

Prueba de Solvencia

Procedimiento de Prueba de Solvencia de criptoactivos utilizando Merkle Sum Tree

Marcelo Cavazzoli¹, Federico Temari¹, Santiago Galiñanes¹, Agostina Rey¹

¹ Lemon Cash, Ciudad de Buenos Aires, Argentina

1. Introducción

En un CEX (*centralized exchange*) los usuarios depositan criptomonedas en la plataforma y ceden la custodia de los mismos a la empresa que los recibe, confiando en que podrán hacer uso de ellos cuando lo deseen. Existen mecanismos para que el usuario no deba recurrir a la confianza sobre la integridad de sus fondos, sino que pueda comprobar por cuenta propia que, efectivamente, la empresa a la que le cede la custodia de sus fondos es solvente y puede responder ante todos sus usuarios en caso de deseen retirar sus fondos de la misma, incluso si lo hicieran al mismo tiempo.

La solvencia de una empresa está definida por la condición de que sus activos superen a sus pasivos: mediante la comparación de las reservas disponibles con las deudas contraídas con todos los usuarios, es que se puede calcular el estado de solvencia de la empresa en un momento dado. Es decir, la Prueba de Solvencia contiene dos elementos centrales: la Prueba de Reservas y la Prueba de Pasivos.

En el presente artículo se exponen mecanismos para generar cada uno de estos componentes de una manera transparente y verificable para construir a la seguridad de los usuarios respecto de sus fondos. Estos procedimientos están basados en la tecnología y no dependen de ningún intermediario, ni son adulterables. A su vez, se comparte una herramienta de código abierto para que sea utilizado por cualquier CEX.

La intención de estas propuestas radica en aportar a la transparencia en el ecosistema de manera tal de fomentar una industria más segura para los usuarios y acompañar al cambio de paradigma, en el que la transparencia es central. Como se presenta a continuación, esto es alcanzable a través de los pilares fundamentales de los criptoactivos.

2. Ratio de colateralización

El ratio de colateralización está dado por la relación entre la Prueba de Reservas y la Prueba de Pasivos. Este valor expresado de manera porcentual indica cuánto de aquella deuda contraída con los usuarios se encuentra en las billeteras de criptomonedas custodiadas por la empresa. Este indicador sirve para poder determinar la solvencia de la empresa y el poder de devolver a los usuarios todos los activos que fueron depositados.

Los valores posibles son:

- Ratio de colateralización $> 100\%$: Posición sobrecolateralizada en la cual la empresa cuenta con mayor cantidad de activos en sus reservas que deuda tomada con el usuario.
- Ratio de colateralización $= 100\%$: Posición completa donde la empresa cuenta con la misma cantidad de activos que de pasivos.
- Ratio de colateralización $< 100\%$: Posición sub colateralizada en la que la empresa cuenta con menor cantidad de activos disponibles bajo su custodia que las deudas contraídas con los usuarios.

3. Prueba de Reservas

También conocida como prueba de reservas o prueba de fondos, está dada por todas las criptomonedas que se encuentran bajo la custodia de la empresa. La misma está compuesta por todas aquellas “Billeteras de criptomonedas” bajo el control de la empresa. Es posible utilizar diferentes herramientas para realizar la gestión y control de cada una de ellas.

La información pertinente y pública que deberá ser otorgada a los usuarios son su dirección pública y el saldo de la misma, incluyendo la cantidad y el tipo de criptomoneda. De esta manera, es posible contrastar esta información con el “explorador” de cada una de las cadenas de bloques según corresponda.

3.1. Método de custodia de reservas

Existen diferentes métodos de custodia que se utilizan para resguardar las criptomonedas. Para el siguiente documento se describirán dos de las más conocidas y utilizadas.

- Billeteras de criptomonedas - Con custodia de clave privada: Este método contempla que la empresa en cuestión almacena las claves privadas de sus billeteras de manera segura y las utiliza para realizar transacciones.
- Servicio de terceros - Sin custodia de clave privada: Este tipo de almacenamiento se realiza utilizando un servicio de una empresa tercera, la misma es la encargada de resguardar las claves privadas y permite gestionar los fondos de la misma.

