About

Description

42-coin is extremely rare cryptocurrency with unique deflationary emission model, fair distribution (no ICO, premine or instamine) and both private and public transaction support. It is based on an open source peer-to-peer internet protocol and hybrid Proof-of-Work / Proof-of-Stake block generation methods.

Technical features (protocol)

Block

The transactions history is permanently recorded in the network through items called blocks. A block is a record of some or all of the most recent transactions that have not yet been recorded in any prior blocks. Each block memorializes what took place immediately before it was created.

Block structure

Field	Type	sizeof	Description
Magic number	unsigned int	4	Always 0xE5E9E8E4
Block size	unsigned int	4	Number of bytes following up to end of block
Block header	struct	80	Consists of 6 items
Transactions count	Variable integer	1 - 9	-
Transactions set	transaction[]	Transactions size	List of transactions
Header signature	unsigned char[]	<= 72	Signature for Proof-of-Stake is placed here.

Transaction

A transaction is a signed section of data that is broadcast to the network and collected into blocks. It typically references previous transaction(s) and dedicates a certain number of 42 from it to one or more new public key(s). Currently there are a few transaction types possible.

User operation

These transactions are typically used to redeem 42 from unspent inputs. It generally references unspent input(s) and creates a new output(s) with specified value(s) and destination(s).

Coinbase

Coinbase have a single input, and this input has a 'coinbase' parameter instead of a scriptSig. The data in 'coinbase' can be anything; it isn't used. 42 puts the current compact-format target and the arbitrary-precision 'extraNonce' number there, which increments every time the Nonce field in the block header overflows. The extranonce contributes to enlarge the domain for the proof of work function.

These transactions are used to reward the Proof-of-Work miners. Proof-of-stake blocks have the coinbase transaction too, but with empty output.

Coinstake

These transactions are used to provide a suitable proof for Proof-of-Stake block header. This type is similar to user transactions, but with some differences.

- First output must be empty;
- First input of this transaction is required to satisfy a current Proof-of-Stake difficulty.
- It's allowed to generate new coins through paying a negative fee.
- A destination of the second output must be Pay-to-Pubkey (described later).

General format of a transaction

Field	Type	sizeof	Description
Version	unsigned int	4	Currently 1
Timestamp	unsigned int	4	Transaction timestamp
Inputs count	variable int	1 – 9	-
Inputs array	TxIn[]	inputs set size	Inputs array or coinbase property
Outputs count	variable int	1 – 9	-
Outputs array	TxOut[]	outputs set size	Array of output structures
Lock time	unsigned int	4	block height or timestamp when transaction is final

Input format

An input is a reference to an output in a different transaction. Multiple inputs are often listed in a transaction. The values of the referenced outputs are added up, and the total is usable in the outputs of this transaction.

Field	Type	sizeof	Description
txid	unsigned char[]	32	ID of previous transaction
n	unsigned int	4	Number indexing an output of the to-be-consumed transaction
scriptSigLength	variable int	1-9	scriptSig length
scriptSig	unsigned	-	first half of script, signatures for the

	char[]		scriptPubKey
nSequence	unsigned int	4	Transaction variant number, irrelevant if nLockTime isn't specified. 0xffffffff by default, see this link for a detailed explanation.

Output format

An output contains instructions for sending 42. Value is the number of Satoshi (1 42 = 100,000,000 Satoshi, 1 Satoshi in 42 is also called “Dent”) that this output will be worth when claimed.

Field	Type	sizeof	Description
nValue	unsigned long int	8	the number of Satoshis($42/10^8$) to be transfered
scriptPubKeyLength	variable int	1-9	scriptPubKey length
scriptPubKey	unsigned char[]	-	second half of script, spending instructions

Destinations

There are three destination types allowed for user and coinbase transactions:

- Public key (Pay-to-Pubkey);
- Public key hash (Pay-to-PubkeyHash);
- Script hash (Pay-to-ScriptHash);
- Empty destination;
- Non-standard script.

