Whitepaper
Bismuth Crypto-Currency

Written by the Bismuth Core Dev Team
Version 1.2
Date: August 13, 2019

Contents

Abstract

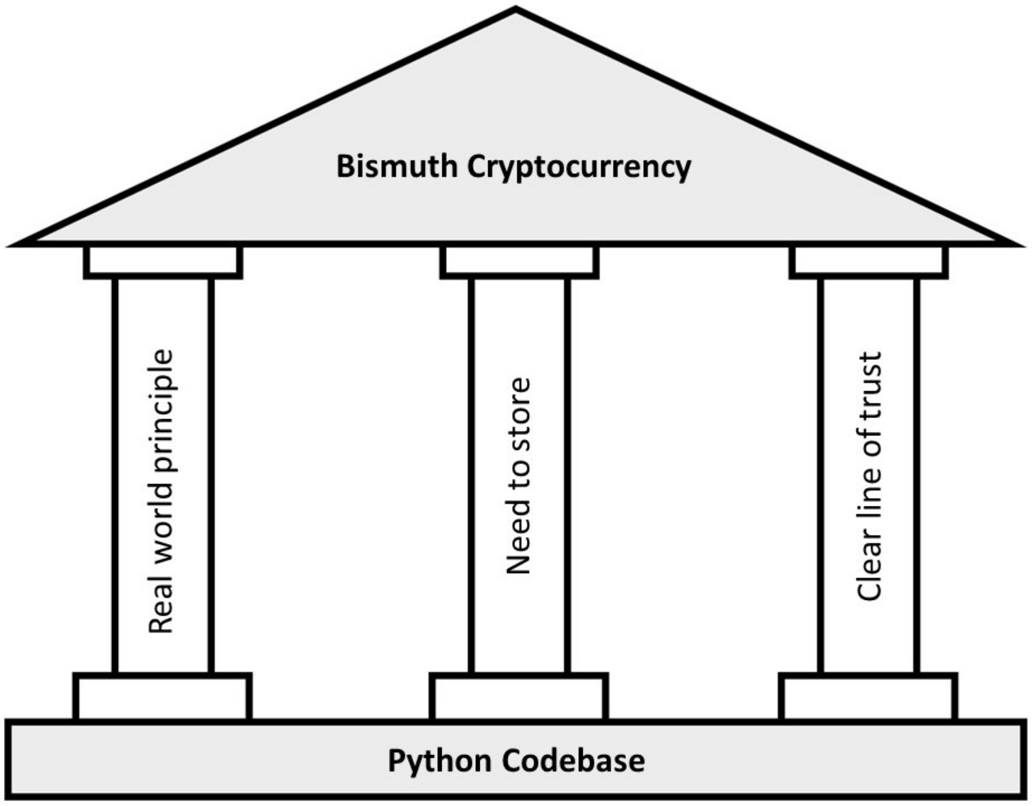
This document contains the whitepaper of the Bismuth crypto-currency. The whitepaper begins with a presentation of the Bismuth philosophy and a description of the three pillars it stands on followed by a presentation of some of the core features, which are: Plugins, Use Cases, Coin Supply and Reward Model, Cryptography, Mining Algorithm, Tail Removal Block Validation, Operation and Data Fields, Private Contracts, Hypernodes and Sidechain, Testnet and Regnet, Education and Research.

Introduction

The crypto-currency Bismuth's initial goal was to build a chain that was as simple as possible, from a brand new fresh python codebase. It started as a personal project by one developer, to learn about the technology, but soon evolved into a feature packed crypto-currency and platform with multiple developers, technical support persons, pool operators, exchanges and social media influencers involved.

Bismuth Pillars

Bismuth stands on three pillars that differentiate it from other crypto-currencies.



1. The Real World Principle

Some chains live in an idealist world where everything can be made perfect, where code fixes everything and users are experts able to understand complex algorithms, their bug free implementation, compile from source, then make no error using the tool. The solutions designed in that spirit are bound to fail because they ignore the real world, real users, and use case of the tool. The interfaces with the real world (users, oracles, network) are not perfect, and Bismuth has to account for that.

