Technical Whitepaper for ZillionCoin

UTXO Proof-of-Work (PoW)

Multi-algo CPU Miner ZillionFLUX

WEBSITE: www.ZillionCoin.com

OPEN SOURCE: https://github.com/zillioncoin/zillioncoin

ZILLIONGRID:
https://www.zillionpress.com/ZillionGrid_WhitePaper_Version1_Phase1.pdf

Executive Summary

ZillionCoin development begain in 2014 and emerged during a pivotal era in cryptocurrency history—just years after Bitcoin's inception—when GPU mining surged and early talks of specialized ASIC hardware threatened to monopolize mining rewards for many years to come. The rise of mining pools further marginalized individual CPU miners, who found themselves powerless against vast mining farms and resource concentration.

Recognizing this challenge, ZillionCoin was designed to restore mining equity and decentralization by radically rethinking the mining protocol. Its groundbreaking approach requires miners to cryptographically sign each mined block themselves, binding mining rewards directly to individual control and eliminating the viability of large-scale mining pools.

At its core, ZillionCoin developed ZillionFLUX, a unique, multi-algorithm mining sequence that rotates through six carefully selected cryptographic hash functions. This design dramatically increases ASIC resistance, enabling CPU miners to remain competitive and actively contribute to network security and growth.

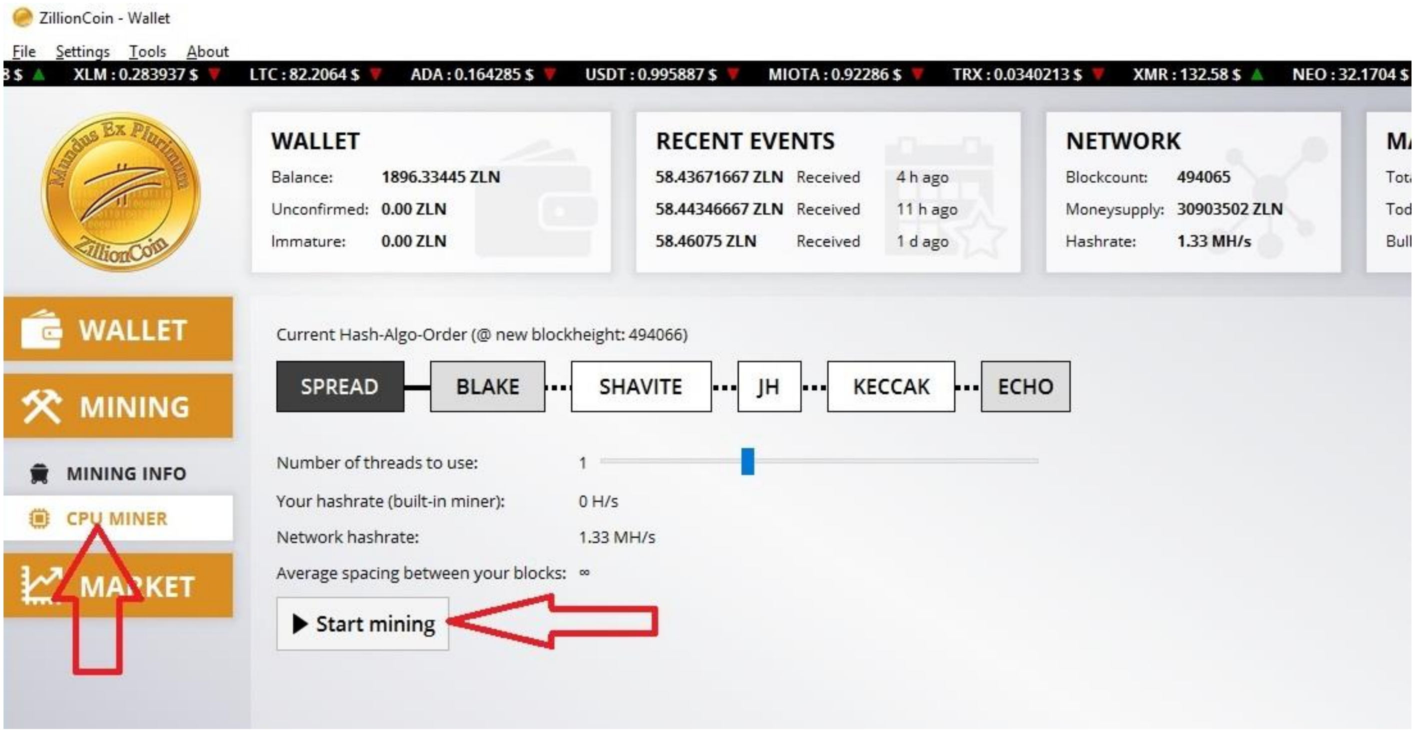
Differentiating from Bitcoin's four-year halving events, ZillionFLUX implements a smooth and continuous block reward reduction. This approach avoids abrupt reward shifts, mitigating speculation-driven price volatility and fostering a more stable value throughout the coin's lifecycle. ZillionCoin prioritizes stability over speculation, aiming to attract a professional business and gaming audience where real-world applications and utility drive value accumulation rather than price manipulation.

