2018

MB8 COIN TECHNICAL DOCUMENT

PUBLISHED BY DANIEL POTTS

2ND SEPTEMBER 2018

Contents

Technical Document

Introduction

MB8Coin uses public-key cryptography, peer-to-peer networking, and proof-of-stake to process and verify payments. MB8Coins are sent (or signed over) from one address to another with each user potentially having many, many addresses. Each payment transaction is broadcast to the network and included in the blockchain so that the included coins cannot be spent twice. After 30 minutes to an hour, each transaction is locked in time by an amount of processing power and collateral value that continues to extend the blockchain. Using these techniques, MB8Coin provides a fast and extremely reliable payment network that anyone can use.

Common Standards

Hashes

Usually, when a hash is computed within mb8coin, it is computed twice. Most of the time SHA-256 hashes are used, however RIPEMD-160 is also used when a shorter hash is desirable (for example when creating a mb8coin address).
Example of double-SHA-256 encoding of string “hello”:

```
hello
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 (first round of sha-256)
9595c9df90075148eb06860365df33584b75bff782a510c6cd4883a419833d50 (second round of sha-256)
```

For mb8coin addresses (RIPEMD-160) this would give:

```
hello
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 (first round is sha-256)
b6a9c8c230722b7c748331a8b450f05566dc7d0f (with ripemd-160)
```

Merkle Trees

Merkle trees are binary trees of hashes. Merkle trees in mb8coin use a double SHA-256, the SHA-256 hash of the SHA-256 hash of something.
If, when forming a row in the tree (other than the root of the tree), it would have an odd number of elements, the final double-hash is duplicated to ensure that the row has an even number of hashes.
First form the bottom row of the tree with the ordered double-SHA-256 hashes of the byte streams of the transactions in the block.
Then the row above it consists of half that number of hashes. Each entry is the double-SHA-256 of the 64-byte concatenation of the corresponding two hashes below it in the tree.
This procedure repeats recursively until we reach a row consisting of just a single double-hash. This is the Merkle root of the tree.
For example, imagine a block with three transactions a, b and c. The Merkle tree is:

```
d1          =
dhash(a) d2 =
dhash(b) d3 =
dhash(c)                # a, b, c are 3. that's an odd number, so we
                        # take the c twice

d5 = dhash(d1 concat d2)
d6 = dhash(d3 concat d4)

d7 = dhash(d5 concat d6)
```

where

```
dhash(a) = sha256(sha256(a))
```

d7 is the Merkle root of the 3 transactions in this block.
Note: Hashes in Merkle Tree displayed in the Block Explorer are of little-endian notation. For some implementations and calculations, the bytes need to be reversed before they are hashed, and again after the hashing operation.

Signatures

uses Elliptic Curve Digital Signature Algorithm (ECDSA) to sign transactions.
For ECDSA the secp256k1 curve from http://www.secg.org/sec2-v2.pdf is used.
Public keys (in scripts) are given as 04 <x> <y> where x and y are 32 byte big-endian integers representing the coordinates of a point on the curve or in compressed form given as <sign> <x> where <sign> is 0x02 if y is even and 0x03 if y is odd.
Signatures use DER encoding to pack the r and s components into a single byte stream (this is also what OpenSSL produces by default).

Transaction Verification

Transactions are cryptographically signed records that reassign ownership of MB8Coins to new addresses. Transactions have inputs - records which reference the funds from other previous transactions - and outputs - records which determine the new owner of the transferred MB8Coins, and which will be referenced as inputs in future transactions as those funds are respent.

Each *input* must have a cryptographic digital signature that unlocks the funds from the prior transaction. Only the person possessing the appropriate private key is able to create a satisfactory signature; this in effect ensures that funds can only be spent by their owners.

Each *output* determines which MB8Coin address (or other criteria, see Script) is the recipient of the funds.

In a transaction, the sum of all inputs must be equal to or greater than the sum of all outputs. If the inputs exceed the outputs, the difference is considered a transaction fee, and is redeemable by whoever first includes the transaction into the block chain.

A special kind of transaction, called a coinbase transaction, has no inputs. It is created by miners or stakers, and there is one coinbase transaction per block. Because each block comes with a reward of newly created MB8Coins, the first transaction of a block is, with few exceptions, the transaction that grants those coins to their recipient (the miner). In addition to the newly created

MB8Coins, the coinbase/coinstake transaction is also used for assigning the recipient of any transaction fees that were paid within the other transactions being included in the same block. Once coin staking takes place, the coinbase transaction becomes empty and instead a coinstake transaction is responsible for minting new coins. In a coinstake transaction the staker consumes a previous UTXO and marks it as collateral, they then pay back this collateral to themselves plus the staking reward, as well as any fees from transactions. If a coinstaker is seen as trying to be malicious with the blockchain, their collateral can be lost as a result. The coinbase/coinstake transaction can assign the entire reward to a single MB8Coin address, or split it in portions among multiple addresses, just like any other transaction. Coinbase transactions always contain outputs totalling the sum of the block reward plus all transaction fees collected from the other transactions in the same block.
The coinbase transaction in block zero cannot be spent. This is due to a quirk of the reference client implementation that would open the potential for a block chain fork if some nodes accepted the spend and others did not[1].
Most MB8Coin outputs encumber the newly transferred coins with a single ECDSA private key. The actual record saved with inputs and outputs isn't necessarily a key, but a script. MB8Coin uses an interpreted scripting system to determine whether an output's criteria have been satisfied, with which more complex operations are possible, such as outputs that require two ECDSA signatures, or two-of-three-signature schemes. An output that references a single MB8Coin address is a typical output; an output actually contains this information in the form of a script that requires a single ECDSA signature (see OP_CHECKSIG). The output script specifies what must be provided to unlock the funds later, and when the time comes in the future to spend the transaction in another input, that input must provide all of the thing(s) that satisfy the requirements defined by the original output script.

