pero en la práctica, la mayoría de los nodos ejecutan la implementación de referencia y la comunidad confía en sus desarrolladores y tienen mucho poder

- Aunque se ha discutido sobre la centralización y la descentralización de manera general, ahora se puede discutir la descentralización de Bitcoin a un nivel técnico más detallado

    - Un término clave en la discusión es el consenso distribuido, dado que el problema técnico más importante a resolver es el de conseguir consenso distribuido al construir un sistema de efectivo digital

        - El consenso distribuido tiene varias aplicaciones, y se ha estudiado por décadas en ciencias computacionales, en donde su aplicación principal era la confianza en los sistemas distribuidos

        - Por ejemplo, grandes redes sociales tienen miles o millones de servidores que forman una gran base de datos distribuida que registrada todas las acciones que ocurren en el sistema. Cada pieza de información debe ser registrada en diferentes nodos en el *backend*, y los nodos deben de estar sincronizados sobre el sistema general del estado

    - Las implicaciones de tener un protocolo de consenso distribuido llegan a aplicaciones más allá de la tradicional: con este protocolo se podrían construir un almacén de valores-clave masivo y distribuido que asigna claves o nombres arbitrarios a valores arbitrarios

        - Un almacén de valores-clave distribuido, a su vez, permitiría a muchas aplicaciones. Por ejemplo, se podría crear un sistema de nombres de dominio distribuido, que es simplemente mapear nombres inteligibles por el ser humano con direcciones IP

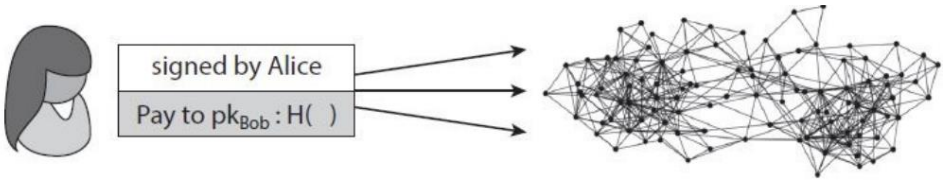
        - Otra potencial aplicación sería la construcción de un directorio de llaves públicas, que es un mapeado entre las direcciones de e-mail (o alguna otra forma de identidad en el mundo real) y las llaves públicas

- En un protocolo de consenso distribuido hay n nodos en donde cada uno tiene un valor de insumo y algunos de los nodos son maliciosos. Un protocolo de consenso distribuido cumple con las siguientes propiedades:

    - El protocolo debe terminar con todos los nodos honestos con un acuerdo sobre el valor

    - El valor debe haberse generado por uno de los nodos honestos

- Para entender como el consenso distribuido funciona para Bitcoin, uno se tiene que acordar que Bitcoin es un sistema P2P: cuando una persona paga a otra, lo que hace es transmitir la transacción a todos los nodos de Bitcoin que componen la red P2P



    - Aunque la persona que recibe la transacción no esté ejecutando ninguno de los nodos que componen la red P2P, no hay necesidad de ello para recibir los fondos (los Bitcoins serán suyos independientemente de si opera un nodo). No obstante, es buena idea ejecutar un nodo para ser notificado de que la transacción ha ocurrido y de que ha sido pagado

    - Dada la variedad de usuarios que están transmitiendo estas transacciones en la red, los nodos deben acordar exactamente qué transacciones transmitir y el orden en el que estas transacciones han ocurrido. Esto resultará en un solo libro mayor global para el sistema

    - Por lo tanto, todos los nodos deben estar de acuerdo sobre qué bloques de transacciones incluir en la *blockchain*

- Como se ha mencionado anteriormente, Bitcoin suele agrupar varias transacciones en un mismo bloque por cuestiones de optimización. Por lo tanto, en cualquier punto, todos los nodos de la red P2P tienen un libro mayor consistiendo de una secuencia de bloques, cada uno conteniendo una lista de transacciones en las cuales han llegado a un acuerdo

    - Adicionalmente, cada nodo tiene un conjunto de transacciones restantes de las cuales ha escuchado pero que no ha incluido en la *blockchain*. Para estas transacciones aún no ha ocurrido el consenso, por lo que cada nodo puede tener una versión un poco diferente del conjunto de transacciones restantes

        - En la práctica, esto último ocurre porque la red P2P no es perfecta y algunos nodos pueden escuchar de una transacción mientras que otros no

        - Para llegar a un consenso en un bloque, cada nodo en el sistema propone su propio conjunto de transacciones restantes para que se incluyan en el siguiente bloque en intervalos regulares de tiempo (cada 10 minutos, por ejemplo). Entonces, los nodos ejecutan un protocolo de consenso, donde cada insumo de cada nodo es su propio bloque propuesto

        - Algunos nodos son maliciosos y pueden poner transacciones inválidas en los bloques, pero se puede asumir que los otros nodos son honestos. Si el protocolo de consenso es exitoso, un bloque válido se seleccionará como el resultado, aunque este solo sea propuesto por un solo nodo (mientras que el bloque sea válido)

        - Podría haber transacciones restantes válidas que no se hayan incluido en el bloque, pero esto no es un problema. Si alguna transacción no se ha incluido en el bloque particular, esta puede esperar e incluirse en el siguiente bloque

    - Aunque este enfoque se parece a como Bitcoin trabaja, no es exactamente así, habiendo varios problemas técnicos

        - Primero, el consenso en general es un problema difícil, dado que los nodos pueden fallar o ser maliciosos

        - Segundo, la red es altamente imperfecta (sobre todo en el contexto de Bitcoin): es un sistema P2P, y no todo par de nodos se conectan a los otros. Podría haber fallos en la red debido a una baja conectividad a Internet, por ejemplo, y ejecutar un protocolo de consenso en la que todos los nodos deben participar no es realmente posible

        - Finalmente, hay mucha latencia en el sistema por ser distribuido en internet. Una consecuencia de la alta latencia es que no hay una noción de tiempo global, haciendo que no todos los nodos puedan estar de acuerdo en un orden común de los eventos basado en ver las etiquetas de tiempo (no todos los nodos pueden acordar qué mensaje se envió primero en un paso del protocolo)

- La falta de tiempo global restringe mucho el conjunto de algoritmos que se pueden usar en los protocolos de consenso. Debido a esto mucha literatura en

consenso distribuido es pesimista y se han probado muchos resultados de imposibilidad, pero algunos no todo aplica a Bitcoin

- Un resultado de imposibilidad muy famoso está relacionado con el problema de los generales bizantinos
    - En el problema, la armada bizantina se separa en divisiones, comandadas por un general por división, los cuales se comunican por mensajeros para poder diseñar un plan de acción. Algunos de estos generales son traidores, de modo que trataran de manipular el proceso de modo que los generales leales no puedan llegar a un plan unificado
    - El objetivo del problema es que todos los generales leales lleguen a un plan sin que los generales traicioneros hagan que adopten un mal plan
    - Se ha demostrado matemáticamente que es imposible que esto ocurra si un tercio o más de los generales son traidores
- Un resultado de imposibilidad más sutil es el resultado de imposibilidad de Fischer-Lynch-Paterson
    - Bajo ciertas condiciones, que incluyen que los nodos actúen de manera determinística, se ha demostrado que el consenso es imposible incluso con un solo proceso defectuoso
- Aunque haya resultados de imposibilidad, la literatura también ha presentado protocolos de consenso, como el protocolo Paxos
    - Por un lado, Paxos nunca produce un resultado inconsistente, pero, por otro lado, acepta el *trade-off* que, bajo ciertas condiciones, aunque son raras, el protocolo puede fallar al intentar obtener cualquier progreso
- Las buenas noticias es que estos resultados de imposibilidad se han demostrado para un modelo específico: se estudiaban bases de datos distribuidas, pero este modelo no es un buen representante de la configuración de Bitcoin, dado que esta viola varias suposiciones de estos modelos
    - Irónicamente, en el estado actual de la literatura, se puede ver como el consenso en Bitcoin funciona mejor en la práctica que en la teoría. Se observa consenso en el sistema, pero la teoría no ha llegado al punto de poder justificar por qué ocurre

        - El desarrollo de esta teoría es importante porque permite predecir ataques no previstos y otros problemas, y solo cuando se tenga una teoría así uno puede tener garantías sobre la seguridad y la estabilidad de Bitcoin

    - Bitcoin introduce la idea de los incentivos, que es nueva para un protocolo de consenso distribuido, violando las suposiciones de los modelos tradicionales

        - Esto solo es posible en Bitcoin porque es una moneda y por tanto tiene un mecanismo natural para incentivar a los participantes a que actúen honestamente

        - Por lo tanto, Bitcoin no resuelve el problema de consenso distribuido en un sentido general, sino que lo resuelve en el contexto específico de un sistema monetario

    - Bitcoin tienen en cuenta la noción de aleatoriedad, dado que su algoritmo de consenso se apoya fuertemente en la aleatorización. Además, también se deshace de la noción de un punto específico de inicio y final para el consenso

        - En vez de la noción de principio y final, el consenso se da en un tiempo largo de más o menos una hora en el sistema práctico. Aun llegando al final de ese tiempo, los nodos no siempre tienen la certeza de que una transacción particular o bloque ha llegado al libro mayor

        - A lo largo del tiempo, la probabilidad de que la visión de cualquier bloque coincida con la visión del consenso eventual incrementa, y la probabilidad de que la visión diverja decrece exponencialmente

Las mecánicas de Bitcoin: transacciones y los *scripts*

- El bloque fundamental de Bitcoin son las transacciones, pero para su discusión se utiliza un modelo simplificado del libro mayor: en vez de bloques, se supone que las transacciones individuales se añaden una a una

    - El primer modelo que se puede pensar con este modelo simplificado es un sistema basado en cuentas, que es como muchas personas piensan que Bitcoin funciona

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

- Uno puede añadir transacciones que crean nuevas monedas y las conceden a otros. Después, uno puede transferir las monedas

- Una transacción puede ser algo como "mover 17 monedas de Alicia a Bob", y sería firmada por Alicia. Esta sería toda la información que habría en el libro mayor de la transacción, y si Alicia tiene una cuenta, entonces se restaría la cantidad transferida al balance de la cuenta (sus monedas restantes)

- Lo malo de este sistema es que cualquiera que quiera determinar si una transacción es válida tendría que hacer seguimiento de todos los balances de cuenta. Aunque se puede hacer de manera un poco más eficiente con algunas estructuras de datos (para hacer seguimiento del balance después de cada transacción), esto requiere mucha infraestructura aparte del libro mayor

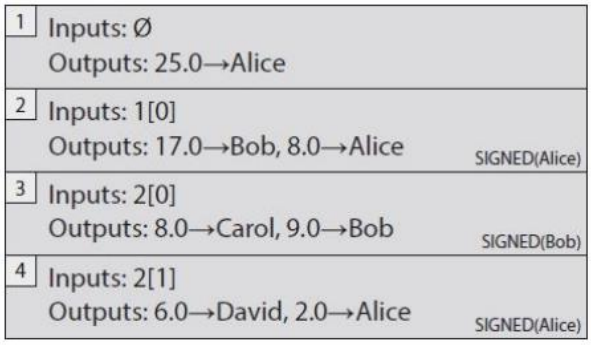
- Es debido a estos problemas que Bitcoin no usa un modelo basado en cuentas. En vez de este, Bitcoin utiliza un libro mayor que solo sigue las transacciones, similar al que se vio con el ejemplo de Scroogecoin

  - Las transacciones solo especifican un número de insumos y resultados (como los $PayCoins$ de Scroogecoin), en donde los insumos se pueden interpretar como las monedas que se están consumiendo (creadas en la transacción previa) y los resultados como las monedas que se crean. Cada transacción tiene un identificador único

  - Para transacciones en las que se emiten nuevas monedas, no hay monedas consumiéndose (como en $CreateCoins$ de Scroogecoin)

  - Los resultados se indexan comenzando por 0, de modo que el primer resultado es el resultado 0

- Se plantea un ejemplo del sistema de libro mayor (simplificado) que usaría Bitcoin para un mejor entendimiento del funcionamiento, basado en el siguiente esquema:



- La primera transacción no tiene insumos, dado que esta transacción está creando monedas, y tiene un resultado de 25 monedas que van a Alicia. En una transacción en donde se crean monedas no se necesita que se firme digitalmente nada

- Si Alicia quiere enviar algunas de sus monedas a Bob, crea la transacción 2 y se refiere a la transacción anterior explícitamente para indicar de dónde vienen las monedas que tiene (se refiere al resultado 0 de la transacción 1, que asignó 25 monedas a Alicia)

- Además, también se tiene que especificar la dirección en el resultado de la transacción (a quién se paga). En el ejemplo de la transacción 2 se especifican dos resultados, uno para Bob y otro para Alicia (para especificar las monedas restantes que le quedan), y se firma por Alicia para que se reconozca la autoría

- En Bitcoin, el resultado de una transacción se tiene que consumir entero (o ninguno), por lo que Alicia tiene que especificar un resultado para su dirección. Se puede utilizar otra dirección que aquella a la que pertenecen las monedas, pero la misma persona tiene que ser dueña, y a esto se le llama *change address* o dirección de cambio

- El ejemplo anterior permite ver que este modelo permite una verificación eficiente, la consolidación de fondos y los pagos conjuntos

  - La verificación de las transacciones es eficiente debido a que se utilizan *hash pointers* para referirse a los resultados de las transacciones previas. Para ver que las monedas no han sido previamente gastadas, se necesita escanear la *blockchain* entre la transacción referenciada y el bloque más reciente

  - Como las transacciones tienen muchos insumos y varios resultados, dividir y fusionar valor es fácil. Como uno quiere gastar las monedas totales de diferentes transacciones que se controlan, uno puede crear una transacción usando los insumos para las transacciones anteriores en las que se reciben las

monedas y poniendo como resultado una dirección que se controle y la cantidad de monedas totales (se consolidan las varias transacciones en una)

- Si varias personas quieren pagar a una sola, estas pueden crear una transacción con varios insumos y un solo resultado. La única diferencia con el caso anterior de consolidación de transacciones es que ahora se necesita firmar la transacción por todas las partes involucradas (excepto el receptor del pago)

- Conceptualmente esto es todo lo que hay en una transacción de Bitcoin, pero es interesante ver su representación a bajo nivel, dado que cualquier estructura de datos que se envía en es un *string* de bits (que se codifica en formato binario después)

- Los metadatos representan información útil como el tamaño de la transacción, el número de insumos, y el número de resultados. Se proporciona el *hash* de la transacción entera (el ID único de las transacciones), que es lo que permite que los *hash pointers* se refieran a las transacciones, y también se proporciona un *lock time*

- Los insumos de la transacción forman un arreglo (cada insumo tiene la misma forma), y estos insumos especifican transacciones previas, de modo que contienen el *hash* de aquellas transacciones, que actúa como un *hash pointer* para este. Además, cada insumo contiene la firma correspondiente para el *hash* de las transacciones

        - Los resultados también son un arreglo, en donde cada resultado tiene un valor y la suma de todos los valores de resultados tiene que ser menor o igual a la suma de los valores de insumos. Si la suma de los resultados es menor a la suma de los valores de insumo, la diferencia es una comisión de la transacción que va al minero que publica la transacción

El almacenamiento y uso de Bitcoins: llaves, carteras y mercados

- Hasta ahora se ha hablado de como almacenar y gestionar los Bitcoins, por lo que se pueden discutir maneras de usar servicios de terceros para la gestión de Bitcoins. El servicio más básico es el de una cartera *online*
    - Una cartera *online* es como una cartera local que uno maneja uno mismo, excepto la información se guarda en la nube, y uno accede a ella utilizando una interfaz web en un computador o en una aplicación. Algunos de estos servicios son Coinbase y blockchain.info
        - Lo que es crucial desde el punto de vista de la seguridad es que el sitio web entrega el código que se ejecuta en el buscador o la aplicación en el *smartphone*, y también guarda las llaves. Como mínimo, tendrá la habilidad de acceder a las llaves
        - Idealmente, el sitio encripta estas llaves bajo una contraseña que solo uno conoce, pero claramente uno tiene que confiar en el sitio de que realmente haga eso. Uno tiene que confiar en que su código no filtrará sus llaves o su contraseña
    - Una cartera tiene ciertos *trade-offs* comparado con tener una cartera local
        - Una gran ventaja es que es conveniente, dado que no se tiene que instalar nada en el computador para usar la cartera, o si es una aplicación de móvil, solo se tiene que instalar esta, pero no descargará la *blockchain*. Además, se puede usar en múltiples plataformas
        - No obstante, hay preocupaciones de seguridad, dado que, si el sitio o los operadores se vuelven maliciosos, los Bitcoins están comprometidos porque el sitio proporciona el código que tiene estos
        - Idealmente, el sitio o servicio está operado por profesionales de la seguridad que están más entrenados o son más diligentes que uno mismo al mantener la seguridad. Por lo tanto, uno espera que

los Bitcoins estén más seguros que si uno mismo los guarda, pero se tiene que confiar en ellos en primer lugar

- Otro servicio que se puede utilizar es el de las bolsas de Bitcoin, y se pueden entender mejor comparándolas con cómo los bancos o servicios parecidos operan en la economía real