Publicly launched and properly announced in August 2017, ZillionCoin was introduced with complete transparency and fairness. It features no pre-mine, no hidden "easter eggs," no administrative block rewards, and no surprises. The network was simply and fairly launched to welcome anyone willing to participate, reinforcing its commitment to true decentralization and equitable access.

By empowering individual miners through miner-bound block signing and algorithmic diversity, ZillionCoin pioneers a fair, secure, and truly decentralized cryptocurrency ecosystem—where every participant can meaningfully engage in building the blockchain network.

MINING ZILLIONCOIN IS AS EASY AS CLICKING START

CPU MINER INCLUDED IN THE ZILLIONCOIN WALLET


ZillionCoin - Wallet
File Settings Tools About
3 $ ▲ XLM : 0.283937 $ ▼ LTC : 82.2064 $ ▼ ADA : 0.164285 $ ▼ USDT : 0.995887 $ ▼ MIOTA : 0.92286 $ ▼ TRX : 0.0340213 $ ▼ XMR : 132.58 $ ▲ NEO : 32.1704 $
Mundus Ex Plurimum
ZillionCoin
WALLET
MINING
MINING INFO
CPU MINER
MARKET
Start mining

WALLET

Balance:	1896.33445 ZLN
Unconfirmed:	0.00 ZLN
Immature:	0.00 ZLN

RECENT EVENTS

58.43671667 ZLN	Received	4 h ago
58.44346667 ZLN	Received	11 h ago
58.46075 ZLN	Received	1 d ago

NETWORK

Blockcount:	494065
Moneysupply:	30903502 ZLN
Hashrate:	1.33 MH/s

MA

Tot

Tod

Bul

Current Hash-Algo-Order (@ new blockheight: 494066)

SPREAD — BLAKE ··· SHAVITE ··· JH ··· KECCAK ··· ECHO

Number of threads to use:	1
Your hashrate (built-in miner):	0 H/s
Network hashrate:	1.33 MH/s
Average spacing between your blocks:	∞

Introduction

ZillionCoin is a next-generation proof-of-work cryptocurrency designed to enhance decentralization by preventing mining pools through novel cryptographic and protocol innovations. It employs a rotating multi-algorithm mining scheme combined with miner private key binding to blocks, enforcing solo mining and increasing resistance to specialized mining hardware.

The ZillionCoin miner is bult directly into the wallet, just click the START button and you’re part of the ZillionCoin crypto experience. You can even throttle CPU usage so we don’t use all of your resources and you can keep working while mining!

This whitepaper details the entire system architecture, mining processes, wallet internals, network operations, and RPC interfaces integral to ZillionCoin’s operation.

System Architecture Overview

The ZillionCoin system extends the foundational Bitcoin protocol with critical modifications targeting miner decentralization:

- **Multi-algorithm Mining Chain:** Mining proof-of-work utilizes six cryptographic hash algorithms in rotation, preventing ASIC dominance on any single algorithm.
- **Miner Private Key Binding:** Miners digitally sign the block header, embedding ownership and preventing pooling.
- **Enhanced Block Structure:** The block header contains additional fields MinerSignature and hashWholeBlock ensuring miners control the entire block content and keys.
- **Wallet Integration:** The wallet manages key generation, encryption, transaction creation, and miner key control.
- **Robust RPC Interface:** Provides comprehensive mining, wallet, and network control commands supporting ZillionCoin's unique protocols.