Addresses

A mb8coin address is in fact the hash of a ECDSA public key, computed this way:

```
Version = 1 byte of 50; on the test network, this is 1 byte of 111
Key hash = Version concatenated with RIPEMD-160(SHA-256(public key))
Checksum = 1st 4 bytes of SHA-256(SHA-256(Key hash))
MB8Coin Address = Base58Encode(Key hash concatenated with Checksum)
```

The Base58 encoding used is home made, and has some differences. Especially, leading zeroes are kept as single zeroes when conversion happens.

Common structures

Almost all integers are encoded in little endian. Only IP or port number are encoded big endian.

Message structure

Field Size	Description	Data type	Comments
4	magic	uint32_t	Magic value indicating message origin network, and used to seek to next message when stream state is unknown
12	command	char[12]	ASCII string identifying the packet content, NULL padded (non-NULL padding results in packet rejected)
4	length	uint32_t	Length of payload in number of bytes
4	checksum	uint32_t	First 4 bytes of sha256(sha256(payload))

Field Size	Description	Data type	Comments
?	payload	uchar[]	The actual data

Known magic values:

Network	Magic Value	Sent over wire as
MB8Coin	0x3F9A2476	3F 9A 24 76
MB8Coin	0xD9B4BEF9	F9 BE B4 D9

Variable length integer

Integer can be encoded depending on the represented value to save space. Variable length integers always precede an array/vector of a type of data that may vary in length. Longer numbers are encoded in little endian.

Value	Storage length	Format
< 0xFD	1	uint8_t
<= 0xFFFF	3	0xFD followed by the length as uint16_t
<= 0xFFFF FFFF	5	0xFE followed by the length as uint32_t
-	9	0xFF followed by the length as uint64_t

Variable length string

Variable length string can be stored using a variable length integer followed by the string itself.

Field Size	Description	Data type	Comments
?	length	var_int	Length of the string
?	string	char[]	The string itself (can be empty)

Network address

When a network address is needed somewhere, this structure is used. Network addresses are not prefixed with a timestamp in the version message.

Field Size	Description	Data type	Comments
4	time	uint32	the Time (version >= 31402). Not present in version message.
8	services	uint64_t	same service(s) listed in version

Field Size	Description	Data type	Comments
16	IPv6/4	char[16]	IPv6 address. Network byte order. The original client only supported IPv4 and only read the last 4 bytes to get the IPv4 address. However, the IPv4 address is written into the message as a 16 byte IPv4-mapped IPv6 address (12 bytes 00 00 00 00 00 00 00 00 00 00 FF FF, followed by the 4 bytes of the IPv4 address).
2	port	uint16_t	port number, network byte order

Inventory Vectors

Inventory vectors are used for notifying other nodes about objects they have or data which is being requested.

Inventory vectors consist of the following data format:

Field Size	Description	Data type	Comments
4	type	uint32_t	Identifies the object type linked to this inventory
32	hash	char[32]	Hash of the object

The object type is currently defined as one of the following possibilities:

Value	Name	Description
0	ERROR	Any data of with this number may be ignored
1	MSG_TX	Hash is related to a transaction
2	MSG_BLOCK	Hash is related to a data block
3	MSG_FILTERED_BLOCK	Hash of a block header; identical to MSG_BLOCK. Only to be used in getdata message. Indicates the reply should be a merkleblock message rather than a block message; this only works if a bloom filter has been set.
4	MSG_CMPCT_BLOCK	Hash of a block header; identical to MSG_BLOCK. Only to be used in getdata message. Indicates the reply should be a cmpctblock message. See BIP 152 for more info.

Other Data Type values are considered reserved for future implementations.

Block Headers

Block headers are sent in a headers packet in response to a getheaders message.

Field Size	Description	Data type	Comments
4	version	int32_t	Block version information (note, this is signed)

Field Size	Description	Data type	Comments
32	prev_block	char[32]	The hash value of the previous block this particular block references
32	merkle_root	char[32]	The reference to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A timestamp recording when this block was created (Will overflow in 2106)
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block… to allow variations of the header and compute different hashes
1	txn_count	var_int	Number of transaction entries, this value is always 0

Differential encoding

Several uses of CompactSize below are "differentially encoded". For these, instead of using raw indexes, the number encoded is the difference between the current index and the previous index, minus one. For example, a first index of 0 implies a real index of 0, a second index of 0 thereafter refers to a real index of 1, etc.

PrefilledTransaction

A PrefilledTransaction structure is used in HeaderAndShortIDs to provide a list of a few transactions explicitly.

