Atomchain: An Alternative to Blockchain

Author: Okpara Okechukwu D.
Email: okpara.net@gmail.com
Website: www.okpara.net
Date: 17-04-2022

Abstract:
We propose the first alternative architecture and technology to blockchain, called the **atomchain** (also, referred to as the **atomsphere**), which aims to eliminate the multiple spending of coins problem, 51% attack, Race attack and Finney attack. The network has a secure way of keeping the sequence of transactions when new entries are added. The network timestamps transactions using a double-hashing encapsulation technique (involving Merkle Root and **Atom Root**), then links the messages to the last valid **atom**, called the **atomsphere**, which is broadcasted to nodes base on their **elevation modes and pools**. Atomchain is perceived to be more efficient than blockchain by design; for instance, the energy and resources that are spent as a result of "**forks**" in blockchain; either does not exist or is absorbed by this new alternative. The overarching aim of atomchain is to avoid both past and present mistakes, bugs, and inadequacies of blockchain, while providing a solid, stable, scalable, sustainable, and secure decentralized architecture for both the present and future stages of web 3.0.

Keywords: web3, atomchain, blockchain, Atom Root, Merkle Root

1. Introduction

The invention of bitcoin (which is a peer-to-peer electronic cash system) and blockchain introduced a new era of using digital signatures to seamlessly make secure and trustless electronic transactions [1]. A blockchain is a distributed, peer-to-peer database across a network of computer nodes that hosts a continuously growing number of transactions. Each transaction is secured through cryptography, timestamped, and validated by all authorized nodes using consensus algorithms.

In cryptocurrencies, users' balances are recorded in a database or **ledger** – the blockchain. It is essential that every node maintains an identical copy of this database [2]. Unfortunately, in blockchain it is quite possible to get a fraudulent transaction by an attacker approved, for instance, the Nakamoto consensus protocol provides no security when 51% of the miners fail – or become Byzantine[1]. There are also issues with Proof of Identity consensus algorithms, such as minorities being side-lined or manipulating smaller blockchain networks. Hence, it is easy for a group of attackers to enter a blockchain with many different devices, and once they form a majority, they can get their transaction approved. Both the "Proof of Identity" and the "Proof of Stake" methods in blockchain can be used in the 51% attack. A successful 51% attack can be used for several malicious acts, such as gaining a mining monopoly in a Proof-of-Work (PoW) network, and preventing some or all other users from mining, thus gaining all the rewards from mining. They can also tamper blocks in the blockchain in such a way that it looks like the multiple spending of the same cryptocurrency transactions never occurred.

Moreover, when a miner uses "Proof of Work" to validate a transaction, they are given a complex mathematical problem that requires a great amount of computational power to solve — which results in high electricity bills and enormous amounts of energy consumption.

In this paper, a more efficient and highly secure alternative to the blockchain is conceptualized.

2. Atomchain versus Blockchain

Network

The steps to run the atomchain network are as follows:

1. New transactions are broadcast to all active **nodes/miners**.
2. Each node/miner collects new transactions into an atom from their respective **transaction pools**.
3. The nodes/miners with valid atoms are placed in an **elevation pool**, and their valid atoms are placed in an **atom pool**.
4. When a node in the elevation pool successfully adds its atom to the atomchain, it broadcasts the **atomsphere** to all nodes.
5. Nodes accept the atomsphere only if all transactions in it are valid and not already spent.
6. Nodes express their acceptance of the atomsphere by working on creating the next atom in the atomchain.

Nodes consider the atomsphere to be the prevailing correct atom and will keep working on extending it.

Problems/Attacks with Blockchain System

1. The 51% Attack: here it is assumed that a miner owns more than 50% of the computing power of the network. The attacker mines a private blockchain where he double-spends (multi-spends) the coins. With majority of computing power, the miner is guaranteed that their private blockchain at some point in time would be longer than the chain of "honest" network. They then release their private blockchain in the system making all the transactions earlier recorded in the honest blockchain to be invalid.
2. Race Attack: this is when an attacker sends the same coin to different vendors in rapid succession, probably by using many different machines. If the vendors do not wait for the block confirmation before delivering the goods, they will soon realize that the transaction was rejected during the mining process.
3. Finney Attack: this is when a miner mines a block with their transaction and does not release it in the system. They now use the same coins in a second transaction and then releases the pre-mined block. Certainly the second transaction would be rejected eventually by other miners, but this will take some time.

Mitigation of Problems/Attacks using Atomchain

1. The 51% Attack: by design, atomchain eliminates this attack, because it is not based on the **longest blockchain rule**.
2. Race Attack: vendors should wait for at least one sphere confirmation (or addition of an atom) before sending out their goods; for example, a digital asset. However, the use of double-hashing functions can prevent this attack from ever happening.
3. Finney Attack: by design, the use of elevation modes and atom pools may not allow this attack to happen after at least two sphere confirmations, or the addition of two atoms.

Resolving Conflicts: Blockchain & Atomchain

1. In Blockchain: when two different miners solve the consensus algorithm, for instance, the Proof-of-Work (PoW), at the same time and add their blocks to the last known block in the blockchain. The longest branch always wins subsequently, and the transactions in the shorter chains are returned to the transaction pool before the shorter chains are purged.

   In Atomchain: this conflict described above is resolved by the miners in question. Before a node adds a valid atom as the atomsphere, it first obtains the hash of the last Nucleus, does some computations, timestamps, checks the elevation status of the miner on the network, and so on. The other miners whose atoms were rejected are granted higher elevation statuses.