Multi-Algorithm Mining and Pool Prevention

Mining Problem in Traditional POW Systems

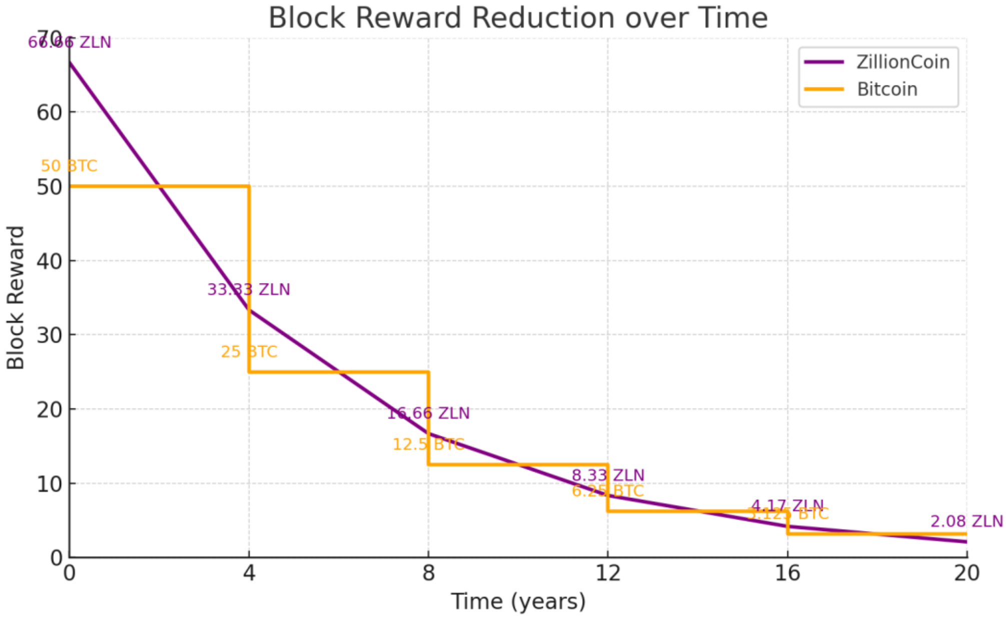
Traditional proof-of-work blockchains suffer from mining centralization through pools, which concentrate hashing power, enabling possible 51% attacks and weakening the decentralized trust model.

ZillionCoin's Solution

ZillionCoin addresses this by requiring miners to:

- Know the **private key** controlling the coinbase transaction to spend mining rewards.
- Possess full knowledge of the **entire block content** before mining.
- Compute a **digital signature** (MinerSignature) on the block header, binding miner identity and preventing delegation.
- Use a **multi-algorithm hash chain** cycling through six strong cryptographic hashes to resist ASIC specialization.

Smooth Supply



Block reward in Bitcoin is computed using the following formula:

$$R_h = R_0 \cdot 2^{-\left\lfloor \frac{h}{p} \right\rfloor}$$

where

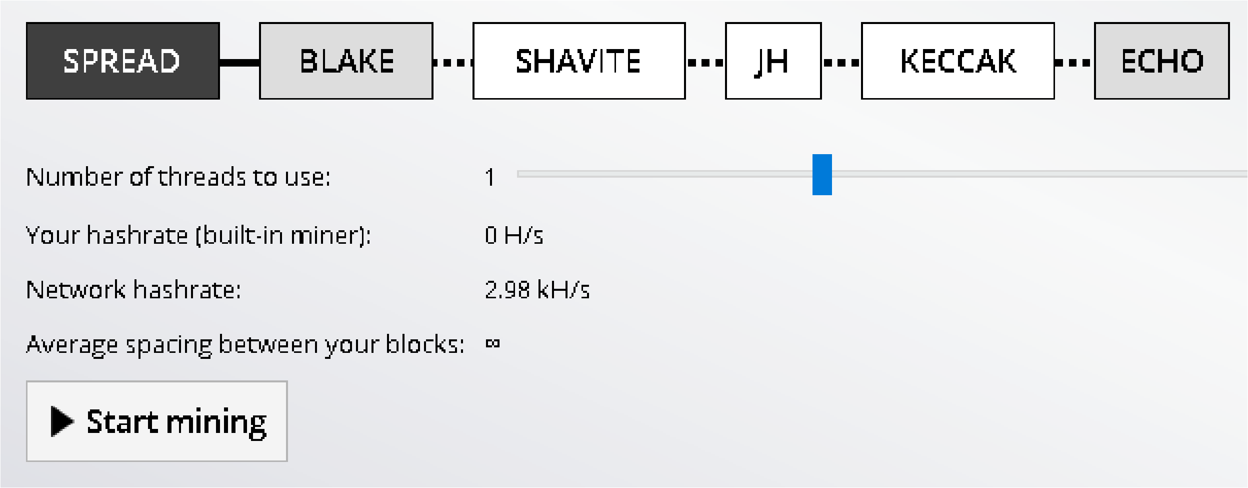
- h – block height,
- p – reward halving period,
- R_0 – initial reward,
- R_h – reward for block h,
- $\lfloor \cdot \rfloor$ – floor function.