Field Name	Type	Size	Encoding	Purpose
index	CompactSize	1, 3 bytes	Compact Size, differentially encoded since the last PrefilledTransaction in a list	The index into the block at which this transaction is
tx	Transaction	variable	As encoded in tx messages	The transaction which is in the block at index index.

HeaderAndShortIDs

A HeaderAndShortIDs structure is used to relay a block header, the short transactions IDs used for matching already-available transactions, and a select few transactions which we expect a peer may be missing.

Field Name	Type	Size	Encoding	Purpose
header	Block header	80 bytes	First 80 bytes of the block as defined by the encoding used by "block" messages	The header of the block being provided
nonce	uint64_t	8 bytes	Little Endian	A nonce for use in short transaction ID calculations

Field Name	Type	Size	Encoding	Purpose
shortids_length	CompactSize	1 or 3 bytes	As used to encode array lengths elsewhere	The number of short transaction IDs in shortids (ie block tx count - prefilledtxn_length)
shortids	List of 6-byte integers	6*shortids_length bytes	Little Endian	The short transaction IDs calculated from the transactions which were not provided explicitly in prefilledtxn
prefilledtxn_length	CompactSize	1 or 3 bytes	As used to encode array lengths elsewhere	The number of prefilled transactions in prefilledtxn (ie block tx count - shortids_length)
prefilledtxn	List of PrefilledTransactions	variable size*prefilledtxn_length	As defined by PrefilledTransaction definition, above	Used to provide the coinbase transaction and a select few which we expect a peer may be missing

BlockTransactionsRequest

A BlockTransactionsRequest structure is used to list transaction indexes in a block being requested.

Field Name	Type	Size	Encoding	Purpose
blockhash	Binary blob	32 bytes	The output from a double-SHA256 of the block header, as used elsewhere	The blockhash of the block which the transactions being requested are in
indexes_length	CompactSize	1 or 3 bytes	As used to encode array lengths elsewhere	The number of transactions being requested
indexes	List of CompactSizes	1 or 3 bytes*indexes_length	Differentially encoded	The indexes of the transactions being requested in the block

BlockTransactions

A BlockTransactions structure is used to provide some of the transactions in a block, as requested.

Field Name	Type	Size	Encoding	Purpose
blockhash	Binary blob	32 bytes	The output from a double-SHA256 of the block header, as used elsewhere	The blockhash of the block which the transactions being provided are in
transactions_length	CompactSize	1 or 3 bytes	As used to encode array lengths elsewhere	The number of transactions provided
transactions	List of Transactions	variable	As encoded in tx messages	The transactions provided

Short transaction ID

Short transaction IDs are used to represent a transaction without sending a full 256-bit hash. They are calculated by:
single-SHA256 hashing the block header with the nonce appended (in little-endian)
Running SipHash-2-4 with the input being the transaction ID and the keys (k0/k1) set to the first two little-endian 64-bit integers from the above hash, respectively.
Dropping the 2 most significant bytes from the SipHash output to make it 6 bytes.

Message types

version

When a node creates an outgoing connection, it will immediately advertise its version. The remote node will respond with its version. No further communication is possible until both peers have exchanged their version.

Payload:

Field Size	Description	Data type	Comments
4	version	int32_t	Identifies protocol version being used by the node
8	services	uint64_t	bitfield of features to be enabled for this connection
8	timestamp	int64_t	standard UNIX timestamp in seconds
26	addr_recv	net_addr	The network address of the node receiving this message
26	addr_from	net_addr	The network address of the node emitting this message
8	nonce	uint64_t	Node random nonce, randomly generated every time a version packet is sent. This nonce is used to detect connections to self.
?	user_agent	var_str	User Agent (0x00 if string is 0 bytes long)
4	start_height	int32_t	The last block received by the emitting node
1	relay	bool	Whether the remote peer should announce relayed transactions or not

A "verack" packet shall be sent if the version packet was accepted.

```
0000   3F 9A 24 76 76 65 72 73 69 6f 6e 00 00 00 00 00  ....version.....
0010   64 00 00 00 35 8d 49 32 62 ea 00 00 01 00 00 00  d...5.I2b.......
0020   00 00 00 00 11 b2 d0 50 00 00 00 00 01 00 00 00  .......P........
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff  ................
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050   00 00 00 00 00 00 00 00 ff ff 00 00 00 00 00 00  ................
0060   3b 2e b3 5d 8c e6 17 65 0f 2f 4d 42 38 43 6f 69  ;..]...e./MB8Coi
0070   6e 3a 31 2e 30 2e 30 2f c0 3e 03 00              n:1.0.0/.>..

Message Header:
```

```
 3F 9A 24 76
-Main network magic bytes
 76 65 72 73 69 6F 6E 00 00 00 00 00
-"version" command
 64 00 00 00
-Payload is 100 bytes long
 3B 64 8D 5A
-payload checksum

Version message:
 62 EA 00 00
-60002 (protocol version 60002)
 01 00 00 00 00 00 00 00
-1 (NODE_NETWORK services)
 11 B2 D0 50 00 00 00 00
- Tue Dec 18 10:12:33 PST 2012
 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00
00 00 - Recipient address info - see Network Address
 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00
00 00 - Sender address info - see Network Address
 3B 2E B3 5D 8C E6 17 65
-Node ID
 0F 2F 4D 42 387 43 6F 69 6E 3A 31 2E 30 2E 30 2F
-"/MB8Coin:1.0.0/" sub-version string (string is 15 bytes long)
 C0 3E 03 00
-Last block sending node has is block #212672
```

verack

The verack message is sent in reply to version. This message consists of only a message header with the command string “verack".