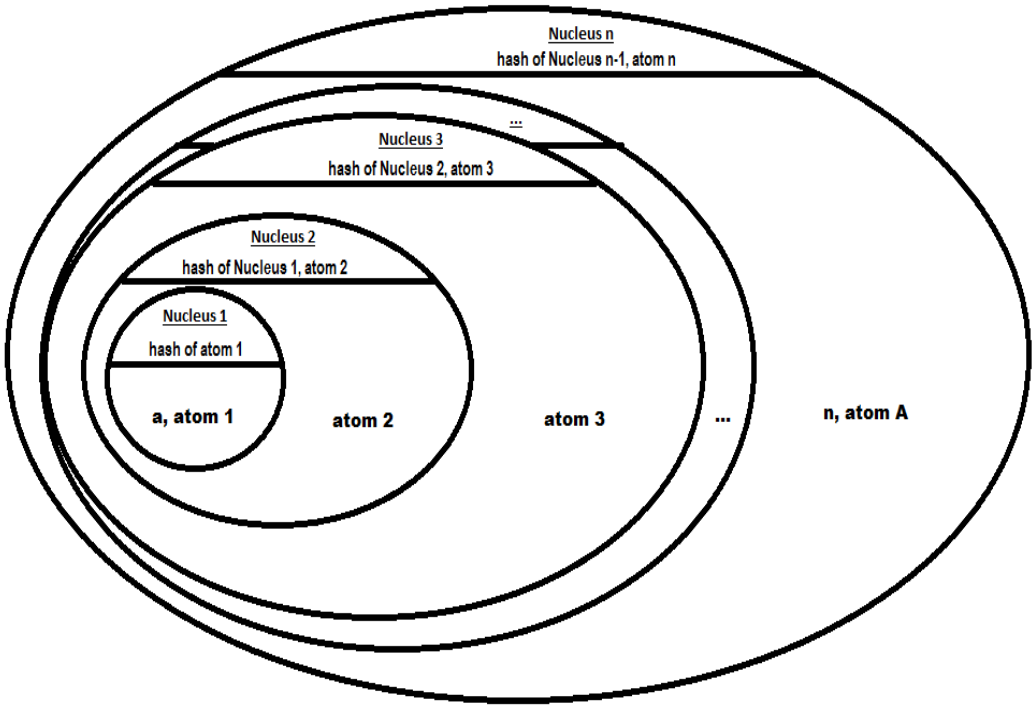
2. In Blockchain: in a situation where the computing powers of many different miners are tremendous, in such a way that, these "super miners" keep having their blocks added to the blockchain even after the "difficulty target" for the PoW algorithm has been automatically adjusted by the blockchain (for instance, in bitcoin) system from a lesser difficulty to a higher difficulty level.

   In Atomchain: this inherent and unfair advantage of super miners over miners with very low computing power in the network is reduced or eliminated with the concept of Elevated Modes and Pools.

   There are attacks targeted at the peer-to-peer (P2P) networks on which both Blockchain and Atomchain are based. For instance, the Sybil attack, in which an attacker pretends to be so many nodes at the same time when connecting to a P2P network, creating multiple fake identities, can be resolved using the Proof-of-Work algorithm, but in Atomchain, this type of attack (as well as the Eclipse attack – which is about eclipsing certain nodes of the network) are prevented using Elevation modes.

3. Concept: The Atomchain Process

While creating the encapsulation of atoms (see the first Figure), we observe the rule that hash of the previous atom is added to the current atom, that is, it then becomes the nucleus of the current atom. Thus, a miner while creating the atom, picks up the hash of the last atom, and another hash in the atom preceding the last atom, then combines it with its own set of messages, and creates a Merkle and atom root for its newly created atom. This newly created atom now becomes the new endpoint for the atomchain, and thus the atomchain keeps on growing as more and more atoms are added to it by miners.



Encapsulation of atoms: a is atomcore, A is atomsphere

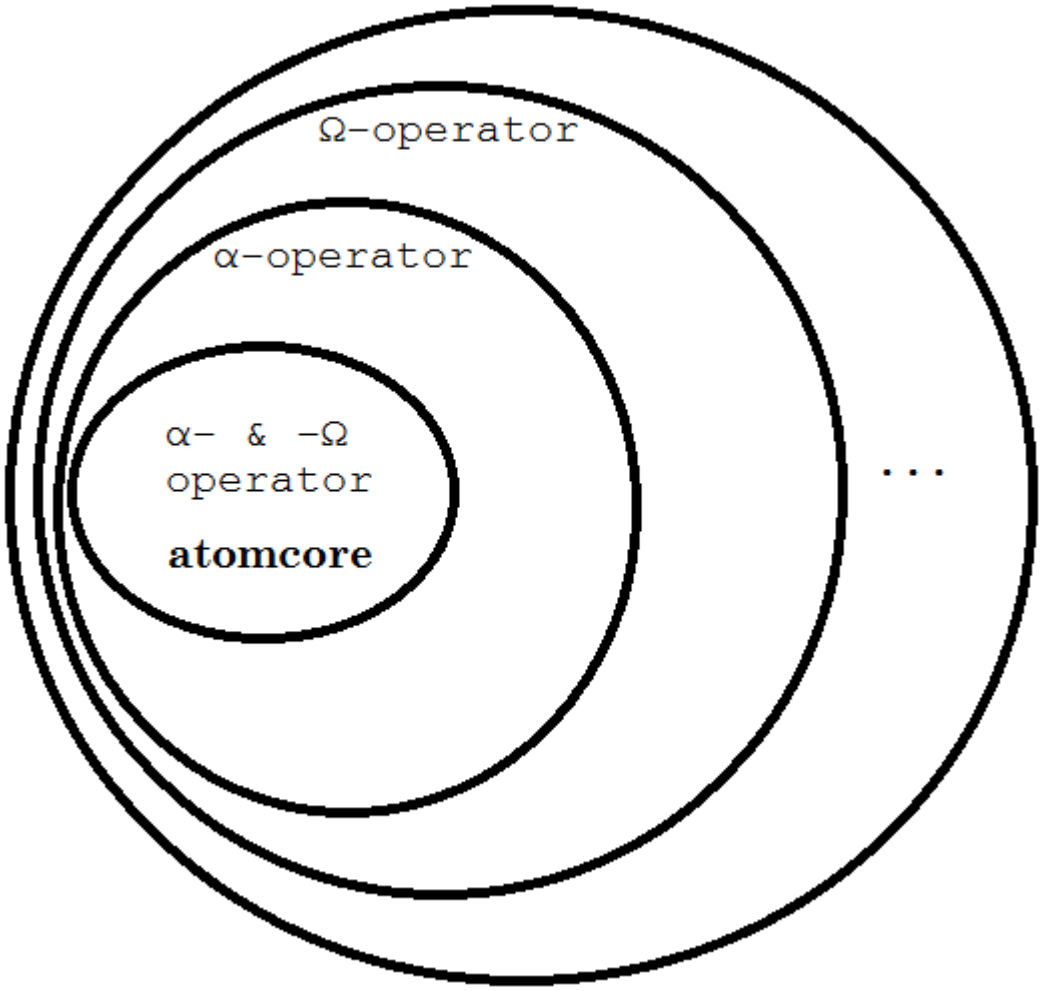
Note: The use of the term, "**atomchain**" was for easy interpretation of the various aspects of the technology that are similar to blockchain. The author of atomchain actually called it "**atmosphere**," and he referred to **atoms** as **spheres**, just like **blocks** in blockchain.