4. Prueba de Pasivos

Los pasivos de la empresa están representados por todas las criptomonedas ingresadas por los usuarios a la plataforma. El mismo es el encargado de la custodia en nombre del usuario, por lo cual, contrae una deuda para con el usuario por el monto ingresado. A su vez, el usuario tiene el control total de sus fondos pudiendo gestionarlos a través de la plataforma.

Por consecuencia, la sumatoria de todas las deudas contraídas con cada uno de los usuarios es el pasivo total de la empresa. Dicha sumatoria se expresa en valores nominales para cada uno de las diferentes criptomonedas dentro de la plataforma. Cabe destacar que la deuda solamente especifica cada una de las criptomonedas pero no en qué cadena de bloques se encuentran las mismas.

5. Árbol de Merkle como mecanismo de prueba

Dicho mecanismo permitirá afrontar el desafío de poder hacer público todos los pasivos de la empresa en cuestión sin sacrificar la privacidad de todos sus usuarios ya que da la posibilidad de que los mismos puedan verificar de manera fehaciente que su balance dentro de la plataforma realmente esté incluido dentro del saldo total de la empresa. A su vez, este mecanismo permite validar mediante la criptografía toda la información presentada sin necesidad de utilizar un ente regulador.

5.1. Árbol de Merkle

El Árbol de Merkle es un tipo de estructura de datos que se utiliza para almacenar información de manera eficiente y que permite la fácil verificación de la integridad de los datos. Se llama “árbol” porque está estructurado como uno, con una serie de “nodos” que se ramifican desde un nodo “raíz” central. Cada nodo en un árbol de Merkle contiene un valor obtenido a partir de una “Función de resumen” llamado de aquí en adelante *hash*, y cada nodo padre contiene un *hash* que es la concatenación en orden de izquierda a derecha de los *hashes* de sus nodos “hijos”. Se puede observar un diagrama en la figura 1

El árbol contiene diferentes niveles comenzando en el nivel 0 en el cual se encuentra el *hash* padre y él mismo está compuesto por la sumatoria de los *hash* de los nodos 1 y 2 que se encuentran

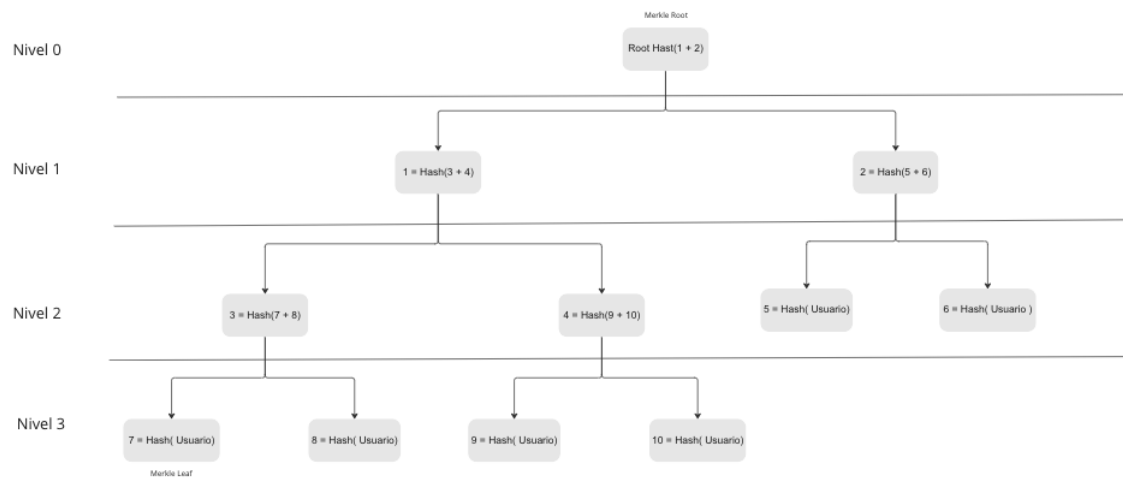


Figure 1. Diagrama de Árbol de Merkle

en el nivel 1. Esto se repite de manera sucesiva hacia los niveles 2 y 3. Por último, los nodos del último nivel representan cada uno a un usuario.

