Metadata subtree for Bitcoin

By Joannes Vermorel (Lokad), Amaury Séchet (Bitcoin ABC), Shammah Chancellor (Bitcoin ABC), Jason Cox (Bitcoin ABC), January 2019

In the present document, Bitcoin always refers to Bitcoin Cash.

Status: **EARLY DRAFT**

Abstract: The original design of Bitcoin includes limited block-level metadata through an 80-byte block header. Unfortunately, much is left to be desired with this design. In particular, there is no provision for later upgrades. A decade after the inception of Bitcoin, newer requirements have emerged, and there are compelling reasons to introduce a limited set of carefully chosen metadata. We propose the *metadata subtree* as a maximally backward-compatible approach to introduce new metadata. This approach comes with its own versioning path, anticipating the need for future evolutions beyond the current proposal.

This proposal should be seen as the second part of a two-part proposal for Bitcoin. The first proposal is intended to rearrange the Merkle tree to facilitate the scaling process. See "Merklix tree for Bitcoin". Keeping those two proposals together simplifies the upgrade path for the Bitcoin ecosystem at large as both impact the block format.

The blocks within the blockchain of Bitcoin include transactions, however they also include other data that are not transactions. These non-transaction data which are inserted in every Bitcoin block are referred to as *metadata*. The metadata serve multiple purposes with varying degrees of criticality for the system as a whole. The original design of the Satoshi's client concentrated the metadata in the *block header*, an 80 byte sequence that most importantly includes the hash of the previous block header and the hash of the merkle tree which contains the transactions.

While Satoshi Nakamoto had taken great care of properly shaping key economic incentives within Bitcoin, the historical design of the Satoshi's client is lacking, to say the least, when it comes to metadata: timestamp is too short, nonce is too short, version number is not usable, etc. More importantly, the original design did not include provisions to standardize how further metadata should be introduced. While it is possible to put arbitrary metadata in the coinbase transaction, this approach presents unnecessary complications if the intent is to modify the consensus rules based on those metadata. None of those limitations represent a terminal problem for Bitcoin though. Those limitations are more akin to an ongoing hindrance, slowing down the professionalization and the scaling of the Bitcoin infrastructure.

Indeed, since the inception of Bitcoin back in 2008, the challenges and use cases have been significantly refined. There is a series of well-identified evolutions of Bitcoin which require

metadata to be added, especially with regards to its transition toward becoming a viable system that operates at scale. One of the most prominent use cases for metadata is the inclusion of a UTXO commitment which provides a straightforward mechanism for *livechain*[1] apps to restrict their processing to the UTXO set, skipping the bulk of the blockchain.

However, it is also reasonable to assume that further desirable metadata use cases will be discovered in the future. Thus, metadata needs a standardized path to make their way into Bitcoin if the corresponding use cases prove to be compelling enough to justify the overhead for the network.

While much is left to be desired in the design of the present 80-byte block header, it is not possible to modify this header without breaking the compatibility with the existing *physical* infrastructure of Bitcoin as most mining devices are hard-wired to this specific design. This concern restricts the options that are available to Bitcoin to improve its capacity to handle metadata. Indeed, the metadata upgrade should preserve backward compatibility as much as possible.

We present a mechanism to introduce metadata in Bitcoin referred to as the *metadata subtree*. This approach provides a series of advantages for Bitcoin while preserving the existing economic incentives.

- Metadata can be introduced in a controlled fashion in Bitcoin.
- The design flaws of the block header are mostly mitigated.
- Bitcoin software and Bitcoin-dependent software can be simplified.

By *publishers*, we refer to the participants in Bitcoin who publish valid blocks. The metadata subtree approach is extensively backward compatible, restricting the need to upgrade toward new Bitcoin software to the sole publishers. In particular, wallet implementations are not required to upgrade, also the expanded set of metadata proposed in the following allows significant simplifications and improvements on the wallet side as well.