This method results in abrupt reward changes near halving points. ZillionCoin uses simple linear interpolation between halving points to make reward decrease much smoother. This is achieved by modifying reward using the following formula:

$$R'_h = \frac{4}{3}\left(R_h - R_h \cdot \frac{h \bmod p}{2p}\right).$$

ZillionCoin uses $p = 2 \times 10^6$ as its reward halving period.

Multi-Algorithm Chain



Number of threads to use: 1

Your hashrate (built-in miner): 0 H/s

Network hashrate: 2.98 kH/s

Average spacing between your blocks: ∞

The mining hash function for a block of height h computes:

- Starting index = h mod 6 to rotate algorithm order.
- Applies six hash functions in sequence starting from the index, cycling through the following:

Index	**Algorithm**
0	SPREAD
1	BLAKE
2	SHAVITE
3	JH
4	KECCAK
5	ECHO

Each hash processes the output of the previous, ensuring miners must efficiently compute all six hashes per nonce iteration.

Algorithm Rotation Benefits

- Resists ASIC optimization for any single algorithm.
- Maintains mining competitiveness for general-purpose hardware.
- Enhances network security by diversifying cryptographic assumptions.

Mining Algorithms: Detailed Implementations

SPREAD Hash

- **Source:** src/hash.cpp
- **Function:** uint256 SpreadHash(const unsigned char* data, size_t len)
- Combines multiple cryptographic primitives to maximize diffusion and resist optimization shortcuts.
- Example logic includes SHA-256 layers combined with custom mixing.

BLAKE Hash

- **Source:** src/sphlib/blake.h & .c
- Implements Blake2b, a fast, secure hash with good software and hardware efficiency.

SHAVITE Hash

- **Source:** src/sphlib/shavite.h & .c
- A SHA-3 finalist based on AES rounds, offering strong cryptographic resistance.

JH Hash

- **Source:** src/sphlib/jh.h & .c
- Another SHA-3 finalist known for robustness against differential cryptanalysis.

KECCAK Hash

- **Source:** src/sphlib/keccak.h & .c
- The NIST standard SHA-3 winner, widely accepted and secure.

ECHO Hash

- **Source:** src/sphlib/echo.h & .c
- High-throughput cryptographic hash designed for efficient computation and strong security.

Multi-Algorithm Hash Chain Code Snippet

```
uint256 MultiAlgoHash(const unsigned char* data, size_t len, int blockHeight) {
    int startIndex = blockHeight % 6;
    uint256 hash = uint256(data, len);

    for (int i = 0; i < 6; i++) {
        int algoIndex = (startIndex + i) % 6;
        switch (algoIndex) {
            case 0: hash = SpreadHash(hash.data(), hash.size()); break;
            case 1: hash = BlakeHash(hash.data(), hash.size()); break;
            case 2: hash = ShaviteHash(hash.data(), hash.size()); break;
            case 3: hash = JhHash(hash.data(), hash.size()); break;
            case 4: hash = KeccakHash(hash.data(), hash.size()); break;
            case 5: hash = EchoHash(hash.data(), hash.size()); break;
        }
    }
    return hash;
}
```

Wallet Architecture and Data Flow

Key Generation and Storage

- The wallet generates secp256k1 elliptic curve key pairs.
- Public keys are hashed and encoded into Base58Check addresses.
- Private keys are stored encrypted using a passphrase (encryptwallet command) and saved persistently in the wallet database (walletdb.cpp).
- A key pool is maintained (keypoolrefill) to allow rapid generation of new addresses.

Address and Account Management

- Addresses are grouped under “accounts” for organizational purposes.
- Commands like getaccount, setaccount, and getaddressesbyaccount facilitate management.

Transaction Creation and Signing

- The wallet constructs transactions by selecting unspent outputs (UTXOs) sufficient to cover the amount plus fees (coincontrol.h).
- Outputs include recipient addresses and change back to the wallet.
- Inputs are signed with private keys using ECDSA, incorporating compact signatures that recover the public key internally to reduce size.

Transaction Broadcast and Confirmation

- Signed transactions are broadcast using sendrawtransaction.
- The wallet listens for inclusion of transactions in blocks, updating balances accordingly.
- Balances tracked as confirmed, unconfirmed, and immature (e.g., coinbase maturity).

Mining Reward Handling

- The wallet generates coinbase transactions paying mining rewards to controlled addresses.
- Coinbase outputs mature after a defined number of confirmations (e.g., 100 blocks).

Transaction Lifecycle

- **Creation:** User or mining software initiates a transaction request.
- **Input Selection:** Wallet selects UTXOs to fund transaction.
- **Construction:** Outputs are created, including recipient and change.
- **Signing:** Wallet signs transaction inputs with private keys.
- **Broadcast:** Transaction sent to network via RPC commands.