    - Normalmente uno da un depósito al banco y este promete repagar el depósito en un momento en el futuro. Este suele invertir esos depósitos para sacar rentabilidad, pero estos no invierten todo

        - Los bancos suelen mantener parte de ese dinero en reserva para asegurarse de que puedan pagar la demanda de retiradas en un día normal o en un día extraordinario

        - Muchos bancos típicamente utilizan lo que se llama reserva fraccional: reservan una cierta fracción de toda la demanda de depósitos para cubrir riesgos de liquidez

    - Las bolsas de Bitcoin son negocios que, al menos desde el punto de vista de la interfaz del usuario, funcionan de manera similar a los bancos: estos aceptan depósitos de Bitcoins y prometen devolverlo en el futuro

        - Uno también les puede transferir moneda fiat a través de la cuenta bancaria. La bolsa promete pagar devuelta ambos o alguno de los tipos de monedas en demanda

    - Esta bolsa permite hacer varias actividades similares a las de un banco normal, por ejemplo, se pueden hacer y recibir pagos de Bitcoin entre particulares o cambiar moneda fiat por Bitcoins

        - Lo que normalmente hace la bolsa para intercambiar monedas es encontrar compradores de Bitcoins que son a su vez vendedores de dólares, y vendedores de Bitcoins que son a su vez compradores de dólares. Si hay un precio aceptado por ambas partes, la transacción se consuma

        - Cuando la transacción ocurre entre uno y otro cliente de la misma bolsa cotizada, no ha ocurrido ninguna transacción en la *blockchain* de Bitcoin. Las bolsas no necesitan recurrir a la *blockchain* para transferir Bitcoins o dólares de una cuenta a otra, todo lo que ocurre es que la bolsa cambia la promesa que le había hecho a cada una de las partes sobre cuánto devolverá de cada moneda

        - Una de las grandes ventajas de las bolsas de Bitcoin es que ayuda a conectar la economía Bitcoin y los flujos de Bitcoin con la

economía de la moneda fiat, haciendo fácil la transferencia de valor. No obstante, la desventaja es la presencia de riesgo similar al que se tiene exposición en los bancos

- Los riesgos a los que uno está expuesto en las bolsas de Bitcoin se pueden clasificar en tres categorías

    - El primer riesgo es el de corrida bancaria, que ocurre cuando varias personas demandan su dinero al mismo tiempo, y debido a la reserva fraccionaria, la institución puede no ser capaz de satisfacer a todos sus clientes. Este riesgo nace del comportamiento de pánico, en donde rumores o creencias sobre si la institución será solvente causa desconfianza

    - El segundo riesgo es que los propietarios de los bancos o las bolsas sean maliciosos y todo sea un esquema Ponzi, causando que se pierda mucho dinero

    - El tercer riesgo es el de *hacking*, en donde alguien pueda penetrar la seguridad de la bolsa y robar los datos y los Bitcoins u otras monedas de los clientes. Por ello, las bolsas deben ser muy cuidadosas con su *software* de seguridad y con sus procedimientos (como lo del almacenamiento *hot and cold* y todo eso)

    - Todo esto ha ocurrido con anterioridad: muchas bolsas han cerrado por la consumación de estos riesgos. El ejemplo más famoso es el de Mt. Gox, que era una de las bolsas de Bitcoin más grandes y se volvió insolvente, sin saber dónde el dinero de sus depositantes se fue

- Los gobiernos regulan estrictamente a los bancos reales, y suelen imponer un requerimiento mínimo de reserva en los bancos y los tipos de inversiones y métodos de gestión que estos pueden usar

    - Todo esto tiene el objetivo de asegurar que los activos bancarios se invierten en lugares con relativamente poco riesgo, dado que sus activos son de sus depositantes

    - A cambio de estas formas de regulación, los gobiernos normalmente hacen cosas para ayudar a los bancos y a sus depositantes, tales como seguros de depósito o concesión de rescates bancarios

    - Aunque los bancos estén regulados de esta manera, eso no ocurre con las bolsas de Bitcoin

- Una bolsa de Bitcoin u otra institución que tenga Bitcoins pueden usar un truco criptográfico llamado *proof of reserve* para asegurarle a los clientes que sus depósitos están seguros. El objetivo es que la bolsa o las instituciones demuestren que tienen reserva fraccionaria de los depósitos

    - El problema del *proof of reserve* se puede dividir en dos piezas. La primera es demostrar cuantas reservas tiene la bolsa y la segunda es demostrar la cantidad de la demanda de depósitos de la bolsa

    - Para la primera parte, la bolsa simplemente debe publicar una transacción de la bolsa a ella misma con la cantidad de reservas que dice tener y mostrar que esta transacción es válida. Entonces, ellos firman un *challenge string* (un *string* aleatorio de bits generado por una parte imparcial) con la misma llave privada que se ha usado para firmar la transacción, demostrando así a cualquiera que alguien que sabía la llave privada ha participado en la *proof of reserve*

    - Estrictamente hablando, este primer proceso no es una demostración de que la parte que dice tener estas reservas es propietaria de estas, solo que aquella entidad que es propietaria está dispuesta a colaborar en el proceso. No obstante, parece una demostración de que alguien que controla o conoce a alguien que controla aquella cantidad de Bitcoins

    - Además, se tiene que ver que uno siempre puede poner una cantidad menor a la que tiene, por lo que esta demostración establece el mínimo que la compañía tiene, pero no necesariamente la cantidad real ni el máximo

- La segunda pieza es la parte difícil, y en este caso el proceso se conoce como *proof of liabilities*. Si la bolsa puede demostrar sus reservas y sus depósitos de demanda, entonces cualquier puede dividir ambas cantidades para determinar la reserva fraccionaria

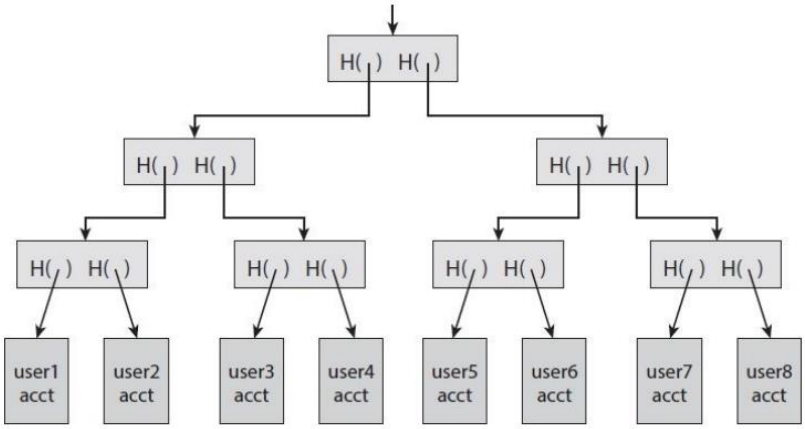
    - El esquema que se desarrolla permite que la bolsa pueda decir que tiene más depósitos de los que realmente tiene, pero no menos. De este modo, si la compañía puede demostrar que sus reservas son de al menos cierta cantidad y sus pasivos son como mucho de cierta cantidad, se establece una cota inferior para su reserva fraccional

    - Si a la bolsa no le importara la privacidad de sus usuarios, podría publicar sus registros de clientes y sus depósitos: de este modo, cualquiera puede calcular los números para ver si hay fraude (en el número de clientes, de depósitos, etc.). Aunque la compañía

pudiera crear usuarios falsos, solo puede incrementar el valor de sus pasivos totales, de modo que se establecería una cota inferior de los depósitos al publicar

- Para poder demostrar esta cota inferior sin tener que comprometer la privacidad, se pueden usar los árboles de Merkle. La bolsa puede ejecutar la demostración construyendo este árbol donde cada hoja corresponde a un usuario y sus cantidades de depósitos, y publicando su *hash* raíz y el valor total de depósitos (asociado a esta raíz)

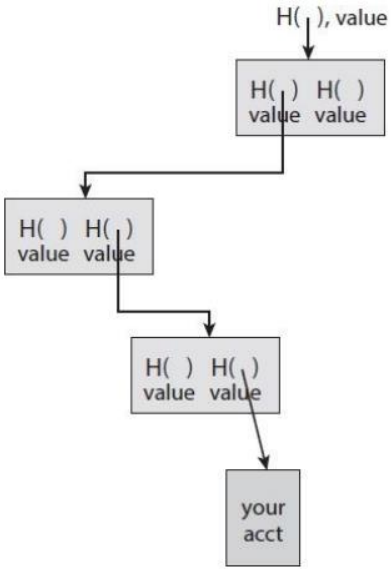
    - En este protocolo, cada usuario es responsable de asegurar que está incluido en el árbol. Además, hay una manera de que los usuarios colectivamente comprueben el total de depósitos publicado por la bolsa



    - A cada uno de los *hash pointers* se le añade un atributo o espacio adicional, que es el número que representa el valor monetario total en Bitcoins de todos los depósitos que están en el subárbol debajo de ese *hash pointer* en árbol. Para que esto sea verdad, el valor correspondiente de cada *hash pointer* debe ser la suma de los valores de los dos *hash pointers* debajo

    - La bolsa construye el árbol, criptográficamente firma el puntero raíz junto con el valor de atributo raíz (el total de pasivos), y lo publica. La bolsa está intentando mostrar que los valores de depósito de sus usuarios están representados correctamente y que los valores se propagan correctamente arriba del árbol, demostrando así que el valor raíz es la suma de los depósitos de todos los usuarios

- De este modo, cualquier cliente puede ir a la organización y pedir una demostración de su inclusión correcta. La bolsa debe mostrarle al cliente el árbol parcial desde la hoja del usuario hasta la raíz



- El cliente, entonces, verifique que el *hash pointer* raíz y el valor raíz son los mismos que la bolsa ha firmado y publicado, que los *hash pointers* son consistentes en todo el camino hacia abajo (a las hojas), que las hojas contienen la información del usuario correcta (ID del usuario y la cantidad de depósito), que cada valor es la suma de dos valores debajo de ese nodo, y que no hay valores negativos

- Si cada cliente hace eso, entonces cada rama del árbol se puede explorar, y alguien verificará que, para todo *hash pointer*, su valor asociado iguala a la suma de los valores de los dos *hash pointers* hijos. Por cómo funciona el árbol, uno se asegura que la bolsa no puede presentar valores diferentes en ninguna parte del árbol

- En resumen, tanto la demostración del mínimo de reservas X y la demostración del máximo de pasivos Y permite obtener una reserva fraccionaria de al menos X/Y

  - Ambas demostraciones, no obstante, revelan mucha información privada: específicamente, revelan todas las direcciones usadas en la bolsa, el valor total de reservas y pasivos y hasta información de los balances individuales de cada usuario. Las bolsas reales son contrarias a publicar esta información, por lo que las demostraciones criptográficas de reservas han sido muy raras en la práctica

  - Un protocolo reciente llamado “Provisions” permite realizar la misma demostración de solvencia, pero sin revelar el total de pasivos o reservas o las direcciones en uso. Este protocolo usa criptografía más avanzada, y es otro ejemplo de como la criptografía puede garantizar la privacidad

- La bolsa, donde se cambian monedas fiat por Bitcoins, se ha discutido desde el punto de vista del negocio, pero ahora se discute como un mercado

    - Estas bolsas suelen operar como los mercados entre dos divisas fiat: el precio fluctúa dependiendo de qué tanto se quiera comprar una moneda fiat y qué tanto se quiera comprar un Bitcoin

        - Explorando webs que proporcionan datos sobre el comercio de monedas y Bitcoins, se puede observar que el volumen de comercio es grande y que los precios se mueven en tiempo real mientras se ejecutan transacciones. Este mercado es muy líquido, con varios sitios donde uno puede comprar y vender Bitcoins

        - Otra opción que existe es encontrarse con personas que comercien Bitcoin, habiendo sitios webs que ayudan a encontrar a esta gente. En estas webs aparecen personas que quieran comprar o vender Bitcoins y la cantidad y el precio que se ofrece

        - Finalmente, algunos lugares tienen reuniones regulares donde las personas quedan para comerciar con Bitcoins. Una razón por la que algunas personas quieren obtener Bitcoins en persona es porque de este modo hay anonimato (todo lo que se pueda en un sitio físico), en comparación a hacerlo *online* por la bolsa, donde normalmente se requiere verificación de identidad gubernamental por regulaciones bancarias

    - Igual que en cualquier mercado, el mercado de la bolsa de Bitcoin junta compradores con vendedores para Bitcoins (a cambio de monedas fiat). Aunque es un mercado relativamente grande (millones de dólares al día), es más pequeño que las bolsas de valores tradicionales, pero tiene el suficiente volumen para establecer un precio de consenso y ser líquido

        - El precio de consenso del mercado se fija por la oferta y la demanda, donde la oferta de Bitcoins es la cantidad de Bitcoins que potencialmente se pueden vender y la demanda es la cantidad de Bitcoins que se querría comprar por las personas con monedas fiat. A través de este mecanismo de mercado, el precio será fijado a un nivel que iguale la demanda y la oferta

    - La oferta de Bitcoins es igual al número de Bitcoins en circulación. En 2015, esta cantidad era de 15 millones, y las reglas de Bitcoin actuales dicen que este número subirá poco a poco hasta llegar a un límite de 21 millones

        - En la oferta también podría incluir depósitos a la vista de Bitcoins: si alguien ha puesto dinero en su cuenta en una bolsa de Bitcoin y esta no mantiene una reserva completa para cubrir todos los

depósitos, entonces habrá depósitos a la vista en ese intercambio que serán mayores que la cantidad de monedas que la bolsa tiene

- Dependiendo de la pregunta que se haga sobre el mercado, puede ser correcto o no incluir los depósitos a la vista. Estos se deberían incluir en el análisis cuando el dinero en estos depósitos se puede vender en el mercado

- Existen dos fuentes principales para la demanda de Bitcoins: la demanda de estos como una manera de mediar transacciones de monedas fiat y como una inversión

  - Si una persona quiere comprar algo de una segunda y quiere transferir un cierto número de dinero fiat, pero encuentran que es más conveniente transferir Bitcoins, entonces la primera persona puede comprar Bitcoins con moneda fiat y transferirlos a la segunda, la cuál recibirá estos y los venderá por moneda fiat. Esto crea demanda de Bitcoins porque los Bitcoins usados para intermediar se han tenido que sacar de circulación (se han tenido que pedir a una tercera persona)

  - Otra posibilidad es que alguien quiera comprar Bitcoin y mantenerlo en cartera con la esperanza de que su precio incrementará en el futuro y se podrá vender para obtener un beneficio en moneda fiat. Cuando las personas compran y mantienen, estas monedas se sacan de circulación

- Es posible realizar un modelaje económico simple para entender como funcionan estos mercados. En este modelo, el foco está en la demanda basada en Bitcoins como medio de transacción y el efecto que puede tener esto en el precio de los Bitcoins

  - El valor total de valor transaccionado a través de Bitcoin en el mercado es T, medido en dólares por segundo; la duración que los Bitcoins necesitan mantenerse fuera de circulación para mediar es D (el proceso explicado anteriormente); la oferta total de Bitcoins para su compra es S, igual a la cantidad de Bitcoins disponibles menos la cantidad de Bitcoins que se usan para inversión; y finalmente, P es el precio del Bitcoin en dólares

  - La cantidad de Bitcoins que se vuelven disponibles para dar servicio a las transacciones cada segundo es de S/D Bitcoins en promedio (la oferta). El número de Bitcoins por segundo que se necesita para dar servicio a las transacciones cada segundo es de T/P Bitcoins

- Las dinámicas de cualquier mercado se respetan: si la oferta es más grande que la demanda, entonces P bajará porque los ofertantes pondrán un menor precio de compra para vender, mientras que, en el caso contrario, P subiría porque los demandantes subirían sus precios para ser capaces de obtener Bitcoins. En equilibrio, la demanda será igual a la oferta, lo cual permite obtener la siguiente fórmula:

$$\frac{S}{D} = \frac{T}{P} \Rightarrow P = TD/S$$

- Como la oferta total S solo incluye los Bitcoins usados para intermediar, entonces el aumento de Bitcoins como inversión haría que se reduzca S, haciendo que el precio crezca

La plataforma Bitcoin: mercados de predicción y *data feeds*

- La idea del arbitraje en este contexto descentralizado permite ir a conceptos más generales como el de introducir datos del mundo real a las criptmonedas a través de *data feeds* o como el de libro de órdenes u *order book*

    - En la mayoría de mercados financieros o de predicción, no hay un solo precio de mercado, sino que hay precios de compra o *bids* y precios de venta o *asks*

        - El *bid price* es el precio más grande al que alguien está dispuesto a comprar una acción, mientras que el *ask price* es el precio más bajo al que alguien está dispuesto a vender la acción. Típicamente el *ask price* es mayor al *bid price*, dado que de otro modo se produciría una transacción y alguna de las órdenes ya no estaría en el *order book*

        - Una persona que quiere comprar una acción inmediatamente lo puede hacer al *ask price*, mientras que un participante que quiera vender la acción inmediatamente lo puede hacer al *bid price*. Estas órdenes se conocen como órdenes a mercado o *market orders*, mientras que las órdenes limitadas o *limit orders* son órdenes registradas en el *order book* que se ejecutan a un precio límite específico (o mejor)

