Bitcoiin B2G – Technical Whitepaper

Abstract: This whitepaper describes Bitcoiin B2G generally, then presents granular descriptions of elements of the Bitcoiin algorithm, which is an implementation of the Ethereum algorithm. The Ethereum algorithm is described, at a general level. Some utilities are described, emphasizing user benefits. Transactions among Bitcoiin B2G holders, and between Bitcoiin holders and third parties are described, with general reference to Bitcoiin's trading platform. Blocks and the coding protocols are described. Finally, Bitcoiin's mining process is described.

General Description:

Bitcoiin B2G (hereinafter "Bitcoiin") is a cryptocurrency, based on the same Ehash algorithm used by Ethereum, with a proof of work implementation very similar to Equihash, as used by Bitcoin Gold. Bitcoiin B2G begins at a genesis state (s=0), and incrementally executes transactions to transition a ledger into a final state (s=n). The state represents the information about the accounts holding the coins. A valid state transition is one which comes about through a transaction.

Transactions are collated into blocks. Blocks are then chained together, using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block, and also an identifier for the final state. Each block does not store the final state itself since a complete journal like that would be far too large. Blocks also punctuate each transaction series with incentives for nodes to mine. This incentivisation takes place as a state-transition function, adding value to a nominated account.

Mining is the process of dedicating effort (working) to bolster one series of transactions (a block) over any other potential competitor block. It is achieved by a cryptographically-secure proof. This scheme is known as a Proof-of-Work (PoW). PoW is the backbone of not only Bitcoiin but

also most of the decentralized consensus-based transaction systems that have been built to date.

The mechanism behind proof-of-work was a breakthrough because it simultaneously solved two problems. First, it provided a simple and moderately effective consensus algorithm, allowing nodes in the network to agree collectively on a set of canonical updates to the state of the Bitcoin ledger.

Second, PoW provided a mechanism for allowing free entry into the consensus process, solving the problem of prioritizing who gets to influence the consensus, while simultaneously preventing "Sybil" attacks. It does this by substituting a formal barrier to participation, such as the requirement to be registered, as a unique entity on a particular list.

Built upon the foundation of the Ethereum consensus-driven blockchain, Bitcoiin's blockchain operates perpetually, censorship-free and collusion-resistant. Blockchain technology is hardened against tampering and revision, with the unanimity of records hosted by every node equally.

Transactions, contract code storage, and code execution fees are recorded on continuous blocks on the network ledger, cryptographically secured and time-stamped.

The peer-to-peer validation model of blockchain transactions has proven elsewhere to be and is here self-governing. The mining process incentivizes PoW effort, through the awarding of block rewards and transaction fees paid to mining participants.

Ethereum Algorithm

The Ethereum algorithm uses accounts and balances to record state transitions. This recording function does not rely upon **unspent transaction outputs** (UTXOs). State denotes the current balances of all accounts, plus extra data. The State is not stored on the blockchain, but on a separate Merkle Patricia tree.

A Merkle Patricia tree provides a cryptographically-authenticated data structure, that can be used to store all (key, value) bindings. In the context of a cryptocurrency, our use of the term is confined to strings. These Patricia trees are fully deterministic, meaning that a Patricia tree with the same bindings is guaranteed to be exactly the same down to the last byte, and therefore has the same root hash. This consistency provides O(log(n)) efficiency for inserts, lookups, and deletes. And that application allows easier coding.

The benefit of this implementation of Patricia trees comes with the addition of some complexity to the data structure. A nonce in a Merkle Patricia tree is one of the following:

NULL (represented as the empty string)

branch (a 17-item node [v0 … v15, vt]

leaf (a 2-item node [encodedPath, value]

extension (a 2-item node) [encodedPath, key]

The result of this implementation is optimized throughput. A thorough presentation of the use of trees throughout the Ethereum blockchain is beyond the scope of this paper.