Existen varios conceptos importantes a la hora de generar un Árbol de Merkle, entre ellos:

- *Hash* padre: Este es el *hash* que se encuentra en el nodo padre de todo el árbol que se encuentra en el nivel 0 del mismo
- *Hash* de hoja: Este es uno de los *hash* que se encuentra en el último nivel del árbol, el mismo representa a un usuario en particular.
- Prueba de Merkle: Este es la conjunción de nodos que se requiere para obtener el *hash* padre partiendo de un *hash* de hoja.

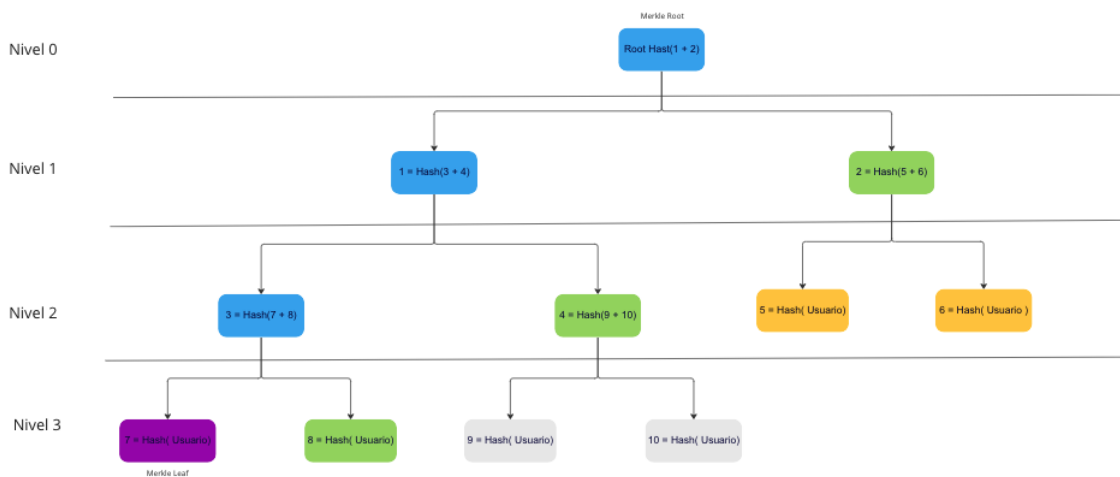


Figure 2. Diagrama de Árbol de Merkle con Validación de Merkle

En la figura 2 partiendo desde el nodo número 7 de color violeta, la prueba de Merkle está representada por todos los nodos que se encuentran en color verde los cuales son necesarios para alcanzar el nodo padre.

5.2. Árbol de Merkle de Sumatoria

El Árbol de Merkle brinda una estructura de datos con características tales que permiten utilizarla como mecanismo de prueba. Sin embargo, para realizar la aplicación del mismo es necesario agregar el balance de cada usuario dentro de la estructura a construir.

Es por ello que para ofrecer dicha estructura como mecanismo de prueba para cada usuario, es necesario realizar un Árbol Merkle de Sumatoria que mantendrá las propiedades de un Árbol de Merkle presentadas en la sección anterior, pero con dos diferencias:

- Cada nodo, además de almacenar el *hash*, almacenará los balances de las criptomonedas de un usuario.
- El algoritmo para crear un nodo padre a partir de dos nodos hijos también incluirá sus balances, tanto en la creación del *hash*, como la sumatoria de balances del nodo a construir.

De esta manera, la nueva estructura de datos cumple los requisitos para ser utilizada como mecanismo de pruebas manteniendo la privacidad de la identidad de cada usuario, así como también de sus balances.