        - Tradicionalmente esto se ha hecho de una manera centralizada con un solo servicio de *order book* (normalmente una bolsa) que recoge todas las órdenes. El problema, típico de servicios centralizados, es que una bolsa deshonesta puede sacar un beneficio a costa de los participantes: al recibir una orden de compra, la bolsa podría comprar ellos mismos al mejor *ask price* antes de poner la orden recibida, y vender estas acciones

compradas a un mayor precio, llevándose la diferencia (el llamado *frontrunning*, que es ilegal)

- En un *order book* descentralizado, uno no se puede apoyar en el respeto de la ley, pero hay una solución inteligente que consiste en olvidrase del *frontrunning*

  - La idea es que cualquiera pueda subir órdenes límite a los mineros a través de transmitir transacciones, y estos puedan hacer coincidir cualquier par de órdenes mientras que el *bid price* sea mayor o igual al *ask price* (para que haya transacción), de modo que se llevan la diferencia como una comisión por la transacción (proveniente del *bid price*). Esto desincentiva al *frontrunning* por parte de los mineros porque en ningún caso será más provechoso

  - Esta es una manera elegante de construir un *order book* descentralizado, pero la gran desventaja son los costes de transacción que los comerciantes deben pagar al minero. Para poder evitar pagar esta comisión, los comerciantes pueden subir órdenes más conservadoras o pueden no querer revelar sus precios directamente, haciendo que el mercado sea menos eficiente

  - Uno aún no sabe cómo funcionaría este tipo de *order book* en la práctica, pero parece una idea prometedora

Las Altcoins y el ecosistema de criptmonedas: relación con Bitcoin

- Por supuesto, estas Altcoins tienen relación con Bitcoin, y se pueden usar diferentes métricas para poder comparar Altcoins entre ellas y discutir la relación que guardan con Bitcoin

  - Tradicionalmente, la capitalización de mercado o *market cap* es un método simple para estimar el valor de una corporación pública basada en multiplicar el precio de la acción por el número total de acciones restantes

    - En el contexto de las Altcoins, el *market cap* se usa de manera similar para estimar el valor total del Altcoin a través de multiplicar el precio de una unidad individual del Altcoin (medido, por ejemplo, con los precios de las bolsas más populares) por el número total de unidades de Altcoin que se piensa que están en circulación

    - Con esta métrica se puede ver que Bitcoin es la más grande, representando en 2015 el 90% del *market cap* de criptomonedas. La clasificación relativa de las otras Altcoins suele variar

muchísimo, pero el punto está en ver que son comparativamente pequeñas en términos de valores monetarios

- Esta métrica no es la más informativa: no expresa necesariamente cuanto le costaría a alguien comprar todas las monedas en circulación (puede ser mayor o menor porque las órdenes grandes pueden mover el precio de la moneda), no incluye las monedas que se incluirán en el futuro a la circulación (lo cual dificulta la interpretación) y no puede estimar precisamente el número de monedas actualmente en circulación (algunos poseedores han perdido claves privadas y no hay manera de saber el porcentaje de monedas perdidas)

- Si dos Altcoins utilizan el mismo puzle de minería, se pueden comprar directamente con cuánto energía o poder de minería tienen todos los mineros de estas Altcoins. A esto se le suele llamar *hash rate* debido a la prominencia de puzles basados en *hashes*

  - Porr ejemplo, Zetacoin utiliza puzles SHA-256 como Bitcoin, y su *hash rate* en la red es de 5 terahashes por segundo (que son 5×10^{12} *hashes* por segundo) a fecha de Diciembre de 2015, lo cual es una parte muy muy pequeña del poder de minería de Bitcoin

  - Es más difícil comparar la potencia de minería entre Altcoins que usan diferentes puzles de minería, dado que estos pueden tomar tiempos diferentes. Además, el *hardware* especializado de minería para una de las monedas puede no ser útil para las otras (incluyendo la posibilidad de atacar)

  - Aunque una Altcoin use un puzle de minería completamente único, aún se puede aprender algo del cambio relativo en la potencia de minería a lo largo del tiempo. Esta métrica indica si hay más participantes que se han unido o que han mejorado sus equipamientos de minería, o si por el contrario más gente lo ha abandonado

- Existen muchos otros indicadores que se pueden mirar para valorar y comparar Altcoins

  - Los cambios en el tipo de cambio a lo largo del tiempo dan una pista sobre la salud de la moneda y tiende a estar correlacionado con cambios en el *hash rate* en largos periodos de tiempo

  - El volumen de intercambios en varias bolsas mide la actividad y el interés en la Altcoin. En cambio, el volumen de transacciones en la *blockchain* del Altcoin no dice mucho, dado que los usuarios

pueden estar mezclando sus propias monedas en su propia cartera con varias transacciones (incluso automáticamente)

- Finalmente, también se puede mirar cuántos comerciantes y procesadores de pago soportan el pago en la Altcoin: solo las criptomonedas más importantes tienden a estar incluidas en este tipo de servicios

- La relación entre Bitcoin y las Altcoins es complicada: las criptomonedas compiten entre sí como método de pago *online*. Si hay dos estándares, protocolos o formatos en competición que son más o menos equivalentes en términos de lo que ofrecen, una de ellas tenderá a dominar por lo que se conoce como efectos de red
  - Esta línea de razonamiento sugiere que una criptomoneda (presumiblemente Bitcoin) dominará, aún si los sistemas sucesores pueden ser técnicamente superiores. No obstante, esto es simplificar demasiado, dado que la competencia entre criptomonedas no es tan hostil por al menos dos razones

  - La primera es que es relativamente fácil para los usuarios convertir una criptomoneda a otra y es fácil para los vendedores aceptar múltiples criptomonedas, de modo que varias criptomonedas pueden coexistir y crecer (hay pocos costes de conversión). Los costes de conversión no son nulos, pero es más fácil convertirlas o usar más de una al mismo tiempo

  - La segunda es que muchas Altcoins tienen características únicas que proporcionan una razón distinta para su existencia: no son sustitutos de Bitcoin, sino que a veces pueden ser ortogonales o incluso complementarias. De este modo, Altcoins complementarias pueden incrementar la utilidad de Bitcoin y no competir contra ella

  - No obstante, esto último es una simplificación, dado que algunas Altcoins (como Litecoin) intentan conseguir la misma funcionalidad que otras, pero de manera diferente o más eficiente

  - Algunos usuarios dicen que tener varias Altcoins divide la potencia de *hash* en diferentes frentes y hacen que cada criptomoneda sea menos segura, y, consecuentemente, menos valiosa. En contraste, otros usuarios dicen que permite que el mercado decide qué tipos de características son las mejores y qué criptomonedas son superiores, y también argumentan que la existencia de varias criptomonedas hace que se reduzca el riesgo idiosincrático de un solo sistema

- Aunque se ha usado un punto de vista económico y político de las Altcoins en las discusiones anteriores, ahora uno se puede centrar en las interacciones técnicas entre Bitcoin y las Altcoins

    - A fecha de 2015, la potencia de *hash* de Bitcoin eclipsa el de cualquier otra Altcoin, teniendo mineros poderosos y grupos de minería que controlan más potencia de minería que el implementado por todas las otras Altcoins

        - Tal minero o entidad podría llevar a cabo un ataque a cualquier Altcoin pequeña que utilice un puzle SHA-256 para causando bifurcaciones y estragos, lo cual es suficiente para matar esta Altcoin. A esto se le conoce como infanticidio de Altcoins

        - Estos ataques no se suelen producir para obtener beneficios, sino que se suelen producir por otras razones. Por ejemplo, el operador de la agrupación de minería Eligius decidió atacar CoiledCoin porque pensó que era una estafa y dañaba el ecosistema de criptomonedas, de modo que lo atacó y desapareció

    - Por defecto, la minería de una Altcoin es exclusiva, por lo que se puede intentar resolver la solución del puzle de minería para encontrar un bloque válido de la Altcoin o de Bitcoin, pero no ambos al mismo tiempo

        - Por supuesto, uno puede dividir sus recursos en minar Altcoin y minar Bitcoin, o dividir la minería entre diferentes Altcoins y asignar recursos de manera dinámica, pero no hay manera de que la potencia de minería haga el doble de trabajo

        - Con la minería exclusiva, los efectos de redes pueden hacer difícil que una Altcoin haga *bootstrapping* (lanzar y establecer la criptomoneda para que gane valor), dado que se tendría que convencer a los mineros de Bitcoin de que dejen de minar Bitcoin para entrar en su red, causando pérdidas inmediatas de las recompensas de Bitcoin. Esto significa que una Altcoin que se lance así probablemente se quede pequeña en términos de potencia de *hashing* y que sea más vulnerable a ataques infanticidas por mineros de Bitcoins

        - Para poder diseñar una Altcoin en la que se puedan minar bloques de esta Altcoin y Bitcoin a la vez, es necesario crear bloques que incluyan transacciones en ambas monedas, haciéndolas válidas en ambas *blockchains*. Es fácil hacer que se incluyan transacciones de Bitcoin en la Altcoin (se diseña a placer), pero hacer el proceso inverso es complicado

- Anteriormente se han discutido maneras de meter datos arbitrarios en las transacciones de Bitcoin, pero la banda ancha de estos métodos es limitada

- Un truco para poder incluirlas en los bloques de transacciones de Bitcoin es poner un resumen de los detalles de la transacción de la Altcoin en forma de *hash pointer* al bloque de la Altcoin

    - Específicamente, todos los bloques de Bitcoin tienen una transacción especial (la transacción de *coinbase*) que el minero utiliza para crear nuevas monedas como una recompensa del bloque. El campo *scriptSig* de esta transacción no tiene significado (no hace falta firmarla porque no gasta ningún resultado de transacciones previas), por lo que se puede usar para guardar datos arbitrarios

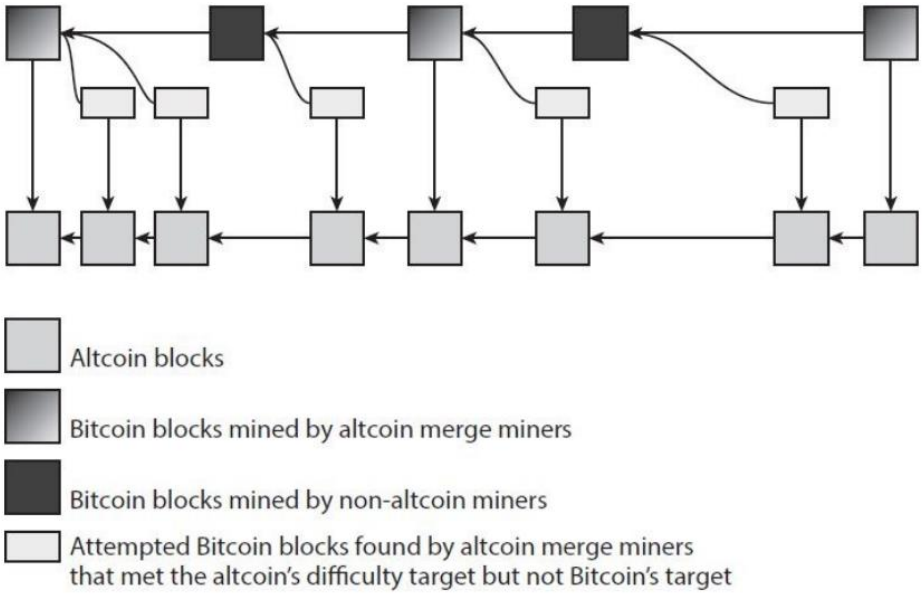
    - En una Altcoin con minería fusionada o *merge-mined*, la tarea de minería es calcular bloques Bitcoin cuya transacción *coinbase* tenga un *scriptSig* que contenga un *hash pointer* al bloque de la Altcoin

    - Para los clientes de Bitcoin, este bloque es un bloque de transacciones normal y corriente, con un *hash* que puede ser ignorado, y los clientes de la Altcoin entienden este bloque y pueden mirar las transacciones en el bloque de la Altcoin a la cual apunta el puntero. Aunque esto no requiere ningún cambio en Bitcoin, si requiere que la Altcoin pueda entender Bitcoin y aceptar bloques *merge-mined*

    - Si una Altcoin es *merge-mined*, se espera que muchos mineros de Bitcoin la minen, debido a que no requiere potencia de *hash* adicional, y solo requiere un módico coste en recursos computacionales adicionales para procesar bloques y transacciones y que se sepa lo mínimo de la Altcoin por parte de los mineros (para que tengan interés en minarla)

    - Haciendo esto, los bloques de la Altcoin dependerían del porcentaje de mineros de Bitcoin que minaran la moneda. Si el porcentaje es bajo debido a que la moneda está aún *bootstrapping*, entonces el tiempo entre bloques podría ser de varias horas o días, lo cual es inaceptable

- Una manera de hacer que la creación de bloques se de a una tasa estable, tan alta o bajo como una quiera, y que sea independiente de la fracción de mineros es alterando el *target* (aunque la tarea de minar sea la misma)



- La Altcoin calcula el *target* y la dificultad de sus bloques independientemente de la red Bitcoin. Igual que Bitcoin ajusta su *target* para que se minen bloques cada 10 minutos, se puede ajustar el objetivo de la Altcoin a un intervalo de tiempo deseado

- El *target* de la Altcoin, por tanto, se pone mucho más bajo que el de Bitcoin, y algunos bloques de la Altcoin no serán apuntados por los bloques válidos de Bitcoin. No hay problema con esto debido a que ambas *blockchains* se pueden interpretar como paralelas, con punteros ocasionales de un bloque de Bitcoin a otro de Altcoin

- En el ejemplo se puede ver como todos los bloques de la Altcoin resultan de intentar minar un bloque de Bitcoin, pero solo un porcentaje pequeño lo consigue, y para el otro porcentaje, la red Altcoin tiene que ser capaz de verificar la solución del puzle de minería. La manera más fácil de hacer esto es transmitir solo el cabezal del bloque de Bitcoin más cercano y la demostración de inclusión en el árbol de Merkle en la transacción de *coinbase* en el bloque de Bitcoin

- También es posible que el nivel de dificultad de los bloques de la Altcoin sea mayor a la de Bitcoin, por lo que ocurriría el análogo a lo que ocurre en el esquema, pero con la Altcoin rechazando los bloques que no llegan al nivel de dificultad requerido. Esto es raro porque normalmente uno quiere que la Altcoin genere bloques más rápido para *bootstrapping*

- Finalmente, se puede ver como el número de Altcoins puede ser *merge mined* simultáneamente con Bitcoin, y cada minero es libre de escoger un subconjunto arbitrario de Altcoins para *merge mine*.

