📗 Hash en Python | El Libro De Python

ellibrodepython.com/hash-python

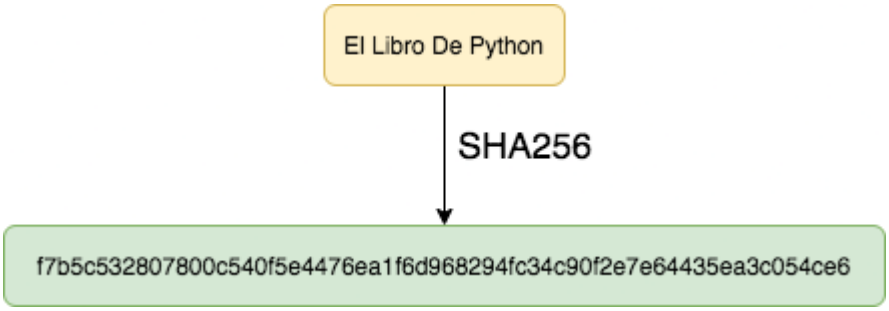
Hash en Python

Una función **hash** es una función que dada una entrada de longitud variable devuelve una secuencia de longitud fija, con algunas propiedades interesantes. Si por ejemplo aplicamos como entrada la cadena `El Libro De Python`, la salida será la siguiente.

```
import hashlib

salida = hashlib.sha256(b"El Libro De Python").hexdigest()
print(salida)

# Salida
# f7b5c532807800c540f5e4476ea1f6d968294fc34c90f2e7e64435ea3c054ce6
```



Existen diferentes funciones hash, y en el caso anterior hemos usado la `sha256`. Una función hash se puede ver como **una función resumen**, ya que nos permite "resumir" un conjunto de datos de longitud variable en una secuencia de longitud fija (y relativamente corta). Podríamos también meter el libro entero de El Quijote un su función hash `sha256` sería:

```
03f22ee1408a1bea9a7a9dfc0431051432c26a8a16fa6925d5246ff3235de3a4
```

Hemos por tanto resumido cientos de páginas en una sola línea.

Propiedades de las Funciones Hash

Las funciones hash tienen unas propiedades que las hacen muy útiles en el mundo de la criptografía y blockchain:

- A pesar de que la entrada tiene una longitud arbitraria, **la salida tiene una longitud fija**. Esta longitud vendrá determinada por el tipo de función hash que se use. Por ejemplo, la `sha256` devuelve siempre 256 bits (o 32 bytes).
- Las funciones hash suelen ser **rápidas de calcular**.

- Siendo `x` la entrada y `hash(x)` su función hash, es imposible (o muy muy difícil) obtener `x` a partir de `hash(x)`. Es decir, que la función hash no es reversible. Si tenemos el hash de El Quijote, no podemos reconstruir el libro a partir del hash. Esto se conoce como **resistencia a la primera preimagen**.
- Tiene que ser muy complicado (por no decir imposible) encontrar una nueva entrada `x'` siendo `x'!=x` tal que `hash(x) = hash(x')`. Es decir, tiene que ser imposible encontrar dos entradas cuya función hash sea la misma. Esto se conoce como **resistencia a la segunda imagen**.
- Por último, la **resistencia a colisiones** o *collision resistance* nos dice que debe ser imposible encontrar dos entradas diferentes y distintas cuyo hash sea el mismo.

Es importante notar que algunas de estas características son de vital importancia, y que si alguna de ellas dejara de cumplirse, el mundo de Internet estaría en serios problemas.

Por ejemplo, si se llegara a poder revertir una función hash, los pagos online que realizamos, contraseñas o incluso blockchains como bitcoin o ethereum podrían estar en problemas. Se dice que la computación cuántica podría romper las funciones hash, pero aún quedan años para eso.