The metadata subtree approach leverages the left subtree of the block merkle tree - starting from the root - in order to consolidate all the metadata in the left subtree, while preserving all the transactions in the right subtree. This approach does not touch the block header, hence preserving the existing physical infrastructure of Bitcoin. Also, this approach does not modify all the backend software infrastructure that supports wallets which expect to navigate blocks through the merkle tree.

[1] See *A taxonomy of the Bitcoin applicative landscape*, Joannes Vermorel, May 2018
https://blog.vermorel.com/journal/2018/5/7/a-taxonomy-of-the-bitcoin-applicative-landscape.html

Motivation for metadata

The use cases supporting metadata in blocks are numerous. Those use cases should not be confused with tokens or other alternative ways to insert arbitrary payloads into the blockchain: here the data of interest only refer to *cashcog* parts, that is, parts strictly required to keep Bitcoin operating as electronic cash.

Let's review some of those use cases for metadata in Bitcoin beyond the original use cases supported by the original block header:

- **Transforming big data into smaller data**: The essence of the UTXO commitment proposal is to offer a straightforward mechanism for *livechain* participants - including publishers - to restrict their data synchronization to the UTXO set, instead of synchronizing the entire blockchain. This proposal requires the inclusion of an extra 32-byte metadata in each block.
- **Self-describing infrastructure**: Publishers are expected to strengthen their networking links between each other. In practice, this process is rather ad-hoc. By having publishers making their endpoints public, some publishers can make themselves more easily reachable by their peers, which is in their interest. For such an approach to be automated, the inclusion of one or several endpoints in each block is required.
- **Facilitating maintenance and monitoring**: While all transaction statistics can be reconstructed by processing the entire block, in practice, if Bitcoin ever reaches gigabyte blocks and above, many mundane operations will be greatly facilitated if some basic statistics are included in every block. Such statistics would be more easily accessible and would help participants to react faster whenever a problem occurs in the network. This approach requires the inclusion of a short list of indicators in each block.
- **Establishing a functional fee market**: Transaction fees are necessary to ensure the long term viability of Bitcoin. However, transaction fees can dysfunction in multiple ways, as illustrated by exploding fees as observed in 2017 and 2018 in the Bitcoin Core network. By having publishers make their prices public through metadata, a functional fee market could be established[2]. This proposal requires the inclusion of a short list of pricing parameters in each block.

Then, revisiting the metadata also provide the opportunity to revisit the very use cases of the block header, as those use cases need to be improved upon (see *Limitations of the block header* below).

Thus, the possibility to have metadata is desirable for Bitcoin. Then, as we will see in the next section, each metadata should prove itself valuable enough to be worth its own overhead. In aggregate, metadata have to remain small.

[2] See *Midas, united non-colluding transaction fees for Bitcoin*, Joannes Vermorel, Mars 2018 (pdf)

Economics of metadata

The burden, in terms of computing resources, associated with metadata is more expensive to the ecosystem at large than UTXO entries because - for many practical purposes - it won't be possible or practical to prune the metadata. In particular, it is reasonable to assume that most *stateful*[3] apps will maintain an up-to-date copy of all the metadata which includes all the block headers plus the metadata subtree as presented below.

In terms of data storage, counting 80 bytes per block header and a cap at 1000 bytes per metadata subtree, the growth of the metadata as a whole remains capped at 57MB per year, which remains compatible with storage capacity of even entry-level devices when looking centuries ahead. While the limit at 1000 bytes - detailed below - for the metadata subtree is sufficiently low to be economically viable, it is also sufficient to cover the most pressing use cases, and probably a large variety of use cases that have not been considered yet.

In terms of computing costs, each standard metadata come with its own computational footprint. As a rule of thumb, any metadata entry representing more than a single digit percentage increase of the computational workload requires a rather intense scrutiny to make sure that it's really worth the extra burden put on the blockchain of Bitcoin.

Limitations of the block header

The block header is an 80-byte sequence which starts every Bitcoin block. This piece of data is of primary importance because it chains the blocks together. The block header is also important because it contains the metadata as originally implemented in Satoshi's client.