An atom contains a nucleus and a body (which has transaction data). A α-transaction list is an ordered list of α-transactions. Atom size is usually limited to 1 megabytes.

The PHP code for adding a non-coinbase n-list of α-transaction (αTXn)

```
/* New αTX
@param $senderHome
@param $senderαAddress
@param $recipientαAddress
@param $amount
@param $TXcount
@return bool
*/

public function createNewTX($senderHome, $senderαAddress,
$recipientαAddress, $amount, $inCount, $outCount){
$TX1 = [
      'inCount' => …,
      'timestamp' => time(),
      'TXin' => $senderαAddress,
      'TXout' => $recipientαAddress,
      'amt' => $amount,
      'outCount' => …
];

$TX2 = [
```

```
      'inCount' => …,
      'timestamp' => time(),
      'TXin' => $senderαAddress,
      'TXout' => $recipientαAddress,
      'amt' => $amount,
      'outCount' => …
];

...

$TXn = [
      'inCount' => …,
      'timestamp' => time(),
      'TXin' => $senderαAddress,
      'TXout' => $recipientαAddress,
      'amt' => $amount,
      'outCount' => …
];

$this -> version = $ver;
$this -> TX1[] = $TX1;
$this -> TX2[] = $TX2;
...
$this -> TXn[] = $TXn;

}

/*
@var_array txList
*/
private $TX1; private $TX2; private $TXn; private $ver;


// home + hash = signature
// atom network nodes can use the signature to deduce the world, and
// use the world to verify the signature while comparing the data
```



α-operator atom is added to the atomchain after the Atomcore followed by an Ω-operator atom. This alternating procedure continues as more atoms are added to the atomchain. An α- and Ω-operator atom should be used for protocol upgrades: hydrogen, helium and so on.

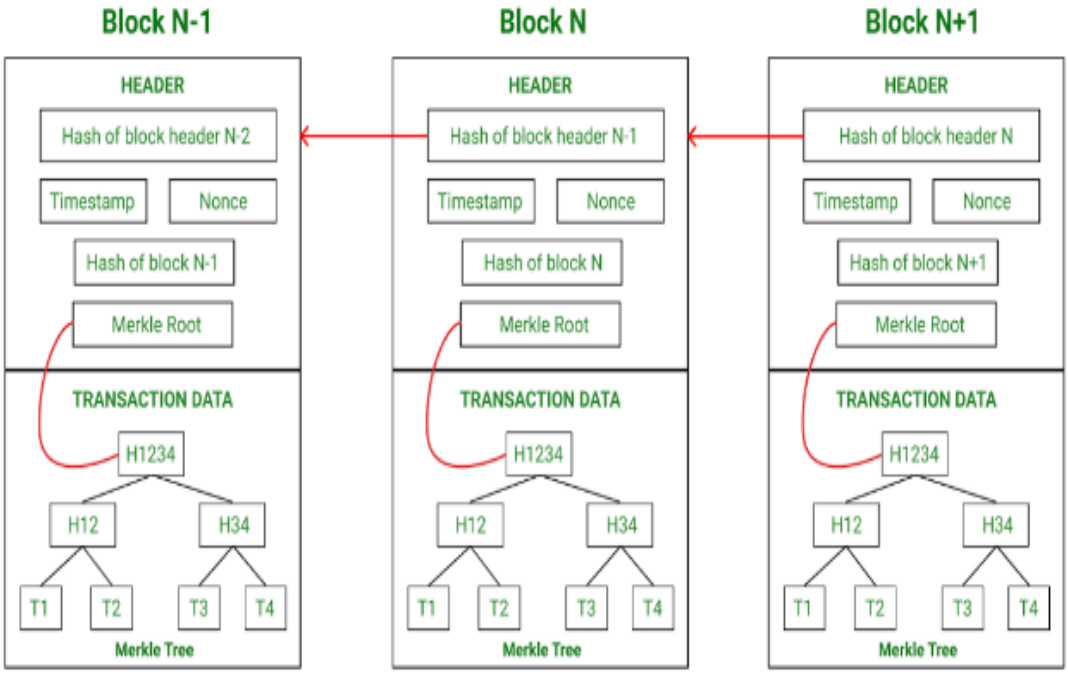
To start the mining process, a node or miner must create a nucleus. This includes at least these pieces of information:

1. Version number.
2. Previous Nucleus Hash. The hash value of nucleus of the previous atom. This ensures the sequence of atoms in the atomchain, and also prevents an attacker from altering the nucleus of old atoms.
3. AM-Roots (Atom/Merkle Roots). Merkle Tree is used as a fingerprint of α Transaction List. This prevents an attack from altering any transaction included in the atom.
4. Timestamp: the approximate creation date and time of the atom displayed in Unix epoch, which is seconds since January 1, 1970.(http://en.wikipedia.org/wiki/Unix_time).
5. The DNA.