There is no need to design a super complex algorithm which ensures a consensus to reach immutability when what it secures is not fail free. The complexity of the code should reflect the real world use case. As an example, you do not need 12 supersonic reactors to run your car. Your usual engine is enough and gives enough guarantee for everyday use. As an example, Bismuth does not use transaction hashes. The block hash verifies all the transactions contained in it using the following code:

```
block_hash = hashlib.sha224((str(transaction_list_converted) +
    db_block_hash_prev).encode("utf-8")).hexdigest()
```

That is in the Bismuth spirit, simple, easy to understand, yet effective and considered good enough.

If the goal is to achieve the mass adoption of cryptocurrencies, then we need to move away from them being seen as something only for developers and experts in cryptography. The way we do this is by making them simple to understand and not over engineering them. There is still work to do in that respect, but this is the way Bismuth moves on: not trying to design a perfect abstract system, then find matching use cases, but starting from real world use cases instead and searching for the simplest possible way to achieve it - with sufficient guarantees.

2. Need to store

Because of the real world principle, some data needs to be stored in the chain. Users do not need to use tricks to do that like they do with BTC or ETH. Some things just HAVE to be stored on chain, let's not try to hide that, and ease the user's job. Bismuth supports by default two abstract fields - operation and data - users can leverage to build any protocol, however complex, on top, while keeping a good level of performance.

However, storing everything on chain because, we have a chain, is kind of the hammer problem: if all you have is a hammer, you see every problem as a nail. That is why some crypto-currencies are so focused on scalability issues: they want to store everything on chain, and it is just too much. Bismuth's philosophy is to store only what needs to be stored on the main chain, and no more. Still, following the real world principle, on chain storage is not strongly discouraged.

Scalability is an application design issue. By ensuring you store only what needs to be, you avoid - from the start - many scalability and long term issues. Why store full documents for proof when you can store only their hash? Store proofs instead of data. Use checkpoints, signatures, fingerprints.

Bismuth also aims to provide a practical framework to store more aside the chain, while still benefiting from the chain safety (See Hypernodes-like side chains, hyperlane).

3. Clear line of trust

Crypto-currencies often define themselves as ”trustless”. This is ignoring the real world principle.

What is stored in a chain is not necessarily ”true”. It is just supposed to be immutable once stored, and requires some hidden trust anyway: in the code, in the bootstrapped data, in the underlying algorithm, in majority of other peers, in service providers. On-chain contracts are not magical, they are not perfect. They can have bugs, the virtual machine (VM) can change, they rely on oracles, you still have to trust but you do not always know what or who you trust.

Bismuth does not try to hide the fact that you sometimes have to trust, and we try to make that clear. When you use a private bis contract like Zircodice or Dragginator for instance, you trust its operator. You still can have a history of the transactions and verify that the operator did what he was supposed to.

The Bismuth execution model and abstract protocols also go in that direction. Not only you know what you have to trust, but you can choose who you trust. In the case of a protocol, you can freely choose the implementation you trust the more, the certifier that suits you, and so on. It is not some magical immutable contract stored on chain, with possible bugs, proxies, backdoors and no way of fixing when it goes wrong.

In other words ”clear line of trust” means that when you have to trust something or someone, you know what/who you have to trust. Bismuth is not presented as trustless, whereas you implicitly trust several layers and people.

Bismuth Value proposal

The Bismuth value proposition can be summarized as follows:

- Lightweight: The node is lightweight and does not require a powerful CPU and a lot of RAM.
- Perfect fit for developers, scholars and academics.
- The codebase is quick to handle and develop upon.
- Allows for fast prototyping of use cases.

- Easy to tweak and experiment on
- Multiple access layers and clients API in several languages are available.

Bismuth for developers
Along its development, Bismuth always tries to:

- be simple: does not copy every complex feature of other chains, but strip down and simplify to the core so it is understandable.
- be innovative: the Bismuth protocol belongs to a core team, which allows for great flexibility and new features to be tested and added quickly.
- be extremely extensible and tweakable.

Bismuth is tailored for developers:

- Use of python means it is largely accessible to an ever growing base of developers.
- Python is currently the language of choice for universities, students, academics, data and machine learning scientists.
- Python does not need time consuming compilation steps. Tweak and test.
- Bismuth allows for interaction at many levels of its infrastructure, from direct DB access - pure SQL - to API clients in several languages.
- Bismuth node comes with hooks and filters, allowing for easy to write plugins, in pure python code - no new language to learn.
- Bismuth Wallet also comes with pluggable crystals, so dApps can be seemlessly integrated in the wallet.
- Bismuth Abstract transaction model and protocols allow for virtually any application to be run on top of Bismuth.