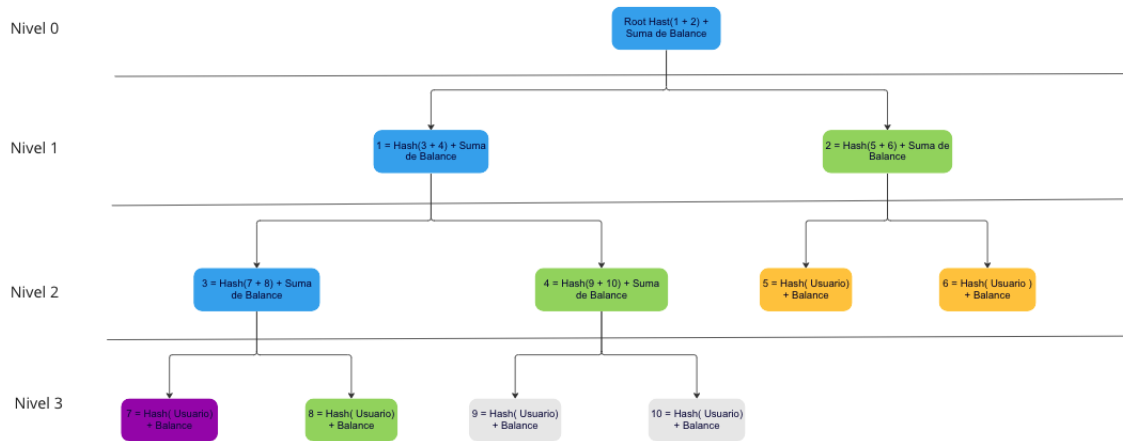


Figure 3. Diagrama de Árbol de Merkle de Sumatoria con balances

5.3. Construcción del Árbol de Merkle de Sumatoria

La construcción del Árbol de Merkle de Sumatoria comienza con la información de cada uno de los usuarios por separado, en este caso el balance y un código identificador único del mismo. El objetivo principal es partir del balance individual de cada uno de los usuarios, representado en nominales de cada una de las criptomonedas, para alcanzar una sumatoria total nominal de cada una de ellas.

A continuación se presenta un ejemplo de cómo podría ser generado un Árbol de Merkle desde la información de varios usuarios, la cual está detallada en la siguiente tabla.

Usuario ID	Balance	Criptomonedas
Usuario 1	1.5	BTC
Usuario 2	3	BTC
Usuario 3	5	BTC

Table 1. Ejemplo de listado de usuarios con sus respectivos identificadores y balances

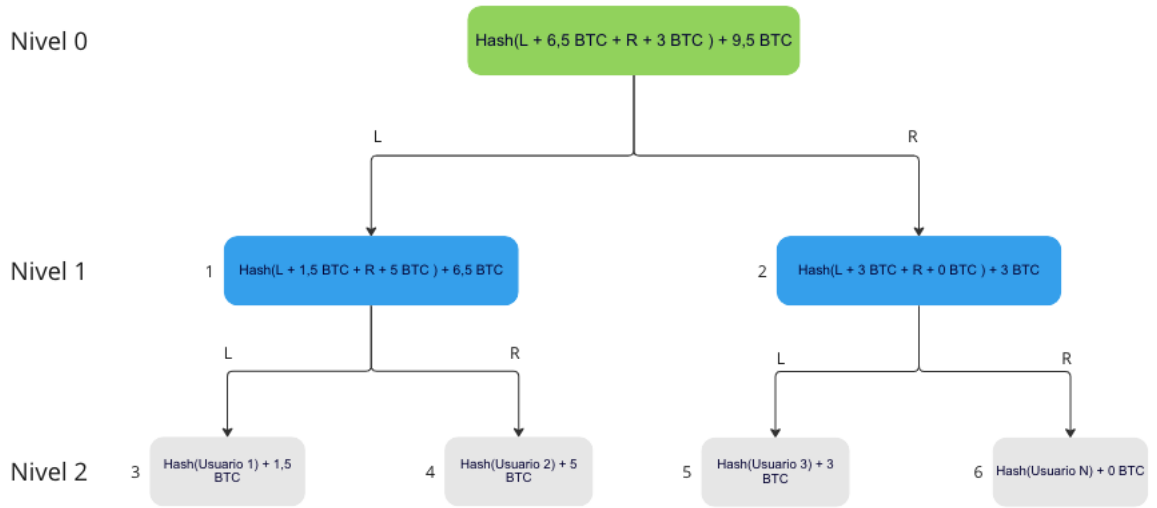


Figure 4. Diagrama de Árbol de Merkle de Sumatoria construido a partir de los datos del ejemplo

5.3.1. Generación de hojas

Nodo 3:

$$(4c7f961a9ea01b1b7b01013d4060e66be114bf001a6f5dcf592f0c70a7834, 1.5 \text{ BTC}) \\ = (\text{Sha256}(\text{Usuario 1}), 1.5 \text{ BTC})$$

Nodo 4:

$$(dab30e44f84eff3e0b08a1200cf2c39d752920d205fe1707946f96a28455ef09, 5 \text{ BTC}) = \\ (\text{Sha256}(\text{Usuario 2}), 5 \text{ BTC})$$

