Funciones hash criptográficas

Introducción

Una función hash es una función de programación que se utiliza para asignar datos de tamaño random a datos de una longitud fija. Las funciones de hash se originaron de la necesidad de comprimir información para reducir la cantidad de memoria requerida para almacenar archivos grandes. El caso de uso más popular para una función hash es para una estructura de datos que se llama tabla hash. Las tablas hash se usan ampliamente para la búsqueda rápida de datos. También ayudan a minimizar las etiquetas de archivos como textos, mp3, PDF o imágenes para que el trabajo con estos archivos grandes en tamaño sea manejable. Un requisito clave de las funciones hash es que generan una serie de caracteres alfanuméricos de longitud fija entonces se pueden utilizar como resumen o digestión (por el uso de la palabra en inglés 'digest') de archivos.
La razón inicial para el uso función hash provino de la necesidad de comprimir información, un uso y beneficio secundario que apareció es uso del hashing como identificadores únicos y singulares. Cuando se "hashean" (palabra que no estoy seguro de que exista pero que se utiliza) muchos mensajes, hay una poca probabilidad que dos terminen resultando en un hash idéntico. Si una función hash es elegida con ciertas propiedades entonces, los mensajes diferentes dan como resultado hashes de salida diferente. En cambio, sí "hasheamos" dos mensajes diferentes y resulta que ambos producen el mismo hash entonces decimos que tuvo una colisión. Desde la perspectiva la base de datos, esto significa que dos objetos diferentes terminan siendo almacenados en la misma celda, lo que no es bueno si uno busca definir identificadores singulares. Para que una función hash sea útil necesitamos que posea tres propiedades:

1. Que su entrada puede ser cualquier cadena de cualquier tamaño.
2. Que produzca salidas de tamaño fijo.
3. Que sea eficientemente computable.

Esas propiedades son las que definen a las funciones hash en general, y gracias a estas las funciones se pueden usar para construir estructuras de datos como tablas hash, árboles de Merkle o blockchains (cadenas de bloques), que analizaremos en siguientes artículos.
Para que una función hash sea criptográficamente segura, requerimos que tenga tres propiedades adicionales:

1. Resiste a colisiones
2. Ocultar
3. Amigables para construir rompecabezas criptográficos

De estas últimas tres propiedades es que voy a hacer foco, porque son parte de las tecnologías que se usan en la criptografía que usamos en Signatura, en Bitcoin y en la mayoría las criptomonedas.

Propiedad 1: Resistente a colisiones

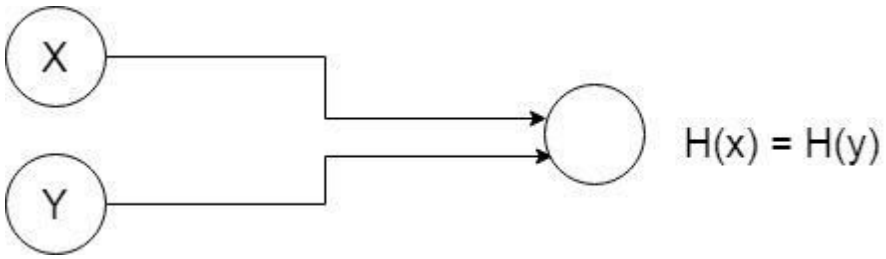
Nota: Una cadena (string) de información genérica se expresa como "x" y la función hash asociada a esa cadena como H(x).

Una colisión se produce cuando dos entradas distintas nos dan como resultado la

misma salida. Una función hash H(x) es resistente a colisiones si nadie puede encontrar una colisión.
Esto se puede expresar matemáticamente como:

$$\{ x \; e \; y \; / \; xyH(x) = H(y)\}$$

Para todos los "x" e "y" que sean distintos es imposible encontrar un par de "x" e "y" que al aplicarles la función hash se obtengan los mismos valores de salida.



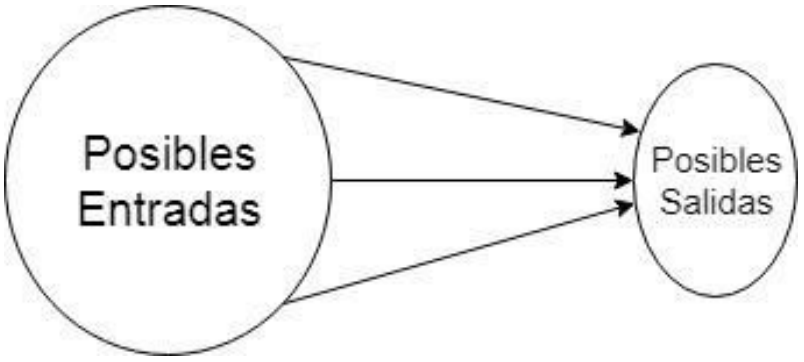
Ejemplo de colisión, "x" e "y" son distintos y sin embargo cuando les aplicamos la función hash producen la misma salida