Hexdump of the verack message:

```
0000   3F 9A 24 76 61 64 64 72  00 00 00 00 00 00 00
00    ....addr........
0010   1F 00 00 00 ED 52 39 9B  01 E2 15 10 4D 01 00
00    .....R9.....M...
0020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00
FF    ...............
0030   FF 0A 00 00 01 20 8D                                ..... .

Message Header:
 3F 9A 24 76                                      - Main network magic
bytes
 61 64 64 72  00 00 00 00 00 00 00 00              - "addr"
 1F 00 00 00                                       - payload is 31 bytes
long
 ED 52 39 9B                                       - checksum of payload
```

```
Payload:
 01                                                - 1 address in this
message

Address:
 E2 15 10 4D                                       - Mon Dec 20 21:50:10
EST 2010 (only when version is >= 31402)
 01 00 00 00 00 00 00 00                           - 1 (NODE_NETWORK
service - see version message)
 00 00 00 00 00 00 00 00 00 00 FF FF 0A 00 00 01 - IPv4: 10.0.0.1,
IPv6: ::ffff:10.0.0.1 (IPv4-mapped IPv6 address)
 20 8D                                             - port 8333
```

inv

Allows a node to advertise its knowledge of one or more objects. It can be received unsolicited, or in reply to getblocks.

Payload (maximum 50,000 entries, which is just over 1.8 megabytes):

Field Size	Description	Data type	Comments
?	count	var_int	Number of inventory entries
36x?	inventory	inv_vect[]	Inventory vectors

getdata

getdata is used in response to inv, to retrieve the content of a specific object, and is usually sent after receiving an inv packet, after filtering known elements. It can be used to retrieve transactions, but only if they are in the memory pool or relay set - arbitrary access to transactions in the chain is not allowed to avoid having clients start to depend on nodes having full transaction indexes (which modern nodes do not).

Payload (maximum 50,000 entries, which is just over 1.8 megabytes):

Field Size	Description	Data type	Comments
?	count	var_int	Number of inventory entries
36x?	inventory	inv_vect[]	Inventory vectors

notfound

notfound is a response to a getdata, sent if any requested data items could not be relayed, for example, because the requested transaction was not in the memory pool or relay set.

Field Size	Description	Data type	Comments
?	count	var_int	Number of inventory entries

Field Size	Description	Data type	Comments
36x?	inventory	inv_vect[]	Inventory vectors

getblocks

Return an inv packet containing the list of blocks starting right after the last known hash in the block locator object, up to hash_stop or 500 blocks, whichever comes first.
The locator hashes are processed by a node in the order as they appear in the message. If a block hash is found in the node's main chain, the list of its children is returned back via the inv message and the remaining locators are ignored, no matter if the requested limit was reached, or not.

To receive the next blocks hashes, one needs to issue getblocks again with a new block locator object. Keep in mind that some clients may provide blocks which are invalid if the block locator object contains a hash on the invalid branch.

Payload:

Field Size	Description	Data type	Comments
4	version	uint32_t	the protocol version
1+	hash count	var_int	number of block locator hash entries
32+	block locator hashes	char[32]	block locator object; newest back to genesis block (dense to start, but then sparse)
32	hash_stop	char[32]	hash of the last desired block; set to zero to get as many blocks as possible (500)

getheaders

Return a headers packet containing the headers of blocks starting right after the last known hash in the block locator object, up to hash_stop or 2000 blocks, whichever comes first. To receive the next block headers, one needs to issue getheaders again with a new block locator object. The getheaders command is used by thin clients to quickly download the block chain where the contents of the transactions would be irrelevant (because they are not ours). Keep in mind that some clients may provide headers of blocks which are invalid if the block locator object contains a hash on the invalid branch.

Payload:

Field Size	Description	Data type	Comments
4	version	uint32_t	the protocol version
1+	hash count	var_int	number of block locator hash entries
32+	block locator hashes	char[32]	block locator object; newest back to genesis block (dense to start, but then sparse)
32	hash_stop	char[32]	hash of the last desired block header; set to zero to get as many blocks as possible (2000)

tx

tx describes a mb8coin transaction, in reply to getdata

Field Size	Description	Data type	Comments
4	version	int32_t	Transaction data format version (note, this is signed)
0 or 2	flag	optional uint8_t[2]	If present, always 0001, and indicates the presence of witness data
1+	tx_in count	var_int	Number of Transaction inputs (never zero)
41+	tx_in	tx_in[]	A list of 1 or more transaction inputs or sources for coins
1+	tx_out count	var_int	Number of Transaction outputs
9+	tx_out	tx_out[]	A list of 1 or more transaction outputs or destinations for coins
0+	tx_witnesses	tx_witness[]	A list of witnesses, one for each input; omitted if flag is omitted above
4	lock_time	uint32_t	The block number or timestamp at which this transaction is unlocked: 0 = Not Locked < 500000000 = Block number at which this transaction is unlocked >= 500000000 = UNIX timestamp at which this transaction is unlocked. If all TxIn inputs have final (0xffffffff) sequence numbers then lock_time is irrelevant. Otherwise, the transaction may not be added to a block until after lock_time (see NLockTime).