- **Validation:** Network nodes validate transactions before including them in blocks.
- **Confirmation:** Transactions confirmed as blocks are mined on top.
- **Wallet Update:** Wallet updates balances and transaction history accordingly.

Mining RPC Commands Walkthrough

Below is a full explanation of all listed RPC commands, focusing on mining and related wallet/network operations:

Mining Control and Info

- **getmininginfo**: Provides mining statistics such as difficulty, hash rate, block height, and active algorithm rotation.
- **getblocktemplate**: Returns data needed to build a candidate block, including transactions and current difficulty. Miners must follow the multi-algorithm sequence indicated by block height.
- **submitblock**: Accepts mined block data, which must include the miner's signature and comply with multi-algo hash requirements.
- **getnetworkhashps**: Estimates total network hash rate, including all hashing algorithms.
- **setgenerate**: Enables or disables mining on the local node, allowing control over CPU thread usage.
- **gethashespersec**: Returns the local miner's current hash rate.
- **getwork**: Legacy command to request and submit work (less used with getblocktemplate).

Wallet and Transaction Management

- Commands like sendtoaddress, createrawtransaction, signrawtransaction, and sendrawtransaction facilitate creation, signing, and broadcasting of transactions, essential for spending mined coins or sending funds.
- dumpprivkey and importprivkey allow key export/import, critical for miner control of private keys required by MinerSignature.

Network and Node Management

- addnode, getpeerinfo, getconnectioncount provide tools to manage and monitor network connectivity essential for block propagation.

Account and Address Operations

- getnewaddress, listreceivedbyaddress, setaccount help manage receiving addresses, balances, and organizational grouping.

Network Layer and Peer Protocol

- The P2P protocol is Bitcoin-compatible with enhancements to support extended block headers (MinerSignature, hashWholeBlock).
- Network messages include version, block, tx, and RPC over HTTP/JSON interfaces.
- Peers verify multi-algo proof-of-work and miner signatures before relaying blocks.

Security and Cryptographic Foundations

- **Elliptic Curve Cryptography:** secp256k1 curve for key pairs and signatures.
- **Multi-Algorithm Hashing:** Combining six secure hashes increases resistance to collision and pre-image attacks.
- **Miner Signature:** Authenticates miner's ownership of block and private key, preventing unauthorized block submission.
- **Wallet Encryption:** Protects private keys with AES encryption derived from passphrases.

ZillionCoin innovatively combines multi-algorithm proof-of-work, miner key binding, and wallet integration to create a decentralized, ASIC-resistant cryptocurrency. The carefully designed mining algorithms, RPC interfaces, and wallet lifecycle ensure network security and fair mining opportunities.

Source Code Walkthrough: Mining Algorithms

SPREAD Hash (src/hash.cpp)

- This is a custom hash specific to ZillionCoin (originally from SpreadCoin).
- It combines SHA-256 rounds and a mixing function to maximize diffusion.

  Excerpt (simplified):

  ```
  uint256 SpreadHash(const unsigned char* data, size_t len) {
     uint256 hash1 = SHA256(data, len);
     uint256 mixed = CustomMix(hash1);
     return SHA256(mixed.data(), sizeof(mixed));
  }
  ```
- CustomMix applies rotations and XORs on hash bytes for extra entropy.

BLAKE Hash (src/sphlib/blake.h and .c)

- Wrapper uses the Blake2b variant optimized for 512-bit output.
- Called via:

  ```
  sph_blake512_context ctx;
  sph_blake512_init(&ctx);
  sph_blake512(&ctx, data, len);
  sph_blake512_close(&ctx, hash);
  ```
- Returns a 512-bit digest truncated to 256 bits as uint256.

SHAVITE Hash (src/sphlib/shavite.h and .c)

- Implements AES-based SHA-3 finalist.
- Similar use as Blake, via:

  ```
  sph_shavite256_context ctx;
  sph_shavite256_init(&ctx);
  sph_shavite256(&ctx, data, len);
  ```

```
sph_shavite256_close(&ctx, hash);
```

JH Hash (src/sphlib/jh.h and .c)

- Provides strong differential resistance.
- Usage pattern:

```
sph_jh512_context ctx;
sph_jh512_init(&ctx);
sph_jh512(&ctx, data, len);
sph_jh512_close(&ctx, hash);
```

KECCAK Hash (src/sphlib/keccak.h and .c)

