Zerocash: Decentralized Anonymous Payments

Nicolazzi Robertino, robertnn@outlook.com

ZeroCash introduce la noción de un sistema descentralizado de pago, el cual es anónimo pero que brinda todas las funcionalidades y garantías de un sistema completo de moneda digital. Agregando fuertes garantías de anonimato, utilizando avances en áreas de zero-knowledge, mas específicamente z-k Succint Non-Interactive Arguments of Knowledge.

Ademas se reduce el tamaño de las transacción, se aumenta la velocidad de verificaron de las mismas. Permitir transacciones anónimas de diferentes valores, ocultar estos monto y permitir pagos directos entre usuarios.

Construccion progresiva de un esquema descentralizado-anónimo de pago

Paso 1. Valores fijos que ayudan al anonimato. Descripción con monedas de valor fijo, por ejemplo 1 BTC. Se logra esconder el origen de un pago, utilizando zk-SNARKs y un esquema de "commit". Definamos COMM al esquema de commit no-interactivo (dada aleatoriedad r y un mensaje m. El commit $\mathsf{c} := \mathsf{COMM}_{\mathsf{r}}(\mathsf{m})$; (donde se puede revelar que $\mathsf{c} := \mathsf{COMM}_{\mathsf{r}}(\mathsf{m})$ con m y r). Cada moneda es acuñada cuando un usuario u genera aleatoriamente un numero de serie sn y una trapdoor r, y genera un $\mathsf{cm} := \mathsf{COMM}_{\mathsf{r}}(\mathsf{sn})$. La moneda se define como $\mathsf{c} := (\mathsf{r}, \mathsf{sn}, \mathsf{cm})$ y la mint transaction $\mathsf{TX}_{\mathsf{Mint}}$, que contiene cm, se envía al "ledger". Sea CMList la lista de commits en la ledger, si un usuario quiere gastar, puede enviar una spend transaction $\mathsf{Tx}_{\mathsf{Spend}}$ que contenga el numero de serie sn de la moneda y una prueba π zk-SNARKS de la sentencia NP "Conozco algún r tal que $\mathsf{COMM}_{\mathsf{r}}(\mathsf{sn})$ aparece en la CMList"

El anonimato se alcanza ya que la prueba es zero-knowledge, se revela información de sn pero no de r, además lograr encontrar cual commit corresponde a una $\mathsf{Tx}_{\mathsf{Spend}}$ en particular es equivalente a invertir $\mathsf{f}(\mathsf{x}) = \mathsf{COMM}_{\mathsf{x}}(\mathsf{sn})$ lo cual es inviable.

Paso 2. Optimizando la lista de commits

La CMList definida solo como una lista de commit puede traer problemas de rendimiento. Ya que la mayoría de los algoritmos van a crecer linealmente con la CMList. Además que los commit que corresponden a monedas ya gastadas no pueden ser eliminadas, ya que por (1) no pueden ser identificadas de forma rápida. Para evitarlo se utiliza una función CRH resistente a colisiones. Se mantiene un árbol de Merkle CRH-based $\mathsf{Tree}(\mathsf{CMList})$ sobre la lista CMList donde rt denota al raíz de $\mathsf{Tree}(\mathsf{CMList})$, de esta forma la raíz puede ir actualizándose, reduciendo el tiempo lineal a logarítmico. Por lo tanto modificaremos nuestra sentencia NP a "Conozco algún r tal que $\mathsf{COMM}_{\mathsf{r}}(\mathsf{sn})$aparece como una hoja en el Árbol de Merkle CRH-based cuya raíz es rt"

Paso 3. Extendiendo las monedas para pagos directos. Como sabemos el commit de una moneda c establece una relación con el sn de la misma, esto es un problema cuando se quiere transferir c a otro usuario. Si un usuario $\mathsf{u_A}$ crea una c y se la envía a un usuario $\mathsf{u_B}$. $\mathsf{u_A}$ conoce el sn, haciendo no anónima la transferencia, luego de transferirla podría reconocerla. Por ende $\mathsf{u_B}$ debe transferir c y generar una c' para protegerse. Otro problema es que al querer transferir montos por ejemplo 100 BTC se necesitan 100 transferencias, revelando en cierta forma el monto, además si queremos enviar valores que no son múltiplos de 1BTC no es posible. Por lo tanto este esquema es inservible.

Esta situación se resuelve modificando como se deriva un commit, y usando funciones pseudoaleatorias para generar números de serie y direcciones destino de pago . Se utilizan tres funciones aleatorias, tal que dado x (semilla) definimos $\mathsf{PRF}^{\mathsf{addr}}_{\mathsf{x}}()$, $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{x}}()$, y $\mathsf{PRF}^{\mathsf{pk}}_{\mathsf{x}}(\cdot)$.

