SonoCoin Protocol Documentation

Release 0.0.1

SonoCoin Foundation

Sep 13, 2018

Contents:

SonoCoin - the first cryptocurrency to transact via encrypted audio files simplifies blockchain-based payments giving anyone the power to transact utilizing common methods of delivery.

CHAPTER 1

Scope

This documentation aims to provide enough information to enable an avid reader to build his own node implementation conforming to the SonoCoin protocol. SonoCoin was inspired by many existing platforms and aims to combine the best parts of all existing solutions.

1.1 Introduction

SonoCoin is a decentralized blockchain platform with a native crypto currency based on proof of stake (PoS). As such, there exist a variety of appilcations surrounding the SonoCoin ecosystem.

This documentation focuses on the protocol specification that allows nodes to communicate with each other and agree on the state of the blockchain in a decentralized manner.

1.1.1 SonoCoin Nodes

A SonoCoin node is any piece of software that behaves according to the present SonoCoin protocol specification. Nodes form the basis of a SonoCoin network and perform several important functions:

- Peer-to-peer communication over TCP / UDP
- Blockchain storage
- Consensus establishment
- Gateway functionality

An open source *Reference Implementation* of a SonoCoin node was implemented by the SonoCoin foundation.

1.1.2 SonoCoin Networks

A SonoCoin network is formed by a large number of nodes interacting with each other according to the protocol specification.

Currently, two public networks exists:

- Main network
- Test network

1.1.3 SonoCoin Clients

A SonoCoin client is any piece of software that interacts with a SonoCoin network. Clients use the gateway functionality of nodes to publish transactions or query the blockchain state. Several different open source clients implementations have been developed by the SonoCoin foundation:

- Android Client
- iOS Client
- Desktop Client

Note: Github links will be added soon.

1.2 Standards

1.2.1 Cryptography

Hashes

All hashes use the SHA256 algorithm as defined in FIPS 180-4. The reference implementation uses the sha256 package from golang's standard library.

Example

The SHA256 hash of the message "Hello World!" results in:

```
0x7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069
```

Signatures

All signatures use EdDSA with Curve25519. The reference implementation uses the go-libsodium library.

Example

Given the following keypair:

```
0x75a8c71c874456844313e4e0766d9aee0b225a95497b576f0c914a19ce1c5f96 // private
0xbe725676384aa8b7ce1faf7a0c20b7cb3e862c139bc0a9479a8789346e4af234 // public
```

Signing the msg "Hello World!" with the given private key would result in the following signature.

```
0xa4a0e361cfb0a28c8f577fe6ba440d8e4a441d42a1d93b695cdd470d14ced778adb70a8e1762f42366235610cebd98e58f
```

1.2.2 Encoding

Some networking data is encoded using RLP to allow for more efficient usage of bandwidth.

Recursive Length Prefix (RLP)

Recursive Length Prefix (RLP) is a serialization format originally created by the Ethereum developers. Unlike other binary serialization formats, RLP doesn't differentiate between different data types but only encodes data structure.

Definition

- Single bytes between **0x00** and **0x7f** are their own RLP encoding.
- Byte arrays with a length of **0** to **55** bytes are encoded using a special prefix of **0x80** plus the length of the array.

Note: Special case: Single bytes between **0x80** and **0xff** are encoded as a byte array of length 1 and are thus prefixed with **0x81**.

- Byte arrays larger than **55** bytes are encoded using two prefixes. The first prefix **pf1** defines how many bytes are going to be used to encode the length of the array. The second prefix **pf2** defines the length of the array. **pf1** is calculated by adding the length of **pf2** to **0xb7**. **pf2** is simply the length of the byte array and might be several bytes long.
- Lists of length **0** to **55** are prefixed with 0xc0 added to the length of the list in bytes. List items can be any other RLP encoded data (lists, byte arrays, bytes).
- List of length greater than **55** are similarly encoded using two prefixes. The first prefix **pf1** defines how many bytes are going to be used to encode the length of the list. The second prefix **pf2** defines the length of the list. **pf1** is calculated by adding the length of **pf2** to **0xf7**. **pf2** is simply the length of the list in bytes and might be several bytes long.