An example of feedback from a dev to the core team: *With Python's simplicity, with a few hours you can get some poc apps done. The main hassle trying to bring talent to blockchain is all the research that is needed. Most of the time when we were going to do afternoon workshops with eth-solidity it just got too complex.*

Hack with Bis repository is the place to start to learn the basis and get started. Bismuth core concepts for developers, see: This GitHub Link

Bismuth Execution Model
The current Bismuth model is very different from the Ethereum one. You simply can not transpose what is done with smart contracts and solidity. Bismuth does not need public ”smart” contracts at the moment, and does not have a VM where every node executes the same code.

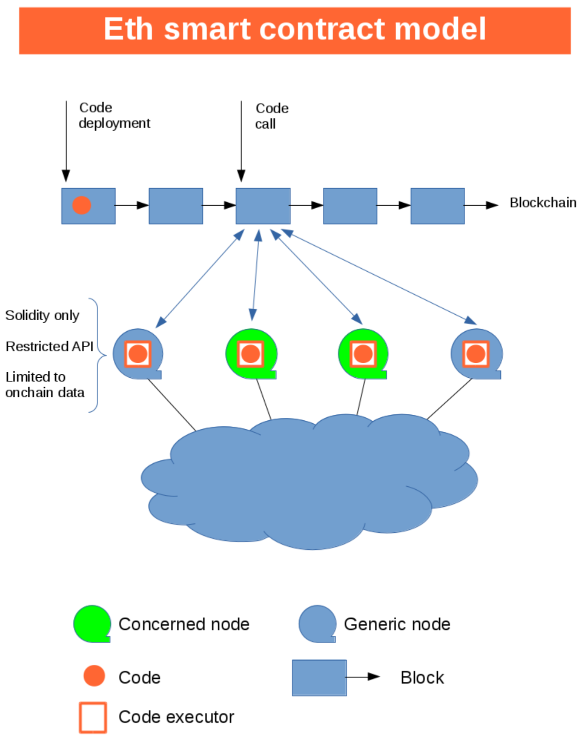
Although it could be seen as a limitation, it is in fact quite a strength, and some exploits that have taken place with ETH smart contracts could not have been successful on a Bismuth like architecture.

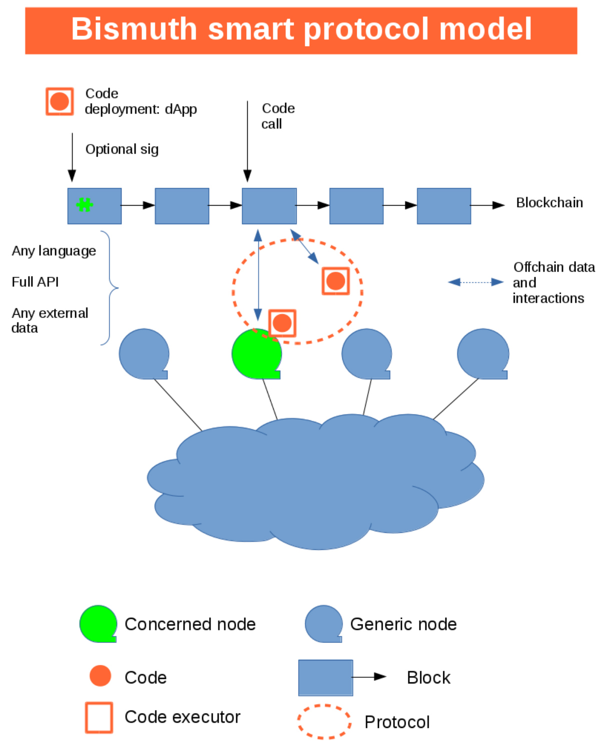
”Smart” contracts vs ”smart” protocols in a nutshell
ETH like smart contracts are written in a specific language, stored on chain, and run **IN** every node. Bismuth like smart protocols are implemented **ON TOP** of the Bismuth chain and only run by concerned dApps.