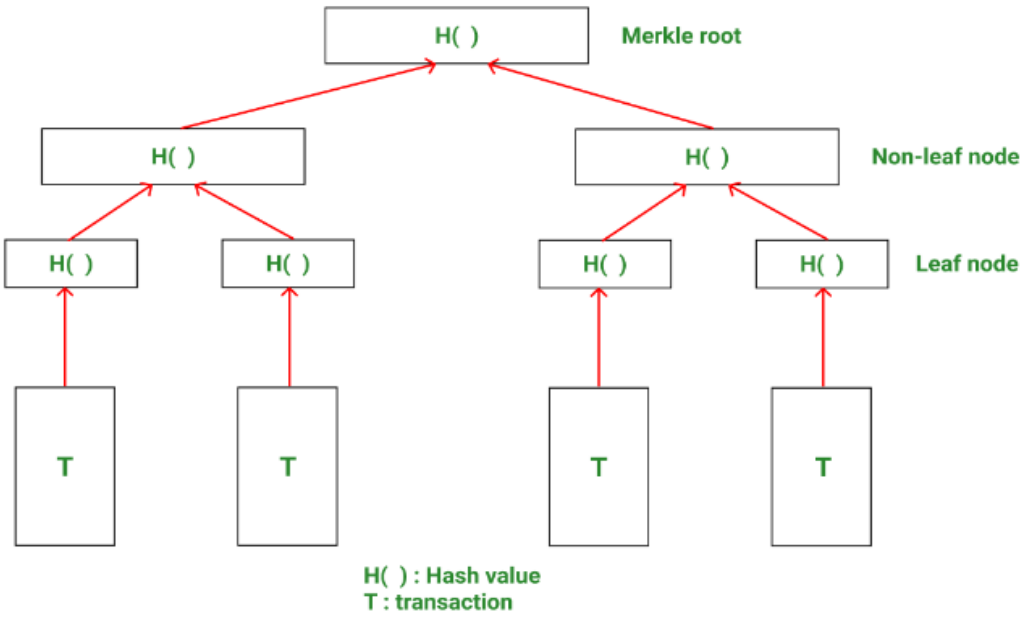
4. Methods: The Two Technologies

The basic blockchain is a hash-based linked list of blocks where each block consists of header and transactions. The transactions are arranged in a tree-like fashion known as the Merkle Tree [3] [4]

A Merkle tree is a binary tree formed by hash pointers. A hash pointer is a pointer to where data is stored and with the pointer, the cryptographic hash of the data is also stored. A hash pointer points to the data and also allows us to verify the data. It can be used to build data structures. The structure of the blockchain is a linked list of hash pointers, which are created from hash functions such as SHA-256. The red arrows in the diagram below represent hash pointers. [4]



Each block comprises of block header + Merkle tree



Structure of Merkle tree

The Block Header contains:

- Hash of the previous block header
- Hash of the current block
- Timestamp
- Merkle Root
- Cryptographic Nonce (Number only used once)

Additionally, block header may include:

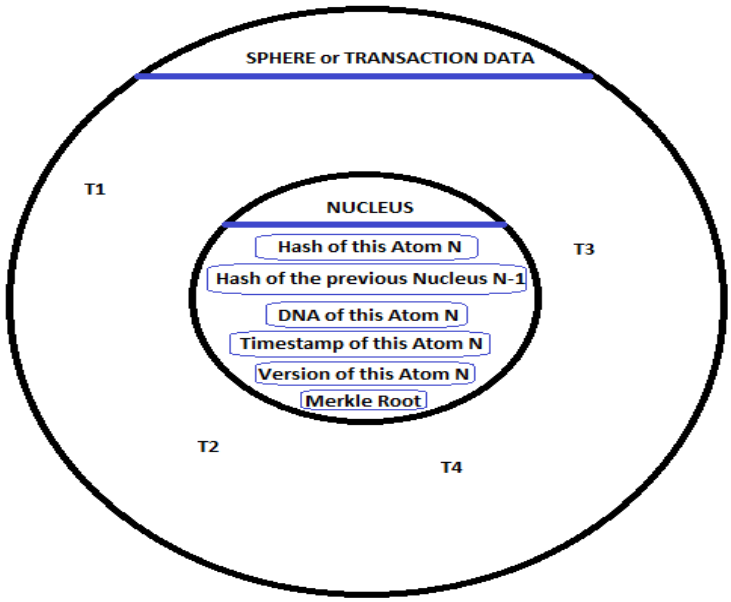
- Block version number (that indicates which set of block validation rules to follow)
- Difficulty Target (that may involve the use of algorithms like SHA-3 (Secure Hash Algorithm 3), RIPEMD160 (RACE Integrity Primitives Evaluation Message Digest 160-Bits), MD5 (Message Digest Method 5), BLAKE2 and so on.

The Nucleus of an atom contains:

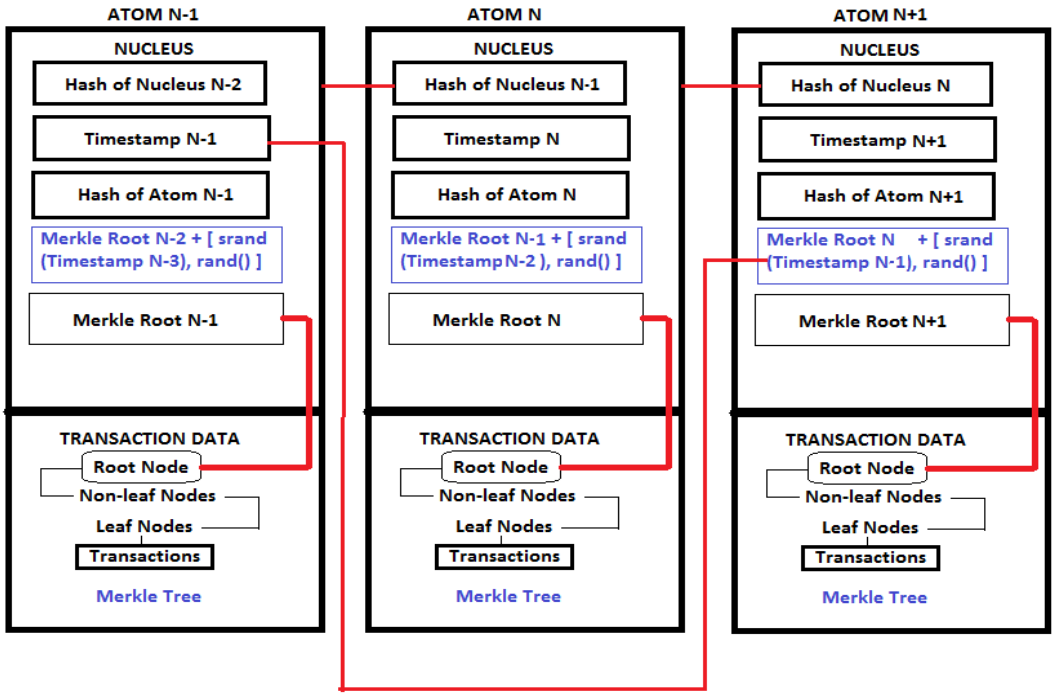
- Hash of the previous nucleus: the Keccak 256-bit hash of the parent nucleus, in its entirety
- Hash of the current atom
- Timestamp: a scalar value equal to the reasonable output of Unix's time() at this atom's inception
- Merkle Root
- Atom version number
- Atom Index: a scalar value equal to the number of ancestor atoms. The atomcore has an index of zero, 0.
- Atom Root

- The DNA
- stateRoot: The Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied.
- transactionsRoot: The Keccak 256-bit hash of the root node of the trie structure populated with each transaction in the transactions list portion of the atom.
- receiptsRoot: The Keccak 256-bit hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list portion of the atom.
- atomOkebase**:** The 160-bit address to which all fees collected from the successful mining of this atom will be transferred to. (See Section 4, "Inside atomcore.json State File" of the Atomchain Network Whitepaper).
- energyMAX: A scalar value equal to the current limit of energy expenditure or consumption per atom.
- energyUsed: A scalar value equal to the total energy used in transactions in this atom.
- atomBank: An arbitrary byte array containing data relevant to this atom. It can occupy a maximum of 32 bytes.

The hash of the previous nucleus is like a reference to the previous atom, and a Merkle Root is like a list of all of the transactions that have taken place since the previous atom.



First draft of the basic structure of an Atom, N, with Transactions/Messages T1, T2, T3, T4



First Draft of the Atomchain

From the first draft of the atomchain, it could be observed that the data stored in an atomchain must have the following characteristics just as in blockchain: immutable, “unhackable”, persistent (no loss of data), and distributed. These qualities are necessary to maintain the integrity of the network, and the security of the network within which the transactions occur [5].

A simple PHP programming language code for a block in a Blockchain is written below:

```
<?php

class Block
{

public $nonce;                                  //declared so that it can be changed externally

public function __construct($blockindex, $timestamp, $merkleRoot, $data, $proof, $prevHash = null)
        {
        $this->blockindex = $blockindex;    //index number of the current block
        $this->timestamp = $timestamp;    //timestamp for the current block
        $this->merkleRoot = $ merkleRoot;          //Merkle Root of the Merkle Tree of the //transactions in
                                                   the body of the current block
        $this->proof = $proof;                     //for example, proof-of-work or proof-of-stake.
        $this->data = $data;                       //transactions or messages of the current block
        $this->prevHash = $prevHash;               //hash of the previous block header
        $this->hash = $this->calcHash();           //function to calculate the hash of the current block
        $this->nonce = 0;                       //the Nonce, set to zero by default

        }

        public function calcHash()
```

```
        {
        return hash("sha256", $this->blockindex.$this->prevHash.$this->merkleRoot.$this-
        >timestamp.((string)$this->data).$this->nonce);                    //returns the SHA-256 hash of
        the current block
        }
}
?>
```

A simple PHP programming language code alternative to the above blockchain code in an Atomchain – where the block has been replaced by the atom, is shown below:

```
<?php

class Atom
{
        public $merkle_root_N-1 = "...";          // Merkle Root of the N-1 atom
        public $timestampN-2 = "...";             //timestamp of the N-2 atom

        public function __construct($atomIndex, $timestamp_N, $merkleRoot_N, $data, $prevNucleusHash =
null)
        {
        $this->atomIndex = $atomIndex;    //atom index of N atom
        $this->timestamp_N = $timestamp_N;        //timestamp of N atom
        $this->merkleRoot_N = $merkleRoot_N;      //Merkle Root of N atom
        $this->data = $data;                      //transaction list of N atom
        $this->prevNucleusHash = $prevNucleusHash;                  //hash of the previous Nucleus
        $this->hash = $this->calcHash();          //function to calculate the hash of the current atom
        $this->hash = $this->calcAtomDNA();       //a function to get the atom's DNA




        /*
        private function calcHash()               //calcHash() function private implementation
        {
        $atomArray = [
                'atomIndex'         => $this->atomIndex,
                'timestamp_N'  => $this->timestamp_N,
                'data'  => $this->data,
                'prevNucleusHash'  => $this->prevNucleusHash,
                'merkleRoot _N'  => $this->merkleRoot _N,
                'calcAtomDNA'  => $this->calcAtomDNA(),
                'ver'  => $this->ver,
                'calcAtomRoot'  => $this->calcAtomRoot()

                ];
                $atom = json_encode($atomArray); // helps to keep the array sorted before hashing
                return hash('sha256', $atom);
        }
        */

        /* Atom Root code section */
        }

        public function calcHash()          //calcHash() function public implementation
        {
```

```
        return hash("sha256", $this->atomIndex.$this->prevAtomHash.$this->merkleRoot_N.$this-
        >timestamp_N.((string)$this->data).$this->calcAtomDNA().$this->ver.$this.calcAtomRoot());
                //returns the SHA-256 hash
        }

        public function calcAtomDNA()
        {
        return hash("sha256", $this->calcAtomRoot().$this->merkleRoot_N-1);  //returns SHA-256 hash
        }

}
?>
```