Note: The first byte of the RLP encoding thus implies the data structure as follows:

Byte range	Data structure
[0x00, 0x7f]	Single byte
[0x80, 0xb7]	Byte array of size 0-55
[0xb8, 0xbf]	Byte array of size > 55
[0xc0, 0xf7]	List of size 0-55
[0xf8, 0xff]	List of size > 55

Examples

Data	RLP Encoding
0x05	[0x05]
“H”	[0x68]
“Hello”	[0x85, 0x68, 0x65, 0x6c, 0x6c, 0x6f]
“World”	[0x85, 0x77 0x6f 0x72 0x6c 0x64]
[“Hello”,”World”]	[0xcc, 0x85, 0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x85, 0x77 0x6f 0x72 0x6c 0x64]
[[], [[]], [[], [[]]]]	[0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0]

Note: Since RLP doesn’t know any data types, examples encode strings using ASCII for simplicity.

1.2.3 Decimals

The value of SonoCoin is represented by a 64 bit unsigned integer (uint64). In order to allow fractions of a SonoCoin to be transfered, a constant is defined that determines how many decimal places are included in the value. SonoCoin uses 8 decimal places, meaning that the amount of SonoCoins is the uint64 value divided by 10^8.

Examples

uint64 Value	SonoCoin
1	0.00000001
10	0.0000001
100	0.000001
1000	0.00001
10000	0.0001
100000	0.001
1000000	0.01
10000000	0.1
100000000	1
1000000000	10
10000000000	100
100000000000	1000
1000000000000	10000
10000000000000	100000
100000000000000	1000000
1000000000000000	10000000
10000000000000000	100000000

1.3 Blockchain

The following section covers data structures used in SonoCoin’s blockchain.

Note: Sizes are measured in bytes.

1.3.1 Transactions

Transactions represent a blockchain state transition function. They are the only means to change the state of the blockchain and are consequently used to split coins, combine coins and reissue coins. Transactions in SonoCoin work similar to Bitcoin's. Each transaction contains a list of inputs and outputs. Transaction inputs reference and claim previous transaction outputs. Transaction outputs set up conditions for claiming ownership of the value held in them. The state transition specified by the transaction happens when the transaction is validated and added to a block.

The structure of a transaction can be seen in the following table.

Table 1: Transaction Data Structure

Size	Name	Type	Comment
32	Hash	[32]byte	Hash of the transaction
4	Version	uint32	For forward compatibility (Current Version = 1)
4	LockTime	uint32	Time after which the transaction can be added to the blockchain
var.	Inputs	[]TransactionInput	Claimed transaction inputs
var.	Outputs	[]TransactionOutput	Spendable transaction outputs (UTXO)

The following illustration shows an example transaction that takes 4 inputs and generates two outputs (excluding the commission output):

Fig. 1: An example of a SonoCoin transaction

Transaction Input

Transaction inputs reference a spendable transaction output (UTXO) captured in the transaction input outpoint data structure. Furthermore a script needs to be added with a valid signature allowing the party creating the transaction to claim the referenced UTXO.

Table 2: TransactionInput Data Structure

Size	Name	Type	Comment
44	PreviousOutput	TransactionInputOutpoint	Reference to UTXO
4	Sequence	uint32	The index of the specific input in the transaction
var.	Script	[]byte	Signature to verify public key ownership of the referenced UTXO

Transaction Input Outpoint

Transaction input outpoints are references to previous transactions outputs. The reference is captured with a transaction hash, index of the to be claimed UTXO and the value contained in the UTXO.

Table 3: TransactionInputOutpoint Data Structure

Size	Name	Type	Comment
32	Hash	[32]byte	The hash of the referenced transaction
4	Index	uint32	The index of the specific output in the transaction
8	Value	uint64	UTXO value

Transaction Output

Table 4: TransactionOutput Data Structure

Size	Name	Type	Comment
4	Index	uint32	Ordering parameter
8	Value	uint64	Transaction output value
var.	Script	[]byte	Script defining conditions to claim this output
32	NodeID	[32]byte	Public key of node that wants to participate in PoS

Note: the script field usually contains the public key of the new UTXO owner.

Commission Transaction

Every transaction in SonoCoin is subject to a commission that is used as an incentivization mechanism for the POS consensus algorithm. The first output of every transaction is defined to be a commission output spendable by whoever mines the block containing the transaction. A valid commission output thus has an index of 0 and an empty script enabling the miner to claim the output.