Utilities

A **cryptocurrency wallet** stores public and private "keys" or "addresses," which may be used when an owner of Bitcoiin wishes to receive or spend the coin. These address keys could be generated through BIP 39 style mnemonics, for a BIP 32 "HD Wallet." But Bitcoiin, using the Ethereum algorithm, does not operate in a UTXO scheme. With the private key, it is possible to write in the blockchain, effectively making a transaction. To send Bitcoin to an account, the holder needs the public key of the destination account. Bitcoiin accounts are pseudonymous—meaning accounts are not linked to individual persons, but rather to one or more specific addresses. Owners can store these addresses in software, or on paper.

Addresses

Bitcoiin’s Ethereum algorithm addresses are composed of the prefix "0x", a common identifier for hexadecimal, concatenated with the rightmost 20 bytes of the Keccak-256 hash (big endian) of the ECDSA public key. In hexadecimal, 2 digits represent a byte, meaning addresses contain

40 hexadecimal digits. One example is 0xb794f5ea0ba39494ce839613fffba74279579268, the Poloniex ColdWallet. Contract addresses are in the same format, however, they are determined by sender and creation transaction nonce. User accounts are indistinguishable from contract accounts given only an address for each and no blockchain data. Any valid Keccak-256 hash put into the described format is valid, even if it does not correspond to an account with a private key or a contract.

Comparison to Bitcoin

Bitcoiin, like ether, is different from the original Bitcoin in several notable aspects:

- The block posting interval is 14 to 15 seconds, compared with 10 minutes for Bitcoin.
- Mining Bitcoiin generates new coins at a usually consistent rate, occasionally changing during hard forks.
- Transaction fees on Bitcoiin differ by computational complexity, bandwidth use, and storage needs, while Bitcoin transactions compete by means of transaction size, in bytes.
- Ethereum gas units each have a price that can be specified in a transaction. Bitcoin transactions usually have fees specified in satoshis per byte.
- Transaction fees are generally considerably lower for Bitcoiin than for Bitcoin. In December 2017, the median transaction fee for ether corresponded to $0.33, while for Bitcoin it corresponded to $23.
- Bitcoiin's implementation of Ethereum uses an account system where values in Bitcoiin debited from accounts and credited to another, as opposed to Bitcoin's UTXO system, which is more analogous to spending cash and receiving change in return. Both systems have their pros and cons, in terms of storage space, complexity, and security/anonymity.

Implications for Users

Bitcoiin's use of the Ethereum blockchain provides a block refresh rate of 14-15 seconds. This refresh rate reduces the blockchain's exposure to Sybil attacks and other mischiefs substantially, as compared to the original Bitcoin's posting rate of approximately one chain per 10-12 minutes.

Transactions:

Each transaction is a single cryptographically-signed instruction, constructed by a user externally (in Bitcoiin wallet) and then sent to the blockchain for inclusion in blocks during the mining process (assuming validity). Each transaction consists of the following fields:

1. Nonce – a scalar value equal to the number of transactions sent by a particular address. It keeps count of the transactions and also acts as a measure to restrict double spending.
2. Fee – transaction fee (in units of Bitcoiin) the account user is willing to give for a transaction as an incentive to the miner.
3. Value – the amount of Bitcoiin the user is willing to send to the receiver.
4. To – the receiver's Bitcoiin address.
5. Signature – the cryptographic signature of the initiator of the transaction, i.e., the account from which the Bitcoiin will be deducted.

Transactions can be of several kinds, including:

- Buying Bitcoiin from a peer user, in exchange for *fiat* currencies or other cryptocoins
- In the ICO period, buying pre-mined Bitcoiin from the organizers
- Selling Bitcoiin on the Bitcoiin exchange
- Selling Bitcoiin on a third-party exchange
- Receiving payments in Bitcoiin from mining activities, or from other peer users
- Trading Bitcoiin for other cryptocurrencies, including Ether and other leading tokens
- It is conceivable futures markets, exotic trades and derivative instruments might surface

Block:

The block consists of the relevant pieces of information that fulfills the consensus protocol. It constitutes of following fields :