Resistente a las colisiones no significa que las colisiones no existan, significa que las colisiones existen pero que es extremadamente difícil encontrarlas. Vamos a demostrarlo con un ejemplo muy muy muy sencillo.
Vamos a crear la función hash más sencilla que puede existir, la función SHA1 (las siglas SHA viene del acrónimo en inglés "Secure Hash Algoritm" en español Algoritmo Hash Seguro). Nuestra función va tener como parámetro de entrada múltiple data y va a tener como salida 1 bit (recordemos que un bit es la menor unidad de información en una computadora y posee un valor binario es un 0 o un 1).
Si de todas las posibles entradas elijo el abecedario {a,b,c,d,.....} cuando ingrese la letra {a} a la función hash voy a obtener el 0 de mi único bit y cuando ingrese la letra {b} obtendré el valor 1 de mi bit. Ahora que es lo que va a suceder cuando quiera ingresar la letra {c}, la función hash me va a devolver o un 0 o un 1, efectuando así la primera colisión, y como existe 1 colisión existen infinitas colisiones. Entonces como conclusión sacamos que la función de 1 solo bit es insegura.
En Bitcoin se utiliza el estándar criptográfico SHA256 eso quiere decir que la salida tiene 256 bits, puede parecer poco pero la cantidad de posibilidades que tenemos a la salida de esta función es 2^{256} . Como está escrito puede parecer que no son muchas posibilidades, pero si traducimos ese número a notación decimal lo que obtenemos es astronómico:

Decimal approximation:

$$1.157920892373161954235709850086879078532699846656405 6\ldots \times 10^{77}$$

¿Por qué elegí esa palabra? porque ese número está a solo 3 órdenes de magnitud de todos los átomos del universo. Es prácticamente imposible encontrar una colisión en este espacio, pero las colisiones existen simplemente por el hecho que el espacio de entrada es infinito y el espacio de salida es finito.



Debido a que el número de entradas excede el número de salidas, tenemos la garantía de que debe haber al menos una salida a la que la función hash asigne más de una entrada.

Si uno quisiera calcular la mitad del espacio de las salidas, o sea 2^{128} posibilidades distintas, llevaría muchísimo tiempo de calcular. Si una computadora calcula 10.000 hashes por segundo, se necesitarán más de un octillón (10^{27}) años para calcular 2^{128} hashes.

Aplicación: Digestor de mensajes

La pregunta que emerge de todo lo que hablamos, ¿Para qué sirve esto?
Si sabemos que una función hash "H(x)" es resistente a colisiones, entonces es seguro asumir que "x" e "y" son diferentes.
Este argumento nos permite usar salidas hash como un resumen del mensaje.
Imaginemos un sistema de almacenamiento de archivos online que permite a los usuarios subir archivos y garantizar su integridad cuando los descarguen. Las funciones hash libres de colisiones nos permiten una solución eficiente a este problema. Solo se necesita recordar el hash del archivo original, cuando se descarga nuevamente el archivo se calcula su hash y lo comparamos con el hash almacenado. Si los valores hash son iguales, se puede asumir que el archivo es exactamente el mismo, pero si son diferentes, entonces es seguro asumir que el archivo sufrió una modificación.
En Bitcoin se utiliza la función hash de diversas formas, la primera como resumen de mensaje para garantizar su inmutabilidad, también como parte de la prueba de trabajo y la última como parte del protocolo de firmas digitales ECDSA que veremos más adelante.

Propiedad 2: Ocultar

La propiedad de ocultar afirma que si se nos da el resultado de la función hash H(x), no hay forma posible de descubrir cuál fue la entrada "x". Esta propiedad solamente se puede aplicar en el caso que no se conozcan cuáles son las posibles entradas.
Pensemos el siguiente ejemplo: Un simple juego de tirar una moneda al aire, en donde el resultado puede ser "cara" o "cruz".

Si el resultado del lanzamiento de moneda fue cara, vamos a mostrar el hash H(cara). Si el resultado fue cruz, mostraremos el hash de la cadena H(cruz), esto nos dá 2 resultados.

Input 1: H(cara) => f61f5c5e22749aeba2bd920352dbd4c9f63e0ebaed6df623b09c836ffcbec2eb
Input 2: H(cruz) => 231f6523efb1f10c654e2b6a16d792296e7497119f561721890b3221ff1a94c8

Un atacante, que no vio el lanzamiento de moneda, pero solo vio la salida de la función hash, puede descubrir el lanzamiento de la moneda simplemente computando los hashes de "cara" y "cruz" y compararlos con el que salió. El atacante es capaz de adivinar cuál fue el lanzamiento porque solo había dos valores posibles de x, y es fácil para él probarlos. Para poder lograr la propiedad de ocultar, debe darse el caso de que no haya ningún valor de x que sea particularmente probable.

Una función hash H tiene la propiedad de ocultar si se elige un valor secreto r de una distribución de probabilidad que tiene una entropía alta tal que dado el hash de ‘x’ concatenado con ‘r’, H (r | x), es imposible encontrar x.

Aplicación: Comprometerse con un resultado

Un esquema de compromiso (en inglés "commitment scheme") es el análogo digital de tomar un valor, sellarlo en un sobre y poner ese sobre en la mesa donde todos puedan verlo. Cuando hacemos esto, nos comprometimos con lo que está dentro del sobre. Pero no lo abrimos, así que, aunque nos hayamos comprometido con un valor, el valor permanece en secreto para todos los demás. Cuando sea necesario, se puede abrir el sobre y revelar el valor con el que nos comprometimos anteriormente.