Para los pagos se utilizan direcciones, para lo cual cada usuario u genera un par $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$, direcciones publica y privada. Cada moneda de u contiene $\mathsf{a_{pk}}$ y pueden ser gastadas si se conoce $\mathsf{a_{sk}}$. Un par $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$ se genera eligiendo $\mathsf{a_{sk}}$ como semilla y haciendo $\mathsf{a_{pk}} := \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a_{sk}}}(0)$. De esta forma se pueden generar infinitos pares.

Para generar una moneda c de cierto valor v, u genera un p, un valor secreto que determina el numero de serie como $\mathsf{sn} := \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a_{sk}}}(\mathsf{p})$. Luego hace un commit $(\mathsf{a_{pk}}, \mathsf{v}, \mathsf{p})$ de la forma
1) dado r aleatorio $\mathsf{k} := \mathsf{COMM}_{\mathsf{r}}(\mathsf{a_{pk}}||\mathsf{p})$
2) dado s aleatorio $\mathsf{cm} := \mathsf{COMM}_{\mathsf{s}}(\mathsf{v}||\mathsf{k})$

Del acuñamiento resulta $\mathsf{c} := (\mathsf{a_{pk}}, \mathsf{v}, \mathsf{p}, \mathsf{r}, \mathsf{s}, \mathsf{cm})$ y una mint transaction $\mathsf{tx_{Mint}} := (\mathsf{v}, \mathsf{k}, \mathsf{s}, \mathsf{cm})$. De esta forma se oculta el dueño de la moneda o el numero de serie ya que están ocultos en k.

Las monedas se gastan utilizando un operación "pour". Si el usuario u, con direcciones $(\mathsf{a}^{\mathsf{old}}_{\mathsf{pk}}, \mathsf{a}^{\mathsf{old}}_{\mathsf{sk}})$, quiere consumar una moneda $\mathsf{c}^{\mathsf{old}} = (\mathsf{a}^{\mathsf{old}}_{\mathsf{pk}}, \mathsf{v}^{\mathsf{old}}, \mathsf{p}^{\mathsf{old}}, \mathsf{r}^{\mathsf{old}}, \mathsf{s}^{\mathsf{old}}, \mathsf{cm}^{\mathsf{old}})$ y producir dos nuevas monedas $\mathsf{c}^{\mathsf{new}}_1$ y $\mathsf{c}^{\mathsf{new}}_2$ de valor total $\mathsf{v}^{\mathsf{new}}_1 + \mathsf{v}^{\mathsf{new}}_2 = \mathsf{v}^{\mathsf{old}}$, que tendrán como objetivos otro par de direcciones . El usuario u para cada $\mathsf{i} \in \{1, 2\}$, realiza:
(i) u genera la aleatoriedad p del numero de serie $\mathsf{p}^{\mathsf{new}}_{\mathsf{i}}$;
(ii) u computa $\mathsf{k}^{\mathsf{new}}_{\mathsf{i}} := \mathsf{COMM}_{\mathsf{r}^{\mathsf{new}}_{\mathsf{i}}}(\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},\mathsf{i}}||\mathsf{p}^{\mathsf{new}}_{\mathsf{i}})$ dado $\mathsf{r}^{\mathsf{new}}_{\mathsf{i}}$;
(iii) u computa $\mathsf{cm}^{\mathsf{new}}_{\mathsf{i}} := \mathsf{COMM}^{\mathsf{new}}_{\mathsf{s}\,\mathsf{i}}(\mathsf{v}^{\mathsf{new}}_{\mathsf{i}}||\mathsf{k}^{\mathsf{new}}_{\mathsf{i}})$ dado aleatoriamente $\mathsf{s}^{\mathsf{new}}_{\mathsf{i}}$.

Esto retorna monedas $\mathsf{c}^{\mathsf{new}}_1 := (\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},1}, \mathsf{v}^{\mathsf{new}}_1, \mathsf{p}^{\mathsf{new}}_1, \mathsf{r}^{\mathsf{new}}_1, \mathsf{s}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_1)$ y $\mathsf{c}^{\mathsf{new}}_2 := (\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},2}, \mathsf{v}^{\mathsf{new}}_2, \mathsf{p}^{\mathsf{new}}_2, \mathsf{r}^{\mathsf{new}}_2, \mathsf{s}^{\mathsf{new}}_2, \mathsf{cm}^{\mathsf{new}}_2)$. Luego el usuario genera una prueba π_{POUR} para la sentencia NP POUR:
"Dada la raíz rt del árbol de Merkle, el número de serie $\mathsf{sn}^{\mathsf{old}}$ y los commit $\mathsf{cm}^{\mathsf{new}}_1, \mathsf{cm}^{\mathsf{new}}_2$.
Conozco monedas $\mathsf{c}^{\mathsf{old}}, \mathsf{c}^{\mathsf{new}}_1$ y $\mathsf{c}^{\mathsf{new}}_2$ y una dirección secreta $\mathsf{a}^{\mathsf{old}}_{\mathsf{sk}}$ tal que:

- $\mathsf{c}^{\mathsf{old}}$ cumple $\mathsf{k}^{\mathsf{old}} = \mathsf{COMM}^{\mathsf{old}}_{\mathsf{r}}(\mathsf{a}^{\mathsf{old}}_{\mathsf{pk}}||\mathsf{p}^{\mathsf{old}})$ y $\mathsf{cm}^{\mathsf{old}} = \mathsf{COMM}^{\mathsf{old}}_{\mathsf{s}}(\mathsf{v}^{\mathsf{old}}||\mathsf{k}^{\mathsf{old}})$; igual para $\mathsf{c}^{\mathsf{new}}_1$ y $\mathsf{c}^{\mathsf{new}}_2$.
- $\mathsf{a}^{\mathsf{old}}_{\mathsf{pk}} = \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a}^{\mathsf{old}}_{\mathsf{sk}}}(0)$.
- numero de serie correcto: $\mathsf{sn}^{\mathsf{old}} := \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a}^{\mathsf{old}}_{\mathsf{sk}}}(\mathsf{p}^{\mathsf{old}})$.
- el commit $\mathsf{cm}^{\mathsf{old}}$ aparece como hoja en el Merkle-tree con raíz rt.

- Los valores : $v_1^{new} + v_2^{new} = v^{old}$."

Una pour transaction $tx_{Pour} := (rt, sn^{old}, cm_1^{new}, cm_2^{new}, \pi_{POUR})$ es agregada al ledger. Veamos que si el usuario u no conoce $a_{sk,1}^{new}$, este no podrá transferir ni gastar la moneda, como característica de la transacción pour. Si conoce $a_{sk,1}^{new}$, luego no podrá distinguirla porque no se revela información del numero de serie de la nueva moneda. Esta transacción tampoco revela el valor de las monedas consumidas ni que commit han estado involucrados.

Paso 4. Envío de una moneda. Para lograr un envío seguro sin agregados al sistema (e.j. mensajes privados ya que un usuario debe mandar información secreta de c_1^{new} al otro usuario u_1). Se modifica la estructura de los pares de direcciones, el par $(addr_{pk}, addr_{sk})$ donde $addr_{pk} = (a_{pk}.pk_{enc})$ y $addr_{sk} = (a_{sk}.sk_{enc})$. Donde el par (pk_{enc}, sk_{enc}))sirve para un esquema de encriptación.

El usuario u genera el texto cifrado C_1, compuesto de $(v_1^{new}, p_1^{new}, r_1^{new}, s_1^{new})$ usando $pk_{enc,1}^{new}$ (parte de dirección publica de u_1) y agrega C_1 en la transacción pour,

Paso 5. Si queremos convertir monedas a BaseCoin, debemos introducir en la transacción pour un v_{pub} e información en el parámetro info, puede utilizarse para definir el destino de este valor. El valor de la sentencia POUR cambia a $v_1^{new} + v_2^{new} + v_{pub} = v^{old}$. Estos parámetros son opcionales.

Paso 6. Para prevenir malleability-attacks en transacciones pour. Se agrega a la sentencia POUR la utilización de firmas digitales. El usuario u genera un par (pk_{sig}, sk_{sig}), luego computa $h_{sig} = CRH(pk_{sig})$. Se genera dos valores $h_1 = PRF^{pk}_{a^{old}_{sk,1}}(h_{sig})$ y $h_2 = PRF^{pk}_{a^{old}_{sk,2}}(h_{sig})$

Zk-SNARKs

Primero definiremos dos conceptos básicos. Dado un cuerpo $\mathbb{F}$, un **circuito aritmético** de $\mathbb{F}$ toma como entrada elementos del cuerpo $\mathbb{F}$ y retorna elementos del conjunto $\mathbb{F}$. Recordemos que un circuito aritmético es un gráfico acíclico donde los nodos cuyo grado de entrada es cero son llamados input gate. Para modelos no-determinísticos se considerara circuitos que tengan de entrada un $x \in \mathbb{F}^n$ y una entrada auxiliar $a \in \mathbb{F}^h$ denominada "*witness*".

Una asignación valida para un circuito aritmético C:$\mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ es una tupla $(a_1, ..., a_N) \in \mathbb{F}^N$ donde $N = (n + h) + l$ de tal manera que $C(a_1, ..., a_{n+h}) = (a_{n+h+1}, ..., a_N)$

Programas aritméticos cuadráticos.(QAP). Los programas aritméticos-cuadraticos tienen como utilidad en zk-SNARK brindarle al usuario herramientas para construir una prueba π que demuestre que este conoce una asignación valida para un circuito C.