Pay-to-Pubkey

```
scriptPubKey: [pubKey] OP_CHECKSIG

scriptSig: [sig]
```

Pay-to-PubkeyHash

```
scriptPubKey: OP_DUP OP_HASH160 [pubKeyHash] OP_EQUALVERIFY OP_CHECKSIG

scriptSig: [sig] [pubKey]
```

Pay-to-ScriptHash

Send to script hash:

```
scriptPubKey: OP_HASH160 [20-byte-hash of {[pubkey] OP_CHECKSIG} ] OP_EQUAL

scriptSig: <depending on inputs type>
```

Redeem example:

```
scriptPubKey: OP_HASH160 [20-byte-hash of {[pubkey] OP_CHECKSIG} ] OP_EQUAL

scriptSig: [signature] {[pubkey] OP_CHECKSIG}
```

Empty destinations

```
scriptPubKey: (empty)

scriptSig: (empty)
```

Non-standard scripts

Anyone-can-spend:

```
scriptPubKey: (empty)

scriptSig: OP_TRUE
```

Proof-of-Work

A proof-of-work is a solution for difficult (costly) mathematical task. This solution must be trivial to check whether data satisfies claimed requirements.

Currently proof-of-work remains the most practical way of providing initial minting of a crypto-currency. So we decided to keep it as part of our hybrid design.

42-coin uses the hashcash method to provide proofs of the work. The difficulty of this work is adjusted so as to limit the rate at which new blocks can be generated by the network to required target spacing rate (from 10 to 30 minutes). Due to the very low probability of successful proof generation, this makes it unpredictable which worker computer in the network will be able to generate the next solution.

How long will it take me to generate a proof-of-work?

No one can say exactly. But there is an estimation of how long it might take.

Imagine that you have a hardware with 1 MH/s hashing speed. Let's estimate how much time the generation of proof-of-work will take from you in average, with a current 42-coin proof-of-work difficulty.

Difficulty 1.0 is represented by

0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

value of target. So, to get a successful proof of work we need to perform 0xFFFFFFFFFFFFFFFF / 0x00000000FFFFFFFF or ~ 4294967297 attempts.

At difficulty 360 we need ~ 360 * 4294967297 = 1546188226920 attempts. If you have 1 MH/s or 1000000 hashes per second, then you will be able to scan such amount of hashes within 1546188226920 / 1000000 = 1546188 seconds or 1546188 / 86400 = 17,89 days.

Probability of successful block generation during one day could be calculated from available hashrate using formula:

P = nHashesPerSecond * 86400 / (4294967297 * difficulty)

How does it work?

Each block header represented by structure of 6 fields, a some of this fields could be varied pretty freely.

Field	Type	Sizeof	Requirements
nVersion	unsigned int	4	Shouldn't be modified manually

hashPrevBlock	unsigned char[]	32	Shouldn't be modified manually
hashMerkleRoot	unsigned char[]	32	It's a merkle tree hash. Could be modified through modification, addition or removal of transactions.
nTime	unsigned int	4	Can be updated manually to any value from [max tx timestamp, time() + 3600] interval.
nBits	unsigned int	4	Shouldn't be updated manually
nNonce	unsigned int	4	Attempts counter for Proof-of-Work. You need to update this field for every new hashing attempt.

What about rewards?

The block reward is zero, miners can only gather fees from transactions.

Proof-of-Stake

Proof of stake was introduced by Sunny King in Peercoin, alongside proof of work on the 19th August 2012. Proof-of-Stake is term referring to the use of currency itself (ownership) to achieve certain goals. In the 42 it is used to provide mining and transaction processing on a par with Proof-of-Work.

42-coin uses the mixed Coin-Age/CoinDayWeight approach to provide proofs of the stake. The Proof-of-Stake difficulty is adjusted so as to limit the rate at which new blocks can be generated by the network to 7 minutes target spacing rate. Due to the very low probability of successful proof generation, this makes it unpredictable which computer in the network will be able to generate the next solution.

Coin Age

Coin age refers to the age of txn inputs. Coin age is equal to the number of coins sent times the average age on these coins. Age is measured in days. Age is reset to zero whenever a coin is sent AND whenever a coin provides a signature. Coin age could be used to calculate mandatory fees, block reward or proofhash target.

CoinDayWeight

It's similar to coin age but age is calculated using 42-hours offset without upper limit. CoinDayWeight is a parameter of proofhash target in the proof-of-stake system.

nBlockTarget = CoinDayWeight * nNetworkTarget

Proof hash must satisfy the nBlockTarget, so greater CoinDayWeight means higher probability for generation of proof-of-stake block.