When mining a block, a miner creates and adds an additional transaction to the block called the commission transaction that claims and combines all the commission outputs into a single utxo. The commission transaction is always added to the beginning of the transaction list.

Fig. 2: Illustration of how commission outputs are claimed by a miner in the commission transaction. **c** represents the commission amount (0.01 SNC)

Hash Calculation

Transaction hashes are calculated using sha256(sha256(flatTx)), where flatTx stands for the flattened *Transaction* data structure (Excluding the Hash field). The flattened array of a transaction with one input and one output (excluding the commission output) is of the following form:

Table 5: Flattened data structure for a Transaction with one input and one output (excluding commission output)

Size	Name	Type	Endianness	Data Origin
4	Version	uint32	Little	Transaction
4	LockTime	uint32	Little	
32	Hash	[32]byte	n/a	Transaction Input Outpoint
4	Index	uint32	Little	
8	Value	uint64	Little	
4	Sequence	uint32	Little	Transaction Input
var.	Script	[]byte	n/a	
4	Index	uint32	Little	Transaction Output (Commission)
8	Value	uint64	Little	
4	Index	uint32	Little	Transaction Output
8	Value	uint64	Little	
var.	Script	[]byte	n/a	

Validation

A transaction is considered valid if it satisfies the following conditions:

- The transaction contains at least one valid input and two outputs.
- The first output of the transaction has an empty Script and a value of 0.01 SNC (Commission Output).
- The input scripts are valid claims of the referenced transaction outputs.
- The sum of the input values is equal to the sum of output values.

Special cases

- The commission transaction only has one output that combines all the comission outputs of a block.
- The genesis transaction does not have any inputs.

1.3.2 Blocks

There are two types of blocks in SonoCoin:

- Normal blocks (Containing transactions)
- Epoch blocks

Both blocks share the same block header but use a different block body.

Block

A block is a signed aggregation of transactions, that is linked to the previous block by a hash.

Fig. 3: Illustration of block ordering by hash linkage.

Table 6: Normal Block Data Structure

Size	Name	Type	Comment
var.	Header	BlockHeader	block header
var.	Transactions	[]Transaction	Block transactions, in format of “tx” command

Epoch Block

Epoch blocks are part of SonoCoin’s PoS consensus algorithm. They hold a list of public keys called the advice list, that is used to determine the miners for the following epoch in sequential order. Epoch blocks themselves are not mined, but generated by each node independently. For further details please refer to *POS*.

Table 7: Epoch Block Data Structure

Size	Name	Type	Comment
var.	Header	BlockHeader	block header
var.	Advices	[][32]byte	List of public keys determining the miners of the following epoch in sequential order

Block Header

Table 8: Block Header Data Structure

Size	Name	Type	Comment
4	Type	uint32	Block type (0 = Epoch block, 1 = Normal block)
32	Hash	[32]byte	Block Hash
4	Height	uint32	Height
8	Size	uint64	Size
4	Version	uint32	For forward compatibility (Current Version = 1)
32	PrevBlock	[32]byte	The hash value of the previous block this particular block references
32	MerkleRoot	[32]byte	The root hash of the merkle tree of all transactions / advice nodes
4	Timestamp	uint32	A timestamp recording when this block was created (Will overflow in 2106)
4	Bits	uint32	Not Used
4	Nonce	uint32	Not used
32	Seed	[32]byte	Legacy part of PoS algorithm
4	TxnCount	uint32	Number of transactions contained in the block
4	AdviceCount	uint32	Number of advice nodes
var.	Script	[]byte	Node signature

Hash Calculation

Merkle Root Calculation

Note: Will be specified soon.

Block Hash Calculation

Note: Will be specified soon.

1.3.3 Genesis Blocks and Transaction

SonoCoin's gensesis blocks are hard coded blocks that are used to kickstart the blockchain. In SonoCoin two such blocks need to be defined. The first block creates the total supply of 100,000,000 SNC and assigns them to a key pair. The second block is needed to kickstart the PoS algorithm with a list of advisors.