Nodo 5:

$$(0d16e0cdc6c8e85378d9f152bc12784a83d10a8f95b916ffbd48329ae9b04913, 3 \text{ BTC}) = \\ (\text{Sha256}(\text{Usuario 3}), 3 \text{ BTC})$$

Nodo 6:

$$(00000000000000000000000000000000, 0 \text{ BTC})$$

El nodo 6 en este caso es un agregado que no necesariamente se relaciona con un usuario de la plataforma. Se utiliza para que el nivel quede completo y además para ayudar con la privacidad de los usuarios ya que de esta manera cada usuario podrá tener menos información acerca de los balances de otros usuarios (aunque siempre cabe destacar que nunca conocerán la identidad de ese usuario).

Calcular la cantidad de nodos, $N(nodos)$, requeridos en un nivel se puede hacer con la siguiente definición, donde n representa el nivel

$$N(nodos)(n) = 2^n \quad (1)$$

El nivel 0 representa el nivel donde se encuentra el nodo raíz. Por otro lado, partiendo en base a la cantidad de usuarios $N(usuarios)$ (en este caso 3) para obtener la cantidad de niveles necesarios surge a partir de:

$$\frac{\log(N(usuarios))}{\log(2)} \quad (2)$$

Dada esta función la cantidad de niveles siempre estará representada por el valor entero mayor más cercano. En este caso:

$$\frac{\log(3)}{\log(2)} = 1.87 \quad (3)$$

obteniendo el valor número 2 como el entero mayor más cercano.

Por último la cantidad de nodos del tipo vacío que se deben agregar al Árbol de Merkle de Sumatoria es la cantidad restante para que el último nivel quede completo en cuanto cantidad de Nodos. Los nodos vacíos siempre se ingresan con saldo 0 (cero) y *hash* 0x00.

5.3.2. Generacion de niveles intermedios

Para generar los nodos de los niveles intermedios hasta llegar al nodo raíz, se combinan los nodos hijos de manera tal de concatenar secuencialmente los *hashes* y sumando los balances correspondientes

Nodo 1:

(122d9df1ae765225db247cca3302366d29a528e0ceb046684a0d58c7a4a14473, 6.5 BTC)
) = (Sha256(4c7f961a9ea01b1b7b01013d4060e66be114bf001a6f5dcf592f0c70a7834
 + 1.5 BTC + dab30e44f84eff3e0b08a1200cf2c39d752920d205fe1707946f96a28455ef09
 + 5 BTC), 6.5 BTC)

Nodo 2:

(f49ae9d2e55ecedf8ff5c9e87b91f774c1d3187d581810eb9d0e98bc8c24c111, 3 BTC)=
 (Sha256(0d16e0cdc6c8e85378d9f152bc12784a83d10a8f95b916ffbd48329ae9b04913 +
 3 BTC + 00000000000000000000000000000000 + 0 BTC), 3 BTC)

Nodo 0 (raíz):

(7de08b7bba1cfa3ea211a4ff4d214be0764affdf6a1e347f9ce96f90497f5ba5, 9.5 BTC) =
 (Sha256(2e4134db717903572d3bc9c3bfdc431b94c3cdefa38a70fd69c1c51c20d6762a +
 6.5 BTC + 18ff753ac58e00aedaed9daceb058988ba1092cf343da67abe8d025a3b20a176
 + 3 BTC), 9.5 BTC)

Luego de realizar todos los niveles se obtiene el *hash* raíz correspondiente al [Nivel 0 ; Nodo 0] el cual en el ejemplo realizado anteriormente es:

7de08b7bba1cfa3ea211a4ff4d214be0764affdf6a1e347f9ce96f90497f5ba5

Además, se obtiene el balance total que en este caso suma un total de

9.5 BTC

5.4. Proceso de validación de Arbol de Merkle de Sumatoria

El proceso de validación consiste en obtener una prueba de Merkle dados un *hash* de hoja y un *hash* padre y validar que es posible reconstruir la información hasta el nodo raíz correctamente. Esta validación o verificación permite al usuario saber que el balance que se le indica en su hoja forma parte del balance total de los pasivos de la empresa y que además el mismo se encuentra incluido dentro de esta estructura de datos, junto con los balances del resto de los usuarios.