Ethereum

- You have to learn a new language, Solidity.
- There are some specific pitfalls (underflow, visibility, access rights).
- Flawed contract code can give an infinity of coins to a user.
- Several hacks and horror stories already in Ethereum smart contracts History.
- Smart contracts can ”own” funds
- Smart contracts live in the chain forever and can not be stopped nor upgraded unless the author provided a kill switch.
- If there is a kill switch, the owner can get all the funds of a contract.
- Every contract invocation is processed by every single eth node in a VM - Virtual Machine - and consumes gas.
- Contracts can not directly access outside resources.

ETH model has some strengths and some use cases that you could simply not replicate with BIS, but BIS has other uses.





Bismuth

- No new language to learn. You can write contracts and protocols in almost any language, Python being the native language.
- No more pitfalls than with your usual code.
- Contracts can not overspend.
- No VM, no ”on chain” code, no public contracts.
- Users can run private contracts.
- Owners then have full control - including fix and upgrade - over the contract.
- The contract, if its inner working is published, is fully auditable and verifiable.
- Contract invocation is only run by the clients that have an interest in that specific contract.
- Private contracts can do anything, including accessing outside data without the need for on chain oracles.

Bismuth tokens

Things that are to be widely used, like tokens, are not handled via a generic VM and user code. If a use case is wanted enough, it can be integrated in Bismuth core. The Bismuth dev team does not have the weight of Bitcoin or ETH, and can move forward very very fast.

That's the case with tokens.

- native tokens.
- optimized, resource savy tokens, indexed db of tokens.
- still can be overloaded with extra features.
- code is tested, public, the same for everyone: potential bugs are identified and can be fixed globally.

To match ETH terminology, current Bismuth tokens are partially ERC20 compliant. They do not allow delegation: you can not have someone else spend your tokens and approve. More feature packed tokens types may be added soon.

The Bismuth tokens make use of the operation and data fields for creation and transfer. The two main operations are token:issue and token:transfer. For more information about Bismuth tokens, see This Link

Bismuth "Smart" protocols

Rather than having immutable on chain code, that has all power on the funds and can have them destroyed or locked up, plus is run by every single node, Bismuth favors the concept of "smart" protocols. Quotes are used, because no contract or protocol in blockchain world is really "smart". It is just code that is as smart - or dumb - as the developer who wrote it.

A protocol is based upon the Bismuth transactions, that can be considered as abstract data. It is an agreement between two or more parties on what that data means, and what to do when an event occurs.

- Only clients that are involved in a protocol need to read the data and run the code. Not every node.
- Code is not on chain. Can be updated, fixed, does not clobber the chain, does not consume node resources.
- They are a "contract" between agreeing parties, with the logic ideally being public.
- Anyone can run the logic over the on chain data and verify that everyone acted as they should.

- Protocols can evolve, be overloaded, or serve as a basis for more evolved protocols.
- Protocols can use protocols... for instance, a protocol could define valid implementations (with on chain hash) of itself.

For a list of existing Bismuth protocols, see: This GitHub Link

Bismuth Features

This section contains a presentation and description of some of the core features of the Bismuth crypto-currency.

Python and Plugins

Developers who plan to build dApps on top of the Bismuth node, are encouraged to make use of the Bismuth feature called "plugin". Plugins reside in the directory ˜/Bismuth/plugins. The Bismuth plugin system, although very lightweight, allows for action and filter hooks on critical events, for easy feature addition. For example, a plugin wanting to implement a "block" action hook only has to declare a simple function:

```
plugins/900_test/__init__.py:

def action_block(block):
    print(block)
```

For more information about Bismuth plugins, see This Link

To activate a new plugin, the Bismuth node (node.py) must be restarted.

Use Cases

1. Lab Use Case
It is very easy for a student to set up his/her own private network by using virtual machines and then changing the default mainnet port number (5658) to something else. By using local network addresses (for example 10.0.x.x or 192.168.x.x) and whitelisting these in Bismuth configuration file (config.txt), it is possible to isolate the lab network from the outside world. With such a network, a student, researcher or a developer could test out new features or dApps without the need for a hardfork in the regular main or testnet.