- NIST SHA-3 standard implementation.
- Used as:

```
sph_keccak512_context ctx;
sph_keccak512_init(&ctx);
sph_keccak512(&ctx, data, len);
sph_keccak512_close(&ctx, hash);
```

ECHO Hash (src/sphlib/echo.h and .c)

- High-throughput and security-oriented hash.
- Usage:

```
sph_echo512_context ctx;
sph_echo512_init(&ctx);
sph_echo512(&ctx, data, len);
sph_echo512_close(&ctx, hash);
```

Source Code Walkthrough: Mining RPC Commands (src/rpcmining.cpp)

getmininginfo

- Returns JSON object with mining stats:

```
UniValue getmininginfo(const UniValue& params, bool fHelp) {
   UniValue obj(UniValue::VOBJ);
   obj.push_back(Pair("blocks", nBestHeight));
   obj.push_back(Pair("currentblocksize", nLastBlockSize));
   obj.push_back(Pair("difficulty", GetDifficulty()));
   obj.push_back(Pair("networkhashps", GetNetworkHashPS()));
   obj.push_back(Pair("hashalgoorder", GetAlgoOrderString(nBestHeight)));
   return obj; }
```

- GetAlgoOrderString returns current multi-algo order based on block height.

getblocktemplate

- Supplies miner with candidate block data, including transaction list and block header.
- Miner uses this to construct valid blocks compliant with ZillionCoin's rules.

submitblock

- Validates submitted block data, including MinerSignature and multi-algo hash chain.
- On success, broadcasts block to network.

setgenerate

- Enables/disables mining on local node with thread control.

```
void setgenerate(const UniValue& params, bool fHelp) {
```

```
bool generate = params[0].get_bool();

int genprocLimit = params.size() > 1 ? params[1].get_int() : -1;

SetMining(generate, genprocLimit);}
```

- Mining thread invokes multi-algo hashing during work.

Wallet Transaction Signing and Mining Reward Generation

Wallet Key Usage in Mining

- The wallet manages secp256k1 private keys required for signing coinbase transactions and generating the MinerSignature in block headers.
- Miner signs the entire block header (except hashWholeBlock) using its private key (CKey::Sign() in key.cpp).
- This signature authenticates the miner's ownership and control, preventing mining pools from unauthorized block submission.

Coinbase Transaction Construction

- During block template creation (getblocktemplate), the wallet creates a **coinbase transaction** paying the mining reward to an address it controls.
- The coinbase transaction output becomes the miner's reward after maturity.
- Coinbase transaction includes extra nonce data to allow mining iteration without reconstructing the entire block.

Transaction Signing Flow

- Transactions are constructed using inputs selected from available UTXOs (wallet.cpp).
- Inputs are signed individually by calling SignSignature() for each input, producing ECDSA signatures.
- The wallet supports compact signatures with public key recovery to minimize transaction size.

- Signed transactions can be serialized and broadcasted via RPC or mining operations.

Reward Maturity and Spending

- Coinbase outputs mature after a defined number of blocks (e.g., 100).
- Until maturity, these outputs cannot be spent or considered part of the confirmed balance.
- Once mature, rewards enter the wallet’s confirmed balance, usable for spending or staking (if applicable).

Example JSON-RPC Request and Response Samples

getmininginfo

Request:

```
{
  "jsonrpc": "1.0",
  "id": "mininginfo",
  "method": "getmininginfo",
  "params": []
}
```

Response:

```
{
  "result": {
    "blocks": 500000,
    "currentblocksize": 123456,
    "difficulty": 2300000000,
    "networkhashps": 1400000000,
    "hashalgoorder": ["SPREAD", "BLAKE", "SHAVITE", "JH", "KECCAK", "ECHO"],
    "pooledmining": false
  },
  "error": null,
  "id": "mininginfo"
}
```

getblocktemplate

Request:

```
{
  "jsonrpc": "1.0",
  "id": "template",
  "method": "getblocktemplate",
  "params": []
}
```

Response: (Simplified)

```
{
  "result": {
    "version": 4,
    "previousblockhash": "000000000abc1234...",
    "transactions": [ ... ],
    "coinbasevalue": 1250000000,
    "bits": "1a2b3c4d",
    "height": 500001,
    "algoorder": ["BLAKE", "SHAVITE", "JH", "KECCAK", "ECHO", "SPREAD"]
  },
  "error": null,
  "id": "template"
}
```

submitblock

Request:

```
{
  "jsonrpc": "1.0",
  "id": "submit",
  "method": "submitblock",
  "params": [
    "02000000abcdef..."  // Serialized block hex with MinerSignature
  ]
}
```

Response:

```
{
  "result": null,
  "error": null,
  "id": "submit"
}
```

Indicating acceptance and broadcasting.