1. ParentHash: The Keccak 256-bit hash of the parent block's header, in its entirety
2. OmmersHash: The Keccak 256-bit hash of the ommers list portion of this block
3. Beneficiary: The 160-bit address to which all fees collected from the successful mining of this block be transferred.
4. StateRoot: The Keccak 256-bit hash of the root node of the state tree, after all transactions are executed and finalizations applied.
5. 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 block.
6. 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 block.
7. LogsBloom: The Bloom filter composed from indexable information (logger address and log topics) contained in each log entry from the receipt of each transaction in the transactions list.
8. Difficulty: A scalar value corresponding to the difficulty level of this block. This can be calculated from the previous block's difficulty level and the timestamp.
9. Number: A scalar value equal to the number of ancestor blocks. The genesis block has a number of zero.
10. Fee-limit: This is block fee limit which defines the number of transactions that can be included in the block. The collective fees of all the transactions included in the block must be less than the block fee limit.
11. Timestamp: A scalar value equal to the reasonable output of Unix's time() at this block's inception.
12. ExtraData: An arbitrary byte array containing data relevant to this block. This must be 32 bytes or fewer.
13. MixHash: A 256-bit hash which proves combined with the nonce that a sufficient amount of computation has been carried out on this block.
14. Nonce: A 64-bit hash which proves combined with the mix-hash that a sufficient amount of computation has been carried out on this block.

All the above fields are included in the Block Header and the Block consists of two more fields:

1. Transaction Receipt - In order to encode information about a transaction concerning which it may be useful to form a zero-knowledge proof, or index and search, we encode a receipt of each transaction containing certain information from concerning its execution. Each receipt is placed in an index-keyed trie and the root recorded in the header.
2. Holistic Validity. We can assert a block's validity if and only if it satisfies several conditions. It must be internally consistent with the ommer and transaction block hashes and the given transactions, when executed in order on the base state (derived from the final state of the parent block), result in a new state of the identity.

TARGET BLOCK TIME - 15 seconds

In the context of financial applications, people tend to expect a much more rapid response time for the confirmation of transactions. Therefore, for the purpose of a stable AND efficient transactional platform, having a blockchain that is faster than the 10 minute blocktime for Bitcoin is essential. Arguments have been postulated that a faster block time creates more orphaned blocks (uncles) due to the delay of block propagation over the entire network. Our goal is less than 0.5% uncles with a 15-second block time – which is statistically insignificant. Since difficulty adjustment in Bitcoin (BTC) takes place every two weeks, we are adjusting to the Bitcoin Gold adjustment time – i.e., adjustments to every block, which in our case means every 15 seconds.

Genesis Block

Bitcoiin's genesis block will have 50 million pre- mined coins for the purpose of building out infrastructure, funding the ICO, and providing operating capital for continued operations. The genesis block code might have the following code structure:

```
"config": {
  "chainId":
  6755,
  "ethash": {}
```

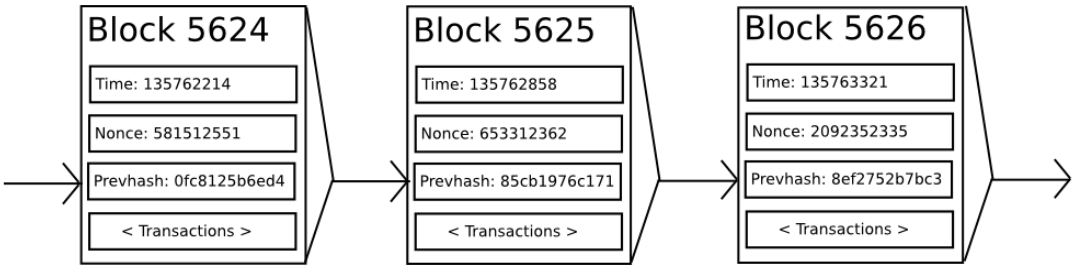
```
  },
  "nonce": "0x0000000000000023",

  "timestamp": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData":
  "0xce02dec31ca49f3c8f149b3b931a0155121d2ca0",
  "gasLimit": "0x1388",
  "difficulty": "0x40000000",
  "mixhash":
  "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "alloc": {
    "0xafc571db22ca80776def02150955026ff7329467": {
"balance": "500000000000000000000000" },

}
```

The similarity between this genesis block code, and Ethereum is notable. Bitcoiin was built to use the Ethereum blockchain algorithm. Aside from security, the primary benefit of this adoption is the speed of blockchain posting. Also, this algorithm improves scalability, since blocks are not limited in size, as with the original Bitcoin. Benefits include throughput, transaction speed, and security.