Coinstake kernel

Coinstake kernel it's a virtual structure which created during Proof-of-Stake block validation attempt. This structure exists in database and memory, but not on the network. The kernel parameters are described in the following table:

Field	Type	sizeof	Description
nStakeModifier	unsigned long int	8	Deterministic modifier, scrambles computation to make it very difficult to precompute future Proof-of-Stake at the time of the coin's confirmation.
nTimeBlockFrom	unsigned int	4	Timestamp for block which provided previous transaction, prevent nodes from guessing a good timestamp to generate transaction for future advantage.
nTxPrevOffset	unsigned int	4	Offset of previous transaction inside the block, used to reduce the chance of nodes generating kernel coinstake at the same time.
nTxPrevTime	unsigned int	4	Timestamp of previous transaction, used to reduce the chance of nodes generating coinstake kernel at the same time.
nPrevoutNum	unsigned int	4	Output number of previous transaction, used to reduce the chance of nodes generating coinstake kernel at the same time.
nTimeTx	unsigned int	4	Current timestamp

How it's supposed to work?

It's performed through scanning all available inputs in order to find lucky one that satisfies following condition:

SHA256(SHA256(KERNEL)) < CoinDayWeight * NetworkTarget

Miner has to find a SHA256 hash that is under the target value. Target is derived from network target using CoinDayWeight parameter. The proof is presented by kernel hash and header signature. Each coinstake kernel represented by structure of 6 fields, a some of this fields could be varied pretty freely.

Field	Type	sizeof	Requirements
nStakeModifier	unsigned long int	8	Shouldn't be modified manually.
nTimeBlockFrom	unsigned int	4	Timestamp for block which provided previous transaction.
nTxPrevOffset	unsigned int	4	Offset of previous transaction inside the block.
nTxPrevTime	unsigned int	4	Timestamp of previous transaction.

nPrevoutNum	unsigned int	4	Output number of previous transaction.
nTimeTx	unsigned int	4	Current timestamp.

The hashing result of a valid Proof-of-Stake value must be lower than block target. Miner tries to find this suitable solution by scanning all available unspent inputs with suitable CoinDayWeight.

How long will it take me to generate a Proof-of-Stake?

Just like with Proof-of-Work, no-one can say exactly. But there is an estimation of how long it might take. Calculations are quite similar with Proof-of-Work, but instead of hash we have coin * day-in-seconds here. So, at difficulty 1.0 we need ~ 4294967297 coin * day-in-seconds to find a block.

Probability of successful block generation during one day could be calculated from available CoinDayWeight using formula:

P = CoinDayWeight * 86400 / (4294967297 * difficulty)

Average block generation time could be calculated as:

T = 4294967297 * difficulty / (CoinDayWeight * 86400)

When can I start generating Proof-of-Stake blocks?

If you have balance, then 42d would automatically try to generate proof hashes for you.

What about rewards?

The block reward is zero and, and fees are destroyed in PoS blocks - this makes 42 a deflationary coin.

Mining and Blockchain

Mining is a term referring to the generation of new blocks for 42-coin blockchain. It's required to provide confirmations for transactions and to protect the history of operations.

Block header

The block header is a metadata structure which is used to link blocks in the blockchain. Block header has a size of 80 bytes and consists of 6 fields:

Field	Type	sizeof	Description	Updated when
nVersion	unsigned int	4	Block header version	You upgrade the software and it specifies a new version
hashPrevBlock	unsigned char[]	32	Previous block header hash, used to link block headers into list	New block is accepted
hashMerkleRoot	unsigned char[]	32	Merkle tree hash, used to link block header and block contents	Transactions pool is updated

nTime	unsigned int	4	Unix timestamp	Every few seconds for Proof-of-Work, every successful attempt for Proof-of-Stake
nBits	unsigned int	4	Compact representation of claimed proof difficulty	The difficulty is adjusted
nNonce	unsigned int	4	Attempts counter for Proof-of-Work	New Proof-of-Work hash tried, or never with Proof-of-Stake

Each block header is required to satisfy the claimed proof.

How is it supposed to work?

All miners have a copy of every unconfirmed valid transaction in own transactions pool. Normally mining process is performed in the four stages:

1. Get a set of transactions from transactions pool;
2. Calculate a merkle tree hash for this set of transactions;
3. Create block header template and link it with this set of transactions using its merkle tree hash;
4. Try to find suitable proof hash for block header created before.

Getting a proof is quite a difficult operation, the difficulty depends on the current number of participants. Each block header is linked to previous block header, so we have linked list of block headers, and a list consistency is guaranteed by difficulty. This linked list is also known as blockchain.

