Bitcoin Cryptography - Hashing Algorithms

Author	josh
Date	2020-01-20 14:05:13

Overview

At the core of *crypto*currencies lies the science of cryptography. These mathematically secured and provable algorithms allow currencies like Bitcoin to be built in a way that's peer-to-peer instead of based on corporate or governmental trust.

One of the key classes of cryptographic algorithms used in cryptocurrencies is *hashing algorithms* - powerful one-way functions with a broad set of interesting applications. Let's learn about some of the important properties of hash functions and how they are used in Bitcoin.

All About Hashes

Properties of Hash Functions

Hash functions have a few key properties that make them incredibly useful and secure. Hash functions are one-way, they are preimage resistant (for crypto-secure versions), and are deterministic.

The first property is that hash functions are *one-way* - you cannot go backwards from a hash output (called the *digest* or the *hash*) back to the original input. These functions are sometimes referred to as "trapdoor functions", because once an input falls into the trapdoor of the hash function, there is no coming back to the original message!

For example, let's look at the SHA-256 hash of the message "Hello there!":

```
>>> from hashlib import sha256
>>> sha256(b"Hello there!").hexdigest()
'89b8b8e486421463d7e0f5caf60fb9cb35ce169b76e657ab21fc4d1d6b093603'
```

The message "Hello there!" is run through the function to create the digest "89b8b8e...". There is no algorithm for going backwards - so if all you have is the hash output "89b8b8e..." and you want to know the original input, the *only way* to find it is to guess inputs until you find a matching digest! For example, this is how cracking passwords works. Attackers compromise a database of hashes, not plaintext passwords. They have dictionaries of common passwords that users use, and run those guesses through the hashing algorithm until a matching hash is found in the database.

This leads us to the second useful property of cryptographically secure hashing algorithms known as *preimage resistance*. These algorithms are designed such that there is not a *predictable pattern* from inputs to outputs that might make it easier to guess. If you have the hash output "89b8b8e..." from above, there's nothing at all in that hash that points you in the direction of what the original input might be. An interesting part of good algorithms that relates to preimage resistance is the *waterfall effect*, where changing the input even slightly results in a drastically different output:

```
>>> from hashlib import sha256
>>> sha256(b"Hello there!").hexdigest()
'89b8b8e486421463d7e0f5caf60fb9cb35ce169b76e657ab21fc4d1d6b093603'
>>> sha256(b"Hello there").hexdigest()
'4e47826698bb4630fb4451010062fadbf85d61427cbdfaed7ad0f23f239bed89'
```

Notice that simply changing one character in the input (the '!') results in a hash that's nowhere near the first hash.

The third useful property of these functions is that they are *deterministic*. We know that they are one way - but it also turns out that the output is *always the same* for a particular input. From our example, feeding "Hello there!" into a properly-implemented SHA-256 function will always give the output value "89b8b83...". This is critical for the use of hash functions, as it means someone with a hash and an input can verify the correctness of a message. For example, a software developer can give you an expected hash for their program along with the binary. You can then run the binary through the hash algorithm to verify the program has not been tampered with on its way to you!

Hashing in Bitcoin

There are two primary hash functions used in Bitcoin: SHA-256 and RIPEMD160. SHA2 was developed by the US National Security Agency (NSA) and first published in 2001. RIPE on the other hand was developed by a group of researchers in the EU and released in 1992 with an update in 1996. It's interesting to note that Satoshi chose algorithms created in two very different ways, so if one does not trust the NSA or not trust open-source developed algorithms, there's layers of security built in since both algorithms are used in the system.

Now what about how these are used? The first application is in *Proof-of-Wor*k *Mining*. POW is used to validate transactions and issue new coins on the peer-to-peer network without the need for trust. Every 10 minutes, miners construct a potential *block* of transactions. They take the block header information and a random value called the *nonce* through the SHA-256 algorithm. If the hash output treated as a 256 bit number is less than a *difficulty target*, the network accepts the Proof-of-Work solution as valid. It take millions of nonce guesses to get to a hash output that meets the difficulty, but once a solution is found anyone on the network can verify it's correct in one step thanks to the deterministic nature of hashes.

```
Ex:
Difficulty target is 0123abc...
"block header info" + 0 -> SHA-256 -> d3a2bb9...
"block header info" + 1 -> SHA-256 -> 110dbb9...
"block header info" + 2 -> SHA-256 -> 04a4173...

0123abc < 04a4173, so this meets POW difficulty requirement
```

The second application of hashes in Bitcoin is in address derivation. Bitcoin uses Elliptic Curve Cryptography as part of the address derivation scheme (which will be covered in a later tutorial :), but hashes are also a critical part of the algorithm. Deriving a Bitcoin address looks like this:

```
Ex (not real outputs):
Private key: 0123456789abc....
->
ECDSA
->
Public key: af325bc23d...
->
SHA-256
->
RIPEMD160
->
base58 encoding
->
1aDF578G...
```

ECDSA provides one layer of security, but the hashing algorithms provide another! When a user sends funds to another user's address, they are not paying directly to the public key of the receiver. They are actually paying to a "Pay to Public Key *Hash*" address. The public key is hashed twice, once with SHA-256 and again with RIPEMD160, then encoded. What this does then is masks the receiver's public key *until the funds are later spent*. So if a user avoids address reuse, other parties can never know the public key of an address containing any funds. If there's a vulnerability in the way the wallet handles ECDSA or some future problem is found, this provides an extra layer of security over the owner's funds.

Hashing - A Critical Part of Bitcoin

Looking at the fascinating properties of hashing algorithms, it's clear why they're so useful in a peer-to-peer application like Bitcoin. Thanks to the one-way, preimage resistant, deterministic nature of these algorithms, we can build systems where other users can verify things like mined transactions securely without the need for trust. As well, hashes provide layers of security over fund ownership on the blockchain, thanks to P2PKH addresses. Us Bitcoin fans trust the provability of math more than the fallibility of corporations and governments - who needs trust when you have cryptography!