TxIn consists of the following fields:

Field Size	Description	Data type	Comments
36	previous_output	outpoint	The previous output transaction reference, as an OutPoint structure
1+	script length	var_int	The length of the signature script
?	signature script	uchar[]	Computational Script for confirming transaction authorization
4	sequence	uint32_t	Transaction version as defined by the sender. Intended for "replacement" of transactions when information is updated before inclusion into a block.

The OutPoint structure consists of the following fields:

Field Size	Description	Data type	Comments
32	hash	char[32]	The hash of the referenced transaction.

Field Size	Description	Data type	Comments
4	index	uint32_t	The index of the specific output in the transaction. The first output is 0, etc.

The Script structure consists of a series of pieces of information and operations related to the value of the transaction.

The TxOut structure consists of the following fields:

Field Size	Description	Data type	Comments
8	value	int64_t	Transaction Value
1+	pk_script length	var_int	Length of the pk_script
?	pk_script	uchar[]	Usually contains the public key as a MB8Coin script setting up conditions to claim this output.

The TxWitness structure consists of a var_int count of witness data components, followed by (for each witness data component) a var_int length of the component and the raw component data itself.

```
000000     3F 9A 24 76 74 78 00 00  00 00 00 00 00 00 00
00   ....tx..........
000010     02 01 00 00 E2 93 CD BE  01 00 00 00 01 6D BD
DB   .............m..
000020     08 5B 1D 8A F7 51 84 F0  BC 01 FA D5 8D 12 66 E9   .
[...Q........f.
000030     B6 3B 50 88 19 90 E4 B4  0D 6A EE 36 29 00 00 00   .;P......j.
6)...
000040     00 8B 48 30 45 02 21 00  F3 58 1E 19 72 AE 8A
C7   ..H0E.!..X..r...
000050     C7 36 7A 7A 25 3B C1 13  52 23 AD B9 A4 68 BB 3A   .
6zz%;..R#...h.:
000060     59 23 3F 45 BC 57 83 80  02 20 59 AF 01 CA 17 D0   Y#?E.W...
Y.....
000070     0E 41 83 7A 1D 58 E9 7A  A3 1B AE 58 4E DE C2
8D   .A.z.X.z...XN...
000080     35 BD 96 92 36 90 91 3B  AE 9A 01 41 04 9C 02 BF
5...6..;...A....
000090     C9 7E F2 36 CE 6D 8F E5  D9 40 13 C7 21 E9 15 98   .~.
6.m...@..!...
0000A0     2A CD 2B 12 B6 5D 9B 7D  59 E2 0A 84 20 05 F8 FC   *.+..].}
Y... ...
0000B0     4E 02 53 2E 87 3D 37 B9  6F 09 D6 D4 51 1A DA 8F
N.S..=7.o...Q...
0000C0     14 04 2F 46 61 4A 4C 70  C0 F1 4B EF F5 FF FF FF   ../
FaJLp..K.....
0000D0     FF 02 40 4B 4C 00 00 00  00 00 19 76 A9 14 1A
A0   ..@KL. .... v....
```

```
0000E0     CD 1C BE A6 E7 45 8A 7A  BA D5 12 A9 D9 EA 1A
FB   .....E.z........
0000F0     22 5E 88 AC 80 FA E9 C7  00 00 00 00 19 76 A9 14
"^. ......... v..
000100     0E AB 5B EA 43 6A 04 84  CF AB 12 48 5E FD A0 B7   ..
[.Cj.....H^...
000110     8B 4E CC 52 88 AC 00 00  00 00                    .N.R......


Message header:
 3F 9A 24 76                                        - main network magic
bytes
 74 78 00 00 00 00 00 00 00 00 00 00                - "tx" command
 02 01 00 00                                        - payload is 258 bytes
long
 E2 93 CD BE                                        - checksum of payload

Transaction:
 01 00 00 00                                        - version

Inputs:
 01                                                 - number of
transaction inputs

Input 1:
 6D BD DB 08 5B 1D 8A F7  51 84 F0 BC 01 FA D5 8D   - previous output
(outpoint)
 12 66 E9 B6 3B 50 88 19  90 E4 B4 0D 6A EE 36 29
 00 00 00 00

 8B                                                 - script is 139 bytes
long

 48 30 45 02 21 00 F3 58  1E 19 72 AE 8A C7 C7 36   - signature script
(scriptSig)
 7A 7A 25 3B C1 13 52 23  AD B9 A4 68 BB 3A 59 23
 3F 45 BC 57 83 80 02 20  59 AF 01 CA 17 D0 0E 41
 83 7A 1D 58 E9 7A A3 1B  AE 58 4E DE C2 8D 35 BD
 96 92 36 90 91 3B AE 9A  01 41 04 9C 02 BF C9 7E
 F2 36 CE 6D 8F E5 D9 40  13 C7 21 E9 15 98 2A CD
 2B 12 B6 5D 9B 7D 59 E2  0A 84 20 05 F8 FC 4E 02
 53 2E 87 3D 37 B9 6F 09  D6 D4 51 1A DA 8F 14 04
 2F 46 61 4A 4C 70 C0 F1  4B EF F5

 FF FF FF FF                                        - sequence

Outputs:
```

```
 02                                                  - 2 Output
Transactions

Output 1:
 40 4B 4C 00 00 00 00 00                            - 0.05 MB8 (5000000)
 19                                                 - pk_script is 25
bytes long

 76 A9 14 1A A0 CD 1C BE  A6 E7 45 8A 7A BA D5 12   - pk_script
 A9 D9 EA 1A FB 22 5E 88  AC

Output 2:
 80 FA E9 C7 00 00 00 00                            - 33.54 MB8
(3354000000)
 19                                                 - pk_script is 25
bytes long

 76 A9 14 0E AB 5B EA 43  6A 04 84 CF AB 12 48 5E   - pk_script
 FD A0 B7 8B 4E CC 52 88  AC

Locktime:
 00 00 00 00                                        - lock time
```