SonoCoins Genesis block is defined as follows:

Table 9: Genesis Block Header

Parameter	Value
Type	1
Hash	114478c6875b7bfe44c9af34c2cf8e93043d59e76ee7180218c65bdc84c0dbcb
Height	1
Size	592
Version	1
PrevBlock	0
MerkleRoot	641693ef03a89b2fd0022ef794294f10be8d38f1c69dcae4ea813d6d0170d85e
Timestamp	1511859600
Bits	0
Nonce	0
Seed	(Empty)
TxnCount	1
AdviceCount	0
Script	[]

The genesis block consists of a single transaction with no input but one output of 100‘000‘000 SonoCoins to the following Key Pair:

Table 10: Genesis Key Pair

Parameter	Value
Private Key	4a9c464e848424c9197e09b85ed47a51d2c07cc43a6b923c9a805686a59b311f
Public Key	b2d29213085e152ec752ff87f1a61cba9523997bfbc9021e7d08a401e31659af

Genesis Epoch

After the genesis block is created a gensis epoch is defined by adding an additional Epoch block. The genesis epoch block header is specified in the following table:

Table 11: Genesis Epoch Block Header

Parameter	Value
Type	0
Hash	cda2ebb07d5224b572723af96cf937e8f2c317bd2e2a585d4e3c7a7d93e2a6ef
Height	2
Size	49611
Version	1
PrevBlock	114478c6875b7bfe44c9af34c2cf8e93043d59e76ee7180218c65bdc84c0dbcb
MerkleRoot	01a4cbcf526be3f153b1dea54cdf3a8f5752a4e35c2c380e9a94f17efc08a4fa
Timestamp	1511952548
Bits	0
Nonce	0
Seed	(empty)
TxnCount	0
AdviceCount	600
Script	[]

The genesis Epoch block body contains a list of 600 identical advisors. The public key of the advisor can be found in the following list:

```
[
    64b8f1da790f5f1fe2e8dce38c3b9e99752b6fe8325693f4909e4203eadcdc92, // 1.
    64b8f1da790f5f1fe2e8dce38c3b9e99752b6fe8325693f4909e4203eadcdc92, // 2.
    ...
    64b8f1da790f5f1fe2e8dce38c3b9e99752b6fe8325693f4909e4203eadcdc92  // 600.
]
```

1.4 Coin

In SonoCoin value is often transferred by sharing a special data structure with a recipient. This data structure is aptly called a coin. A coin is made up of the following elements:

Size	Name	Type	Comment
64	Key	[64]byte	Private key for a utxo
32	TxHash	[32]byte	hash of the transaction
4	Index	uint32	Index of the utxo

1.4.1 Coin Reissuance

When a coin is transmitted from one user to another, the private key is shared between the users which causes a race condition. The recipient has to *reissue* their coin to make the transfer final. This is done by creating a new key pair and issuing a transaction that spends the UTXO associated with the private key to the generated public key.

1.4.2 Coin Transfer

Coins can be transmitted between SonoCoin clients by a variety of methods (Sound, Light, QR-Code). The following illustration shows how SonoCoin's mobile clients create the sound file.

Fig. 4: A simplified view of the SonoCoin sound generation algorithm.

1.5 P2P

This section describes how peers are discovered in the SonoCoin network and how information is shared between nodes. SonoCoin's P2P network layer is based on Ethereum's well established RLPx protocol and can be divided into the following three parts:

- Discovery Protocol (UDP)
- Base Protocol (TCP)
- SonoCoin Protocol (Base Protocol)

Any number of protocols can be built on top of the base protocol and can be used side by side with the SonoCoin Protocol.

1.5.1 Discovery Protocol

SonoCoin uses a variant of the Kademlia distributed hash table (DHT) to form a structured p2p overlay network. This UDP based DHT represents the base layer of the SonoCoin network, allowing for a highly scalable ($O(\log n)$ for node lookup and space complexity) and general decentralized network infrastructure.

The following table summarizes the parametrization of Kademlia used in SonoCoin's reference implementation:

Parameter Name	Value
hash bits	256
k (bucket size)	16
α (Concurrency Factor)	3

For node distance calculations using Kademlia's XOR metric the sha256 hash of the nodeID is used.

Packet header

All packets are prefixed with a header of the following structure:

Table 12: Discovery Packet Header

Size	Name	Type	comment
32	PublicKey	[32]byte	The node's public key
64	signature	[64]byte	sig(packet-type ‖ packet-data)
1	packetType	uint	packet identifier

Note: The ‖ operator stands for concatenation

Data Structures

The following data structure will be reused in the packet definitions:

Table 13: rpcEndpoint data structure

Size	Name	Type	Comment
4 / 16	IPAddress	[]byte	Little endian encoded IPV4 / IPV6 address
2	UDP	uint16	UDP port number
2	TCP	uint16	TCP port number

Packet Types

SonoCoin's discovery protocol defines the following 4 RPC packet types. Packet size is limited to 1280 bytes. Larger packets are discarded.