What can be used as a proof?

The only required property of proof is that it's extremely difficult to obtain, but very easy to check. There are a lot of proof concepts in existence, such as proof-of-work, proof-of-stake or proof-of-burn. 42-coin supports usage of stake or work to provide suitable proof for block header.

Configuration file example

Please note that the following example isn't purposed for production use. It was placed here to help describe the purpose of some of the settings.

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

# Network-related settings:

# Run on the test network instead of the real 42-coin network.
#testnet=0

# Connect via a socks4 proxy - default none
#proxy=127.0.0.1:9050
# Accepting incoming connections
#listen=1
# Use as many addnode= settings as you like to connect to specific peers
#addnode=193.23.181.148
#addnode=91.235.143.61:4242

# ... or use as many connect= settings as you like to connect ONLY
```

```
# to specific peers:
#connect=193.23.181.148
#connect=91.235.143.61:4242

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

# JSON-RPC options (for controlling a running 42d process)

# You must set rpcuser and rpcpassword to secure the JSON-RPC api
#rpcuser=Ulysseys
#rpcpassword=YourSuperGreatPasswordNumber_DO_NOT_USE_THIS_OR_YOU_WILL_GET_ROBBED

# How many seconds 42-coin will wait for a complete RPC HTTP request after the HTTP
connection is established.
#rpctimeout=30

# By default, only RPC connections from localhost are allowed.  Specify as many
rpcallowip= settings as you like to allow connections from other hosts (and you may
use * as a wildcard character):
#rpcallowip=10.1.1.34
#rpcallowip=192.168.1.*

# Listen for RPC connections on this TCP port:
#rpcport=2121

# You can use 42d to send commands to 42d running on another host using this option:
#rpcconnect=127.0.0.1

# Use Secure Sockets Layer (also known as TLS or HTTPS) to communicate with 42d
#rpcssl=1

# OpenSSL settings used when rpcssl=1
#rpcsslciphers=TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!AH:!3DES:@STRENGTH
#rpcsslcertificatechainfile=server.cert
#rpcsslprivatekeyfile=server.pem

# 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

# Data directory path, your keys store, copy of blockchain and unspent outputs index
are stored here.
#datadir=D:\42

# Wallet file name
#wallet=wallet.dat

# Checkpoints policy (possible values are strict and advisory)
#cppolicy=strict

# Require confirmations for change (disabled by default)
#confchange=0

# Enforce transaction scripts to use canonical PUSH operators
enforcecanonical=1
```

Exchanges and markets

Cryptopia.co.nz: 42/BTC, 42/DOGE, 42/LTC - *Deposit Confirmations: **200** blocks, Trade Fee: 0.2%, Withdrawal Fee: 0.00000002, Minimum trade: 0.00005 BTC*

LiveCoin.net: 42/BTC, 42/USD, 42/ETH - *Deposit Confirmations: 8 blocks, Trade Fee: **0.2**-%, Withdrawal Fee: 0.000001, Minimum trade: 0.0001 BTC*

TradeSatoshi.com: 42/BTC, 42/BCH, 42/LTC, 42/DOGE - *Deposit Confirmations: 12 blocks, Trade Fee: 0.2%, Withdrawal Fee: 0.00000002, Minimum trade: 0.000005 BTC*

CoinsMarkets.com: 42/BTC - *Deposit Confirmations: ? blocks, Trade Fee: **0.25**%, Withdrawal Fee: 0.00000003, Minimum trade: 0.00001 42*

NovaExchange.com: 42/BTC, 42/DOGE, 42/ETH, 42/LTC, 42/MOON, 42/KIC - *Deposit Confirmations: 40 blocks, Trade Fee: 0.2%, Withdrawal Fee: **0.00000001**, Minimum trade: 0.0000002 BTC*

Useful Links

Official website: https://42-coin.org/

Official twitter: https://twitter.com/42newchain

Official thread on bitcointalk.org forum: https://bitcointalk.org/index.php?topic=1502028.0

GitHub repository: https://github.com/42-coin/42/

Wallet download: https://github.com/42-coin/42/releases

Blockchain explorer: https://chainz.cryptoid.info/42/, https://prohashing.com/explorer/42/

Paper/Brain Wallet Generator: https://42-address.github.io/

Capitalization: http://coinmarketcap.com/currencies/42-coin, https://bitscreener.com/coins/42-coin

Charts: https://bitinfocharts.com/42coin/