La verificación se obtiene mediante un proceso iterativo generando nodos padre partir del *hash* de sus hijos, siguiendo el algoritmo descrito en la sección anterior y verificando en el paso final que el *hash* padre obtenido es igual al indicado por la empresa generadora del Árbol de Merkle. De esta manera podemos comprobar que el saldo en cuestión esta contemplado dentro del resultado final del Árbol de Merkle.

Es decir, la prueba consiste de una lista de nodos donde se indica la posición del nodo con respecto al padre, este dato es necesario para poder realizar el camino correcto hacia el *hash* raíz.

6. Proceso de implementación: Árbol de Merkle de Sumatoria

A continuación se realizará una explicación de una implementación de Árbol de Merkle de Sumatoria utilizando la teoría explicada anteriormente y mostrando el código utilizado en el repositorio *open-source* creado para tal fin [9].

6.1. Construcción del Árbol de Merkle de Sumatoria

La construcción del Árbol de Merkle de Sumatoria sigue los lineamientos establecidos en la sección anterior, con la diferencia de que en el ejemplo descrito se utiliza un único balance, pero en el caso de un *exchange* tradicional se ofrecen múltiples criptomonedas por lo que el balance pasará a ser una lista de balances, donde se tendrá un monto por cada criptomoneda.

Salvando esta diferencia, la construcción se realiza de la misma manera y se establecen las siguientes condiciones adicionales:

- Si el balance de una criptomoneda es cero para un usuario entonces el mismo no estará almacenado como balance. Por ejemplo, si el usuario tuviera fondos únicamente en BTC, su balance sería definido como “BTC:monto”. Se encuentra implícito el hecho de que tenga balance cero en el resto de las criptomonedas.
- El monto de cada criptomoneda que posea el usuario se debe definir con 8 decimales.
- El árbol siempre será completo, en el caso en donde la cantidad de usuarios no llenará el último nivel completamente, se completará con nodos definidos con un identificador en 0 y sin balances (que se define como un *string* vacío “”).
- Los nodos no pueden tener balances negativos, a medida que se construye el árbol se verifica que el balance sea siempre mayor a cero.

6.2. Consumo de información de balances de usuarios

Para obtener los datos referentes a los balances de los usuarios se obtienen todos los balances de los usuarios para cada una de las criptomonedas ofrecidas por la empresa, luego se filtran a aquellos usuarios que no cuentan con balances para ninguna criptomoneda.

Una vez hecho esto, debe generarse un archivo CSV en donde se almacenarán estos datos de manera tal de que sigan el siguiente formato:

id, balances

en donde el *id* será el identificador del usuario y los balances se definen como un string con el formato *cryptocurrency₁ : amount₁, ..., cryptocurrency_N : amount_N* para cada una de las criptomonedas.

6.3. Construcción de Árbol de Merkle de Sumatoria

Una vez generado este archivo CSV, se utiliza un script que puede consultarse en el enlace al repositorio correspondiente [1] en el cual se lee el archivo y se crea el Árbol de Merkle de Sumatoria correspondiente.

En la presente implementación se genera el Árbol de Merkle mediante la clase `MerkleSumTree` que se puede instanciar de la siguiente manera

```
mst = MerkleSumTree(user_balances, hash_type = 'sha256', salt = audit_id, shuffle = True)
```

Este `MerkleSumTree` se instancia enviando como parámetro las hojas del árbol (que representan los identificadores de los usuarios y sus balances). A su vez, es posible indicar otros parámetros optativos:

- *Shuffle*: Permite elegir si uno desea que las hojas pasadas como parámetro sean reordenadas o no.
- *Salt*: Define el salt que se utilizará para crear el hash para un usuario, como default se genera un código alfanumérico aleatorio.
- *Hash_type*: Permite elegir entre distintos algoritmos de hash, como default se utiliza sha256.

En este caso, se utiliza sha256 como algoritmo de hash, se utiliza un salt que se pasa como parámetro al script y luego se realiza el reordenamiento.

La implementación de la librería define que el árbol se almacene en el objeto creado como una lista plana, en donde el nodo i es el padre de los nodos $2i$ y $2i + 1$, respectivamente. Cabe destacar que el nodo con posición 0 no se encuentra definido, es decir, el árbol comienza en la posición 1 de la lista.