All packet contents are serialized using *Recursive Length Prefix (RLP)* encoding.

Ping Packet (packet-type = 0x01)

Size	Name	Type	comment
4	Version	uint32	For Forward compatibility (Current Version = 1)
8 / 20	From	rpcEndpoint	
8 / 20	To	rpcEndpoint	
8	Expiration	uint64	Unix timestamp after which packet is considered expired

Pong Packet (packet-type = 0x02)

Size	Name	Type	comment
8 / 20	To	rpcEnpoint	
32	ping-hash	[32]byte	hash of corresponding ping packet
8	Expiration	uint64	Unix timestamp after which packet is considered expired

FindNode Packet (packet-type = 0x03)

Size	Name	Type	comment
32	Target	[32]byte	The identifier of the node
8	Expiration	uint64	Unix timestamp after which packet is considered expired

Neighbors Packet (packet-type = 0x04)

Size	Name	Type	comment
var.	Nodes	[]rpcEndpoint	
8	Expiration	uint64	Unix timestamp after which packet is considered expired

Neighbors packets are split up into multiple packets to stay below the 1280 byte limit.

1.5.2 Base Protocol

The TCP-based base protocol is used to authenticate nodes and upgrade to a higher layer protocol (eg. SonoCoin Protocol).

Message structure

All messages are structured as follows:

Size	Name	Type	Comment
3	StartSymbol	[3]byte	[0x73, 0x63, 0x6d] (“scm”)
4	Magic	uint32	1 = MainNet, 2 = TestNet
8	Command	uint64	command
8	Length	uint64	Length of payload in number of bytes
4	Checksum	uint32	First 4 bytes of sha256(sha256(payload))
16	UUID	[16]byte	Unique Packet Identifier
var.	Payload	[]byte	Payload Data

Available Commands

Command Name	Command ID
cmdEncHandshake	0x00
cmdHandshake	0x01
cmdDisc	0x02
cmdPing	0x03
cmdPong	0x04

Handshakes

After a peer is discovered by the discovery protocol, two handshakes are performed to authenticate peers and communicate protocol capabilities. The described process is illustrated in the following sequence diagram:

Fig. 5: The base protocol handshaking process.

Data Structures

The following data structure will be reused in the packet definitions:

Table 14: Cap data structure

Size	Name	Type	Comment
var.	Name	string	Protocol name (“SNC” for SonoCoin Protocol)
4	Version	uint32	For forward Compatibility (current version = 1)

Commands in Detail

cmdEncHandshake

The Encryption Handshake (0x00) authenticates the nodes and establishes a shared secret between them.

> **Warning:** Shared secret isn’t used for encrypted communication yet.

The payload is different for the initiating peer and the target peer.

Request Payload

The initiator generates a random key pair and a random nonce and signs the nonce with the generated private key.

Size	Name	Type	comment
64	Signature	[64]byte	Signed Nonce
32	InitiatorPubKey	[32]byte	Randomly generated public key
32	Nonce	[32]byte	Randomly generated nonce
4	Version	uint32	For forward compatibility (Current Version = 1)

Response Payload

Size	Name	Type	comment
32	random_pub_key	[32]byte	Randomly generated public key
32	Nonce	[32]byte	Randomly generated nonce
4	Version	uint32	For forward compatibility (Current Version = 1)

cmdHandshake

The protocol handshake (0x01) determines the capability of the nodes and is the same for the initiator and the target.

Size	Name	Type	comment
32	NodeID	[32]byte	
8	Version	uint64	
var.	Name	string	
var.	Caps	[]Cap	
8	ListenPort	uint64	

cmdDisc

cmdDisc is used when disconnecting from a peer. It lets the peer know why the TCP connection will be dropped.