Mining



An illustration of the mining process

(Source: https://github.com/ethereum/wiki/wiki/White-Paper#Bitcoin-as-a-state-transition-system)

We could simply code these transactions as pictured, using a centralized server's hard drive to keep track of the states(s) and transitions. But the first principle of cryptocurrencies is decentralization. Accordingly, we need to couple the state transaction system pictured above with a transparent consensus posting system – so that everybody can agree on the order of transactions. The Ethereum algorithm, implemented in a decentralized consensus process, means network mining nodes must attempt continuously to produce packages of transactions called "blocks." (See our description of blocks in the Ethereum algorithm above.) If implemented securely, and maintained continuously, the network of mining nodes is intended to produce roughly one block 15 seconds. This is a substantial improvement over the original Bitcoin's latency time of approximately every ten minutes. The continuous posting means that, over time, this creates a persistent, ever-growing blockchain that constantly updates to represent the latest state of the Bitcoiin ledger.

The algorithm for checking if a block is valid, expressed in this paradigm, is as follows:

- Check if the previous block referenced by the block exists and is valid.
- Check that the timestamp of the block is greater than that of the previous block[2] and less than 2 hours into the future
- Check that the proof of work on the block is valid.
- Let `S[0]` be the state at the end of the previous block.
- Suppose `TX` is the block's transaction list with `n` transactions. For all `i` in `0...n-1`, set `S[i+1] = APPLY(S[i],TX[i])` If any application returns an error, exit and return false.
- Return true, and register `S[n]` as the state at the end of this block.

Each transaction in the block must provide a "valid" result, i.e., a valid state transition, from what the canonical state before the transaction was executed. Note that the state is not encoded in the block in any way. The state is purely an abstraction, documented by the validating node. The state can only be computed safely and securely for any block, by starting from the genesis state and sequentially applying every transaction in every block. Order of transactions matters. If there are two transactions A and B in a block such that B spends a UTXO created by A, then the block will be valid if A comes before B but invalid of B comes for A.

Another consideration for mining is the Proof-of-Work paradigm. The hash of every block, treated as a 256-bit number, must be less than a dynamically-adjusted target. This target will grow over time, as the Bitcoiin blockchain grows. Certainly it will be become very large, depending upon the volume of transactions generated. This is not a bad thing; in fact, this growth is the essence of security in the blockchain. The longer the blockchain, the harder block creation becomes, speaking computationally. This difficulty prevents Sybil attackers from remaking the entire blockchain in their favor. Because our hash function is designed to be a completely-unpredictable pseudo-random function, the only way to create a valid block is simply trial and error, repeatedly incrementing the nonce and seeing if the new hash matches.

In order to compensate miners for the brute-force computational work of working these trial-and-error problems over and over in real time, the miner of every block is entitled to include a transaction fee. The original Bitcoin pays 12.5 BTC out of nowhere, at present. This compensation halves at specified intervals, until the entire Bitcoin issue has been computed. At that point, there will be no more mining. Also, if any transaction has a higher total denomination in its inputs than in its outputs, the difference also goes to the miner as a "transaction fee". Incidentally, this is also the only mechanism by which Bitcoiin, Ether, or the original Bitcoin are issued. For each of these cryptocurrencies, the genesis state contained no coins at all.

