CRYPTONOTE STANDARD 003 Albert Werner
Category: Main Track Montag
Ardolabar
Tereno
Antonio M. Juarez
CryptoNote
September 2012

CryptoNote Blockchain

Abstract

This document is part of the CryptoNote Standards describing a peer-to-peer anonymous payment system. It defines the way data is stored within blocks and the blockhain along with the corresponding data structures.

Table of Contents

1. Introduction

CryptoNote's distributed public ledger is organized in the form of blockchain similar to Bitcoin's blockchain described in [BITCOIN]. The blockchain consists of blocks chained together: each block refers to the previous one through block identifiers. The blockchain has a tree-like form with a single root block and branches that may start from any other blocks. At any moment only one branch is considered to be the main branch. The main branch is chosen by a consensus

algorithm at each network node.

Each block has a block header and an additional payload. The additional payload consists of a base transaction and a list of transaction identifiers. The transactions themselves (except for the base transaction) are not included in the block. The block's payload (the transactions attached to the block via identifiers) is expected to be non-self-contradictory and not contradict the blockchain branch it is attached to. This ensures that the entire blockchain is non-contradictory.

The blockchain acts as an append-only database. Each block contains certain information that is appended on top of some branch (usually the main branch).

2. Definitions

base transaction: a transaction that generates new coins

block: a set of data (payload) with a block header

block header: metadata at the beginning of each block

block height: the distance between the block and the genesis block in the blockchain

blockchain: a tree structure of blocks

genesis block: the root of the blockchain

3. Variable-Length Encoding of Integers (varint)

Most integers in CryptoNote are encoded in a variable-length prefix-free representation. The same encoding is used in the .xz file format [XZ].

Integers between 0 and 127 (inclusive) are represented by a single byte having the same value as the integer. Larger integers are first represented in base 128. Let a[0], a[1], ..., a[n-1] be such representation, where a[0] is the least significant base-128 digit and a[n-1] is the most significant one. Then, the encoding is the sequence of n bytes with values a[0]+128, a[1]+128, ..., a[n-2]+128, a[n-1]. Since a[n-1] > 0, such encoding does not contain null bytes.

Decoding such an integer requires scanning the stream until a byte with the value less than 128 is found. If it is not the first byte and its value is 0, the encoding is invalid. Otherwise, the sequence of values of the bytes from the first byte to the byte that was found before (inclusive) with their most significant bits cleared (except for the last one) is interpreted as a little-endian base-128 integer. This integer is the result of the decoding.

4. Block Structure

A block consists of three parts:

- block header,

- base transaction body,

- list of transaction identifiers.

The list starts with the number of transaction identifiers that it contains.

4.1 Block Header

Each block starts with a block header. The major version defines the block header parsing rules (i.e. block header format) and is incremented with each block header format update. The table below describes version 1 of the block header format. The minor version defines the interpretation details that are not related to block header parsing.

It is always safe to parse the block header of a particular major version with a parsing procedure suitable for said version, even if the minor version is unknown. Parsing the block header with an unknown major version is not safe as the content of the block header may be misinterpreted.

Field	Type	Content
major_version	varint	Major block header version (always 1)
minor_version	varint	Minor block header version
timestamp	varint	Block creation time (UNIX timestamp)
prev_id	hash	Identifier of the previous block
nonce	4 bytes	Any value which is used in the network consensus algorithm

Table 4.1: Block header structure description

4.2 Base Transaction

Each valid block contains a single base transaction. The base transaction's validity depends on the block height due to the following reasons:

- the emission rule is generally defined as a function of time;

- without the block height field, two base transactions could be indistinguishable as they can have the same hash (see [BH] for a description of a similar problem in Bitcoin).

Field	Type	Content
version	varint	Transaction format version
unlock_time	varint	UNIX timestamp. See [CNS004]
input_num	varint	Number of inputs. Always 1 for base transactions
input_type	byte	Always 0xff for base transactions
height	varint	Height of the block which contains the transaction
output_num	varint	Number of outputs
outputs	array of outputs	Array of outputs. See [CNS004]
extra_size	varint	Number of bytes in the Extra field
extra	array of bytes	Additional data associated with a transaction

Table 4.2: Base transaction structure description

For general transaction structure description see [CNS004].

4.3 List of Transaction Identifiers

Base transaction is followed by a list of transaction identifiers. A transaction identifier is a transaction body hashed with the Keccak hash function. The list starts with the number of identifiers and is followed by the identifiers themselves if it is not empty.

Field	Type	Content
tx_num	varint	Number of transaction identifiers
identifiers	array of hashes	Array of transaction identifiers

Table 4.3: List of transaction identifiers structure description

5. Calculation of Block Identifier

The identifier of a block is the result of hashing the following data with Keccak:

- size of [block_header, Merkle root hash, and the number of transactions] in bytes (varint)
- block_header,
- Merkle root hash,
- number of transactions (varint).

The goal of the Merkle root hash is to "attach" the transactions referred to in the list to the block header: once the Merkle root hash is fixed, the transactions cannot be modified.