Un QAP Q(C) := $(\vec{A}, \vec{B}, \vec{C}, Z)$ dado un circuito aritmético C, retorna tres grupos de

polinomios

$\overrightarrow{A} = (A_i(z))_{i=0}^{m} \qquad \overrightarrow{B} = (B_i(z))_{i=0}^{m} \qquad \overrightarrow{C} = (C_i(z))_{i=0}^{m} \qquad (m >= N)$

Los cuales tienen coeficientes en $\mathbb{F}$ y un polinomio de salida $Z(z) \in \mathbb{F}[z]$ tal que el polinomio $Z(z)$ divide a

$$P(z) := \underbrace{(A_0(z) + \sum_{i=1}^{m} a_i A_i(z))}_{A(z)} \underbrace{(B_0(z) + \sum_{i=1}^{m} a_i B_i(z))}_{B(z)} - \underbrace{(C_0(z) + \sum_{i=1}^{m} a_i C_i(z))}_{C(z)}$$

Si y solo si $(a_1, ..., a_N) \in \mathbb{F}^N$ es una asignación valida para el circuito C. El "prover" puede fácilmente construir $P(z)$ para la prueba π y chequear la divisibilidad de $P(z)$por $Z(z)$

Definición:

El problema de satisfacibilidad de un circuito aritmético $\mathbb{F}$, sea *C:* $\mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ un circuito, es capturado por la relación $R_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, a) = 0^l\}$.

Y su lenguaje esta definido como $L_C = \{x \in \mathbb{F}^n : \exists a \in \mathbb{F}^h s.t. C(x, a) = 0^l\}$.

Dado un cuerpo $\mathbb{F}$, un zk-SNARKs para un circuito aritmético de $\mathbb{F}$ es una 3-upla de algoritmos de tiempo polinomial $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{KeyGen}(1^\lambda, C)(pk, vk)$. El cual tiene como entrada un parámetro de seguridad λ y un circuito aritmético de $\mathbb{F}$, C . El algoritmo tiene como salida una clave de prueba, pk y una clave de verificación vk. Las cuales son publicadas como parámetros públicos y pueden ser usados cualquier cantidad de veces para probar o verificar pertenencia al conjunto L_C.

En los esquemas DAP se utiliza este algoritmo en la inicialización del sistema. Luego de la construcción de un circuito C_{POUR} se computa $(pk_{POUR}, vk_{POUR}) = \mathsf{KeyGen}(1^\lambda, C_{POUR})$

- $\mathsf{Prove}(pk, x, a) \to \pi$. Como entrada recibe una clave pk y cualquier par $(x, a) \in R_C$, y genera como salida una "non-interactive proof" π para la sentencia $x \in L_C$.

En esquemas DAP el algoritmo se utiliza cuando se vaya a generar una transacción POUR. Como resultados se obtiene $\pi_{POUR} := \mathsf{Prove}(pk_{POUR}, x, a)$, donde pk_{POUR} es parámetro de sistema , $x = (rt, sn_1^{old}, sn_2^{old}, cm_1^{new}.cm_2^{new}, v_{pub}.h_{Sig}, h_1, h_2)$ y a alberga mas información de estas monedas.

- $\mathsf{Verify}(vk, x, \pi) \to b$. Como entrada recibe una clave de verificación vk, x, y una prueba π. Si el verificador es convencido que $x \in L_C$ retornara $b = 1$.

Se utiliza en esquemas DAP en el algoritmo para verificar transacciones calculando $\mathsf{Verify}(vk_{POUR}, x, \pi_{POUR})$, vk_{POUR} es un parámetro de sistema, x es una instancia de la sentencia POUR del esquema y π_{POUR} se obtiene a partir de la estructura de una transacción pour TX_{POUR}]

Esquema descentralizado y anónimo de pago (DAP)

Estructuras de datos

Basecoin ledger. El protocolo es aplicado sobre una moneda digital ledger-based como bitcoin. Esta denominación es llamada *Basecoin.* Sea cualquier momento T los usuarios pueden acceder al *ledger*, la cual es una secuencia de transacciones que se van agregando al final.
Parámetro públicos. Una lista de parámetros públicos pp esta disponible para todos los usuarios del sistema. Generados en un momento inicial por una parte confiable del mismo.
Direcciones. Cada usuario genera al menos un par denominado *address key* $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$. La clave publico $\mathsf{addr}_{\mathsf{pk}}$, es publicada para que otros usuarios puedan realizar pagos directos, y la clave secreta $\mathsf{addr}_{\mathsf{sk}}$ se utiliza para recibir pagos enviados a la anterior.