Note that, in addition to the Merkle Root, the atomchain has another hashing element in its **Nucleus** called the **Atom Root**. A simple implementation of the Atom Root in PHP language is written below:

```
<?php

/*
where, N refers to the last atom, the atomsphere
N-1 refers to the first to the last atom,
N-2 refers to the second to the last atom
*/

//define ("MAX_SEED_LENGTH", 128); //use first 128 bits entries for $n, ignore this line
$merkle_root_N-1 = "..." //the Merkle Root of the previous atom, N-1
$timestamp_N-2 = "..."; //get the value of the timestamp of N-2 atom
srand($timestamp_N-2);  //set the seed for the rand() function
$atom_seedN-2 = rand(); //generates a random integer
$timestamp_N-1 = "..."; //get the value of the timestamp of N-1 atom
srand($timestamp_N-1);  //set the seed for the rand() function
$atom_seedN-1 = rand(); //generates a random integer
$atom_hash = hash("sha512", $merkle_root_N-1.$atom_seedN-2); //concatenation



Function atom_root($atom_hash){   //the Atom Root is returned by this function
        $root_seed = md5($atom_seedN-1);
        //the above line can be replaced with the following line (uncommented) below:
        //$root_seed = md5(uniqid($atom_seedN-1, true));


/*
Note that the uniqid() function is used to create a salt, and it is set to true so as to increase the chance of the generated number being unique. However, we may use SHA1() or RIPEMD160() function which produces 160-bit(20 bytes) hash, instead of the md5() function which creates 128-bit hash.
*/

        //$fnx_n = substr($n, 0, MAX_SEED_LENGTH); //trims $n to 128 entries, ignore this line
        $root_salt = hash("RIPEMD160",$root_seed);
        return hash("sha512", $atom_hash.$root_salt); //produces 512-bit hash

        }

?>
```

When implementing an atomchain, it is easy to write program codes for it in different programming languages, for instance, the PHP code for the function calcHash() can be written in JavaScript as:

```
constcalcHash = (atom) =>{
const data = JSON.stringify(atom.data);
constatomData = data +atom.prevNucleusHash + atom.timestamp_N.toISOString() + atom.
calcAtomDNA().toString();
return createHash("sha256").update(atomData).digest("hex");
//
// the code above can be simply be written as:
// return SHA256(this.prevNucleusHash + this.timestamp_N + JSON.stringify(this.data)).toString();
};
```

Another example, thecalcHash() function can be defined in JavaScript Atom class as:

```
this.hash = this.calcHash(); //in php, it is: $this->hash = $this->calcHash();
```

Moreover, the Atom class that was written in PHP earlier on, may be written in Python language as:

```
class Atom:
    def __init__(self, atomindex, data, timestamp_N, prevNucleusHash, merkleRoot_N, AtomDNA=0):
        self.atomindex = atomindex;                    #atom index of N atom
        self.timestamp_N = timestamp_N;                #timestamp of N atom
        self.merkleRoot_N = merkleRoot_N;              #Merkle Root of N atom
        self.data = data;                              #transaction list of N atom
        self.prevNucleusHash = prevNucleusHash;        #hash of the previous Nucleus
        self.AtomDNA = AtomDNA;                        #the atom's DNA
```

A sample atom hash calculation in Python is given below:

```
import datetime
import hashlib
import binascii

version = "…" # current version number is 1
hashPrevNucleus = \
    "…"
hashMerkleRoot =  \
    "…"
time = datetime.datetime(2022,5,22,10,20,30) # May 21, 2011 10:20:30 AM
time = int(time.timestamp()) # in POSIX timestamp
time = hex(int(0x100000000)+time)[-8:] # in 4-byte hexadecimal notation
#print(time)

# Converting to little-endian hexadecimal notation
version = binascii.hexlify(binascii.unhexlify(version)[::-1])
hashPrevNucleus = \
    binascii.hexlify(binascii.unhexlify(hashPrevNucleus)[::-1])
hashMerkleRoot = \
    binascii.hexlify(binascii.unhexlify(hashMerkleRoot)[::-1])
time = binascii.hexlify(binascii.unhexlify(time)[::-1])

#Concatenating nucleus values
nucleus = version+hashPrevNucleus+hashMerkleRoot+time+...
```

```
#Calculate the double-SHA256 hash value
nucleus = binascii.unhexlify(header)
hash = hashlib.sha256(hashlib.sha256(nucleus).digest()).digest()
hash = binascii.hexlify(hash)

#Convert the hash value to big-endian hexadecimal notation
hash = binascii.hexlify(binascii.unhexlify(hash)[::-1])
```

Finally, writing program codes for Atomchain in any other object-oriented programming languages; such as C/C++ and GO, should not be difficult to implement.