Mining and Wallet Workflow Diagram (Conceptual)

User/Mining Software

|

v

getblocktemplate --> Wallet creates coinbase tx & block template (with miner key)

```
|
v
Local mining:
For nonce in range:
Compute MinerSignature(private key)
Compute multi-algo hash chain
Check difficulty target
|
v
If valid:
submitblock(serialized block with signature)
|
v
Network validates block --> adds to blockchain
|
v
Wallet updates balances, UTXOs, transaction history
```

Wallet and Address Commands

- addmultisigaddress <nrequired> <'["key","key"]'> [account]
  Create a multisig address with nrequired signatures.
- backupwallet <destination>
  Backup wallet.dat file.
- createmultisig <nrequired> <'["key","key"]'>
  Create raw multisig transaction.
- createrawtransaction [{"txid":txid,"vout":n},...] {address:amount,...}
  Create a raw unsigned transaction.
- decoderawtransaction <hex string>
  Decode a raw transaction hex.
- dumpprivkey <zillioncoinaddress>
  Export private key for an address.
- encryptwallet <passphrase>
  Encrypt wallet with passphrase.
- getaccount <zillioncoinaddress>
  Get account name for address.
- getaccountaddress <account>
  Get address for account.
- getaddressesbyaccount <account>
  List addresses for account.
- getbalance [account] [minconf=1]
  Get wallet or account balance.
- getnewaddress [account]
  Get new address for receiving.
- importprivkey <zillioncoinprivkey> [label] [rescan=true]
  Import private key into wallet.
- keypoolrefill
  Refill key pool.
- listaccounts [minconf=1]
  List accounts and balances.

- listaddressgroupings
  List address groupings.
- listlockunspent
  List locked unspent outputs.
- listreceivedbyaccount [minconf=1] [includeempty=false]
  List received amounts by account.
- listreceivedbyaddress [minconf=1] [includeempty=false]
  List received amounts by address.
- listsinceblock [blockhash] [target-confirmations]
  List transactions since block.
- listtransactions [account] [count=10] [from=0]
  List recent transactions.
- listunspent [minconf=1] [maxconf=9999999] ["address",...]
  List unspent outputs.
- lockunspent unlock? [array-of-Objects]
  Lock or unlock unspent outputs.
- makekeypair [prefix]
  Generate new key pair.
- move <fromaccount> <toaccount> <amount> [minconf=1] [comment]
  Move balance between accounts.
- sendfrom <fromaccount> <tozillioncoinaddress> <amount> [minconf=1] [comment] [comment-to]
  Send from account to address.
- sendmany <fromaccount> {address:amount,...} [minconf=1] [comment]
  Send to multiple addresses.
- sendrawtransaction <hex string>
  Broadcast raw transaction.
- sendtoaddress <zillioncoinaddress> <amount> [comment] [comment-to]
  Send to address.
- setaccount <zillioncoinaddress> <account>
  Set account for address.

- setmininput <amount>
  Set minimum input for spending.

- settxfee <amount>
  Set transaction fee.

- signmessage <zillioncoinaddress> <message>
  Sign message with address private key.

- signrawtransaction <hex string>
  [{"txid":txid,"vout":n,"scriptPubKey":hex,"redeemScript":hex},...]
  [<privatekey1>,...] [sighashtype="ALL"]
  Sign raw transaction.

- validateaddress <zillioncoinaddress>
  Validate an address.

- verifychain [check level] [num blocks]
  Verify blockchain integrity.

- verifymessage <zillioncoinaddress> <signature> <message>
  Verify signed message.

Network and Node Commands

- addnode <node> <add|remove|onetry>
  Add or remove node.

- getaddednodeinfo <dns> [node]
  Info on added nodes.

- getconnectioncount
  Number of connections.

- getpeerinfo
  Peer info.

- getrawmempool
  List unconfirmed txs.

- stop
  Stop the node.

Blockchain Commands

- getbestblockhash
  Get latest block hash.
- getblock <hash> [verbose=true]
  Get block info.
- getblockcount
  Get block height.
- getblockhash <index>
  Get block hash by height.
- getdifficulty
  Get current difficulty.
- getinfo
  General node info.
- getrawtransaction <txid> [verbose=0]
  Get raw transaction data.
- getreceivedbyaccount <account> [minconf=1]
  Amount received by account.
- getreceivedbyaddress <zillioncoinaddress> [minconf=1]
  Amount received by address.
- gettransaction <txid>
  Transaction info.
- gettxout <txid> <n> [includemempool=true]
  Get transaction output info.
- gettxoutsetinfo
  UTXO set stats.