Monedas. Una *moneda* es una objeto de datos **c** asociada a:
- Compromiso (*coin commitment*) denotado por $\mathsf{cm}(\mathsf{c})$, una cadena de caracteres que aparece en el *ledger* luego que **c** es acuñada (*minted*).
- Valor denotado $\mathsf{v}(\mathsf{c})$, valuado en *Basecoin,* entero entre 0 y un valor máximo $\mathsf{v}_{\mathsf{max}}$.
- Un número serie denotado por $\mathsf{sn}(\mathsf{c})$, una cadena única para identificar monedas.
- Una dirección $\mathsf{addr}_{\mathsf{pk}}(\mathsf{c})$ que representa a quien le pertenece la moneda

Transacciones. Se agregan dos nuevos tipos de transacciones.
- Mint transactions. Una mint transaction $\mathsf{tx}_{\mathsf{mint}}$ es una tupla $(\mathsf{cm}, \mathsf{v}, *)$, donde cm es un coin commitment, v un valor y * denota otra información que varia según la implementación. Esta transacción deja sentado que una moneda a sido creada.
- Pour transactions.
Una pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ es una tupla $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, \mathsf{v}_{\mathsf{pub}}, \mathsf{info}, *)$. Donde rt es la raíz del árbol de Merkle. Estas transacciones consumen dos monedas y dejan asentado el "pasaje" de estas dos monedas a nuevas con sus respectivos commitment

Commits y números de serie. Dado algún tiempo T.
- $\mathsf{CMList}_{\mathsf{T}}$ denota la lista de todos los commits que aparecen en mint y pour transactions L_{T} .
- $\mathsf{SNList}_{\mathsf{T}}$. Lista de todos los números de serie que aparecen en pour transactions en L_{T}.

Árbol de Merkle. Dado un tiempo T denotaremos como $\mathsf{Tree}_{\mathsf{T}}$al árbol de Merkle sobre $\mathsf{CMList}_{\mathsf{T}}$ y rt_{T} su raíz.

Algoritmos: Un esquema DAP es una tupla de algoritmos
(Setup, CreateAddress, Mint, Pour, VerifyTransaction, Receive)

System Setup. Genera una lista de parámetros públicos. Se realiza una única vez

- INPUTS: parámetro de seguridad λ
- OUTPUTS: parámetros públicos $\mathsf{pp} = (\mathsf{pk}_{\mathsf{POUR}}, \mathsf{vk}_{\mathsf{POUR}}, \mathsf{pp}_{\mathsf{enc}}, \mathsf{pp}_{\mathsf{sig}})$

  Se construye el circuito $\mathsf{C}_{\mathsf{POUR}}$ y se utiliza KeyGen para generar los parámetros públicos de firma, encriptación y claves de verificación

CreateAddress

- INPUT: parámetros públicos pp
- OUTPUTS: par $(\mathsf{addr}_{\mathsf{pk}}.\mathsf{addr}_{\mathsf{sk}})$

  Se generan las claves de encriptación publica y privada, y a_{sk} y a_{pk}. Para poder retornar el par de direcciones

Minting coins

- INPUTS:
  - parámetros públicos pp
  - un valor v para la moneda
  - dirección de destino $\mathsf{addr}_{\mathsf{pk}}$
- OUTSPUTS: moneda c y una transacción $\mathsf{tx}_{\mathsf{Mint}}$

  Se generan el número de serie y las dos trapdoors necesarias (r,s) para obtener los valores k y cm que servirán para identificar una moneda

Pour

- INPUTS
  - parámetros públicos pp
  - raíz rt de Merkle
  - monedas viejas $\mathsf{c}_1^{\mathsf{old}}, \mathsf{c}_2^{\mathsf{old}}$
  - direcciones viejas $\mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}$
  - camino de autenticación path_1 desde $\mathrm{cm}(\mathsf{c}_1^{\mathsf{old}})$ a rt
  - camino de autenticación path_2 desde $\mathrm{cm}(\mathsf{c}_2^{\mathsf{old}})$ a rt
  - nuevos valores $\mathsf{v}_1^{\mathsf{new}}, \mathsf{v}_2^{\mathsf{new}}$
  - nuevas direcciones
  - valor publico $\mathsf{v}_{\mathsf{pub}}$
- OUTPUTS: nuevas monedas $\mathsf{c}_1^{\mathsf{new}}, \mathsf{c}_2^{\mathsf{new}}$ y una transacción pour

  Además de construir una instancia POUR y obtener la prueba para dicha sentencia utilizando Prove. Se generan las firmas y se encriptan los textos planos que son

necesario como información adicional en la transacción POUR

VerifyTransaction

- INPUTS:
  - parámetros públicos pp
  - una transacción mint o pour tx
  - el "ledger" L actual
- OUTPUTS: un bit b, que es 1 si la transacción es valida

Si se recibe una transacción mint la uncia verificación necesaria, es ver si el commit que se obtiene utilizando v y k es el mismo que el que se encuentra en la transacción mint. Si la transacción es Pour debe verificarse, que estén los números de series antiguos en el "ledger", si la raíz es correcto, verificar la firma del mensaje y además, utilizando Verify, realizar la prueba de zero-knowledge.