Aplicaciones de Función Hash

Las funciones hash tienen aplicaciones en diferentes sectores. Explicamos a continuación sus casos de uso más relevantes:

- **Integridad de información**: Podemos usar las funciones hash para asegurarnos de que un determinado contenido digital no ha sido modificado. Si por ejemplo calculamos el hash de un vídeo o un libro y lo almacenamos, tendremos una "huella digital" de dicho contenido. Si en un futuro nos envían ese mismo vídeo o libro, podemos calcular el hash otra vez y compararlo con el que teníamos almacenado anteriormente. Esto nos ahorra tener que ir fotograma a fotograma o página a página comparando ambos archivos.
- **Generar números aleatorios**: Podemos usar las funciones hash para generar números aleatorios, o para ser más preciso para generar números pseudoaleatorios.
- **Firma digital**: En la firma digital se suele firmar sólo el hash del mensaje en vez del contenido entero, lo que resulta más eficiente y reduce ciertos vectores de ataque.
- **Merkle Trees**: Los *merkle trees* también pueden ser usados para resumir información, donde la misma es dividida en pequeños trozos y su hash es calculado recursivamente hasta obtener un único hash llamado *merkle root*. Estos son muy utilizados en la blockchain.

Tipos de Funciones Hash

Existen diferentes funciones hash, donde cada una tiene sus casos de uso. Algunas de las características más importantes son la longitud de la salida y el algoritmo que usan:

- **BLAKE**: Tiene variantes como la BLAKE-2, BLAKE-3, siendo la última anunciada en 2020. Existen diferentes variantes en función del número de bits de su salida.
- **MD**: Tiene múltiples variantes como la MD1, MD2, MD3, MD4 y MD5. El MD5 es muy usado para integridad de datos y fue introducido en la RFC 1321.
- **SHA**: Tiene variantes como la SHA256, SHA512, SHA224, SHA384. El SHA256 es el usado por la criptomoneda Bitcoin.
- **KECCAK-256**: Usado por la criptomoneda Ethereum.

Funciones Hash en Python

Gracias a la librería `hashlib` de Python disponemos de prácticamente todas las funciones hash que existen. Veamos por ejemplo como usar la `sha256`.

```
import hashlib

m = hashlib.sha256()
m.update(b"El Libro De Python")
salida = m.hexdigest()

print(salida)
# f7b5c532807800c540f5e4476ea1f6d968294fc34c90f2e7e64435ea3c054ce6
```

También podemos acceder al `digest_size`, es decir a la longitud de la salida. Este será un valor fijo dentro de cada función hash, y por ejemplo en el caso de `sha256` es 32 bytes, o lo que viene siendo lo mismo, 256 bits. De ahí viene su nombre.

```
print(m.digest_size)

# Salida
# 32
```

También podemos hacer el hash de múltiples entradas:

```
import hashlib

m = hashlib.sha256()
m.update(b"Secuencia 1")
m.update(b"Secuencia 2")
m.update(b"Secuencia 3")
salida = m.hexdigest()

print(salida)

# Salida:
# 8bfe6a71680cdf4cc4c4024b3808f0d865a729c58695b145e7408555c24aab29
```

A continuación podemos ver ejemplos para varias funciones hash, donde todas usan la misma entrada. Podemos ver como las salidas son diferentes y tienen distinta longitud.

```
import hashlib

print(hashlib.sha256(b"El Libro de Python").hexdigest())
print(hashlib.sha224(b"El Libro de Python").hexdigest())
print(hashlib.sha512(b"El Libro de Python").hexdigest())
print(hashlib.blake2b(b"El Libro de Python").hexdigest())
print(hashlib.blake2s(b"El Libro de Python").hexdigest())
print(hashlib.blake2s(b"El Libro de Python").hexdigest())
print(hashlib.md5(b"El Libro de Python").hexdigest())

# Salida:
# 3f0eb88c12b73f8235f3bc5a19336d32a41bbe6743291c97e8fb00ea2d3520e0
# a46e8c9207522d305f79b818f37170c8b3094acb607ed3645854ea38
#
01a489b59eb18cc2d297e3be6918ba1cff2875514900ce781a95c006a05eac68c9dffc0ae9bb64c00d
bc628fa1b2a28159e8cee2875e86157d82c0998a786beb
#
2728dcb655d2572bff0f0ecad1518ddc005a7896ea18e0f47c07cf54efcb639266c3056ebd4ef62aba
d34d8ccd074925012cde3da4a351f0f14831b7c36f6a48
# cae14ea7acc2827b66212f3da5ea35d8d65fe248fc00030ddb9f9e5113f120a7
# cae14ea7acc2827b66212f3da5ea35d8d65fe248fc00030ddb9f9e5113f120a7
# 77dd74c5fa2c54e26cc67b5ab47b38fd
```