En este caso, el *scriptSig* de la transacción *coinbase* sería un árbol de Merkle de *hash pointers* para varios bloques de Altcoins

- El nivel de complejidad de esto último es grande, debido a que requiere verificar una demostración de Merkle de la inclusión de la transacción de la Altcoin en el bloque de la Altcoin, una demostración de Merkle de la inclusión del *hash* del bloque de la Altcoin en el bloque en el *scriptSig* de la transacción *coinbase* y una demostración de Merkle de la inclusión del *scriptSig* de la transacción *coinbase* en el bloque de Bitcoin o en el bloque más cercano

Ethereum: la red y sus aplicaciones

- Se han visto varias maneras de usar lenguaje de *scripting* de Bitcoin para que se soporten aplicaciones interesantes y cómo de limitado es al no ser completo en el sentido de Turing. Por ello, una Altcoin llamada Ethereum nació con tal de solucionar este problema
    - El objetivo primordial es construir una criptomoneda que pueda soportar cualquier aplicación futura sin tener que lanzar un nuevo sistema, que es también la idea detrás de la completitud de Turing
        - Un lenguaje de programación completo en el sentido de Turing permite especificar cualquier funcionalidad que sea posible programar en una máquina de Turing, un modelo abstracto de una computadora que se cree que es capaz de calcular cualquier función calculable
        - Por lo tanto, cualquier lenguaje de programación completo en el sentido de Turing (Python, Java, etc.) son idénticos en el conjunto de cálculos que permiten expresar
    - Ethereum es una Altcoin que quiere proporcionar un lenguaje de programación completo en el sentido de Turing para escribir *scripts* y contratos
        - Aunque existen otras alternativas similares, Ethereum es la más notable, consiguiendo financiación rápida y adoptando elecciones agresivas para sus parámetros (como el tiempo de bloqueo)
    - El término “contrato inteligente” o “*smart contract”* se usó para describir a los sistemas computacionales (u otros métodos automatizados) para obligar el cumplimiento de los contratos

- Como ejemplo, uno puede pensar en una máquina vendedora como un contrato inteligente que obliga a que se respete un acuerdo entre uno mismo y el propietario de la máquina para la compra de un snack

- En Ethereum, un contrato es un programa que vive en la *blockchain*. Cualquiera puede crear un contrato de Ethereum, por una pequeña comisión, al subir su código en una transacción especial

- Este contrato se escribe en Solidity y se ejecuta por una máquina virtual específica de Ethereum, normalmente llamada EVM. Una vez que se ha subido, el contrato vivirá en la *blockchain*

- El tendrá su propio balance de fondos, otros usuarios pueden hacer llamadas a los procedimientos a través de cualquier API que el programa exponga (invocación de funciones), y el contrato puede enviar y recibir dinero

- Un ejemplo de cómo Ethereum puede implementar cualquier funcionalidad específica de una Altcoin es ver cómo implementar una funcionalidad como la de Namecoin en un contrato de Ethereum

```
contract NameRegistry {
      mapping(bytes32 => address) public registryTable;
      function claimName(bytes32 name) {
            if (msg.value < 10) {
                  throw;
            }
            if (registryTable[name] == 0) {
                  registryTable[name] = msg.sender;
            }
      }
}
```

- El contrato del ejemplo implementa un almacén de nombre/valor crudo o un registro de nombres, en el que a los nombres se les asignan valores de una vez por todas

- El contrato define una variable de datos $registryTable$, que es un mapeado de *strings* de 32 bytes a llaves públicas e inicialmente mapea cualquier *string* a la dirección nula 0x0000…000. También se define un solo punto de entrada, llamado $claimName$, el cual acepta un solo argumento para el nombre

- Primero, el contrato asegura de que el que lo invoca haya enviado un valor de al menos 10 wei, donde el wei es la unidad monetaria más pequeña en Ethereum. Si se han enviado fondos insuficientes,

el contrato termina con un error y no se realiza ninguna acción, y si se envían fondos suficientes, entonces se le asigna el valor de la dirección que ha invocado la función permanentemente, siempre y cuando el nombre no se haya cogido antes

- También se podrían añadir otras características de Namecoin como guardar más datos en cada mapeado aparte de la dirección de la entidad que invocó la función, requerir a los propietarios de nombres que se registren otra vez periódicamente o hacer que los usuarios puedan coger nombres que no se han actualizado en mucho tiempo. Incluso se podría añadir una segunda función que permita sacar dinero del contrato sin que se vaya acumulando eternamente y se identifique al invocador de la función a través de su firma digital

- En el universo de Ethereum existe una computadora única y canónica llamada *Ethereum Virtual Machine* (EVM) cuyo estado es acordado por toda la red de Ethereum y todos los nodos de la red guardan una copia del estado global de esta computadora

  - Adicionalmente, cualquier participante puede transmitir una petición para que esta computadora realice un cálculo arbitrario. Cuando esta petición se ha transmitido, los otros participantes de la red tienen que verificar, validar y ejecutar el cálculo

  - Esta ejecución causa un cambio en el estado de la EVM, que se confirma y se propaga a través de la red entera

  - Las peticiones de cálculo se conocen como peticiones de transacción. El registro de todas las transacciones y el estado presente del EVM se guarda en la *blockchain*, que a su vez se guarda en y se acuerda con todos los nodos

- La máquina virtual de Ethereum (EVM) es un entorno virtual descentralizado que ejecuta código consistentemente y con seguridad entre todos los nodos de Ethereum. Los nodos ejecutan el EVM para ejecutar contratos inteligentes, usando el gas para medir el esfuerzo computacional requerido por las operaciones, asegurando asignación eficiente de los recursos y la seguridad de la red

  - Normalmente se usa la analogía de un libro mayor distribuido para describir *blockchains* como Bitcouin, que permiten que la criptomoneda sea descentralizada usando herramientas fundamentales de la criptografía. Ethereum, aunque tiene una moneda nativa que sigue la misma intuición, permite una función más poderosa, los contratos inteligentes, por lo que se necesita

una analogía más poderosa: la de una máquina de estado distribuido

- El estado de Ethereum es una gran estructura de datos que mantiene no solo todas las cuentas y balances, sino que también el estado de la máquina, que puede cambiar de bloque a bloque acorde a un conjunto predefinido de reglas y permite ejecutar código de máquina arbitrario (las reglas para cambiar el estado de bloque a bloque se definen por la EVM)

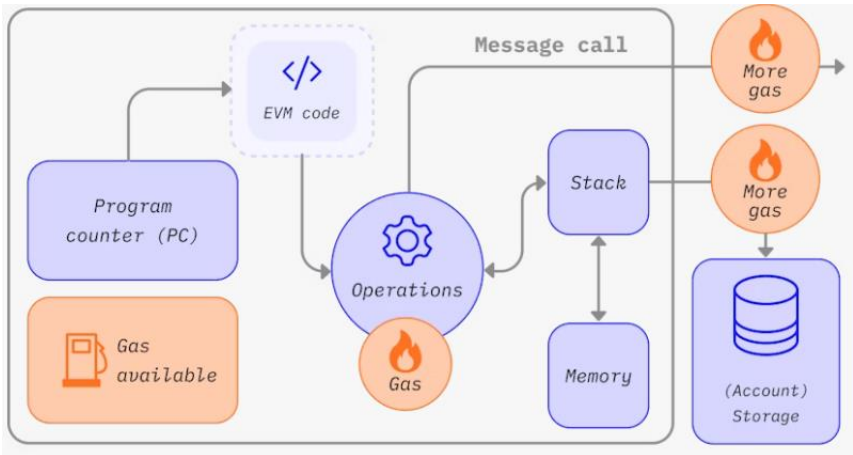
- La EVM se comporta como una función matemática: dado un insumo, produce un resultado determinista. Por lo tanto, es muy útil describir la máquina virtual como teniendo una función de transición de estado

$$Y(S_t, T_{t+1}) = S_{t+1}$$

- Dado un estado S_t y un nuevo conjunto de transacciones válidas T_{t+1}, la función de transición de estado de Ethereum produce un nuevo estado válido resultante S_{t+1}

- En el contexto de Ethereum, el estado es toda esta estructura de datos (el árbol de Merkle Patricia, que mantiene todas las cuentas vinculadas por *hashes* y reducible a un solo *hash* raíz guardado en la *blockchain*

- La EVM se ejecuta como una máquina de *stack* (computadora que utiliza una pila para almacenar datos y ejecutar operaciones secuencialmente) con una profundidad de 1024 elementos, donde cada elemento es una palabra de 256 bits (por facilidad de uso con la criptografía de 256 bits)



- Durante la ejecución, la EVM mantiene una memoria transitoria (como un arreglo de bytes dirigida por palabras) que no persiste entre transacciones. No obstante, los contratos contienen un árbol de almacenamiento de Merkle Patricia asociado con la cuenta en cuestión y que es parte del estado global

        - El *bytecode* compilado de los contratos inteligentes se ejecuta como un número de *opcodes* de EVM, que realizan operaciones de *stack* estándar como XOR, AND, ADD, SUB, etc. El EVM también implementa un número de operaciones específicas para la *blockchain*, tales como ADDRESS, BALANCE, BLOCKHASH, etc.

- Muchas de las aplicaciones financieras que se han visto anteriormente se pueden implementar en Ethereum. Aunque hay sutilezas en cada una de las aplicaciones, todas son posibles de implementar y hasta de manera más simple que con los diferentes protocolos vistos para Bitcoin

    - Anteriormente se han discutido dos maneras de implementar un libro mayor: basado en cuentas o basado en transacciones. Bitcoin utiliza un libro mayor basado en transacciones, de modo que la *blockchain* solo guarda transacciones

        - Para facilitar la validación de las transacciones, Bitcoin trata las monedas como inmutables y los resultados de las transacciones deben gastarse en su totalidad, utilizándose *change addresses* si es necesario. Efectivamente, las transacciones operan en un estado global, que es una lista de resultados de transacciones no gastados, pero este estado nunca se hace explícito en el protocolo Bitcoin y es simplemente algo que los mineros crean por su cuenta para acelerar la verificación

        - En contraste, Ethereum usa un modelo basado en cuentas: dado que Ethereum ya almacena una estructura de datos que asigna las direcciones del contrato al estado global, es natural almacenar también el saldo de la cuenta de cada dirección regular (también llamada "dirección propia") en el sistema. Entonces, en lugar de representar los pagos mediante un grafo de transacciones acíclico, donde cada transacción gasta algunas entradas y crea algunas salidas, Ethereum simplemente almacena un saldo para cada dirección como un banco tradicional podría almacenar el saldo de cada número de cuenta

    - Ethereum tiene estructuras de datos para poder mantener los registros del libro mayor basado en cuentas: cada bloque contiene un resumen del estado actual (balance y contador de transacciones) para toda dirección y el estado (balance y almacenamiento) de todos los contratos

        - El árbol de almacenamiento de cada contrato asigna direcciones arbitrarias de 256 bits a palabras de 256 bits, lo que genera la friolera de $2^{256} \times 256 = 2^{264}$ bytes de almacenamiento. Por supuesto, nunca se podría llenar todo este almacenamiento, pero ese es el espacio teórico

        - El resumen facilita la prueba de que una dirección determinada tiene un saldo o estado de almacenamiento determinado. Por ejemplo, Alicia puede demostrarle a Bob cuál es su saldo sin que Bob tenga que escanear toda la cadena de bloques para verificar la prueba

    - El simple árbol binario Merkle utilizado en Bitcoin funcionaría para este propósito, ya que permite pruebas eficientes de inclusión (siempre que los mineros se aseguren de que ningún árbol incluya dos estados diferentes para la misma dirección). Pero también se quiere búsquedas rápidas y la capacidad de actualizar de manera eficiente el valor de una dirección

        - Para hacer esto, Ethereum utiliza una estructura de árbol un poco más complicada llamada árbol Patricia, también conocido como árbol de prefijos, trie o árbol de base: cada bloque de Ethereum incluye la raíz de un árbol Merkle Patricia (es decir, un árbol Patricia con *hash pointers*) que se compromete con el estado de cada dirección, incluidas las direcciones de contrato. El estado de cada contrato, a su vez, incluye un árbol que se compromete con el estado completo de su almacenamiento

    - Otro problema complicado con un libro mayor basado en cuentas es evitar ataques de repetición

        - En Bitcoin, dado que cada transacción consume sus entradas y salidas de transacciones no gastadas, la misma transacción firmada nunca puede ser válida dos veces. Con el diseño de Ethereum, uno debe asegurarse de que si Aliciaa firma una transacción que dice "paga 1 ether a Bob", Bob no pueda transmitir la transacción una y otra vez hasta que se agote la cuenta de Alicia

        - Para evitar esto, cada cuenta en Ethereum tiene un contador de transacciones que rastrea cuántas transacciones ha enviado. La declaración que Alicia realmente firma es: "Autorizo que mi enésima transacción sea un pago de 1 ether a Bob". Esta transacción no se puede reproducir porque después de procesarla, el contador de transacciones de Alice aumentará y será parte del estado global

Ethereum: ether, dapps y Web3

- Uno de los elementos fundamentales para entender Ethereum es su criptomoneda fundamental, el ether (ETH), de la cual se pueden discutir varios aspectos importantes

- Ether (ETH) es la criptomoneda nativa de la red Ethereum, y su objetivo es permitir un mercado para la computación, el cual proporcionará incentivos económicos a los participantes para verificar y ejecutar peticiones de transacciones y proporcionar recursos computacionales a la red

    - Cualquier participante que transmita una petición de transacción deben ofrecer también alguna cantidad de ETH a la red como recompensa. La red quemará parte de la recompensa y se da el resto a quien quiera que verifique la transacción, la ejecute, la confirme en la *blockchain* y la transmita a la red entera

    - La cantidad de ETH pagada corresponde a los recursos requeridos por hacer los cálculos. Estas recompensas también previenen a los participantes maliciosos de que atasquen la red al pedir la ejecución de infinitos cálculos u otros *scripts* intensivos en recursos, dado que los participantes tienen que pagar por esos recursos

    - Los ether también se usan para proporcionar seguridad económica a la red de tres maneras: se usa como medio para recompensar a los validadores; se apuestan por estos validadores, actuando como garantía contra el comportamiento deshonesto; si los validadores intentan comportarse mal, su ETH puede ser destruido; y se usa para ponderar votos para nuevos bloques propuestos, alimentándose en la elección de bifurcación del mecanismo de consenso

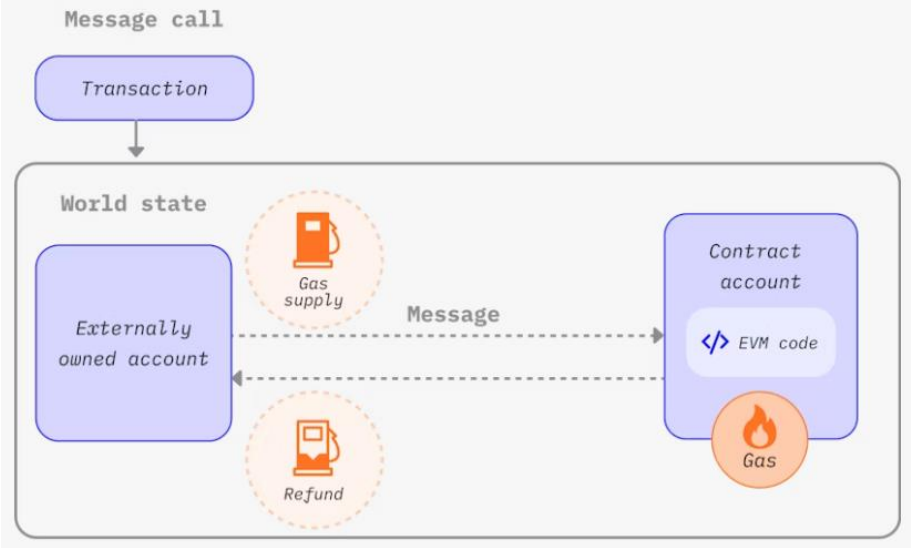
- Ether es la única forma aceptable de pago para comisiones por transacciones y se requiere para validar y proponer bloques en la *blockchain* principal

    - Los ether también se utilizan como colateral en los mercados de crédito descentralizados, como una unidad de cuenta en los mercados NFT, como pagos por la realización de servicios o productos en el mundo real, y otros

    - Esta criptomoneda permite crear aplicaciones descentralizadas, llamadas dapps, que comparten un grupo de potencia de computación común. Este grupo compartido es finito, de modo que Ethereum necesita un mecanismo para determinar quién lo usa

- A diferencia de Bitcoin, Ethereum sí que permite que existan bucles, por lo que una preocupación es que haya bucles infinitos que permitieran que algunos contratos se ejecuten para siempre por una variedad de

razones. Por lo tanto, se quiere ver si existe alguna manera de poder prevenir que haya contratos eternos y, más generalmente, si se pueden limitar contratos que tomen mucho tiempo en ejecutarse (aunque sea finito) porque los recursos de computación de la red son limitados



- El mecanismo que utiliza Ethereum usa un mecanismo llamado gas para conseguir este propósito: esencialmente, ejecutar una instrucción de la EVM cuesta una cantidad pequeña de dinero llamada gas

- Diferentes operaciones cuestan diferentes cantidades de dinero: operaciones básicas como la adición cuestan 1 gas, calcular un *hash* SHA-3 cuesta 20 gas, escribir una palabra de 256 bits para su almacenamiento persistente cuesta 100 gas y realizar una transacción cuesta 21000 gas. La lista completa de instrucciones disponibles y su precio es fijo, y cambiar esto requeriría una bifurcación fuerte

- El gas se paga por usar la moneda inherente de Ethereum, llamada “ether” (solo se llama gas cuando se usa para pagar por la ejecución del contrato). Toda transacción puede especificar el precio del gas, que es la cantidad de ether que se paga por cada unidad de gas consumida, y todo minero puede publicar transacciones a cualquier precio del gas que quieran y decidir su estructura de comisiones (es como una comisión por transacción de Bitcoin), reflejando la oferta y la demanda

- Toda llamada debe especificar con anterioridad cuanto gas se está dispuesto a gastar (el límite de gas). Si este valor se alcanza (uno se queda sin gas), la ejecución del contrato se detiene y todos los cambios en el estado del programa se deshacen (pero el minero se queda con el gas de cualquier manera)

- El requerimiento del gas significa que cálculos muy costosos no son adecuados para Ethereum, ya que el sistema no está diseñado para ser un servicio de computación en la nube (para realizar cálculos costosos que uno no puede hacer), sino que proporciona un servicio en el cual dos o más partes anónimas pueden confiar para que se comporten como especifica un contrato

- El *minting* o acuñación es el proceso por el cual se crea ether en el libro mayor de Ethereum. El protocolo de Ethereum subyacente crea el nuevo ether, y no es posible para un usuario crearlo por sí mismo

    - Ether se acuña como recompensa por cada bloque propuesto y en cada punto de control del *epoch* por otras actividades del validador relacionadas con el logro del consenso. La cantidad total emitida depende del número de validadores en y qué tanto ether han apostado

    - Esta emisión total se divide de manera equitativa entre los validadores en el caso ideal de que todos los validadores fueran honestos y estuvieran en línea, pero en realidad varía dependiendo del rendimiento de los validadores

    - Cerca de 1/8 de la emisión total va al que propone el bloque, y el resto va a los otros validadores. Los que proponen el bloque también reciben propinas de comisiones de transacciones y de ingresos relacionados con MEV, pero provienen de ethers reciclados y no de nueva emisión