A su vez, se almacena un mapa en donde cada *key* es un *hash* de una hoja del árbol y su valor correspondiente es el índice de ese nodo en la lista descrita anteriormente. De esta manera, se simplifica el proceso de encontrar un *hash* ya que se cuenta con acceso directo a un nodo hoja una vez que se tiene su *hash*.

La creación del árbol se realiza definiendo el algoritmo por el cual se generan los nodos intermedios. Para ello, se siguen dos pasos:

1. Computar el *hash* del nodo padre, lo que se logra concatenando el *hash* del hijo izquierdo con su respectivo balance y luego el *hash* del hijo derecho con su respectivo balance.

$$H(p) = H(lhash + lbalance + rhash + rbalance) \quad (4)$$

2. Sumar los balances almacenados en cada uno de los nodos hijo donde quedó definido que si un nodo no cuenta con una criptomoneda como balance entonces debemos sumar con cero (pues la no existencia implica que el balance es 0 para esta criptomoneda).

El *script* creado finaliza generando dos *outputs*. Por un lado, se genera un *output* del árbol en un archivo CSV en donde se escribe una línea por cada uno de los nodos de manera ordenada (de arriba hacia abajo, de izquierda a derecha). Por otro lado, se genera un *output* de las pruebas para cada usuario (cuya implementación se describe en el apartado siguiente), nuevamente en un archivo CSV en donde cada línea define una prueba de Merkle para cada usuario.

6.4. Generación de Pruebas Merkle

Las pruebas para un usuario se pueden obtener mediante el método `get_proof` del `MerkleSumTree`, el cual toma como parámetro el identificador del usuario. Este identificador es luego usado con el *salt* y hasheado para obtener el *hash* correspondiente a una de las hojas del árbol. Luego se busca dicho *hash* en el mapa, obteniendo así la posición de su nodo.

Finalmente, se crea la prueba para dicho nodo, para ello se recorre el árbol hacia arriba y se almacenan los datos correspondientes al hermano de cada uno de esos nodos, lo que definirá finalmente la prueba de Merkle que se busca obtener.

```
for _ in range(proof_length):
    is_right = current_index % 2 != 0

    if is_right:
        sibling = self.tree[current_index - 1]
    else:
        sibling = self.tree[current_index + 1]

    proof.append(
```

```

    ProofStep(Side.RIGHT if not is_right else Side.LEFT, sibling.hash, sibling.balances)
)
current_index = current_index // 2

```

Puede observarse cómo debido a que el árbol se encuentra almacenado en una lista se puede iterar el mismo dividiendo el índice por 2 en cada paso.

A su vez, la prueba es una lista de objetos `ProofStep`, siendo cada uno de ellos un objeto que indica dónde se encuentra el nodo, su *hash* y sus balances.

Una prueba de Merkle será una lista de estos objetos, que luego puede iterarse para realizar una verificación.

6.5. Verificación de balances para un usuario

En el repositorio existe un archivo llamado `verify.py` que permite que el usuario ejecute la verificación de los resultados de los procesos anteriores. El mismo toma como entrada los datos iniciales del *hash* del nodo raíz, sus balances, la hoja de Merkle referente al usuario que efectúa la prueba, y sus pruebas de Merkle correspondientes.

```

def verify_merkle_proof_from_leaf(
    root_node: Node,
    steps: list[ProofStep],
    leaf_node: Node,
    hash_type: str = 'sha256'
) -> tuple[str, str]:
    hash_fun = getattr(hashlib, hash_type)
    current = leaf_node

    for idx, step in enumerate(steps):
        left = right = None

        require(
            all(amount >= 0 for amount in step.balances.values()),
            "At least one balance was negative"
        )
        require(
            all(amount >= 0 for amount in current.balances.values()),
            "At least one balance was negative"
        )

        if step.side == Side.RIGHT:
            left = current.hash + str.encode(to_string(current.balances))
            right = step.hash + str.encode(to_string(step.balances))
            sum_balances = combine_balances(current.balances, step.balances)
        else:
            left = step.hash + str.encode(to_string(step.balances))
            right = current.hash + str.encode(to_string(current.balances))
            sum_balances = combine_balances(step.balances, current.balances)

        current = Node(hash_fun(left + right).digest(), sum_balances)
        print(f"Step {idx + 1}, hash: {current.hash.hex()}")

    require(current.hash == root_node.hash, "Root hash is not equal to obtained hash")
    require(
        current.balances == root_node.balances,
        "Root balances are not equal to obtained balances"
    )

```

```
)

return current.hash.hex(), to_string(current.balances)
```

6.6. Generación de hoja Merkle

La hoja Merkle de un usuario puede generarse a partir de sus datos, concretamente: se parte de su identificador y sus balances en cada criptomoneda.