To illustrate this mining process with an example, consider the possibility of a malicious attacker targeting Bitcoiin. Since the Ethereum cryptography is known to be secure, the attacker will target the only part of the process unprotected cryptography directly: the order of transactions. The attacker's strategy would likely be:

1. Send 10 Bitcoiin to a merchant in exchange for some product (preferably a rapid-delivery digital good);
2. Wait for the delivery of the product;
3. Produce another transaction sending the same 10 Bitcoiin to himself; and then
4. Try to convince the mining network that his transaction to himself was the one that came first.

After step one, in less than a minute some miner will include the transaction in a block. After about one hour, over 300 blocks will have been added to the chain after that block, with each of those blocks indirectly pointing to the transaction and thus "confirming" it. The merchant will accept the payment as finalized and deliver the product. Since we have posited an instant-delivery digital good, the delivery is made and the horse is out of the barn. Now, the attacker creates another transaction sending the 10 Bitcoiin to himself. If the attacker were to release the transaction code “into the wild,” it would be filtered out as invalid, and the blockchain would remain accurate. Miners will run `APPLY(S,TX)` and notice that `TX` consumes a UTXO which is no longer in the state. So instead, the attacker creates a "fork" of the blockchain, starting by mining another version of block 270000 pointing to the same block 269999 as a parent but with the new transaction in place of the old one. Because the block data is different, this requires redoing the proof of work. Furthermore, the attacker's new version of block 270000 has a different hash, so the original blocks 270001 to 270005 do not "point" to it. Thus, the original chain and the attacker's new chain are completely separate. The rule is that in a fork the longest blockchain is taken to be the truth, and so legitimate miners will work on the 27305 chain while the attacker alone is working on the 270000 chain. In order for the attacker to make his blockchain the longest, he would need to have more computational power than the rest of the network combined in order to catch up (hence, "51% attack").

This feature of the blockchain prevents the problem of double-posting. And it is why mining is as crucial to the safety, security and grown of Bitcoiin as any element in the ecosystem.

Conclusion

This whitepaper describes Bitcoiin B2G generally and provides some concrete examples to illustrate how the public blockchain works to keep the ledger safe – not because nobody is looking, but because everybody is looking.

We have also presented a granular description the Bitcoiin algorithm, which is an implementation of the Ethereum algorithm. The Ethereum algorithm has been described, at a general level.

Some utilities were described, particularly wallets. The single most important build feature to Bitcoiin, other than the blockchain algorithm, is the way Bitcoiin keeps wallets safe and secure. The double-key system (public and private) ensures both the safety of wallets, and the fungibility of Bitcoiin, where fungibility can be defined as the ability to exchange Bitcoiin B2G for *fiat* currencies (USD, UKP, EUR and others), and also for other cryptocoins, including Bitcoin and Ethereum.

Transactions among Bitcoiin B2G holders, and between Bitcoiin holders and third parties were described, with general reference to Bitcoiin's trading platform. Blocks and the coding protocols were also described.

Finally, Bitcoiin's mining process is described.

End

References

1. White Paper · ethereum/wiki Wiki · GitHub. https://github.com/ethereum/wiki/wiki/White-Paper

2. The Safest cryptocurrency In the World | By Coin Labs. https://www.swisscoinlab.ch/wp-content/uploads/2017/12/Swisscoin-White-Paper.pdf

3. White Paper - Decred. https://decred.org/research/buterin2014.pdf

4. Ethereum - Wikipedia. https://en.wikipedia.org/wiki/Ether_(currency)

5. Patricia Tree · ethereum/wiki Wiki · GitHub. https://github.com/ethereum/wiki/wiki/Patricia-Tree

6. Transaction Tests — Ethereum Homestead 0.1 documentation. http://ethdocs.org/en/latest/contracts-and-transactions/ethereum-tests/transaction_tests/

7. Blockchain Tests · ethereum/tests Wiki · GitHub. https://github.com/ethereum/tests/wiki/Blockchain-Tests

8. Gender neutral terming for siblings of parents. · ethereum https://github.com/ethereum/yellowpaper/commit/8dd35d4e194a49564756ad3c552bb088f4a75031

9. What is Blockchain Technology - AS Blog. http://blog.astemplates.com/what-is-blockchain-technology/