Mining Commands

- getgenerate
  Mining enabled status.
- gethashespersec
  Local miner hash rate.
- getmininginfo
  Mining stats including multi-algo order.
- getnetworkhashps [blocks] [height]
  Network hash rate.
- getwork [data, coinbase]
  Legacy mining work distribution.
- submitblock <hex data> [optional-params-obj]
  Submit mined block.
- setgenerate <generate> [genproclimit]
  Enable/disable mining.

Raw Transaction and Block Construction

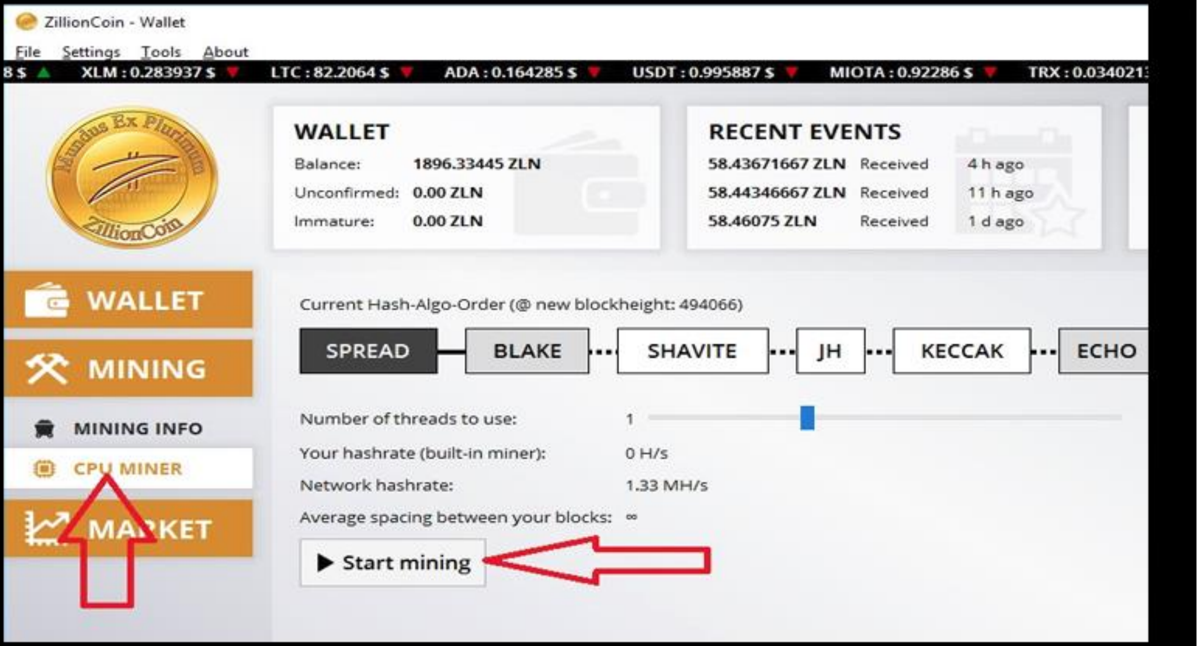
- createrawtransaction [{"txid":txid,"vout":n},...] {address:amount,...}
  Create raw transaction.
- decoderawtransaction <hex string>
  Decode raw transaction.

COIN SPECS:

- ZillionFLUX CPU Algo
- 250 Mln Total Supply
- Block Reward: 66ZLN
- Block Time: 1 min
- Supporting the ZillionGRID

How to get
FREE ZillionCoins

1. Download wallet at www.ZillionCoin.com
2. Run Wallet
3. Sync Wallet
4. Go to “Mining”
5. Go to “CPU Mining”
6. Click “Start Mining”



WALLET

Balance: 1896.33445 ZLN
Unconfirmed: 0.00 ZLN
Immature: 0.00 ZLN

RECENT EVENTS

58.43671667 ZLN	Received	4 h ago
58.44346667 ZLN	Received	11 h ago
58.46075 ZLN	Received	1 d ago

Current Hash-Algo-Order (@ new blockheight: 494066)

Number of threads to use: 1
Your hashrate (built-in miner): 0 H/s
Network hashrate: 1.33 MH/s
Average spacing between your blocks: ∞

www.ZillionCoin.com

ZILLIONCOIN IS PART OF THE ZILLIONGRID

ZillionGrid:
Decentralized
Hybrid
Blockchain
Infrastructure

https://www.zillionpress.com/ZillionGrid_WhitePaper_Version1_Phase1.pdf