block

The block message is sent in response to a getdata message which requests transaction information from a block hash.

Field Size	Description	Data type	Comments
4	version	int32_t	Block version information (note, this is signed)
32	prev_block	char[32]	The hash value of the previous block this particular block references
32	merkle_root	char[32]	The reference to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A Unix timestamp recording when this block was created (Currently limited to dates before the year 2106!)
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block… to allow variations of the header and compute different hashes
?	txn_count	var_int	Number of transaction entries
?	txns	tx[]	Block transactions, in format of "tx" command

The SHA256 hash that identifies each block (and which must have a run of 0 bits) is calculated from the first 6 fields of this structure (version, prev_block, merkle_root, timestamp, bits, nonce, and standard SHA256 padding, making two 64-byte chunks in all) and not from the complete block. To calculate the hash, only two chunks need to be processed by the X13 algorithm. Since the nonce field is in the second chunk, the first chunk stays constant during mining and therefore

only the second chunk needs to be processed. However, an MB8Coin hash is the hash of the hash, so two rounds are needed for each mining iteration. See block hashing algorithm for details.

headers

The headers packet returns block headers in response to a getheaders packet.

Payload:

Field Size	Description	Data type	Comments
?	count	var_int	Number of block headers
81x?	headers	block_header[]	Block headers

Note that the block headers in this packet include a transaction count (a var_int, so there can be more than 81 bytes per header) as opposed to the block headers that are hashed by miners.

getaddr

The getaddr message sends a request to a node asking for information about known active peers to help with finding potential nodes in the network. The response to receiving this message is to transmit one or more addr messages with one or more peers from a database of known active peers. The typical presumption is that a node is likely to be active if it has been sending a message within the last three hours.

No additional data is transmitted with this message.

ping

The ping message is sent primarily to confirm that the TCP/IP connection is still valid. An error in transmission is presumed to be a closed connection and the address is removed as a current peer.

Payload:

Field Size	Description	Data type	Comments
8	nonce	uint64_t	random nonce

pong

The pong message is sent in response to a ping message. In modern protocol versions, a pong response is generated using a nonce included in the ping.

Payload:

Field Size	Description	Data type	Comments
8	nonce	uint64_t	nonce from ping

reject

The reject message is sent when messages are rejected.

Payload:

Field Size	Description	Data type	Comments
1+	message	var_str	type of message rejected
1	ccode	char	code relating to rejected message
1+	reason	var_str	text version of reason for rejection
0+	data	char	Optional extra data provided by some errors. Currently, all errors which provide this field fill it with the TXID or block header hash of the object being rejected, so the field is 32 bytes

CCodes

Value	Name	
0x01	REJECT_MALFORMED	
0x10	REJECT_INVALID	
0x11	REJECT_OBSOLETE	
0x12	REJECT_DUPLICATE	
0x40	REJECT_NONSTANDARD	
0x41	REJECT_DUST	
0x42	REJECT_INSUFFICIENTFEE	
0x43	REJECT_CHECKPOINT	

sendheaders

Request for Direct headers announcement.

Upon receipt of this message, the node is be permitted, but not required, to announce new blocks by headers command (instead of inv command).

No additional data is transmitted with this message.

Block Hashing Algorithm - X13

Security

The algorithm uses eleven hashing functions from the Blake algorithm to the Keccak algorithm making it very secure which really is needed for coins that do so well for CPU's

Effectiveness

The X13 Algorithm gives amazingly fast hashes for both GPU's and CPU's. It also uses less power than other hashing algorithms such as SH256. This makes it ideal for staking as it can effectively hash without costing the user a high electricity bill.

Power Cost

Due to how effective the X13 algorithm is, cpus do not require that much power in order to mine it. Therefore, you will see significantly lower electricity costs at the end of the month. This makes coins running this algorithm to be a favourite in places where electricity costs are far from bearable and great for running on a cloud server.

MB8Coin Core Wallet

The MB8Coin Core client stores private key information in a file named wallet.dat following the so called "bitkeys" format.
It contains:

- keypairs for each of your addresses
- transactions done from/to your addresses
- user preferences
- default key
- reserve keys
- accounts
- a version number
- Key pool
- Since 0.3.21: information about the current best chain, to be able to rescan automatically when restoring from a backup.