2. Child Chain Use Cases
The child chain use cases can deal with scalability and flexibility. A chain with specific properties (block time, entry barrier - or not -) is needed and is not limited by the existing mainnet, proof-of-work (pow) chain. A Hypernode-like

pos chain can be run on top of the pow chain. It uses the same technology, and a developer could have a unique chain for his/her app only, with unique chain settings, and define unique transactions types (be it currency or non currency data or both).

3. Event Sourcing
Event sourcing is an object/data model that stores the events leading to the current state of objects, rather than the current state itself. A proof-of-concept (poc) for event sourcing with sample dApps using Bismuth is available at This Link. Event sourcing would work well in combination with a private child chain. For example, a network of customers/providers/partners, agreeing on operating a shared database. It could be tracking of goods, shipments, invoicing, etc. A child chain could be operated, with every actor running a node (Hypernode-like, with private registration) then the event sourcing poc could be used. This means that the actors would share a distributed and replicated database, where every change to the data is an event, with immutable timestamp and source of an event, plus rights and so on, fully auditable.

4. File Fingerprinting
The legacy wallet contains a feature for fingerprinting one or more files by using the following code:

```
def fingerprint():
    root.filename = filedialog.askopenfilename(multiple=True,
        initialdir="", title="Select files for fingerprinting")
    dict = {}
    for file in root.filename:
        with open(file, 'rb') as fp:
            data = hashlib.blake2b(fp.read()).hexdigest()
            dict[os.path.split(file)[-1]] = data
    openfield.insert(INSERT, dict)
```

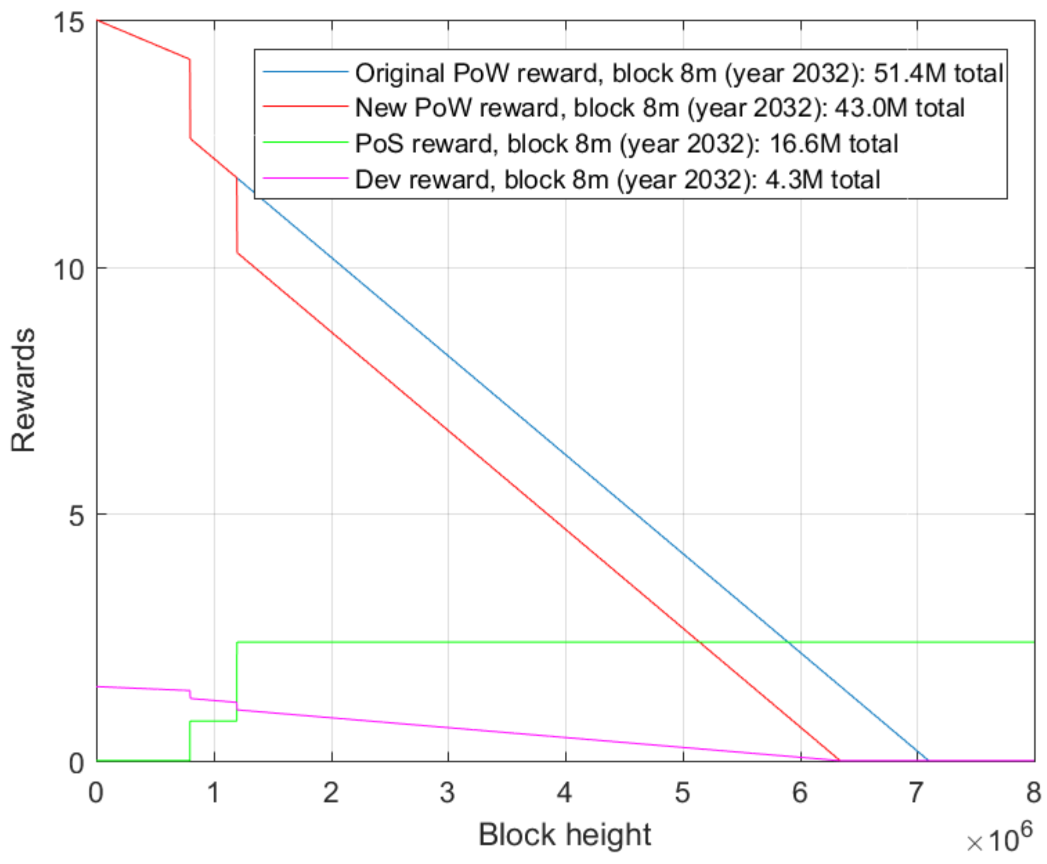
The hash of the file(s) is then inserted into the ”Data” field and can be sent to yourself or someone else. The recipient could then use the received message to validate the authenticity of these files.