- Igual que se puede acuñar ether, este también se puede destruir a través de un proceso de incineración o *burning*. Cuando el ether se incinera, entonces se elimina para siempre de la circulación

    - La quema de ethers ocurre en cada transacción de Ethereum: cuando los usuarios pagan por su transacción, la comisión de gas, fijada por la red acorde a la demanda de transacciones, se destruye

    - Cuando la demanda de la red es alta, los bloques pueden quemar más ether del que acuñan, efectivamente compensando la emisión

    - Quemar la comisión base reduce la habilidad del productore del bloque de manipular las transacciones. Por ejemplo, si los productores de bloque recibieran esta comisión, podrían incluir sus propias transacciones para obtener la comisión

- Debido a que el valor de varias transacciones en Ethereum es pequeño, ether tiene varias denominaciones que pueden referenciarse como unidades de cuenta más pequeñas. Las denominaciones de este tipo más importantes son el wei y el gwei

Denomination	Value in ether	Common Usage
Wei	10^{-18}	Technical implementations
Gwei	10^{-9}	Human-readable gas fees

  - El wei es la cantidad más pequeña posible de ether, mientras que el gwei o giga-wei se suele usar para describir los costes de gas en Ethereum

- Dos de las acciones más importantes que se hacen con ether, desde el punto de vista del usuario, es su transferencia y su consulta

  - Cada transacción de Ethereum contiene un campo llamado "valor", que especifica la cantidad de ether que se transfiere, denominado en wei, para enviársela de la dirección del emisor a la dirección del receptor
  - Los usuarios pueden consultar o *query* su balance de ether de cualquier cuenta a través de inspeccionar el campo de balance, que muestra cuánto ether (denominado en wei) se posee

  - Etherscan es una herramienta popular para comprobar los balances de diferentes direcciones

Ethereum: elementos fundamentales

- Una cuenta de Ethereum es una entidad con un balance de ETH que puede enviar transacciones en Ethereum, la cual puede ser controlada por un usuario o ser implementada como un contrato inteligente

  - Ethereum tiene dos tipos de cuentas: las cuentas de propiedad externa o *externally-owned accounts (EOA)* y las cuentas de contrato

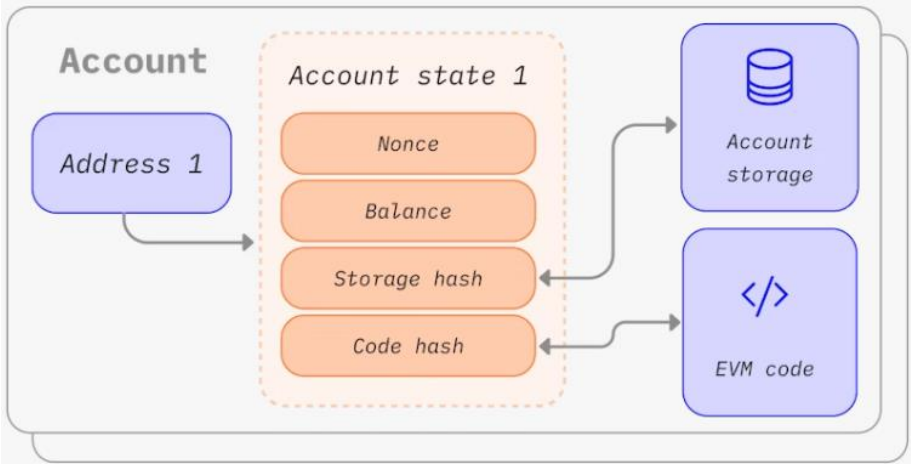
    - La cuenta de propiedad externa es una cuenta controlada por cualquier con llaves privadas, mientras que una cuenta de contrato es un contrato inteligente implementado en la red y controlada por código. Ambos tipos de cuenta permiten recibir, mantener y enviar ETH y tokens

    - Una cuenta de propiedad externa se crea sin coste alguno, puede iniciar transacciones, solo permite transacciones entre cuentas de

propiedad externa en ETH o tokens, y está hecha de un par de llaves criptográficas: una llave pública y otra privada

- Una cuenta de contrato tiene un coste porque se está utilizando el almacenamiento de la red, solo puede enviar transacciones en respuesta a recibir transacciones, las transacciones de una cuenta de propiedad externa a este tipo pueden hacer que se ejecuten varias acciones del código (como crear un nuevo contrato o transferir tokens) y no tienen llaves privadas (se controlan por la lógica del código del contrato inteligente)

- Una cuenta se compone de cuatro campos importantes: el *nonce*, el balance, el *codeHash* y el *storageRoot*



- El *nonce* es un contador que indica el número de transacciones enviadas desde una cuenta de propiedad externa o el número de contratos creados por una cuenta de contrato. Solo una transacción con un *nonce* dado se puede ejecutar para cada cuenta, protegiendo así de ataques de repetición donde las transacciones firmadas se transmiten repetidamente y se ejecutan otra vez

- El balance es el número de wei que tiene en propiedad la dirección, donde hay 10^{18} wei por cada ETH

- El *codeHash* es un *hash* que se refiere al código de una cuenta en la EVM, dado que las cuentas de contrato tienen fragmentos de código programados para poder realizar diferentes operaciones. Este código EVM se ejecuta si la cuenta recibe una llamada, y no puede cambiarse, a diferencia de los otros apartados que sí se pueden cambiar

- Todos esos fragmentos de código están contenidos en la base de datos del estado global bajo sus *hashes* correspondientes para

sacarlos otra vez en el futuro, llamados *codeHash*. Para cuentas de propiedad extern, el campo de *codeHash* es un *string* vacío

- El *storageRoot* es un *hash* de 256 bits del nodo raíz de un árbol de Merkle Patricia que codifica los contenidos de almacenamiento de la cuenta (un mapeado entre valores enteros de 256 bits), codificado en el árbol como un mapeado desde el *hash* de 256 bits Keccak de las llaves enteras de 256 bits de esta cuenta, y está vacío por defecto

- Una cuenta consiste en un par de llaves criptográficas públicas y privadas, las cuales ayudan a demostrar que una transacción se ha firmado por el emisor y evita falsificaciones

  - Una cuenta tiene fondos asociados, dado que en verdad uno nunca tiene criptomonedas. Uno tiene llaves privadas, porque los fondos siempre están en el libro mayor de Ethereum

  - Cuando se crea una cuenta, la mayoría de librerías generan una llave privada, que suele ser un *string* de 64 caracteres y puede ser encriptada como una contraseña. La llave pública se genera a partir de la privada, usando los últimos 20 bytes del *hash* Keccak-256 de la llave pública y añadiendo "0x" al principio

  - Es posible generar nuevas llaves públicas a partir de una misma llave privada, pero no al revés

  - Las cuentas de contratos tienen direcciones de 42 caracteres hexadecimales. La dirección del contrato normalmente se da cuando el contrato se implementa en la *blockchain* de Ethereum

  - Una cuenta no es una cartera: una cartera es una interfaz o aplicación que permite interactuar con la cuenta de Ethereum, ya sea una cuenta de propiedad externa o una cuenta de contratos

- También existen otro tipo de llave en Ethereum, que se introdujo cuando Ethereum pasó del *proof of work* al *proof of stake*: las llaves BLS

  - Estas llaves BLS sirven para identificar validadores. Estas oueden ser eficientemente agregadas para reducir la banda ancha requerida para que haya consenso en la red

  - Sin esta llave, la agregación del *stake* mínimo para el validador sería más grande

- Las transacciones son instrucciones firmadas criptográficamente por cuentas. Una cuenta iniciará una transacción para actualizar el estado global de la red de

Ethereum, siendo la transacción más simple aquella que pasa ETH de una cuenta a otra

- Una transacción de Ethereum se refiere a una acción iniciada por una cuenta de propiedad externa y no un contrato

  - Las transacciones, que cambian el estado global de la EVM, necesitan ser transmitidas a toda la red. Cualquier código puede transmitir una petición para que una transacción se ejecute en la EVM, y después de esto, un validador debe ejecutar la transacción y propagar el cambio resultante en el estado al resto de la red

  - Las transacciones requieren una comisión y deben de estar incluidos en un bloque validado

- Una transacción subida a la red incluye los siguientes apartados de información:

  - El apartado *from*, que es la dirección del emisor que a la vez firma la transacción. Esta será una cuenta de propiedad externa, ya que las cuentas de contratos no pueden enviar transacciones

  - …

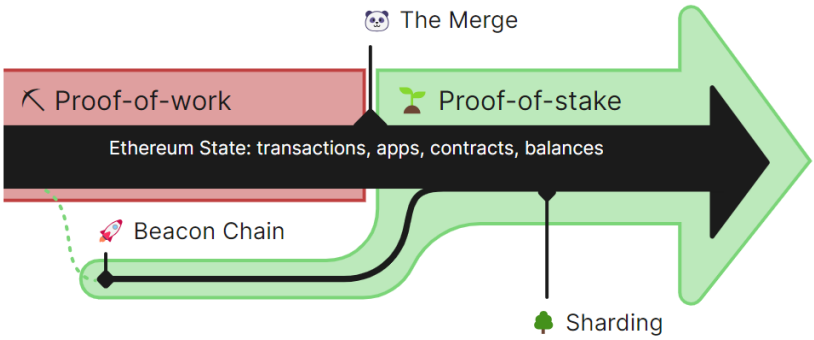
- Bloques

  - …

- Mecanismos de consenso

  - Ethereum utiliza un mecanismo de consenso basado en *proof of stake* o demostración de participación, que se basa en la idea de que cualquiera que quiera añadir nuevos bloques a la *blockchain* debe poseer ether como colateral y ejecutar un *software* de validación

    - Anteriormente, Ethereum tenía un mecanismo de *proof of work* en su *blockchain* principal llamada *mainnet*, pero se fusionó con una cadena llamada *Beacon Chain* para tener una cadena principal unificada que funcione con un mecanismo de *proof of stake*



- Estos validadores se pueden escoger aleatoriamente para proponer bloques que otros validadores revisan y añaden a la *blockchain* si están de acuerdo

- Hay un sistema de recompensas y penalizaciones para incentivar a los participantes a que sean honestos y a que estén disponibles *online* todo lo que puedan

Ethereum: MEV y oráculos

- El valor extraíble máximo o *máximum extractable value* (MEV) se refiere al máximo valor que se puede extraer de la producción de bloque en exceso a la recompensa estándar del bloque y las comisiones de gas al incluir, excluir o cambiar el orden de las transacciones en un bloque

    - En teoría, el MEV va enteramente para los validadores porque son la única parte que puede garantizar la ejecución de una oportunidad de MEV provechosa. Sin embargo, en la práctica, una gran porción de MEV se extrae por participantes independientes en la red llamados buscadores

        - Los buscadores o *searchers* ejecutan algoritmos complejos en datos de la *blockchain* para detectar oportunidades de MEV provechosas y tienen bots para subir automáticamente esas transacciones provechosas a la red

        - Los validadores toman una porción del MEV entero de cualquier manera porque los buscadores están dispuestos a pagar comisiones de gas altas (que van al validador) a cambio de una mayor probabilidad de inclusión de sus transacciones provechosas en el bloque. Asumiendo que los buscadores son económicamente racionales, la comisión de gas que un buscador querría pagar será un 100% del MEV del buscador (porque si el gas es mayor, el buscador perdería dinero)

        - Con ello, para algunas oportunidades de MEV altamente competitivas tales como el arbitraje DEX, los buscadores pueden

pagar hasta un 90% o más de su total de ingresos de MEV en comisiones de gas al validador debido a que mucha gente quiere ejecutar la misma oportunidad. Esto se da porque la única manera de garantizar que la transacción de arbitraje se ejecute es si la transacción tiene el precio de gas más alto

Los *market makers* automáticos: funciones constantes

- Los *market makers* automáticos o *automated market makers* (AMMs) son parte del ecosistema de finanzas descentralizadas que permite que activos digitales sean comerciados de una manera automática y sin necesidad de permisos, a través de fondos de liquidez y no de compradores y vendedores
    - Los AMMs usan fondos de liquidez, donde los usuarios pueden depositar criptomonedas para proporcionar liquidez. Estos fondos usan algoritmos para fijar los precios de los tokens basado en la *ratio* de activos en el fondo
        - Cuando un usuario quiere comerciar, se cambia un token por otro directamente desde el AMM, con los precios determinados por el algoritmo del fondo
        - Los AMMs proporcionan liquidez continua para un gran rango de activos, facilitando así el comercio de criptomonedas menos populares. Además, cualquiera puede proporcionar liquidez a la AMM y participar en el comercio de tokens por menores comisiones (comparado con bolsas tradicionales) y no utilizan ningún intermediario centralizado
        - Algunos de los AMMs más famosos son Uniswap, Sushiswap y Balancer
        - Los AMMs principalmente facilitan el comercio de criptomoneda a criptomoneda. Para poder comerciar con moneda fiat, los usuarios normalmente necesitan ir a una bolsa centralizada u otros servicios antes de interactuar con un AMM
    - Los AMMs son contratos inteligentes que permiten que activos digitales sean comerciados sin necesitar permiso y de manera automática, usando fondos de liquidez en vez de compradores y vendedores
        - En una plataforma de bolsa tradicional, los compradores y los vendedores ofrecen un activo a diferentes precios. Cuando otros usuarios encuentran que el precio cotizado es aceptable, entonces ejecutan una transacción y ese precio se vuelve el precio de mercado

- No obstante, los AMMs utilizan otro enfoque para comerciar con criptomonedas. Los AMMs son una herramienta financiera única de Ethereum y de las finanzas descentralizadas, de modo que nadie controla el sistema y cualquiera puede crear soluciones y participar

- Los comerciantes pueden comerciar contra ese contrato inteligente en vez de comerciar directamente con una contraparte como en las bolsas con *order books*

- Antes de que existieran los AMMs, la liquidez era un reto para las bolsas descentralizadas (DEX) en Ethereum: era una tecnología nueva con una interfaz complicada, de modo que el número de compradores y vendedores era bajo y costaba encontrar personas que comerciaran regularmente

    - En las plataformas AMM, en vez de comerciar entre compradores y vendedores, los usuarios comercian contra un fondo de tokens, un fondo de liquidez (un contrato inteligente). En su núcleo, un fondo de liquidez es un fondo compartido de tokens en donde los usuarios proporcionan los tokens y el precio de estos se determina por una fórmula matemática

    - Es a través de la modificación de la fórmula que los fondos de liquidez se pueden optimizar para diferentes propósitos

    - Cualquiera con conexión a internet y en posesión de cualquier tipo ERC-20 de tokens se puede volver un proveedor de liquidez a través de proporcionar tokens a un fondo de liquidez de un AMM. Los proveedores reciben normalmente una comisión por proporcionar tokens al fondo, y estas comisiones se pagan por los comerciantes que interactúan con el fondo

    - Actualmente, los proveedores de liquidez pueden ganar rendimientos en la forma de tokens de proyecto a través del *yield farming*, que consiste en apostar o bloquear criptomonedas dentro de un protocolo *blockchain* para generar recompensas tokenizadas

- Los AMMs se han convertido en una manera principal de comerciar con activos en un ecosistema de finanzas descentralizadas, y el elemento más importante es una fórmula matemática simple que puede tomar muchas formas

$$f\big(x(p), y(p)\big) = k$$

- En este caso, el balance de cada uno de los tokens $x(p)$ (para un token A) e $y(p)$ (para un token B) es una función de su precio (dado que solo se puede comerciar con estas monedas), y k es una constante

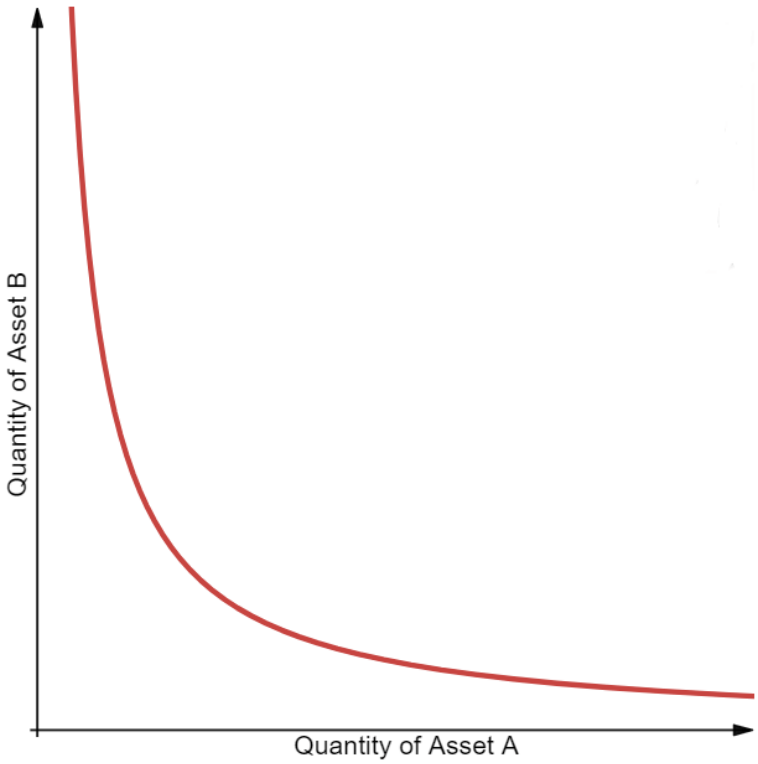
- La presencia de la constante, representada por k, significa que hay un balance constante de activos que determina el precio de los tokens en el fondo de liquidez. Por ejemplo, si un AMM tiene ETH y BTC, toda vez que se compre ETH, el precio de ETH sube porque ahora hay menos ETH en el fondo, mientras que el precio de BTC es mayor porque quedan más