The wallet.dat file is located in the MB8Coin data directory and may be encrypted with a password.
It is intended that a wallet file be used on only one installation of MB8Coin at a time. Attempting to clone a wallet file for use on multiple computers will result in "weird behavior".
The format of this file is Berkeley DB.

Addresses

An **MB8Coin address**, or simply **address**, is an identifier of 26-35 alphanumeric characters, beginning with the letter `M`, that represents a possible destination for an mb8coin payment. Addresses can be generated at no cost by any user of MB8Coin. It is also possible to get an MB8Coin address using an account at an exchange or online wallet service.

An MB8Coin address is a single-use token

Like e-mail addresses, you can send MB8Coins to a person by sending mb8coins to one of their addresses. However, *unlike* e-mail addresses, people have many different MB8Coin addresses and a unique address should be used for each transaction.

Addresses can be created offline

Creating addresses can be done without an Internet connection and does not require any contact or registration with the MB8Coin network. It is possible to create large batches of addresses offline using freely available software tools. Generating batches of addresses is useful in several scenarios, such as e-commerce websites where a unique pre-generated address is dispensed to each customer.

Addresses are case sensitive and exact

MB8Coin addresses should be copied and pasted using the computer's clipboard wherever possible. If you hand-key a MB8Coin address, and each character is not transcribed exactly - including capitalization - the incorrect address will most likely be rejected by the MB8Coin software. You will have to check your entry and try again.
The probability that a mistyped address is accepted as being valid is 1 in 2^{32}, that is, approximately 1 in 4.29 billion.

Proving you receive with an address

The MB8Coin wallet has a function to "sign" a message, proving the entity receiving funds with an address has agreed to the message. This can be used to, for example, finalise a contract in a cryptographically provable way prior to making payment for it.
Some services will also piggy-back on this capability by dedicating a specific address for authentication only, in which case the address should never be used for actual MB8Coin transactions. When you login to or use their service, you will provide a signature proving you are the same person with the pre-negotiated address.
It is important to note that these signatures only prove one receives with an address. Since MB8Coin transactions do not have a "from" address, you cannot prove you are the *sender* of funds.

Misconceptions

Address reuse

Addresses are not intended to be used more than once, and doing so has numerous problems associated.

Address balances

Addresses are not wallets nor accounts, and do not carry balances. They only receive funds, and you do not send "from" an address at any time. Various confusing services and software display *mb8coins received with an address, minus mb8coins sent in random unrelated transactions* as an "address balance", but this number is not meaningful: it does not imply the recipient of the mb8coins sent to the address has spent them, nor that they still have the MB8Coins received.
An example of mb8coin loss resulting from this misunderstanding is when people believed their address contained 3MB8. They spend 0.5MB8 and believe the address now contains 2.5MB8 when actually it contains zero. The remaining 2.5MB8 is transferred to a change address which is not backed up and therefore lost. This has happened on a few occasions to users in the MB8Coin network.

"From" addresses

MB8Coin transactions do not have any kind of origin-, source- or "from" address.

Key pool

New public and private keys are pre-generated and stored in a queue before use.
This pooling feature exists so backups of the wallet would have a certain number of keys that would be used in the future. By default, the number of entries in the queue is 100.
A command line option allows a greater or lesser number of keys to be maintained in the keypool.

mb8coin.conf Configuration File

All command-line options (except for -conf) may be specified in a configuration file, and all configuration file options may also be specified on the command line. Command-line options override values set in the configuration file.
The configuration file is a list of setting=value pairs, one per line, with optional comments starting with the '#' character.
The configuration file is not automatically created; you can create it using your favorite plain-text editor. By default, MB8Coin (or mb8coind) will look for a file named 'mb8coin.conf' in the MB8Coin data directory, but both the data directory and the configuration file path may be changed using the -datadir and -conf command-line arguments.

Operating System	Default MB8Coin datadir	Typical path to configuration file
Windows	%APPDATA%\MB8Coin\	C:\Users\username\AppData\Roaming\MB8Coin\mb8coin.conf
Linux	$HOME/.MB8Coin/	/home/username/.mb8coin/mb8coin.conf
Mac OSX	$HOME/Library/Application Support/MB8Coin/	/Users/username/Library/Application Support/MB8Coin/mb8coin.conf

Note: if running MB8Coin in testnet mode, the sub-folder "testnet" will be appended to the data directory automatically.