BYTES	BITS	NAME	DATA TYPE	DESCRIPTION
1	8	WFLG	int8	Wallet Flag. 0 = α & Ω nucleus operation, 1 = α nucleus operation, 2 = Ω nucleus operation
1	8	Ver	int8	Version bits
32	256	CMRH	char[32]	Current Merkle Root Hash. For only one TXID, CMRH = coinbaseTXID, If with coinbaseTXID & another TXID1, CMRH = SHA256(SHA256(coinbaseTXID + TXID1)), If with many TXIDs, CMRH = SHA256() of the pairs SHA256(coinbaseTXID+TXID1)+ SHA256(TXID2+TXID3)… If the last TXID is an odd number, then it is paired to itself SHA256(TXIDn + TXIDn)
32	256	PNH	char[32]	Previous Nucleus Hash
16	128	Time	int16	Current Timestamp, in Unix Epoch Time. e.g. 1305998791is 4dd7f5c7 in hex and c7f5d74d in little-endian
32	256	CAH	char[32]	Current Atom Hash, SHA256(CMRH+DNA+PNH+Ver+Time+CAH +ART+AtomID+AI)
64	512	ART	char[64]	Atom Root`, SHA512(ASH+RST)
16	128	AtomID	int16	Atom ID
32	256	DNA	char[32]	Decentralize Number of the Atom, SHA256(ART+CMRH)
32	256	EVT	char[32]	Events Root
32	256	ENT	char[32]	Evanescent Root
32	256	THG	char[32]	Things Root
32	256	SMP	char[32]	Samples Root
2	16	pTime	uint16	Previous atom’s timestamp
2	16	ppTime	uint16	Last atom to the previous atom’s timestamp
2	16	pAtomseed	uint16	Previous atom’s Atom Seed pAtomseed = srand(pTime), rand()
		ppAtomseed	uint16	Last atom to the previous atom’s Atom Seed ppAtomseed = srand(ppTime), rand()
32	256	PMRH	char[32]	Previous Merkle Root Hash
16	128	RSD	char[16]	Root Seed, MD5(pAtomSeed)
64	512	ASH	char[64]	Atom SHA hash, SHA512(PMRH + ppAtomSeed)
20	160	RST	char[20]	Root Salt, RIPEMD160(RSD)
2	16	AI	int16	Atom Index, that is the atom height

All hashes are in internal byte order; and values are usually in little-endian order. But sometimes during certain data transmission numbers may be implicitly encoded in big-endian integers of lengths that are in multiples of 8 bits.

6. Efficiency of Atomchain: the AM-Roots or Double Hash Roots

Some may argue that the combined use of both Merkle Root and Atom Root in atomchain reduces the speed and increases the bandwidth of the system when compared to a typical blockchain. However, the advantages associated with the concept of AM-Roots outweigh its disadvantages.

The Atom Root is very important since it incorporates an intrinsic **Double Factorial** function. [6]

Mathematically, the double factorial of a number *n*, is denoted by *n*!!, and if *n* is even, then

n!! = *n*(*n*-2)(*n*-4)(*n*-6)...(4)(2). And if *n* is odd, then, *n*!! = *n*(*n*-2)(*n*-4)(*n*-6)...(3)(1)

From the PHP program code for the Atom Root in Section 6, where: N refers to the atom,N-1 refers to the last atom before the N atom, N-2 refers to the last atom before the N-1 atom
We observed that if there is only one atom in the atomchain such that this single atom represented with *n* is both the **atomcore** and the **atomsphere**, then in line with the double factorial convention, *n*!! = 1 for both *n* = 0 (for the N-1 atom)and *n* = -1 (for the N-2 atom).

7. Compatibility and Future Work

Compatibility is crucial. With the current hype surrounding blockchain, the plethora of DApps, NFTs and cryptocurrencies, it is essential to support blockchain in atomchain. Hence, we made it possible to create a typical smart contract with the Solidity language on an atomchain that has been configured as a basic blockchain.

Merkle Trees (cryptographic hash functions) may be replaced with Verkle Trees (Vector Commitments) in atomchain. In Merkle Tree, a parent node is the cryptographic hash of its children, while in Verkle Tree, a parent node is the Vector Commitment of its children [7].

Latest version of the atomchain whitepaper is available online at https://okpara.net/Atomchain.pdf

8. Conclusion

Atomchain is a distributed ledger technology meant to form transparent and decentralized systems that are fool-proof. It is assumed that it will have less scalability issues than current blockchains. Since it has no Proof-of-Work or other similar consensus mechanisms[8] that are present in many blockchains, it is quite energy saving and efficient. The use of Atom Roots and Atom Pools provides the network with added security and efficiency.

Atomchain by design is expected to outperform the blockchain in many areas of applications. It has an excellent and intuitive **technology depth** based on good security and great consensus. It is expected to support multi-atoms on multiple platforms, thus it will have a wide **technology breadth**.

From the Woof Paper [9] under the section titled “Shiba Ecosystem Tokenomics”, a quote from Miyamoto Musashi says, “it is difficult to understand the universe if you only study one planet”. So why use only blockchain, when there are others, such as atomchain?

9. Appendix A: Terminology and Definitions

Account: An object that have an intrinsic balance and transaction count maintained as part of the atomchain state.

Address: a 160-bit code used for identifying accounts, including quark accounts.

Atom: An atom in atomchain is similar to a block in blockchain; and the “encapsulation” or “bonding” of atoms is like the “chaining” of blocks in a blockchain.

Atomchain: is essentially an encapsulation and double-linking architecture containing atoms created by miners.

Atomcoin: this is the first encryption token of atomchain. Transaction fees in atomchain are handled or paid using atomcoins. It is the main internal crypto-fuel of Atomchain.

Atomcore: this is what the innermost sphere (or first atom) is specifically called. It is similar to the genesis block of blockchain.

Atomicity: for any transaction in Atomchain this implies that, a transaction is either valid (and confirmed) or not. Partial transactions cannot be mined and there is no interim state for a transaction. At any point in time a transaction is either in an atom (or validated and mined), or not confirmed/validated.

Atomsphere: this is what the outermost sphere (or last atom) is specifically called.

Atom Chainware: this is software or hardware dedicated to running atomchain, such as Atomchain Virtual Machine Ware(VMWare).

Atom Level: it is a scalar integer that corresponds to an atom’s number on the atomchain. It is a measure of an atom’s size or density. In blockchain, this is similar to the ‘block height’. It will be used for protocol updates on atomchain’s main network.

Atom Pool: During consensus, all valid atoms that were not added to the atomchain are returned to a waiting area called an Atom Pool. In contrast to transaction pools, all transactions in atom pools are confirmed.

Atom Reward: is the amount of encrypted tokens (or cryptocurrencies) credited to a miner’s account after the miner successfully adds an atomsphere. The reward includes the “fees” or charge paid by the initiators of the transactions.