For the sake of clarity, let's briefly point out the content of the block header:

Bytes	Data type	Description
4	int32_t	Version
32	char[32]	Previous block header hash
32	char[32]	Merkle root hash
4	uint32_t	Unix epoch time
4	uint32_t	Target threshold for the proof of work.
4	uint32_t	Nonce

[3] See *A taxonomy of the Bitcoin applicative landscape*, Joannes Vermorel, May 2018 (link)

Some of the fields are not as useful as they were intended to be.

The *version* field is not really usable in Bitcoin because consensus rules have not been clearly established. As a result, the field can be liberally taken advantage of for multiple purposes, such as signaling upgrade intents. As it stands, this field cannot be relied upon for anything at present date.

The Unix epoch time is a 32-bit integer while will create a year 2038 problem[4] unless handled properly. This is feasible but undesirable as it needlessly complicates the software, creating more edge cases.

The nonce is too short considering that individual mining devices are now reaching terahashes per seconds. Thus, for a given merkle root hash, the majority of headers do not have a single solution reaching the difficulty target.

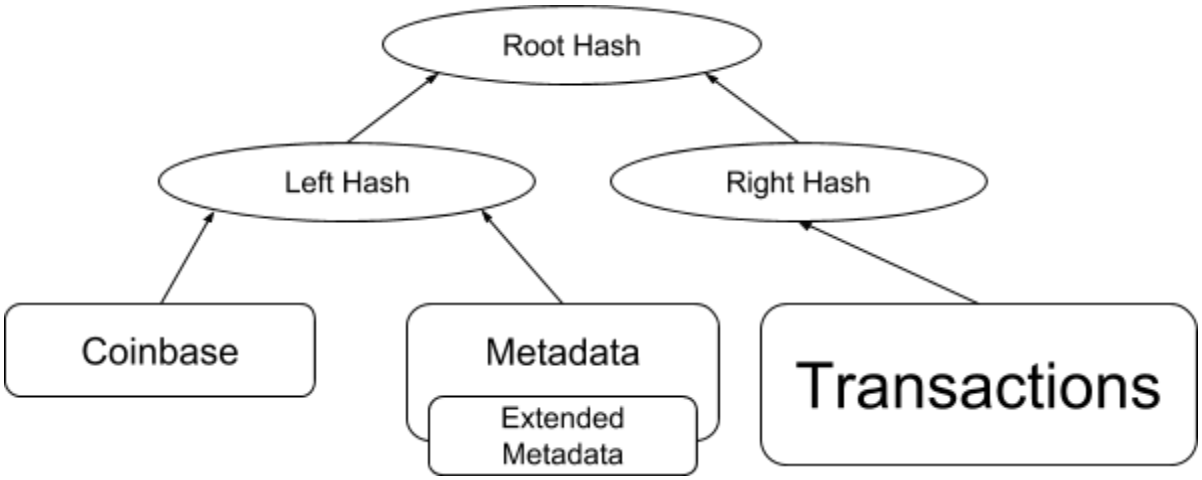
Finally, the main limitation of the block header is that it includes no provision for any extra metadata. Indeed, no matter the amount of careful planning, as technology progresses, some metadata will eventually be required to keep Bitcoin a competitive option against other electronic coins. The opposite proposition is equivalent to stating that Bitcoin cannot be further improved upon, which we believe to be unreasonable. As we will see in the following in greater detail, there are multiple compelling metadata entries that are desirable for Bitcoin beyond what is currently found in the block header.

Unfortunately, as existing mining equipments have been hardwired (asics) to the original header format, this header cannot be modified. Thus, the next best option is to introduce metadata through a secondary location.