Sample mb8coin.conf

```
##
## mb8coin.conf configuration file. Lines beginning with # are comments.
##

# Staking setting
# staking is tuned on, to turn off set to
0 staking=1

# Network-related settings:

# Run on the test network instead of the real mb8coin network.
#testnet=0

# Run a regression test network
#regtest=0

# Connect via a SOCKS5 proxy
#proxy=127.0.0.1:9050

# Bind to given address and always listen on it. Use [host]:port
notation for IPv6
#bind=<addr>

# Bind to given address and whitelist peers connecting to it.
Use [host]:port notation for IPv6
#whitebind=<addr>

############################################################################
##                    Quick Primer on addnode vs connect                  ##
##    Let's say for instance you use addnode=4.2.2.4                      ##
##    addnode will connect you to and tell you about the                  ##
##       nodes connected to 4.2.2.4.          In addition it will tell ##
##       the other nodes connected to         it  that  you  exist  so    ##
##       they can connect to you.                                         ##
##    connect will not do the above when you 'connect' to it.##
##       It will *only* connect you to 4.2.2.4 and no one else.##
```

```
##                                                              ##
##  So if you're behind a firewall, or have other problems      ##
##  finding nodes, add some using 'addnode'.                    ##
##                                                              ##
##  If you want to stay private, use 'connect' to only          ##
##  connect to "trusted" nodes.                                 ##
##                                                              ##
##  If you run multiple nodes on a LAN, there's no need for     ##
##  all of them to open lots of connections.  Instead           ##
##  'connect' them all to one node that is port forwarded       ##
##  and has lots of connections.                                ##
##       Thanks goes to [Noodle] on Freenode.                   ##
##################################################################

# Use as many addnode= settings as you like to connect to specific peers
#addnode=69.164.218.197
#addnode=10.0.0.2:38380

# Alternatively use as many connect= settings as you like to connect
ONLY to specific peers
#connect=69.164.218.197
#connect=10.0.0.1:38380

# Listening mode, enabled by default except when 'connect' is being used
#listen=1

# Maximum number of inbound+outbound connections.
#maxconnections=

#
# JSON-RPC options (for controlling a running Mb8coin/mb8coind process)
#

# server=1 tells Mb8coin-Qt and mb8coind to accept JSON-RPC commands
#server=0

# Bind to given address to listen for JSON-RPC connections.
Use [host]:port notation for IPv6.
# This option can be specified multiple times (default: bind to
all interfaces)
#rpcbind=<addr>

# If no rpcpassword is set, rpc cookie auth is sought. The default
`- rpccookiefile` name
# is .cookie and found in the `-datadir` being used for mb8coind.
This option is typically used
# when the server and client are run as the same user.
#
# If not, you must set rpcuser and rpcpassword to secure the JSON-
RPC api. The first
# method(DEPRECATED) is to set this pair for the server and client:
#rpcuser=Ulysseys
#rpcpassword=YourSuperGreatPasswordNumber_DO_NOT_USE_THIS_OR_YOU_WILL_GET
_ROBBED_385593
#
```

```
# The second method `rpcauth` can be added to server startup argument.
It is set at intialization time
# using the output from the script in share/rpcuser/rpcuser.py
after providing a username:
#
# ./share/rpcuser/rpcuser.py alice
# String to be appended to mb8coin.conf:
#
rpcauth=alice:f7efda5c189b999524f151318c0c86$d5b51b3beffbc02b724e5d09582
8 e0bc8b2456e9ac8757ae3211a5d9b16a22ae
# Your password:
# DONT_USE_THIS_YOU_WILL_GET_ROBBED_8ak1gI25KFTvjovL3gAM967mies3E=
#
# On client-side, you add the normal user/password pair to send commands:
#rpcuser=alice
#rpcpassword=DONT_USE_THIS_YOU_WILL_GET_ROBBED_8ak1gI25KFTvjovL3gAM967mi
e s3E=
#
# You can even add multiple entries of these to the server conf file,
and client can use any of them:
#
rpcauth=bob:b2dd077cb54591a2f3139e69a897ac$4e71f08d48b4347cf8eff3815c0e2
5 ae2e9a4340474079f55705f40574f4ec99

# How many seconds mb8coin will wait for a complete RPC HTTP request.
# after the HTTP connection is established.
#rpcclienttimeout=30

# By default, only RPC connections from localhost are allowed.
# Specify as many rpcallowip= settings as you like to allow
connections from other hosts,
# either as a single IPv4/IPv6 or with a subnet specification.

# NOTE: opening up the RPC port to hosts outside your local
trusted network is NOT RECOMMENDED,
# because the rpcpassword is transmitted over the network unencrypted.

# server=1 tells Mb8coin-Qt to accept JSON-RPC commands.
# it is also read by mb8coind to determine if RPC should be enabled
#rpcallowip=10.1.1.34/255.255.255.0
#rpcallowip=1.2.3.4/24
#rpcallowip=2001:db8:85a3:0:0:8a2e:370:7334/96

# Listen for RPC connections on this TCP port:
#rpcport=38380

# You can use Mb8coin or mb8coind to send commands to Mb8coin/mb8coind
# running on another host using this option:
#rpcconnect=127.0.0.1

# Create transactions that have enough fees so they are likely to
begin confirmation within n blocks (default: 6).
# This setting is over-ridden by the -paytxfee option.
#txconfirmtarget=n

# Miscellaneous options
```

```
# Pre-generate this many public/private key pairs, so wallet backups will
be valid for
# both prior transactions and several dozen future transactions.
#keypool=100

# Pay an optional transaction fee every time you send mb8coins.
Transactions with fees
# are more likely than free transactions to be included in generated
blocks, so may
# be validated sooner.
#paytxfee=0.00

# Enable pruning to reduce storage requirements by deleting old blocks.
# This mode is incompatible with -txindex and -rescan.
# 0 = default (no pruning).
# 1 = allows manual pruning via RPC.
# >=550 = target to stay under in MiB.
#prune=550

# User interface options

# Start Mb8coin minimized
#min=1

# Minimize to the system tray
#minimizetotray=1
```