- El fondo se queda en un balance constante a través de esta afectación de los precios, donde el valor total de ETH siempre será igual al valor total de BTC en el fondo. El fondo solo aumenta de tamaño cuando nuevos proveedores se unen

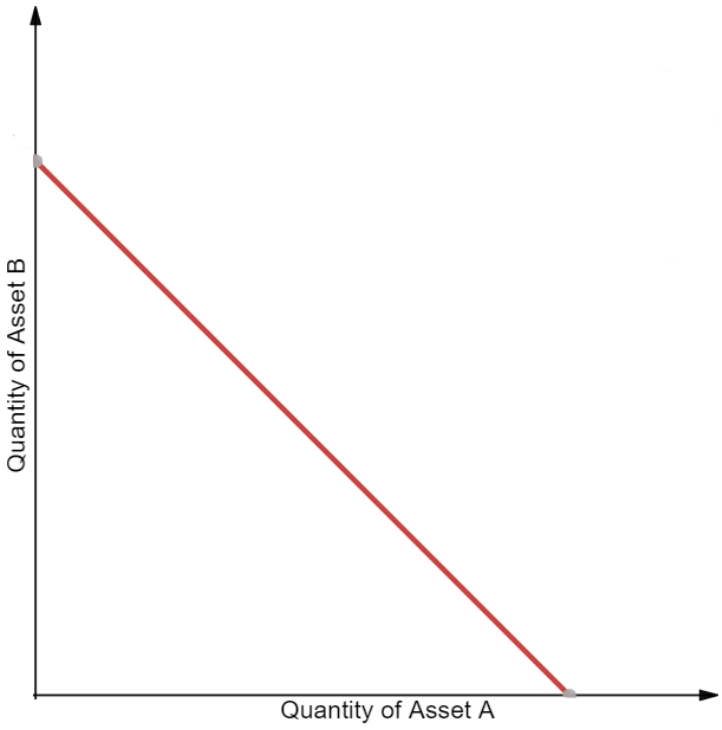
- No importa qué tan volátiles sean los precios, dado que eventualmente habrá un retorno al estado de balance que refleja de manera relativamente precisa el precio de mercado. De no ser así, el modelo incentiva a que los *traders* se aprovechen del arbitraje entre el AMM y las bolsas de criptomonedas, haciendo que se balancee todo otra vez

- La primera generación de AMMs que se hizo popular por protocolos como Bancor, Curve y Uniswap, son los *market makers* de función constante o *constant function market makers* (CFMM). Estos AMM se basan en una función constante en donde las reservas combinadas de activos se tienen que mantener constantes

    - El *market maker* de producto constante o *constant producto market maker* (CPMM) se basa en la función $xy = k$. Cuando se hace el gráfico, el resultado es una hipérbola en donde la liquidez siempre está disponible, pero a precios incrementales que tienden a infinito



- El *market maker* de suma constante o *constant sum market maker* (CSMM) es ideal para transacciones con un impacto en el precio nulo, pero no proporciona liquidez infinita. En este caso, la función es $x + y = k$, y crea una línea recta en donde, por diseño, se puede hacer que se acabe con las reservas de uno de los tokens si los precios no son iguales, y es por eso que no se usan en la práctica



        - El *market maker* de media constante o *constant mean market maker* (CMMM) permite la creación de AMMs con más de dos tokens y que se ponderen por fuera de la distribución estándar de 50/50. En este modelo se usa la función de la media geométrica $(x_1 x_2 \dots x_n)^{1/n} = k$, lo cual permite una exposición variable a diferentes activos en el fondo

    - Muchos de los AMMs de primera generación están limitados por la pérdida impermanente y la poca eficiencia del capital, que impacta tanto en los proveedores de liquidez como en los comerciantes

        - La pérdida impermanente es la diferencia de valor a lo largo del tiempo entre depositar tokens en un AMM comparado con tener estos tokens en cartera. Esta pérdida ocurre porque el precio de mercado de los tokens en la AMM diverge en cualquier dirección

        - Como los AMMs no pueden automáticamente ajustar los tipos de cambio, requieren que haya un comerciante que aproveche el arbitraje en estos AMM con tal de que los precios se vuelvan a igualar. Este beneficio que sacan se extrae de los bolsillos de los proveedores de liquidez, creando una pérdida

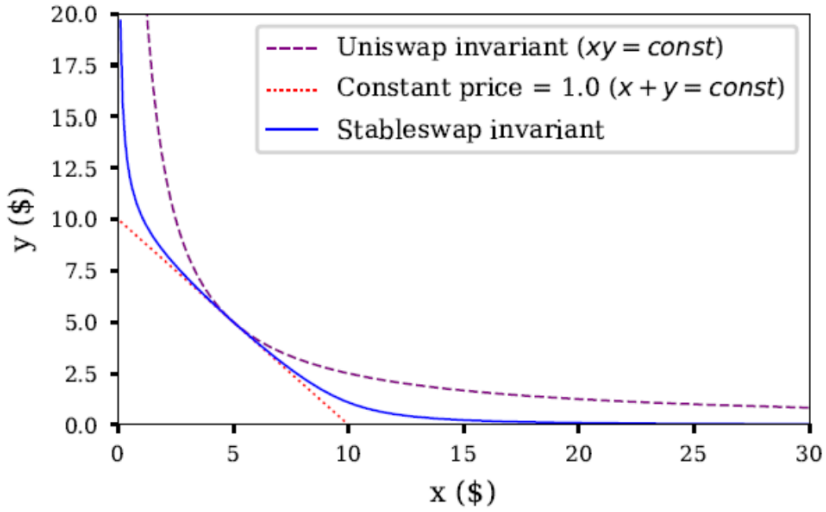
        - Los diseños tradicionales de AMM requieren grandes cantidades de liquidez para conseguir el mismo nivel de impacto de precios que una bolsa con un *order book*. Esto se debe a que una porción sustancial de la liquidez de una AMM está disponible solo cuando la curva de valoración comienza a ser exponencial (como con la hipérbola), haciendo que la mayoría de liquidez nunca se usa por comerciantes racionales (debido al impacto de precio extremo experimentado)

        - En esta situación, los proveedores de liquidez no tienen control sobre los precios que se ofrecen a los comerciantes, haciendo que no sea la mejor asignación de capital. Los *market makers* en *order books* pueden controlar exactamente los precios a los que quieren vender y comprar tokens, llevando así a una mayor eficiencia del capital pero que requiere participación activa y la monitorización de la provisión de liquidez

- Las limitaciones anteriores se están superando con proyectos innovadores con nuevos patrones de diseño, tales como *market makers* automatizados híbridos, dinámicos, proactivos o virtuales

    - Existen CFMMs híbridos avanzados que combinan múltiples funciones y parámetros para conseguir comportamientos específicos, tales como una

exposición al riesgo ajustada para lo proveedores de liquidez o un impacto en el precio reducido para los comerciantes



- Los AMMs, conocidos como *stableswap invariants*, combinan tanto CPMM con CSMM usando una fórmula avanzada para crear bolsillos de liquidez más densos que hacen que el impacto en el precio sea menor dentro de un rango de transacciones dado. El resultado es una hipérbola que se vuelve lineal para partes grandes de la curva y exponencial cuando se está por los límites de los ejes

- Los CFMMs híbridos permiten que haya un impacto de precios muy bajo al usar una curva que es en su mayoría lineal pero que se vuelve parabólica cuando el fondo se empuja a sus límites. Los proveedores de liquidez ganan más comisiones (aunque más bajas por transacción) porque el capital se usa más eficientemente, mientras que los comerciantes que realizan arbitraje también se pueden aprovechar de reequilibrar el fondo

- Esta curva ofrece intercambios con un impacto de precio muy bajo para tokens que ya tenían un tipo de cambio 1:1 relativamente estable. Esto significa que esta solución está predominantemente diseñada para Stablecoins, aunque también permite usar otros pares de tokens más volátiles con liquidez concentrada similar

- Los *market makers* dinámicos automatizados o *dynamic automated market makers* (DAMM) sonmodelos que permiten usar *data feeds* de precios de los tokens junto a su volatilidad implícita para ayudar a distribuir dinámicamente la liquidez a lo largo del precio de la curva

        - Incorporando múltiples variables dinámicas al algoritmo, es posible crear un *market maker* más robusto que se adapta a las condiciones de mercado cambiantes

        - Durante los periodos de baja volatilidad, uno puede concentrar la liquidez cerca del precio de mercado e incrementar la eficiencia del capital, y expandirse durante los periodos de volatilidad alta con tal de proteger a los comerciantes del deterioro

    - Un modelo de *market maker* proactivo o *proactive market maker* (PMM) imita los comportamientos de *market making* de los humanos de un *limit order book* central

        - Este protocolo utiliza precios de mercado precisos a través de un *data feed* para mover proactivamente la curva de precios de cada activo en respuesta a cambios de mercado, incrementando así la liquidez cerca del precio de mercado actual

        - Esto permite una manera de comerciar más eficiente y reduce las pérdidas de deterioro de los proveedores de liquidez

    - Un modelo de *market maker* automatizado virtual o *virtual automated market maker* (VAMM) permite minimizar el impacto en los precios, mitigar la pérdida impermanente y permite que haya exposición a un único token para los activos sintéticos

        - Estos utilizan la misma fórmula que un CPMM, pero en vez de confiar en el fondo de liquidez, los comerciantes depositan colateral en un contrato inteligente

        - Comerciando activos sintéticos y no el activo subyacente, uno puede ganar exposición a movimientos de precios de una gran variedad de criptoactivos de una manera eficiente. No obstante, aquellos que mantienen una posición abierta en un activo sintético tienen riesgo de que su colateral se liquide si el precio se mueve en su contra

Los *market makers* automáticos: Uniswap v2

- Uniswap v2 es una bolsa descentralizada basada en un sistema de contratos inteligentes de la *blockchain* de Ethereum y otras redes. Ahora se estudia el *trading* y el *spot price* de este tipo de fondo de liquidez

    - Esta está formada de fondos de liquidez que permiten el *market making* automatizado: permiten que los *traders* compren y vendan activos contra el protocolo sin la necesidad de una tercera parte

- Cada fondo de liquidez de Uniswap v2 consiste de reservas de dos tokens ERC-20 depositados por los proveedores de liquidez, los cuales se benefician de las comisiones que el protocolo carga a los *traders*

- Las comisiones que se recogen se comparten proporcionalmente entre todos los proveedores de liquidez

- Los fondos de liquidez de Uniswap v2 se basan en la fórmula del producto constante vista anteriormente para la cantidad de reservas de dos tokens en el fondo

$$xy = k$$

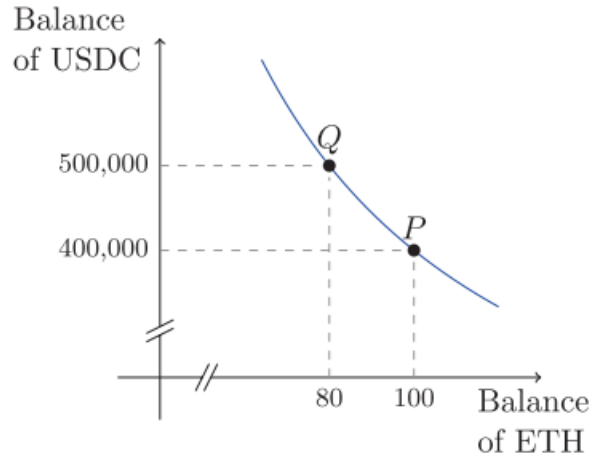
  - Esta fórmula juega un papel muy importante porque determina las cantidades de cada token en cualquier transacción y determina el precio de un token en términos del otro token (del fondo)

  - Además, el precio de un activo en un fondo de liquidez de Uniswap v2 sigue el precio del mercado actual debido a los arbitrajistas externos, los cuales detectarán inconsistencias en los precios y el precio interno del fondo y comerciarán con tal de aprovecharse del arbitraje. Esto, en consecuencia, deja los precios iguales al final

  - Al parámetro k se le conoce como el parámetro de liquidez del fondo, mientras que x e y son las cantidades del token A y el token B

- Asumiendo que el fondo de liquidez no tiene comisiones, se puede analizar la fórmula para obtener varias fórmulas importantes
  - Si un *trader* quiere comprar una cantidad a del token A, entonces se debe depositar una cantidad b del token B para mantener el producto constante al parámetro de liquidez. Por lo tanto, después de la transacción la fórmula sería la siguiente:

$$(x - a)(y + b) = k$$



$P = (100, 400000)$: Pool state before the trade.
$Q = (80, 500000)$: Pool state after the trade.

- Considerando la fórmula después de la transacción, se puede obtener la fórmula para a (la cantidad que se recibe si se pagan b tokens B) y para b (la cantidad que se tiene que pagar por a tokens A)

$$xy + xb - ay - ab = k \;\Rightarrow\; k + xb - ay - ab = k$$

$$\Rightarrow\; xb - ay - ab = 0 \;\Rightarrow \begin{cases} xb - (b+y)a = 0 \\ -ay - (a-x)b = 0 \end{cases}$$

$$\Rightarrow \begin{cases} a = \dfrac{xb}{y+b} \\ b = \dfrac{ya}{x-a} \end{cases}$$

- Un concepto fundamental de los AMMs es el *spot price*, que es el precio que se paga por token A que se recibe del AMM después de depositar una cantidad infinitesimal del token B

  - Si un *trader* deposita b tokens B y recibe una cantidad de a tokens A, entonces el precio que el *trader* paga por cada token A es b/a, que también se conoce como el precio efectivo que paga el *trader* por cada unidad del token A. Por lo tanto, el *spot price* se define de la siguiente manera:

$$p = \lim_{b \to 0} p_e(b) = \lim_{b \to 0} \frac{b}{a}$$

  - Aplicando las fórmulas vistas anteriormente, es posible ver que el el *spot price* con un estado (x, y) del fondo es igual a y/x, que coincide con la pendiente de la línea que pasa a través del origen y el punto (x, y)

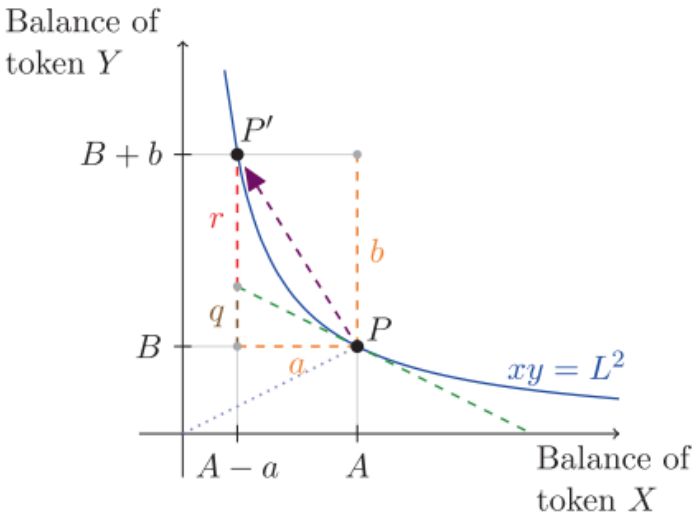
$$p_e(b) = \frac{b}{a} = \frac{b}{\frac{xb}{y+b}} = \frac{y}{x} + \frac{b}{x}$$

$$\Rightarrow\ p = \lim_{b\to 0} p_e(b) = \lim_{b\to 0}\left(\frac{y}{x} + \frac{b}{x}\right) = \frac{y}{x}$$

- Debido a que $b > 0$, entonces $p_e(b) > p$, por lo que el precio efectivo pagado por el *trader* es mayor al *spot price*. La diferencia entre ambos precios indica que los fondos con mucha liquidez tienen precios efectivos y *spot prices* más parecidos

$$p_e(b) - p = \frac{b}{x} > 0$$

- El *spot price* se puede interpretar como el precio que el fondo de liquidez ofrece a los *traders* en un momento particular, aunque estos tengan que pagar más que ese precio para ello



Segment from $(0, 0)$ to P. Its slope is $\frac{B}{A}$, that is, the spot price at P.

Portion of the tangent line to the curve defined by $xy = L^2$ at P. Its slope is $-\frac{B}{A}$, which is the opposite of the spot price at P.

b: Amount of token Y deposited.

a: Amount of token X received.

q: Amount of token Y that would have been paid if the effective price had been equal to the spot price.

r: Price impact (equal to $b - q$).