"Un esquema de compromiso es comprometerse con un valor que permanece oculto, pero con la capacidad de poder revelar este valor en otro momento." David Chaum
Este tipo de esquema lo utilizamos todos los días de nuestras vidas, por ejemplo, cuando nos logueamos en una página de internet. La web no almacena nuestra contraseña tal cual nosotros la ingresamos, sino que, en cambio lo que almacena es el hash de la contraseña. Cuando una persona se quiere loguear, se computa el hash del input de la contraseña y se compara con el valor que tenemos almacenado, en caso de que sean iguales se da acceso. De esta forma, en el caso de que el atacante acceda a la lista de usuarios, le es imposible acceder a los perfiles de los usuarios.
Este es un ejemplo de un esquema de compromiso, nos comprometemos con el valor (nuestra contraseña) que sabemos que computando la función hash nos va a dar como resultado un valor conocido.

Propiedad 3: Amigables para construir rompecabezas criptográficos

Una función hash H es amigable para armar rompecabezas si para cada valor de salida y de n bits posible, y si k se elige de una distribución con entropía alta, no es factible encontrar "x" tal que H(k|x)=y en un tiempo significativamente menor que 2^n.

Traducido al español esto significa que, si alguien quiere obtener un valor de salida particular de una función hash, y si hay una parte de la entrada que se elige de una manera adecuada al azar, es muy difícil encontrar otro valor que nos devuelva el mismo hash inicial, básicamente encontrar una colisión. Y además que no hay ninguna estrategia que sea mejor que otra para probar valores, o sea que probando valores al azar es exactamente igual.

Un rompecabezas criptográfico consiste en:

- una función hash, H
- un valor, id (que llamamos ID (Identificación) del rompecabezas), elegido de una distribución de alta min-entropía
- un objetivo establecido Y

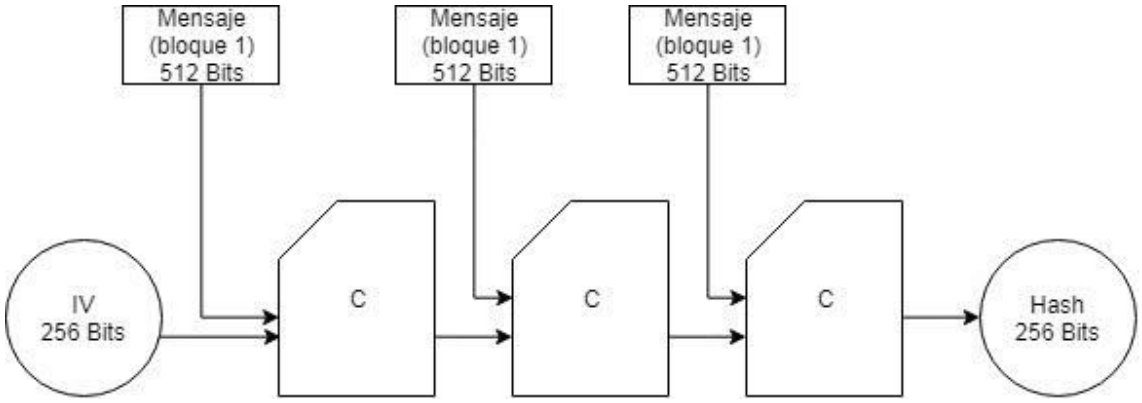
Una solución a este rompecabezas es un valor, x, tal que

$$H(id||x) \in Y.$$

El tamaño de **Y** determina qué tan difícil es el rompecabezas. Si **Y** es el conjunto de todas las cadenas de n bits, el rompecabezas es trivial, mientras que si Y tiene solo 1 elemento, el acertijo es extremadamente difícil.

SHA-256

Hemos discutido acerca de las tres propiedades de las funciones hash y aplicaciones de las mismas. Ahora veamos una función hash particular, esta es la que utiliza Bitcoin principalmente, y se llama SHA-256.
SHA-256 utiliza un método de compresión que se llama Transformada de Merkle-Damgard. En terminología común, como el método de compresión es resistente a las colisiones, entonces la función SHA que lo utiliza también se convierte en resistente a las colisiones.
SHA-256 usa una función de compresión que toma una entrada de 768 bits (256+512) y produce salidas de 256 bits.



Representación del funcionamiento de la función hash SHA-256

Conclusiones

Las funciones de hash se originaron de la necesidad de comprimir y generar datos uniformes estandarizados para que el almacenamiento sea más eficiente, lo que implica que a la salida obtenemos cadenas pseudoaleatorias de una longitud fija. Sin embargo, para crear una función hash completamente resistente a las colisiones, cada mensaje (x) tendría que tener una salida de la misma longitud que la entrada. Si usamos hashes de longitudes variables, perdemos la conveniencia de usarlos en estructuras de datos y si le asignamos una longitud fija nuestra función hash no es completamente libre de colisiones.