Para ello, se utiliza el archivo `merkle_leaf.py` donde se ingresan estos dos datos y el código identificador de la auditoría a evaluar, lo que devuelve el *hash* correspondiente a este *merkle leaf*. Los balances se corresponden con los que ya fueran ingresados, mientras que el *hash* se encuentra en formato hexadecimal.

```
id = '00cf6625-9c3d-4e61-a17a-9820cae615cc'
audit_id = '123456'

# Using SHA256
hash = hashlib.sha256(str.encode(audit_id + id)).digest().hex()

print(f"Obtained merkle leaf hash: {hash}")
```

7. Conclusiones

Mediante este trabajo se presentan los mecanismos que pueden utilizarse para crear una Prueba de Solvencia basada en tecnología transparente y sin recurrir a intermediarios. Para ello, en primer lugar se definen los fondos bajo la custodia de la empresa y luego, utilizando una estructura de datos que permite mantener la privacidad de los saldos de los usuarios, se prueba que los mismos se encuentran incluidos dentro de estas reservas totales.

Estos procedimientos serán utilizados por Lemon Cash para completar una Prueba de Solvencia transparente para que todos los usuarios puedan estar seguros de que sus fondos están protegidos y disponibles correctamente. Bajo la visión de encaminar a un ecosistema transparente, es que se comparten mecanismos que puedan ser aprovechados por otras empresas y establecer, así, un estándar de la industria. Con este objetivo, se disponibiliza código para la generación y verificación de la Prueba de Pasivos, que es *open source* y puede ser tomado como referencia.

Creemos que el futuro de la industria es *trustless*, es decir, que no está basado en la confianza, ya que esta no es necesaria cuando se ofrecen mecanismos transparentes para verificar las premisas por cuenta propia. En este marco es que se ofrecen herramientas que le permitan al usuario realizar las validaciones pertinentes para verificar por cuenta propia la solvencia de la empresa, sin requerir en la confianza en terceros.

Aspiramos a que estos aportes contribuyan a la construcción de un ecosistema más transparente, seguro y descentralizado, focalizado en la seguridad del usuario y aprovechando la tecnología para lograrlo.

References

- [1] Buterin, Vitalik. "Having a Safe CEX: Proof of Solvency and Beyond." Vitalik Buterin's Website, 19 Nov. 2022, Disponible: https://vitalik.ca/general/2022/11/19/proof_of_solvency.html.
- [2] Buterin, Vitalik. "An Approximate Introduction to How Zk-SNARKs Are Possible." Vitalik Buterin's Website, 26 Ene. 2021, Disponible: <https://vitalik.ca/general/2021/01/26/snarks.html>.
- [3] Bitso (2022) Prueba de Solvencia. Disponible: <https://support.bitso.com/hc/es-ar/categories/11192782297876-Prueba-de-Solvencia>.

- [4] Binance (2022) Proof of Reserves (PoR). Disponible: <https://academy.binance.me/en/glossary/proof-of-reserves-por>
- [5] Kraken (2022) Proof of Reserves. Disponible: <https://www.kraken.com/proof-of-reserves>.
- [6] Armanino (2022) proof of reserves merkle tree tool. Disponible: <https://github.com/armaninollp/proof-of-reserves-merkle-tree-tool>
- [7] Philippe Camacho. "Merkle trees: it is not that simple!". Disponible: <https://blog.enumai.io/update/2019/06/10/merkle-trees-not-that-simple.html>.
- [8] Kexin H., Zhenfeng Z., Kaiwen G.(2019). "Breaking the binding: Attacks on the Merkle approach to prove liabilities and its applications", Computers & Security, 87, 101585, <https://doi.org/10.1016/j.cose.2019.101585>
- [9] Lemon Cash (2023). Proof of Liabilities. <https://github.com/lemonatio/proof-of-liabilities>