Understanding the Components of an IPFS CID

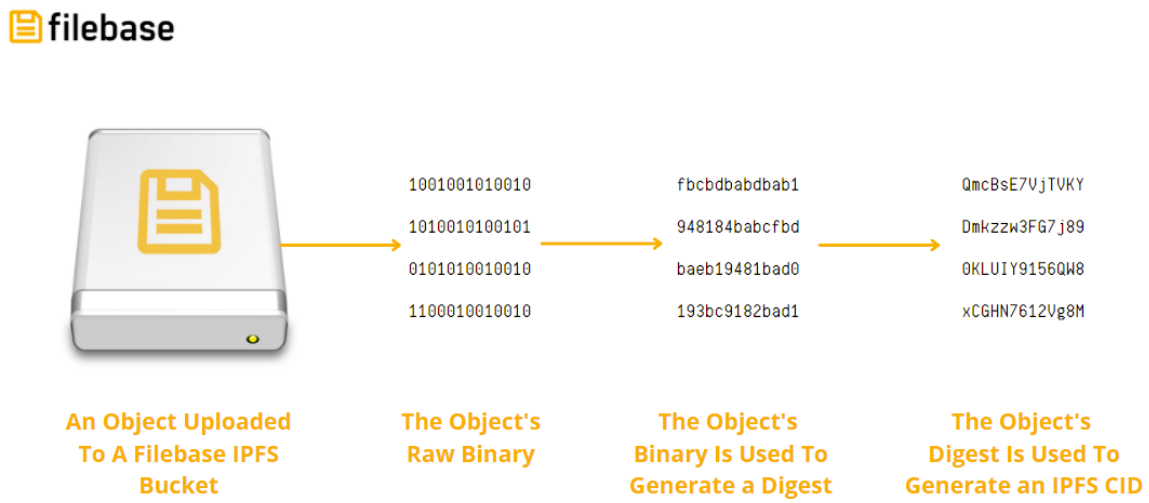
The Filebase Team ⋮ 7-9 minutes ⋮ 3/21/2023

On the decentralized web, content addressing is crucial for securely identifying and locating data. InterPlanetary File System (IPFS) originally developed the CID specification that is now part of Multiformats Project and used by various projects such as IPFS, IPLD, libp2p, and Filecoin. Each of these distributed information systems uses a CID as the primary identifier used for referencing content.

What is a CID?

A CID is a self-describing content-addressed identifier that does not indicate where content is stored. Instead, it creates an identifying address based on the content itself. The characters in a CID are determined by the cryptographic hash of the underlying content, not the content's size for format. As most content in IPFS is hashed using SHA2-256, all CIDs will be the same length, 256 bits or 32 bytes.

For instance, if we store an image of the Filebase on the IPFS network, its CID would be `QmWBaeu6y1zEcKbsEqCuhuDHPL3W8pZouCPdafMCRCSUWk`.



Cryptographic Algorithms

The first step to creating a CID is to transform the input data, using a cryptographic algorithm that maps input of arbitrary size (data or a file) to output of a fixed size. This transformation is known as a cryptographic hash digest or simply a hash.

The cryptographic algorithm used to generate a file's hash must have the following attributes:

- Deterministic: The same input must always produce the same hash.
- Uncorrelated: Any change in the input must always generate a completely different, unique hash.
- One-way: It must be infeasible to reconstruct the data using just the hash.
- Unique: Only one file can produce one unique hash.

If a single pixel in the Filebase logo image is changed or altered, the cryptographic algorithm produces a completely different hash for the image. When the data is retrieved using content addressing, the user retrieving the data is guaranteed to obtain the intended version of that data. This is different from location addressing on

the centralized web where the content at a given address (URL) can be changed or altered over time.

Cryptographic Hashing

Cryptographic hashing is not exclusive to IPFS, and many hashing algorithms such as SHA2-256, blake2b, SHA3-256 and SHA3-512, SHA1, and MD5 are available. IPFS employs SHA2-256 by default, but an IPFS CID can support almost any robust cryptographic hash algorithm.

Occasionally, a hashing algorithm may become insecure and fail to meet the current industry standard, as has already occurred with sha1. In the future, other algorithms may also prove inadequate for content addressing in IPFS and other distributed information systems. Therefore, to support multiple cryptographic algorithms, it is important to be able to determine which algorithm was used to generate the hash of a particular content.

Multihash

In order to support various hashing algorithms, IPFS uses multihash. A multihash is a self-describing hash containing metadata that includes the hash's length and the cryptographic algorithm that generated it. Multiformat CIDs are future-proof because they use multihash to support multiple hashing algorithms instead of relying on a single algorithm.

Multihashes follow the TLV (type-length-value) pattern. TLV patterns are comprised of the following components:

- Type: The identifier unique to the cryptographic algorithm that was used to generate the hash. For example, the identifier of SHA2-256 is 18 - 0x12 in hexadecimal format.
- Length: The actual length of the hash. If using the SHA2-256 algorithm, the length would be 256 bits, which is equal to 32 bytes.
- Value: The hash value.