Preserving backward compatibility through a subtree

As the block header cannot be modified, the metadata can be inserted as a subtree in the Merkle tree of the Bitcoin block itself. The subtree approach has the advantage of being strictly backward compatible for every piece of software that only interacts with the Merkle tree, and not the raw block transaction format. The schema below illustrates the proposed reorganization of the Merkle tree:

[4] See https://en.wikipedia.org/wiki/Year_2038_problem



The coinbase is maintained as the first transaction within the block as there are many apps that rely on this assumption. The coinbase is immediately followed by the metadata subtree. Finally, on the right subtree, the transactions remain arranged in a Merkle tree.

As the metadata subtree does not abide to the serialization format of a regular transaction, it breaks the backward compatibility of existing software which is dealing with flattened blocks. However, the metadata format is designed to ensure that the whole metadata subtree can still be skipped with minimal implementation efforts, not requiring to deserialize the metadata themselves.

An alternative option consists of putting the metadata into the coinbase itself. However, this option is worse than the metadata subtree because it entangles the design of Bitcoin transactions with the design of the metadata. While the present proposal does neither support nor oppose any evolution of the transaction format, putting the metadata in the coinbase would warranty complications for any future attempts at modifying the format of transactions.

Standard set of metadata entries

A metadata entry is referred to as *standard* if its presence is required and its correctness is enforced as part of the consensus rules. Consensus enforcement is required, otherwise the entry cannot be relied upon.

In this section, we review and motivate each one of the standard metadata to be added to the metadata subtree. The serialization format is detailed in the next section.

Secondary nonce. The primary nonce - located in the block header - does not have sufficient entropy to match the capabilities of mining devices that now range in the terahashes per second. Thus, this secondary nonce is included to allow the proof-of-work to happen while remaining maximally decoupled from the management of the blockchain and its transactions.

This design removes the need to modify the coinbase for purposes that are completely orthogonal to the economic reward associated with the successful propagation of a new block.

Blockheight. The blockheight is already present in the coinbase, however the intent is not to provide access to the blockheight value but merely to avoid colliding transaction identifiers. Thus, as the overhead is low, we propose to have this information as part of the metadata subtree to simplify the design of apps that do not require the level of security associated with a direct recomputation of the blockheight. This design also clarify the intent of the design of the coinbase.

Block size in byte and **block transaction count** (coinbase included). The correctness of those statistics is enforced by revised consensus rules. Those statistics have three purposes, listed by decreasing importance:

- They prevent an entire class of complications, related to streaming content on the receiving end without having prior knowledge about the size of the data to be received[5]. While those attacks can be mitigated with heuristics, those heuristics themselves introduce further classes of problems of their own, which are rather undesirable for infrastructure software.
- They facilitate the monitoring of the health of systems interacting with the blockchain. In terms of class of complexity, having those statistics as part of the metadata offers lower computing requirements for many apps. Those apps could be demoted from *livechain*[6] apps to simple *stateful* apps, eliminating big data requirements.
- Most of the proposals for adjustable block sizes resolve around taking into account the number of transactions and their weight, as well as the average observed fees. By having those details in the metadata, this calculation can potentially remain a small data processing.

Unix 64-bit time. This field removes any ambiguity on the time to be associated with the block, allowing a clean resolution of the year 2038 problem associated with the Unix 32-bit time field present in the block header. In order to avoid semantic inconsistencies, the lower 31[7] bits of this field are expected to match those found in the field of the block header.

Reserved fields. Enforced at zero, but already placed in the metadata header because those fields are expected to become standard in a near future.

[5] The Zip-bomb is a famous example of this class of problems. See https://en.wikipedia.org/wiki/Zip_bomb
[6] See *A taxonomy of the Bitcoin applicative landscape*, Joannes Vermorel, May 2018 (link).
[7] The Unix 32-bit time is represented as a signed 32-bit integer. Thus, in year 2038, the counter should reset at zero, thus leaving the sign bit at zero as well.