Receive

- INPUTS:
  - $(\mathsf{addr}_{\mathsf{pk}}.\mathsf{addr}_{\mathsf{sk}})$
  - El "ledger" actual L

Construcción de un DAP

Para el esquema DAP necesario se usa una función de hash resistente a colisiones denominada $\mathsf{CRH} : \{0,1\}^* \to \{0,1\}^{O(\lambda)}$. Se necesita una familias de funciones pseudoaleatorias $\mathsf{PRF} = \{\mathsf{PRF}_{\mathsf{x}} : \{0,1\}^* \to \{0,1\}^{O(\lambda)}\}_{\mathsf{x}}$ donde x es una semilla. De las cuales derivados tres $\mathsf{PRF}^{\mathsf{addr}}_{\mathsf{x}}(\mathsf{z}) = \mathsf{H}(00||\mathsf{z})$, $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{x}}(\mathsf{z}) = \mathsf{H}(01||\mathsf{z})$ y $\mathsf{PRF}^{\mathsf{pk}}_{\mathsf{x}}(\mathsf{z}) = \mathsf{H}(10||\mathsf{z})$.

También se necesita de un esquema de commits que definimos $\{\mathsf{COMM}_{\mathsf{x}} : \{0,1\}^* \to \{0,1\}^{O(\lambda)}\}_{\mathsf{x}}$. Si queremos revelar un commit necesitamos el valor x y el z de esta forma $\mathsf{cm} = \mathsf{COMM}_{\mathsf{x}}(\mathsf{z})$.

One-Time firma digital. Se usa un esquema $\mathsf{Sig} = (\mathsf{G}_{\mathsf{sig}}.\mathsf{K}_{\mathsf{sig}}, \mathsf{S}_{\mathsf{sig}}, \mathsf{V}_{\mathsf{sig}})$. El primero toma un parámetro de seguridad y genera parámetros públicos $\mathsf{pp}_{\mathsf{sig}}$. El segundo, dado $\mathsf{pp}_{\mathsf{sig}}$ genera una clave publica $\mathsf{pk}_{\mathsf{sig}}$ y una secreta $\mathsf{sk}_{\mathsf{sig}}$. $\mathsf{S}_{\mathsf{sig}}$, genera una firma σ a partir de la clave secreta y por ultimo $\mathsf{V}_{\mathsf{sig}}$ devuelve 1 si, dado clave publica un mensaje y la firma σ, esta ultima es valida para el mensajes.

Encriptamiento de clave publica y privada. Se usa un esquema $\mathsf{Enc} = (\mathsf{G}_{\mathsf{enc}}, \mathsf{K}_{\mathsf{enc}}, \mathsf{E}_{\mathsf{enc}}, \mathsf{D}_{\mathsf{enc}})$. $\mathsf{G}_{\mathsf{enc}}$ y $\mathsf{K}_{\mathsf{enc}}$, generan parámetros públicos y las claves publica y privada respectivamente. $\mathsf{E}_{\mathsf{enc}}$ encripta dado mensaje y clave publica, y por ultimo $\mathsf{D}_{\mathsf{enc}}$ desencripta dado texto cifrado y clave secreta

La sentencia POUR

Una instancia de esta sentencia esta definida como $x = (rt, sn_1^{old}, sn_2^{old}, cm_1^{new}.cm_2^{new}, v_{pub}.h_{Sig}, h_1, h_2)$. A su vez definimos una debilidad a de la siguiente manera:

- Una debilidad a es de la forma $(path_1, path_2, c_1^{old}, c_2^{old}, addr_{sk,1}^{old}, addr_{sk,2}^{old}, c_1^{new}, c_2^{new})$ para cada i ∈ {1,2}:
  $c_i^{old} = (addr_{pk,i}^{old}, v_i^{old}, p_i^{old}, r_i^{old}, s_i^{old}, cm_i^{old})$
  $c_i^{new} = (addr_{pk,i}^{new}, v_i^{new}, p_i^{new}, r_i^{new}, s_i^{new}, cm_i^{new})$
  $addr_{pk,i}^{old} = (a_{pk,i}^{old}, pk_{enc,i}^{old})$
  $addr_{pk,i}^{new} = (a_{pk,i}^{new}, pk_{enc,i}^{new})$
  $addr_{sk,i}^{old} = (a_{sk,i}^{old}, sk_{enc,i}^{old})$