- La diferencia entre la cantidad b que el *trader* paga y la cantidad que hubiera pagado si hubiera comprado la cantidad a a un precio igual al *spot price* se conoce como impacto en el precio o *slippage*

$$Price\ impact = b - ap = a\left(\frac{b}{a} - p\right) = a(p_e(b) - p) = \frac{ab}{x}$$

- Si el estado de un fondo antes de la transacción es $P = (x, y)$ y después de esta es $P' = (x', y') = (x - a, y + b)$, entonces el precio efectivo es el siguiente:

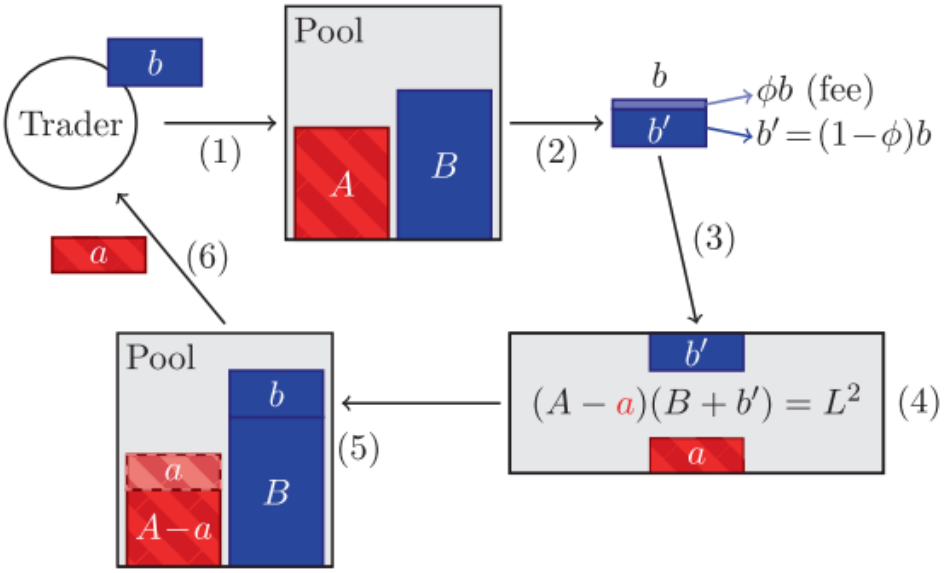
$$p_e = \frac{y' - y}{x - x'}$$

- Como el punto (x', y') pertenece a la curva $xy = k$, se puede entender y' como una función de x', por lo que p_e puede entenderse como una función de x' (donde x e y son fijos), y se puede obtener que el *spot price* es el opuesto a la pendiente de la tangente a la curva en el punto (x, y)

$$p = \lim_{x' \to x} p_e(x') = \lim_{x' \to x} \frac{y' - y}{x - x'} = -\lim_{x' \to x} \frac{y' - y}{x' - x}$$

- Las ecuaciones anteriores son válidas si no se tienen comisiones, pero en general si se tienen. Estas se tienen que añadir a las reservas del fondo y pagarse a los proveedores de liquidez de modo que estos pagos sean proporcionales

    - El protocolo de Uniswap v2 le cobra una comisión al *trader* sobre la cantidad que este deposite, y entonces la cantidad restante es la que se comercia



(1) The trader deposits an amount b of token Y into the pool.
(2) The AMM charges the fee on the amount b.
(3) The remaining amount $b' = (1 - \phi)b$ is actually traded.
(4) The AMM computes the amount a of token X that has to be given to the trader, following the curve described by $xy = L^2$ and using b' as the incoming amount of token Y.
(5) The balance of token X in the pool decreases by a, and the balance of token Y in the pool increases by b.
(6) An amount a of token X is given to the trader.

- Siendo x e y los balances y $\phi \in [0,1)$ la comisión, si un *trader* quiere depositar una cantidad b de tokens B, entonces la cantidad depositada efectiva sería $(1-\phi)b$ y se procedería como si no hubiera comisiones:

$$(x - a)(y + (1 - \phi)b) = k$$

$$\Rightarrow\ xy + (1 - \phi)bx - ay - (1 - \phi)ab = k$$

$$\Rightarrow\ k + (1 - \phi)bx - ay - (1 - \phi)ab = k$$

$$\Rightarrow\ (1 - \phi)bx - ay - (1 - \phi)ab = 0$$

$$\Rightarrow \begin{cases} (1 - \phi)bx - \big((1 - \phi)b + y\big)a = 0 \\ -ay - (a - x)(1 - \phi)b = 0 \end{cases}$$

$$\Rightarrow \begin{cases} a = \dfrac{(1 - \phi)bx}{y + (1 - \phi)b} \\ b = \dfrac{ya}{(1 - \phi)(x - a)} \end{cases}$$

- Es importante ver que después de cada transacción, el parámetro de liquidez k se tiene que actualizar, siendo cada vez un poco más grande en cada transacción siempre que la comisión sea positiva

$$x' = x - a \quad \& \quad y' = y + b$$

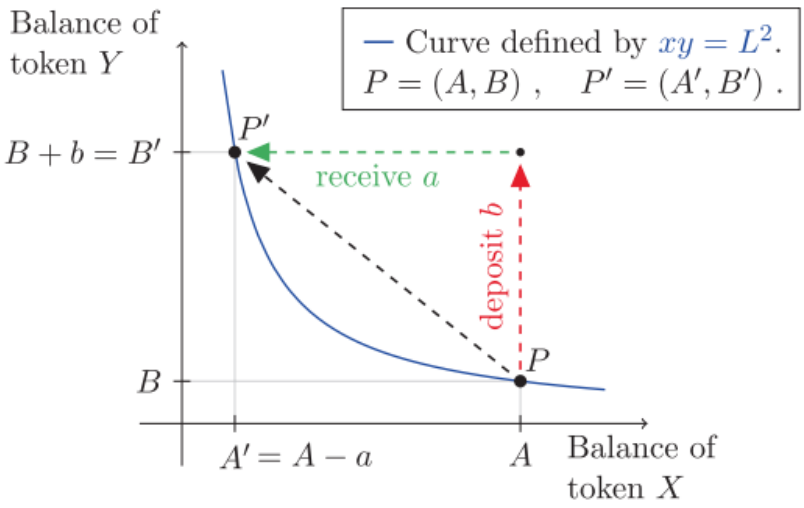
$$\Rightarrow \quad k' = (x-a)(y+b) = xy + bx - ay - ab = xy + (x-a)b - ay$$

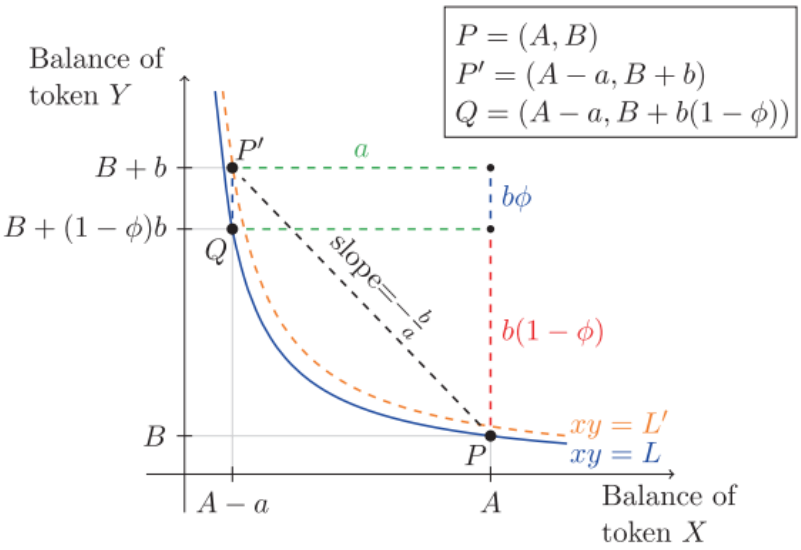
$$\Rightarrow \quad k' = k + (x-a)b - (x-a)(1-\phi)b = k + \phi b(x-a)$$

  - El nuevo *spot price* también cambiará después de cada transacción si la curva cambia, de modo que será el siguiente:

$$p' = \frac{y+b}{x-a}$$

  - En resumen, si el fondo de liquidez no tiene comisiones, su parámetro de liquidez se mantiene constante después de cada *trade* y los nuevos estados (puntos) se localizan en la misma curva. No obstante, si hay comisiones, entonces el parámetro k incrementa cada vez más y los estados se sitúan en curvas diferentes





- Cuando una transacción se ejecuta, los balances de los tokens en el fondo cambian, y como consecuencia de esta transacción, el precio también cambia. Por lo tanto, es interesante realizar estos análisis para mostrar cómo el *spot price* varía cuando las transacciones ocurren

    - Si se considera un fondo de x tokens A e y tokens B con una comisión de ϕ si un *trader* quisiera comprar a del token A, entonces se pueden obtener los siguientes cálculos:

        - Para obtener la cantidad de tokens B que se tienen que pagar, se puede usar la fórmula anteriormente vista para b y calcular el precio medio por unidad:

$$b_1 = \frac{ay}{(1-\phi)(x-a)} \Rightarrow Ave.Price_1 = \frac{b_1}{a}$$

        - Si el *trader* quisiera volver a comprar esa nueva cantidad, ahora se tendrían que usar los valores actualizados de $x' = x - a$ y $y' = y + b_1$, obteniendo el siguiente resultado:

$$b_2 = \frac{ay'}{(1-\phi)(x'-a)} = \frac{a(y+b_1)}{(1-\phi)(x-2a)} > b_1$$

$$\Rightarrow Ave.Price_2 = \frac{b_2}{a} > Ave.Price_1$$

    - Es posible extender el ejemplo anterior para estudiar qué ocurre si en vez de comprar a en una transacción se compra $a/2$ en dos transacciones

        - Si no hay comisiones, entonces se puede calcular el depósito necesario b_0 para una sola transacción y los depósitos b_1 y b_2 necesarios para realizar la transacción en dos transacciones. De

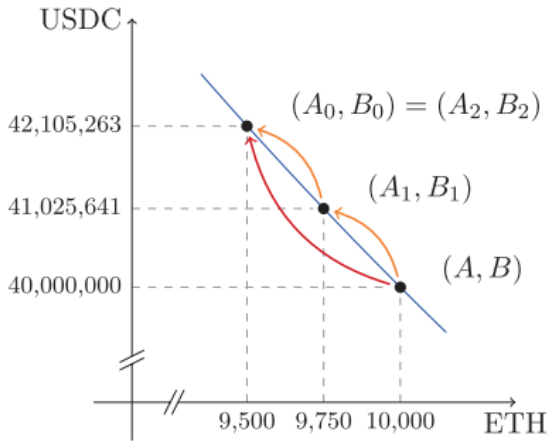
este modo, se puede ver como no hay diferencia entre ejecutar la estrategia en 1 o 2 transacciones

$$(for\ 1\ trans.) \qquad b_0 = \frac{(a_1 + a_2)y}{x - a_1 - a_2}$$

$$(for\ 2\ trans.) \qquad b_1 = \frac{a_1 y}{x - a_1} \quad \& \quad b_2 = \frac{a_2(y + b_1)}{x - a_1 - a_2}$$

$$\Rightarrow\ b_1 + b_2 = \frac{a_1 y}{x - a_1} + \frac{a_2(y + b_1)}{x - a_1 - a_2} = \frac{(a_1 + a_2)y}{x - a_1 - a_2} = b_0$$

- Como el fondo no tiene comisiones, entonces el parámetro k es constante y los estados del fondo se mantienen en la misma curva, haciendo que el punto que se obtiene haciendo una transacción es lo mismo que dividir la transacción en 2



(A, B): Initial pool state.
(A_0, B_0): Pool state after one trade of 500 ETH.
(A_1, B_1): Pool state after one trade of 250 ETH.
(A_2, B_2): Pool state after two consecutive trades of 250 ETH each.

- En cambio, en el caso de que haya comisiones, entonces dividir la transacción en 2 es menos provechoso que realizarla entera

$$(for\ 1\ trans.) \qquad b_0 = \frac{(a_1 + a_2)y}{(1 - \phi)(x - a_1 - a_2)}$$

$$(for\ 2\ trans.) \quad b_1 = \frac{a_1 y}{(1 - \phi)(x - a_1)} \quad \& \quad b_2 = \frac{a_2(y + b_1)}{(1 - \phi)(x - a_1 - a_2)}$$

$$\Rightarrow\ b_1 + b_2 = \frac{\phi a_1 a_2 y}{(1 - \phi)^2 (x - a_1)(x - a_1 - a_2)}$$

$$> \frac{(a_1 + a_2)y}{(1-\phi)(x - a_1 - a_2)} = b_0$$

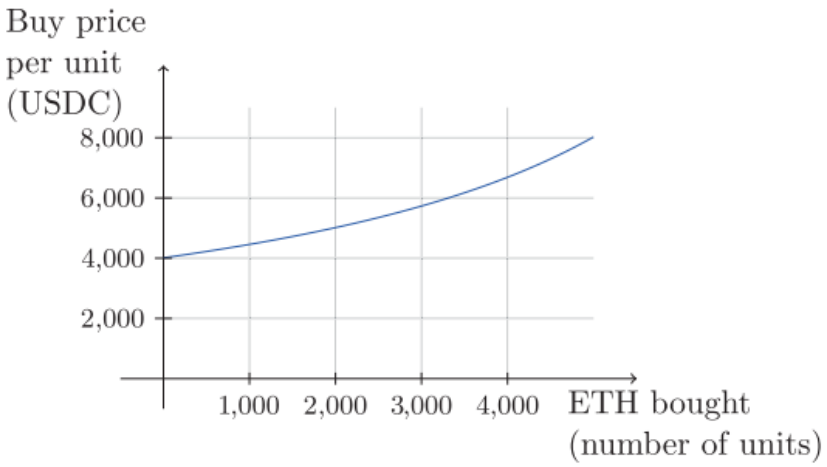
- En consecuencia, $b_0 \leq b_1 + b_2$ y la igualdad se mantiene si, y solo si, $\phi = 0$

- En la práctica, los *traders* normalmente compran tokens una sola vez y los tokens cuestan más con cada compra, por lo que el tamaño de la transacción impacta en el precio de compra y venta promedio

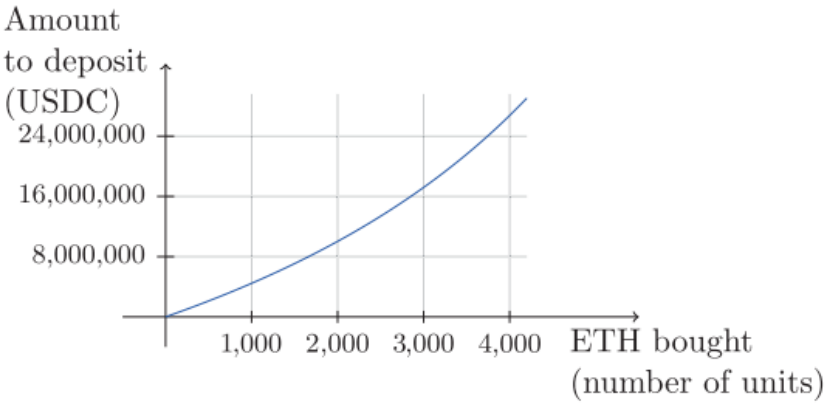
  - La cantidad del token B que se tiene que depositar para pagar cada token A, o el precio promedio del token B por cada token A, se puede expresar a partir de la fórmula para b:

$$b = \frac{ay}{(1-\phi)(x-a)} \Rightarrow \frac{b}{a} = \frac{y}{(1-\phi)(x-a)}$$

  - Como se puede ver, cuanto mayor es a (la cantidad que se quiere comprar), mayor es el precio por unidad b/a, mientras que la cantidad b de tokens B a depositar crece con a, por lo que ambas funciones son convexas. La función de b tiene una pendiente mayor que la de b/a debido a que a crece y divide a, haciendo que el efecto no sea tan pronunciado pero que sea creciente



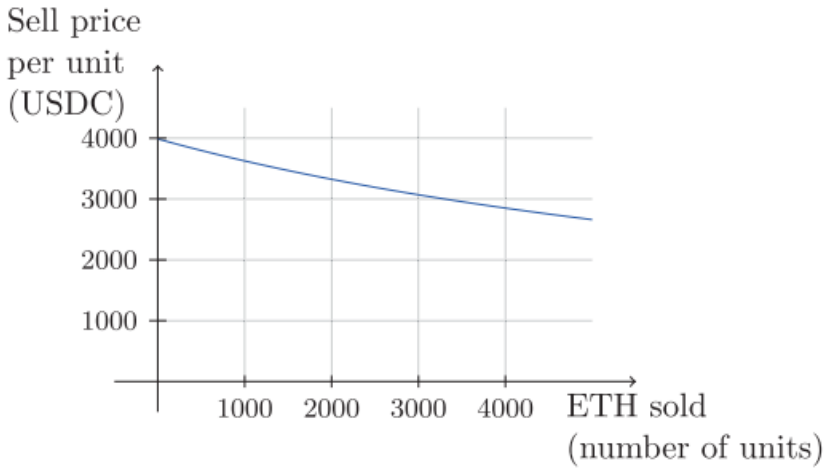
Total amount to deposit



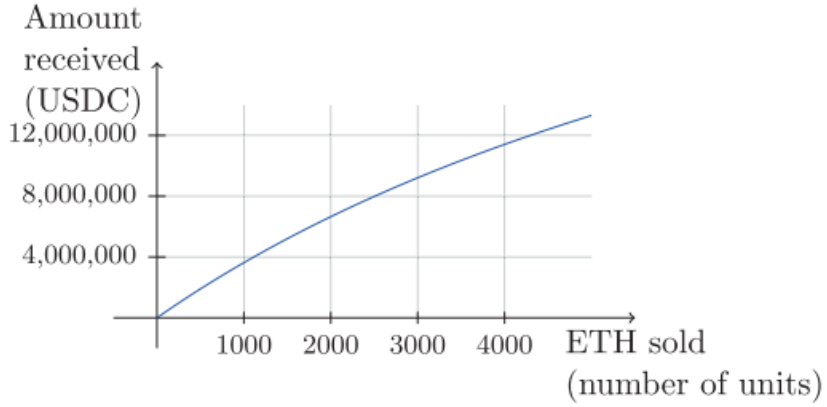
- La cantidad del token A que se tiene que recibir por cada token B pagado, o el precio promedio del token A por cada token B, se puede expresar a partir de la fórmula para a:

$$a = \frac{(1-\phi)bx}{y+(1-\phi)b} \Rightarrow \frac{a}{b} = \frac{(1-\phi)x}{y+(1-\phi)b}$$