Nonstandard metadata entries

A nonstandard entry is not part of the consensus rule. This implies that the network at large cannot trust this data to be present, or correct, or even relevant for the network at large. Nevertheless, nonstandard entries are of interest as we will illustrate through practical examples in this section. In any case, the cap on the maximal size for the metadata subtree ensures an intended degree of scarcity for the metadata, as metadata are not intended to be a replacement for tokens.

Publisher HTTP endpoint: The participant who published the block includes an HTTP endpoint to be reached. This endpoint provides further mechanisms to perform Bitcoin-related services such as transaction propagation.

Public pricing of transaction fees: A functional fee market can emerge if a series of prior conditions are met. One of the potential ways to achieve a functional fee market consists of publishing the parameters of the pricing formula[8].

Also, non-standard entries offer a way to demonstrate how a new entry can prove itself both valuable and production-ready, before being promoted to a standard status. More generally, non-standard entries offer a practical way for miners to signal various intents to the market at large.

Metadata serialization format

The present section details the serialization format proposed for the metadata subtree. Overall, the metadata segment presents itself as a contiguous segment, with the length of the segment that can be obtained by reading a single field and skipping the rest of the subtree altogether.

Metadata subtree

The *metadata subtree* represents the top-level data structure that contains the metadata. The first portion of the data structure contains statically defined metadata entries. The second portion of the data structure contains a compact key-value store which is intended for metadata extensibility.

Bytes	Data type	Description
32	char[32]	**Secondary nonce**.
4	uint32_t	**Blockheight**. The number of blocks preceding the block starting at zero

[8] See *Midas, united non-colluding transaction fees for Bitcoin*, Joannes Vermorel, March 2018 (link)

		for the genesis block.
8	uint64_t	**Block size in byte**. Number of bytes for the whole blocks starting with the header and ending with the last transaction.
8	uint64_t	**Block transaction count**. Number of transactions for the whole block, counting the coinbase.
8	uint64_t	**Unix 64-bit timestamp**. The lower 31 bits of this field are expected to match the timestamp of the header.
32	char[32]	*Reserved field. Intended for a form of UTXO commitment*. Enforced at 0.
32	char[32]	**Hash of the right tree**. Ensures that the medata are covered by proof-of-work before the block is fully download; denying the zip-bomb like attack vector.
32	char[32]	**Hash of the extended metadata**. Starting from the next byte after this field.
1-9	VarInt	**Extended metadata byte length.** Number of bytes for the whole metadata segment, starting from the byte after the present field. Limited to 700 bytes[9].
1*	Extended Metadata Entry	A sequence of extended metadata entries.

The hash of the metadata subtree is computed by hashing all the fields from the *Secondary nonce* (inclusive) to *Extended metadata byte length* (inclusive). This design offers a mechanism to securely isolate the retrieval of the standard non-extended entries from the retrieval the extended entries. As a secondary benefit, this design also provides a way to avoid rehashing the extended metadata if those are not modified. Finally, the *Extended metadata byte length* is included as a standard non-extended field, in order to facilitate parsing - and potentially skipping - the whole metadata subtree.

All standard metadata presented above have a negligible performance impact - i.e. less than 0.1% - on processing the blockchain. The UTXO commitment is the only entry which comes with a non-trivial overhead, which goes beyond the scope of this document.

[9] The intent is to keep the combination of the block header plus the metadata subtree under 1KB, as it has some moderate benefits for embedded devices.

Extended metadata entry

Bytes	Data type	Description
4	uint32_t	**Metadata key**. Values lower than 65536 are not allowed, unless the entry is considered *standard*.
1-9	VarInt	**Entry value byte length**. A length of zero is allowed, possibly to signal an intent via a flag.
1*	char[*]	**Entry value**. A sequence of bytes that matches the length specified by the previous field.

The extended entries are expected to be sorted in increasing order against their respective metadata keys.

The keys with values lower than 65536 are reserved[10] for future entries which would become *standard*. As part of this proposal, there is no *standard* entry within the extended metadata segment.

[10] This is why this second part of the metadata subtree is referred to as the *extended entries* rather than the non-standard entries.