Dada una instancia POUR x, una debilidad a es valida para x si se da:
1. Para cada i ∈ {1,2}:
   (a) El commit cm_i^{old} de la moneda c_i^{old} aparece en el ledger, i.e., $path_i$ es un camino valido para la hoja correspondiente al commit en el árbol de Merkle de raíz rt
   (b) La dirección $a_{sk,i}^{old}$ corresponde c_i^{old} , i.e., $a_{pk,i}^{old} = PRF_{a_{sk,i}^{old}}^{addr}(0)$.
   (c) Numero de serie correcto sn_i^{old} i.e., $sn_i^{old} = PRF_{a_{sk,i}^{old}}^{sn}(p_i^{old})$.
   (d) La moneda c_i^{old} esta bien construida, i.e., $cm_i^{old} = COMM_{s_i^{old}}(COMM_{r_i^{old}}(a_{pk,i}^{old}||p_i^{old})||v_i^{old})$.
   (e) La moneda c_i^{new} esta bien construida, i.e., $cm_i^{new} = COMM_{s_i^{new}}(COMM_{r_i^{new}}(a_{pk,i}^{new}||p_i^{new})||v_i^{new})$.
   (f) La dirección $a_{sk,i}^{old}$ relaciona h_{Sig} con h_i , i.e., $h_i = PRF_{a_{sk,i}^{old}}^{pk}(i||h_{Sig})$.
2. Se preserva el balance: $v_1^{new} + v_2^{new} + v_{pub} = v_1^{old} + v_2^{old}$

ZEROCASH

Para el caso concreto de zerocash, se inicializa el esquema DAP de la siguiente forma:
Sea H la función de compresión de SHA256, inicializamos CRH,PRF, COMM mediante esta función H. Definimos CRH como H(z) para $z \in \{0,1\}^{512}$. Como tiene compresiones "dos-a-uno" sirve para construir un árbol de Merkle.

Definimos $PRF_x(z)$ como $H(x||z)$ con $x \in \{0,1\}^{256}$ y $z \in \{0,1\}^{256}$. Por lo que las funciones derivadas son:
$PRF_x^{addr}(z) = H(x||00||z)$ $x \in \{0,1\}^{256}$ $z \in \{0,1\}^{254}$
$PRF_x^{sn}(z) = H(x||01||z)$
$PRF_x^{pk}(z) = H(x||10||z)$

Para el esquema de commit COMM usamos el patrón $k = COMM_r(a_{pk}||p)$ y $cm = COMM_s(v||k)$.

Donde definimos k como $\mathsf{H}(\mathsf{r}||[\mathsf{H}(\mathsf{a_{pk}}||\mathsf{p})]_{128})$ (r es de 384 bits) y cm como $\mathsf{H}(\mathsf{k}||0^{192}||\mathsf{v})$.

Iniciando la sentencia POUR:

Dado $\mathsf{i} \in \{1, 2\}$ se prueba:
- $\mathsf{path_i}$ es un camino de autenticación para la hoja $\mathsf{cm_i^{old}}$ con respectiva raíz rt en un CRH-based árbol de Merkle
- $\mathsf{a_{pk,i}^{old}} = \mathsf{H}(\mathsf{a_{sk,i}^{old}}||0^{256})$
- $\mathsf{sn_i^{old}} = \mathsf{H}(\mathsf{a_{sk,i}^{old}}||01||[\mathsf{p_i^{old}}]_{254})$
- $\mathsf{cm_i^{old}} = \mathsf{H}(\mathsf{H}(\mathsf{r_i^{old}}||[\mathsf{H}(\mathsf{a_{pk,i}^{old}}||\mathsf{p_i^{old}})]_{128})||0^{192}||\mathsf{v_i^{old}})$
- $\mathsf{cm_i^{new}} = \mathsf{H}(\mathsf{H}(\mathsf{r_i^{new}}||[\mathsf{H}(\mathsf{a_{pk,i}^{new}}||\mathsf{p_i^{new}})]_{128})||0^{192}||\mathsf{v_i^{new}})$
- $\mathsf{h_i} = \mathsf{H}(\mathsf{a_{sk,i}^{old}}||10||\mathsf{b_i}||[\mathsf{h_{Sig}}]_{253})$ donde $\mathsf{b_1} := 0$ y $\mathsf{b_2} := 1$.

Circuito Aritmético Para POUR

Necesitamos combinar varios subcircuitos para lograr la verificación de la sentencia pour ya que esta consiste de varias partes. Primero debemos construir un circuito que demuestre **pertenencia al arbol de Merkle** dado una raíz rt, un camino de identificación path y un commit cm. El camino de identificación contiene un valor de hash $\mathsf{h_i}$y un bit $\mathsf{r_i}$ especificado si $\mathsf{h_i}$ es el hijo izquierdo ($\mathsf{r_i} = 0$) o derecho del nodo padre. Luego verificamos la pertenencia invocando H de abajo hacia arriba. Como $\mathsf{d_{tree}} = 64$ definimos $\mathsf{k_{i-1}} = \mathsf{cm}$, y por cada $\mathsf{i} = \mathsf{d} - 1, ..., 1$ definimos $\mathsf{B_i} = \mathsf{h_i}||\mathsf{k_i}$ si $\mathsf{r_i} = 0$ sino $\mathsf{k_1}||\mathsf{h_1}$ y computamos $\mathsf{k_{i-1}} = \mathsf{H}(\mathsf{B})$. Y verificamos la raíz rt.