5.1 Merkle Root Hash Calculation

Merkle root hash is computed from the list of transactions as follows: let tx[i] be the i-th transaction in the block, where 0 <= i <= n-1 (n is the number of transactions) and tx[0] is the base transaction. Let m be the largest power of two, less than or equal to n. Define the array h as follows:

```
h[i] = H(h[2*i] || h[2*i+1])
  where 1 <= i <= m-1 or 3*m-n <= i <= 2*m-1.
h[i] = H(tx[i-m])
  where m <= i <= 3*m-n-1
h[i] = H(tx[i-4*m+n])
  where 6*m-2*n <= i <= 4*m-1.
```

Where H is the Keccak function that is used throughout CryptoNote,

and || denotes concatenation. Then, h[1] is the root hash.

The figure below illustrates the calculation of Merkle root hash in a block with 9 transactions. Each arrow represents a computation of H.

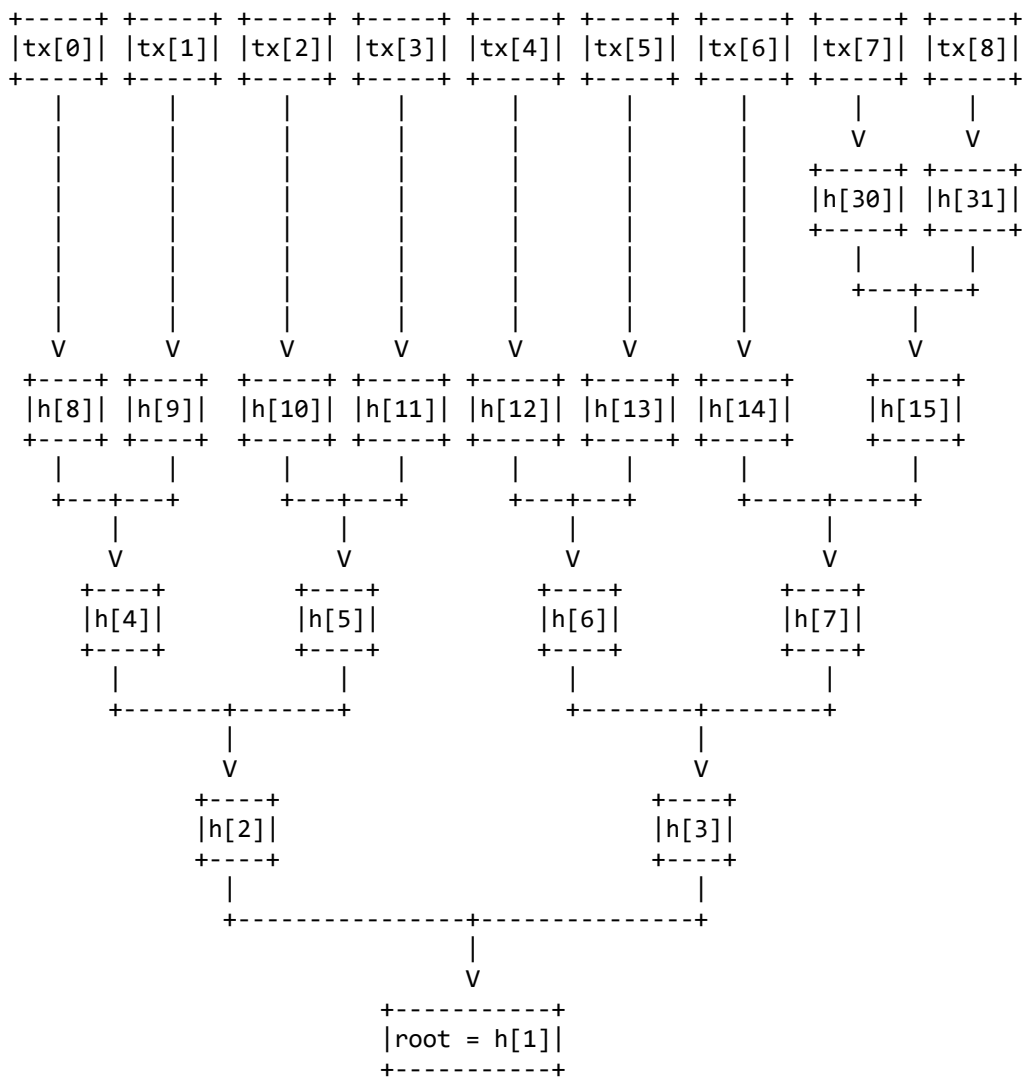

Figure 5.1: Merkle root hash calculation algorithm

6. References

[BITCOIN] Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008.

[BH] Andresen, G., comment to pull request "Transition to requiring block height in block coinbases", June 28, 2012, https://github.com/bitcoin/bitcoin/pull/1526.

[CNS004] "CryptoNote Transactions", CryptoNote Standard 004, September 2012.

[XZ] "The .xz File Format", 2009, http://tukaani.org/xz/xz-file-format-1.0.4.txt.