Size	Name	Type	comment
4	Reason	uint32	Reason for disconnecting. Valid reasons Listed in the following table

Reason List

Name	Code	Description
DiscRequested	0x00	
DiscNetworkError	0x01	
DiscProtocolError	0x02	
DiscUselessPeer	0x03	
DiscTooManyPeers	0x04	
DiscAlreadyConnected	0x05	
DiscIncompatibleVersion	0x06	
DiscInvalidIdentity	0x07	
DiscQuitting	0x08	
DiscUnexpectedIdentity	0x09	
DiscSelf	0x0a	
DiscReadTimeout	0x0b	
DiscSubprotocolError	0x10	

Warning: TODO: define which disconnect reason is used when.

cmdPing

No payload.

cmdPong

Response to ping. No payload.

1.5.3 SonoCoin Protocol V1

The SonoCoin Protocol defines how blockchain related information is exchanged. Nodes that implement Version 1 of the SonoCoin protocol communicate this, by adding ["SNC",1] to their capability list in the base protocol handshake.

Available Commands

Command Name	Command ID
cmdStatus	0x10
cmdPing	0x11
cmdPong	0x12
cmdMsg	0x13
cmdTx	0x14
cmdNewBlockHashes	0x15
cmdGetBlockHeaders	0x16
cmdBlockHeaders	0x17
cmdGetBlockBodies	0x18
cmdBlockBodies	0x19
cmdNewBlock	0x1a
cmdGetNodeData	0x1d
cmdGetReceipts	0x1f

Handshake

The SonoCoin Protocol Handshake involves both peers sending a cmdStatus (0x00) Message that communicates the current state of each peer's blockchain. The process is illustrated in the following sequence diagram:

Fig. 6: The SonoCoin protocol handshake.

Peers that haven't partaken in a handshake but send commands from the SonoCoin Protocol should be dropped (cmdDisc 0x02).

Commands in Detail

cmdStatus

Handshake for the SonoCoin Protocol. Informs peer of blockchain state.

Size	Name	Type	Comment
4	ProtocolVersion	uint32	For forward compatibility (Current version = 1)
8	NetworkID	uint64	Mainnet = 1, Testnet = 2
32	CurrentBlock	[32]byte	Hash of last known block in local blockchain
32	GenesisBlock	[32]byte	Hash of genesis block in local blockchain

cmdPing

No Payload.

cmdPong

Response to cmdPing. No Payload.

cmdMsg

Sends a plain text message.

Size	Name	Type	Comment
var.	Name	string	

cmdTx

Notifies peer of uncofirmed transactions.

Size	Name	Type	Comment
var.	Txs	[]Transaction	

cmdNewBlockHashes

Announces the availability of a number of blocks through a hash notification.

Size	Name	Type	Comment
var.	NewBlockHashes	[]newBlockHashesData	

The newBlockHashesData type is defined as follows:

Size	Name	Type	Comment
32	Hash	[32]byte	
4	Height	uint32	

cmdGetBlockHeaders

Requests block headers starting at hash or height.

Size	Name	Type	Comment
32 / 4	Origin	[32]byte / uint32	Block hash or block height
4	Amount	uint32	Amount of blocks
4	Skip	uint32	Amount of block to skip after origin
1	Reverse	bool	Block header order (1 = towards genesis, 0 = towards leaf)

Note: currently in the json version “reverse”: true, “reverse”: false instead of 1, 0

cmdBlockHeaders

Reply to cmdGetBlockHeaders

Size	Name	Type	Comment
var.	blockHeaders	[]BlockHeader	

cmdGetBlockBodies

Requests block bodies specified by a list of hashes.

Size	Name	Type	Comment
var.	hashes	[][32]byte	Block hashes

cmdBlockBodies

Reply to cmdGetBlockBodies.

Size	Name	Type	Comment
var.	blocks	[]Block	

cmdNewBlock

cmdNewBlock propagates a newly discovered block to a remote peer.

Size	Name	Type	Comment
var.	block	Block	

cmdGetNodeData

Warning: not implemented!

cmdGetReceipts

Warning: not implemented!

1.5.4 Node Bootstrapping

A new node wanting to join the network needs at least one seed node to connect to. It is recommended to optain seed nodes in one of the following ways:

- Receiving a list of seed nodes from a trusted source and connecting to them directly.
- Calling the SonoCoin bootstrap API (https://api.sono.money/v1/nodes).

1.6 POS

Coming Soon...

1.7 Reference Implementation

The reference implementation developed by the SonoCoin Foundation is published on github and is written in golang.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search