The original hash is prefixed with the type of hashing algorithm used and the length of the hash. To represent a CID as a compact string, base encoding is used. When IPFS was first introduced, it used base58btc encoding to create CIDs, which resulted in the CIDv0 format where all CIDs begin with the characters Qm.

Concerns arose about whether this multihash format used for CIDv0 would be sufficient. To address these concerns, CIDv1, was introduced.

CIDv0 allows users to use different hashing algorithms to generate a hash for specific content and later identify the content using this hash. However, when users need to read the data, they may not know the data encoding method used, such as CBOR or plain JSON. To address this, CIDv1 content addressing uses a new prefix that identifies the encoding method used, called the multicodec prefix.

Multicodec

The multicodec prefix specifies the encoding used on the data and each encoding has its own unique short codec identifier. The multicodec prefix for an IPFS CID is always an IPLD codec since IPFS always uses IPLD formats for its data.

It's important to note that multicodec is not only used by IPFS and IPLD but is also a part of the Multiformats Project, along with multihash and a few other self-describing protocols. This project now supports many other projects and protocols, including the CID specification.

After adding a multicodec, a CIDv1 includes the following components:

<cid-version><multicodec><multihash-algorithm><multihash-length><multihash-hash>

However, since Version 0 CIDs only had the **<multihash-*>** components, additional prefixes are required to differentiate between the two versions. Thus, the updated CID format is as follows:

<cid-version><multicodec><multihash>

The **<cid-version>** field indicates the version of the CID (either 0 or 1).

It is possible to convert any CIDv0 to CIDv1, but not all CIDv1 can be converted to CIDv0 - only those with multibase as base58btc, multicodec dag-pb, multihash-algorithm SHA2-256, and multihash-length 32 can be converted into CIDv0.

Multibase Prefixes

In CIDv1, the binary representation of the CID contains the following fields:

<cid-version><multicodec><multihash>.

However, the binary version of a CID is not user-friendly, so we represent them in string form using base encoding. To ensure that we know the type of base encoding used, we use multibase prefixes.

These prefixes are only used in the string form of the CID. For example, in CIDv0, hashes are always encoded using base58btc. CIDv0 CIDs can safely interpret CIDv0 hashes as using base58btc. But, for other versions and encodings, the multibase prefix must be specified. The default encoding used by most IPFS implementations is base32.

The CID Inspector tool is a useful tool that allows you to visualize the prefixes of any IPFS CID.

This tool breaks down each part of the CID into easily readable sections for humans. It identifies components such as the Multibase and the Multicodec. Additionally, the Multihash is broken down into the hashing algorithm used, the length of the hash, and the content hash itself. The original hash of the content, before the appropriate CIDv1 prefixes were added, is also displayed in the "Human Readable CID" breakdown.

HUMAN READABLE CID

base58btc - cidv0 - dag-pb - (sha2-256 : 256 : 748AAECAFD9589CE10B9E61E20AD73E059D1D6C2598505C03BDF5BC32DA814E1)

MULTIBASE - VERSION - MULTICODEC - MULTIHASH (NAME : SIZE : DIGEST IN HEX)

MULTIBASE

PREFIX:
implicit

NAME:
base58btc

MULTICODEC

CODE:
0×70

NAME:
dag-pb

DESCRIPTION:
MerkleDAG protobuf

MULTIHASH

CODE:
0×12

NAME:
sha2-256

BITS:
256

DIGEST (BASE58BTC MULTIBASE):
zQmWBaeu6y1zEcKbsEqCuhuDHPL3W8pZouCPdafMCRCSUWk

DIGEST (HEX):
748AAECAFD9589CE10B9E61E20AD73E059D1D6C2598505C03BDF5BC32DA814E1

CID BYTE LENGTH

AS BASE58BTC STRING (BYTES)
46

AS BASE32 STRING (BYTES)
46

BINARY (BYTES)
34

CIDV1 (BASE32)

bafybeidurkxmv7mvrhhbbopgdyqk247alhi5nqszquc4ao67lpbs3kau4e

- Human Readable CID: An itemized breakdown of each component of the CID.
- Multibase: The code identifier of the base encoding used.
- Multicodec: The code identifier of the encoding type used.
- Multihash: A breakdown of the hashing algorithm used, the length of the hash, and the content hash itself.

IPFS Pinning

To assure that CIDs are always available, they must be pinned to an IPFS node on the network.

IPFS pinning refers to the process of storing a file or folder within an IPFS node's permanent storage instead of in the node's cache storage. Unless a file is pinned, it's stored in cache storage that is periodically cleared by the network's garbage collection process.

You can sign up for a free Filebase account to get started with your IPFS journey today.

If you have any questions, please join our Discord server, or send us an email at hello@filebase.com.