Atom Time: is the time it takes a peer-to-peer network to create an atomsphere in the atomchain. The process requires the network to ensure that the data in the new atom is consistent with the constraints of the network.

Atom Wallet: this is digital software that stores homes and worlds, and also monitors and keeps all the transactions related to them on an atomchain. It allows users to store, send, receive, and manage their single/multiple digital assets (such as cryptocurrencies) on atomchain. One can send a digital asset to another person with their Atom ID the same way one can send money to another person with their bank account number.

Atom's DNA (Decentralization Number of an Atom): this is SHA-256 hash function generated from the Merkle Root of the previous atom and the atom root of the current atom.

Bonding: Bonding mechanisims are atomchain consensus protocols. They help to prevent the threat of degrading the usability of atomchain as a result of increasing wait times for sending transactions and using DApps. Bonding also improves denial-of-service (DoS) attacks resiliency, and replay attack protection on atomchain.

Creation-to-Finalization: this is the confirmation wait time of adding an extra atom to the atomchain.

Cryptocurrency: thisis simply a digital currency - that is not controlled by any government or institution, which are not printed, but are created using computing power, usually through a process called "mining". They are stored electronically as computer data. Encryption techniques are used in regulating and generating of units of cryptocurrency. Atom Wallets allow users to store and manage their atomcoin cryptocurrency; for instance, when they are sending and receiving atomchain transactions.

Cryptographic hash function: is a one-way mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit array of a fixed size (called the "hash value," "hash," or "message digest"). It is practically infeasible to invert or reverse the computation. The SHA-256 hash function used in generating private and public keys for atom wallets is an example of this function.

Elevated Mode: this is a fair-to-all scheme that determines the privilege status of a miner. In this scheme, an integer is decremented for a miner that successfully adds an atom, and incremented for other simultaneous miners with valid atoms that could not be added to the atomchain. Only the Genesis Atom has an elevated mode of zero.

Elevation Pool: before the adding of an atom to the atomchain, all valid atoms with their corresponding miners are placed in an "**elevation pool**". Atomchain first checks and accepts atoms from miners with higher elevation mode. Once an atom is finally added to the atomchain, the miner of that atom is removed from the elevation pool, and given a lower elevation status.

Energy: this is the fundamental atom network's cost unit.

External Actor: this is a person or other entity that is able to interface to an atom network's node. Such an actor can interact with atomchain through depositing signed transactions and inspecting the atomchain and associated state. An actor can have one or more intrinsic accounts – identified by account addresses.

Merkle Tree: is a hash tree in which every "leaf" (node) is labelled with the cryptographic hash of a data structure, and every node that is not a leaf (called a non-leaf, or an inner node, or an inode) is labelled with the cryptographic hash of the labels of its child nodes. Merkle Trees, sometimes called Merkle proofs, prevent malicious or unintentional modifications of the data by providing a unique hash that identifies the data set. Another data set is provably the same only if it computes to the same hash, otherwise the data set is different. Due to this feature, nodes in atom network can verify that they are getting the data they expect, or nodes synchronizing data sets may easily verify that they both ended up with the same results. [10]

Message: a set of bytes (or data sets) and specified atomcoin value, that is passed between two wallet addresses of atomchain accounts.

Multi-spending: is the reusing or spending of the same coins (or digital money) in more than one place for the purchase of digital services or goods from multiple vendors. This may also be referred to as double-spending.

Node: this is a machine on an atom network running a version or copy of the atomchain client or VMWare. Atomchain client is an implementation of the Atomchain Whitepapers.

Nucleus: this is similar to the "Block Header" of a block in blockchain.

Orbit Codes: this nascent technology is a set of executable codes that run on atomchain to facilitate, enforce, and execute transactions between untrustworthy parties

Peer-to-Peer (P2P) Network: is a distributed system architecture that partitions work between different nodes (or workstations) within the network. AtomchainP2P networks incorporate web3-based Layer 1 architectural support for the ecosystem. Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority [11].

Quarks: these are transactions on atomchain that has been mined (that is, transactions that are permanent and immutable in a stored state).

Transaction Pool (transpool): is a "waiting area" for atomchain transactions that each full node maintains for itself. After a transaction is verified by a node, it waits inside the transpool until it is picked up by a miner and inserted into an atom. Transpool is a full node's holding area for all the pending transactions in the node. Each node has a different capacity for storing unconfirmed transactions; hence, each node has its own version of the pending transactions or transpool. Transpool in atomchain is similar to **mempool** in blockchain.

10. References

[1] Satoshi Nakamoto, "A Peer-to-Peer Electronic Cash System," retrieved on 17/04/2022 from https://bitcoin.org/bitcoin.pdf

[2] Binance Academy, "What is a Blockchain Consensus Algorithm?" retrieved on 17/04/2022 from https://academy.binance.com/en/articles/what-is-a-blockchain-consensus-algorithm

[3] R. C. Merkle, "Protocols for Public Key Cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, 1980: 122-133.

[4] "Blockchain Merkle Tree," retrieved on 17/04/2022 from www.geeksforgeeks.org/blockchain-merkle-trees/

[5] "How to Build a Blockchain in Python," retrieved on 17/04/2022 from https://www.activestate.com/blog/how-to-build-a-blockchain-in-python/

[6] "Double factorial," retrieved on 17/04/2022 from www.wikipedia.org/wiki/Double_factorial

[7] Guillaume Ballet &Dankrad Feist, "Verkle Tree Structure," retrieved on 17/04/2022 from http://blog.ethereum.org/2021/12/02/verkle-tree-structure/

[8] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998

[9] “ShibaInu Ecosystem,” Woof Whitepaper (06/25/2021), retrieved on 17/04/2022 from www.shibatoken.com

[10] Kamil Jezek. 2020. Ethereum Data Structures. 1, 1 (August 2020), 27 pages. https://doi.org/10.1145/1122445.1122456

[11] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Content Distribution Technologies. ACM Computing Surveys, 36(4), December 2004.