- Como se puede ver, cuanto mayor es a (la cantidad que se quiere comprar), menor es el precio por unidad a/b, mientras que la cantidad b de tokens B a recibir crece con a, por lo que la primera función es convexa pero la última es cóncava. La función de b tiene una pendiente positiva y la de a/b es negativa debido a que b divide (relación negativa con a), pero la última tiene mayor magnitud porque en a/b, el movimiento de b se compensa entre el numerador y en el denominador



Amount of USDC received



- Igual que con los precios de venta y de compra, el impacto del tamaño de la transacción también se puede cuantificar para la *ratio* del crecimiento del precio efectivo que paga el *trader*

  - En este caso, el *spot price* actual sería de $p = y/x$, y se quiere calcular el crecimiento el *spot price* cuando se quiere comprar una cantidad a. A partir de las ecuaciones anteriormente vistas, es posible derivar el nuevo *spot price*:

$$x' = x - a \qquad y' = y + b = y + \frac{ay}{(1-\phi)(x-a)}$$

$$\Rightarrow\ p' = \frac{y'}{x'} = \frac{y + \frac{ay}{(1-\phi)(x-a)}}{x-a}$$

  - Siendo $r = a/x$ la proporción del token A que el *trader* está comprando (con respecto al balance total en el fondo), es posible obtener una expresión de p' en función de p y r:

$$p' = \frac{y + \frac{ay}{(1-\phi)(x-a)}}{x-a} =$$

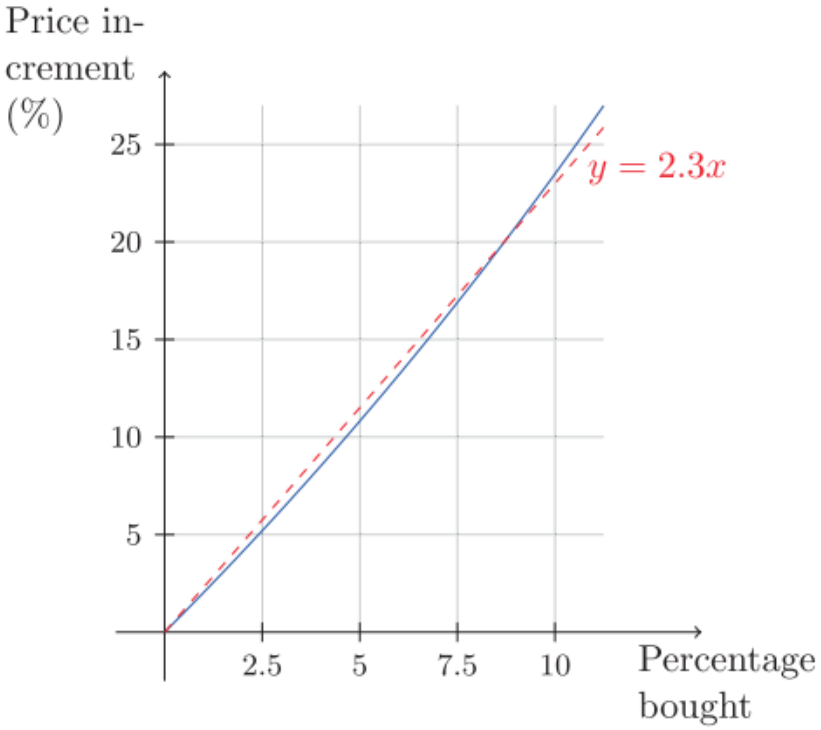
$$= \frac{y/x + \dfrac{ay/x}{(1-\phi)(x/x - a/x)}}{x/x - a/x} = \frac{p + \dfrac{p}{(1-\phi)(r^{-1}-1)}}{1-r} =$$

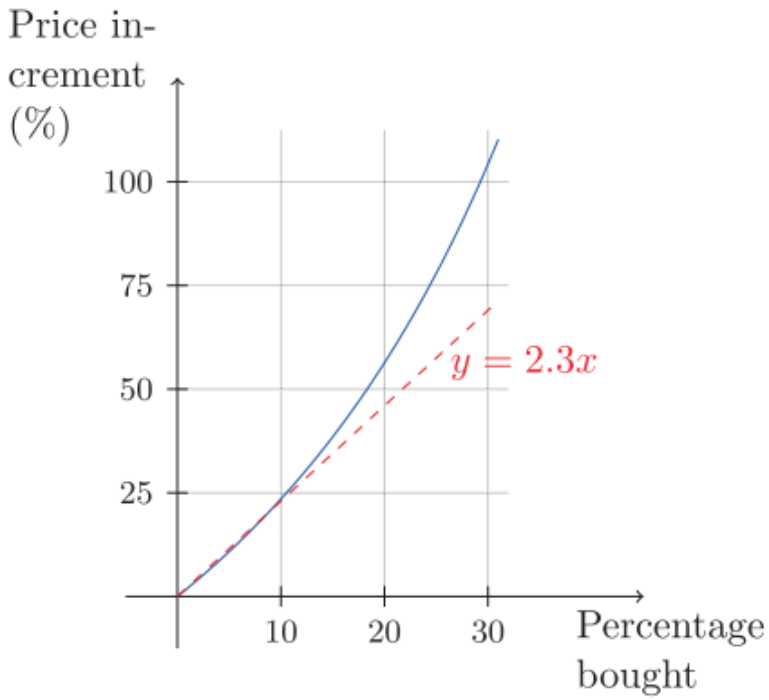
$$= p\frac{1 + (r^{-1}-1)(1-\phi)}{(1-r)(r^{-1}-1)(1-\phi)}$$

- Por lo tanto, la *ratio* de crecimiento del precio es la siguiente, la cual no depende del estado del fondo:

$$\frac{p'}{p} = \frac{1 + (r^{-1}-1)(1-\phi)}{(1-r)(r^{-1}-1)(1-\phi)}$$

- Este crecimiento se puede graficar como una función lineal para porcentajes relativamente pequeños, pero la aproximación lineal falla cuanto uno más se aleja, dado que comienza a tener un comportamiento exponencial





- Ahora que se ha visto el impacto en el precio de las transacciones, uno puede estudiar cómo se añade liquidez a un fondo de liquidez Uniswap v2 y cómo se puede sacar esa liquidez

    - Cuando un proveedor quiere proporcionar liquidez a un fondo, necesita depositar una cantidad de tokens A y B en una proporción concreta determinada por el estado del fondo

        - Por realizar eso obtendrán tokens LP o *liquidity pool tokens*, que representan una parte del fondo del que son propietarios. El proveedor de liquidez también ganará comisiones de *trading* acorde a esta proporción, y esta proporción variará en el tiempo (cuando entren o salgan proveedores del fondo)

        - Si un proveedor de liquidez quiere proporcionar liquidez al fondo, entonces tiene que depositar una cantidad a de tokens A y b de tokens B que satisfagan que la proporción es igual a la proporción de reservas que hay en el fondo

$$\frac{b}{A} = \frac{x}{y}$$

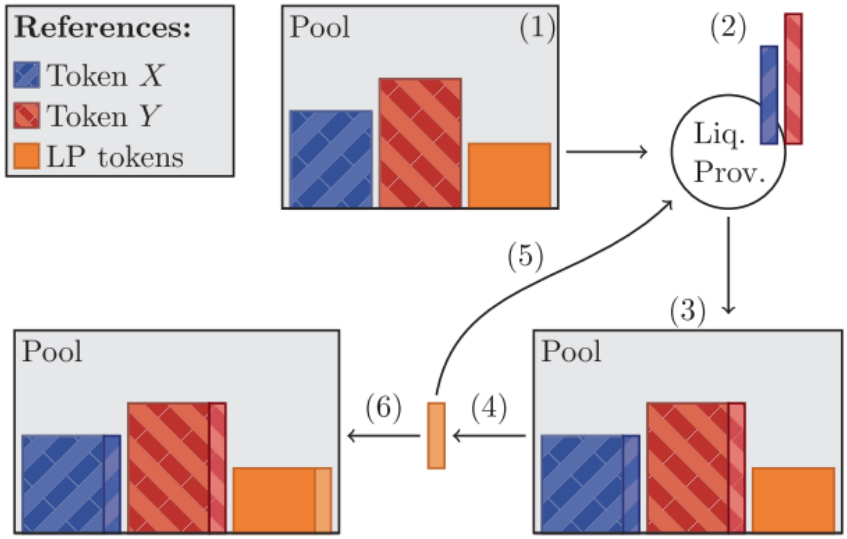
        - Por lo tanto, el *spot price* después del depósito es el mismo que antes debido a la equivalencia anterior

$$p = \frac{y}{x} \quad \Rightarrow \quad p' = \frac{y+b}{x+a} = \frac{y}{x}$$

- Cuando un proveedor proporciona liquidez a un fondo, se le envía tokens LP, los cuales actúan como recibo de la provisión por parte del proveedor de liquidez y representan una porción del fondo

    - La cantidad de tokens LP dentro del fondo es dinámica. El AMM sigue cuantos tokens LP se han creado y cuántos se han dado en retorno por la liquidez

    - Cuando un nuevo usuario proporciona liquidez, el AMM acuña una cantidad apropiada de tokens LP que se envían al nuevo proveedor y representa la porción de la cual es propietario



(1) The pool has a certain amount of reserves and tracks the amount of LP tokens in circulation.
(2) The liquidity provider deposits certain amounts of tokens X and Y into the pool. These amounts must be proportional to the balances of tokens X and Y in the pool.
(3) The tokens deposited are added to the pool reserves.
(4) The AMM mints an amount of LP tokens, which is proportional to the amount of tokens deposited.
(5) The newly minted LP tokens are given to the liquidity provider.
(6) The amount of LP tokens in circulation is updated.

    - Concretamente, si M es la cantidad de tokens LP existentes y un *trader* quiere proporcionar una liquidez de a tokens A y b tokens B, entonces se puede encontrar obtener la cantidad que recibiría el *trader* es qM y que se actualizaría a $(1+q)M$ después de la provisión de liquidez

$$\frac{b}{a} = \frac{y}{x} \Rightarrow \frac{b}{y} = \frac{a}{x}$$

$$if\ \ q = \frac{a}{x} = \frac{b}{y}\ \ \Rightarrow\ \ M + qM = (1+q)M$$

- Los AMM Uniswap v2 no requieren que el proveedor encuentre las cantidades exactas de depósito. En verdad, dadas las cantidades C_x de tokens A y C_y de token B que el proveedor tiene disponibles para depositar, el protocolo calcula las cantidades máximas de cada token que se pueden depositar para mantener la proporción del fondo igual, devolviendo los tokens restantes y dando los tokens LP correspondientes

  - Esta característica es muy útil debido a que el estado del fondo cambia constantemente y no es predecible, debido a que un AMM es descentralizado. Si un proveedor de liquidez intenta calcular en un cierto momento las cantidades que se necesitan depositar, cuando se procese la transacción, el estado podría ser diferente (se han procesado otras transacciones primero) y la proporción necesaria también

  - En el caso en que $C_y/C_x = y/x$, las cantidades y los tokens LP que se reciben son los siguientes:

$$\frac{C_y}{C_x} = \frac{y}{x}\ \Rightarrow\ \ q = \frac{C_x}{x}\ \ \Rightarrow\ qM = \frac{C_x}{x}M = \frac{C_y}{y}M$$

  - En el caso en que $\frac{C_y}{C_x} > y/x$, las cantidades y los tokens LP que se reciben son los siguientes, teniendo en cuenta un diferencial Δ_y para obtener las proporciones adecuadas:

$$\frac{C_y - \Delta_y}{C_x} = \frac{y}{x}\ \ \Rightarrow\ \ \Delta_y = C_y - \frac{yC_x}{x}\ \ \Rightarrow\ \ q = \frac{C_x}{x} = \frac{C_y - \Delta_y}{y}$$

$$\Rightarrow\ qM = \frac{C_x}{x}M = \frac{C_y - \Delta_y}{y}M < \frac{C_y}{y}M$$

  - En el caso en que $\frac{C_y}{C_x} > y/x$, las cantidades y los tokens LP que se reciben son los siguientes, teniendo en cuenta un diferencial Δ_x para obtener las proporciones adecuadas:

$$\frac{C_y}{C_x - \Delta_x} = \frac{y}{x}\ \ \Rightarrow\ \ \Delta_x = C_x - \frac{xC_y}{y}\ \ \Rightarrow\ \ q = \frac{C_x - \Delta_x}{x} = \frac{C_y}{y}$$

$$\Rightarrow\ qM = \frac{C_y}{y}M = \frac{C_x - \Delta_x}{x}M < \frac{C_x}{x}M$$

  - Por lo tanto, el proveedor de liquidez recibirá una cantidad de tokens LP igual al mínimo entre ambas cantidades vistas:

$$\min\left\{\frac{C_x}{x}M, \frac{C_y}{y}M\right\}$$

- Finalmente, una de las características más útiles para los *traders* que interactúan con AMMs son los agregadores DEX

    - Un agregador DEX es un servicio basado en *blockchain* que funciona como un explorador de precios y liquidez ofrecido por diferentes bolsas descentralizadas o DEXs y ayuda a los *traders* a encontrar el mejor precio para sus transacciones

        - Además, los agregadores están equipados con un algoritmo que permite a los usuarios dividir la transacción en varias contra diferentes AMMs, de modo que se puede obtener el mejor precio posible para cada transacción

        - Claramente, este servicio es prometedor, por lo que vale la pena entender la necesidad de dividir una transacción en dos o más partes

    - Suponiendo que se quiere comerciar una cantidad de T en dos fondos de liquidez diferentes para los mismos tokens A y B, uno quiere encontrar la cantidad de B que se debería pagar en cada fondo para recibir la máxima cantidad de A posible

        - Para $j \in \{1,2\}$, las reservas del fondo j son x_j e y_j, y las comisiones son ϕ_1 y ϕ_2 para los fondos 1 y 2, respectivamente. Si se comercia una cantidad de b en el primer fondo y una cantidad de $T - b$ (la cantidad restante) en el otro, entonces se pueden obtener las siguientes cantidades a_j de tokens que se obtendría en cada fondo:

$$a_1 = \frac{(1-\phi_1)by_1}{x_1 + (1-\phi_1)b} \qquad a_2 = \frac{(1-\phi_2)(T-b)y_2}{x_2 + (1-\phi_2)(T-b)}$$

        - Con tal de maximizar la cantidad de a que se recibe cuando se comercia con T tokens B, entonces se tiene que encontrar el máximo de la suma de ambas cantidades, que es una función $f: [0, T] \to \mathbb{R}$. Maximizando la función, se obtienen los siguientes resultados:

$$\max_b f(b) = \max_b \frac{(1-\phi_1)by_1}{x_1 + (1-\phi_1)b} + \frac{(1-\phi_2)(T-b)y_2}{x_2 + (1-\phi_2)(T-b)}$$

$$f'(b) = \frac{x_1 y_1 \overline{\phi_j}[x_2 + \overline{\phi_2}(T-b)]^2 - x_2 y_2 [x_1 + \overline{\phi_1} b]^2}{[x_1 + \overline{\phi_1} b]^2 [x_2 + \overline{\phi_2}(T-b)]^2} = 0$$

$$where \quad \overline{\phi_j} = 1 - \phi_j \quad for \ \ j \in \{1,2\}$$

- El numerador es un polinomio de grado 2, y como $f'(b) = 0$ solo si este numerador es nulo para $b \in [0, T]$, entonces se puede resolver esta ecuación de segundo grado con la fórmula cuadrática. Las raíces b_1 y b_2 permitirán obtener la cantidad de tokens B que se debe asignar a cada uno de los fondos para obtener la máxima cantidad de tokens A

- Como se puede ver, es posible optimizar las cantidades que se quieren, pero es necesario tener toda la información sobre el estado de los dos fondos y realizar los cálculos de la optimización. Los agregadores hacen todo esto por el usuario, por lo que son valiosos