Circuito para comparar. Donde dados 64-bit enteros A,B y C se satisface si C = A + B para ellos vemos $\sum_{\mathsf{i}=0}^{63} 2^{\mathsf{i}}\mathsf{c_i} = \sum_{\mathsf{i}=0}^{63} 2^{\mathsf{i}}(\mathsf{b_i} + \mathsf{a_i})$

Circuito para la suma. Dados dos enteros de 64 bit A y B hay que chequear que $\mathsf{A} + \mathsf{B} < 2^{64}$. Para eso vemos $\sum_{\mathsf{i}=0}^{63} \mathsf{c_i} = \sum_{\mathsf{i}=0}^{63} 2^{\mathsf{i}}(\mathsf{b_i} + \mathsf{a_i})$ para algún $\mathsf{c_i} \in \{0, 1\}$

ANEXO

Algoritmos zk-SNARKs con circuitos aritméticos cuadráticos

Key Generation

Input: Arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^l$

1. Construct the QAP $Q(C) = (\vec{A}, \vec{B}, \vec{C}, Z)$ of C.
2. Randomly sample $\tau, \rho_A, \rho_B \in \mathbb{F}$. Set $\rho_C := \rho_A \rho_B$.
3. Generate the Proving Key $\mathrm{pk} := (\mathrm{pk}_A, \mathrm{pk}_B, \mathrm{pk}_C, \mathrm{pk}_H)$ where

$$\mathrm{pk}_A := (\underbrace{A_i(\tau)\rho_A P_1}_{\mathrm{pk}_{A,i}})_{i=0}^{m} \quad \mathrm{pk}_B := (\underbrace{B_i(\tau)\rho_B P_2}_{\mathrm{pk}_{B,i}})_{i=0}^{m} \quad \mathrm{pk}_C := (\underbrace{C_i(\tau)\rho_C P_1}_{\mathrm{pk}_{C,i}})_{i=0}^{m}$$

$$\mathrm{pk}_H := (\underbrace{\tau^i P_1}_{\mathrm{pk}_{H,i}})_{i=0}^{d}$$

4. Generate the Verification Key $\mathrm{vk} := (\mathrm{vk}_{\mathsf{IC}}, \mathrm{vk}_Z)$ where

$$\mathrm{vk}_{\mathsf{IC}} := (\underbrace{A_i(\tau)\rho_A P_1}_{\mathrm{vk}_{\mathsf{IC},i}})_{i=0}^{n} \quad \mathrm{vk}_Z := Z(\tau)\rho_C P_2$$

Output: $(\mathrm{pk}, \mathrm{vk})$

Prover

Input: $\mathrm{pk}, \vec{x} \in \mathbb{F}^n$, and $\vec{w} \in \mathbb{F}^h$

1. Construct the QAP $(\vec{A}, \vec{B}, \vec{C}, Z)$ of C.
2. Compute a valid distribution $(a_1, \ldots, a_m) = \mathrm{QAPwit}(C, \vec{x}, \vec{w})$.
3. Determine the coefficients $(h_i)_{i=0}^{d}$ of $H(z) = \frac{A(z)B(z) - C(z)}{Z(z)}$
4. Construct the proof $\pi := (\pi_A, \pi_B, \pi_C, \pi_H)$ where

$$\pi_A := \sum_{i=n+1}^{m} a_i \mathrm{pk}_{A,i} \quad \pi_B := \mathrm{pk}_{B,0} + \sum_{i=1}^{m} a_i \mathrm{pk}_{B,i} \quad \pi_C := \mathrm{pk}_{C,0} + \sum_{i=1}^{m} a_i \mathrm{pk}_{C,i}$$

$$\pi_H := \mathrm{pk}_{H,0} + \sum_{i=1}^{d} h_i \mathrm{pk}_{H,i}$$

Output: proof π of the statement "$\vec{x} \in L_C$"

Verifier

Input: $\mathrm{vk}, \vec{x} \in \mathbb{F}^n$, proof π

1. Calculate $\mathrm{vk}_{\vec{x}} = \mathrm{vk}_{\mathsf{IC},0} + \sum_{i=1}^{n} x_i \mathrm{vk}_{\mathsf{IC},i}$.
2. Check QAP divisibility:

$$e(\mathrm{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \mathrm{vk}_Z) \cdot e(\pi_C, P_2). \tag{3}$$

Output: 1 (correct) or 0 (incorrect)

Referencias

zk-SNARK explained: Basic Principles by Hartwig Mayer

Zerocash: Decentralized Anonymous Payments from Bitcoin Eli Ben-Sasson,Alessandro Chiesa Christina Garman, Matthew Green, ian Miers, Eran Tromer, y Madars Virza