Below is a list of some implemented use cases and games at the time of writing:

- anon.py, a private contract anonymizer service.
- Dragginator, a collectible game based on the Bismuth blockchain.
- PokaPoka, a poker game site using Bismuth tokens.
- zircodice, a dice game as private contract.
- autogame, a probabilistic multiplayer game implemented as a private contract.

Coin Supply and Reward Model

The plot below shows the coin supply and rewards for the Bismuth blockchain. At block height 8,000,000 (in year 2032 assuming 1440 blocks per day) the total coin supply will be 63.9 million BIS. From this amount 43.0 million will be miners rewards, 16.6 million BIS will be rewards to Hypernodes and 4.3 million BIS will be developer rewards (10%) of the miner rewards). Any change to this distribution would require a future hard fork which, at the time of writing, is not planned.



The Bismuth blockchain started May 1, 2017 and the inflation rates during the following 10 years are shown in the table below:

Year	Inflation
1	∞
2	93.2%
3	38.9%
4	25.5%
5	18.4%
6	13.9%
7	10.8%
8	8.4%
9	6.6%
10	5.0%

As seen from this table, the inflation is quite high initially while it rapidly drops to 5.0% in year 10. One reason for the relatively high initial inflation is the fact that Bismuth had no pre-mine or ICO: The total coin supply started at zero at the Genesis block (block height 0). Naturally, the inflation will initially be large with such a distribution model. Some motivations behind this coin supply and rewards model is to ensure a fair distribution model of Bismuth in the early phase, while at the same time attract miners to secure the chain with their hash rates. As the project matures, holding Bismuth for the long-term is encouraged by the rapidly falling inflation rates while the miners will continue to get rewarded by collecting transaction fees. For a comparison of Bismuth's coin supply and reward model with some other blockchain projects, see https://hypernodes.bismuth.live/?p=218

Cryptography

Most of today's crypto currencies use Elliptic Curve cryptography - ECC - along with ECDSA signature algorithm. While ECC keys and signature are short and potentially secure with less bits than previous cryptographic keys and signature algorithms, they are a relatively new family and it is always possible that a completely new class of flaw is found, effectively rendering all ECC based chains insecure.

Bismuth - paradoxically - innovates by using an older but very well known asymmetric cryptographic algorithm, RSA, which is studied since its publication in 1977, and relies on core properties of large prime numbers. It is widely used in secure ssh and ssl certificates all over the web since decades.

Key Lengths:

- a RSA key length of 1024 bits is sufficient for many medium-security purposes such as web site logins.
- For high-security applications or for data that needs to remain confidential for more than a few years, a 2048-bit key is recommended and should be safe until 2030.

- To keep data confidential for more than the next two decades, RSA recommends a key size larger than 2048 bits.
- 3072 bits are recommended for usage post 2031.

Bismuth relies on 4096 bits RSA keys, so not to take any risk.

Private Key
The private key is only known by the wallet owner. It is currently stored under PEM - base64 - format.

Example:
```
-----BEGIN RSA PRIVATE KEY-----
MIIBgjAcBgoqhkiG9w0BDAEDMA4ECKZesfWLQOiDAgID6ASCAWBu7izm8N4V
2puRO/Mdt+Y8ceywxiC0cE57nrbmvaTSvBwTg9b/xyd8YC6QK7lrhC9Njgp/
...
-----END RSA PRIVATE KEY-----
```

Public key
The public key is also stored and transmitted in PEM format

Example:
```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAnzgE34oTDlzlPFMsVkNo
foMg9Pm4rG6U8V1fZ/Ewzbtu8UjyvpERblDSaSGBy3C8uZuPpZm/VYTq5KHYJJ6y
...
kLYgWGdQc+MRSkwCwWGQtXECAwEAAQ==
-----END PUBLIC KEY-----
```

Address
The address matching a key is the sha224 hash of the public key PEM, under hex format.

Example:
3e08b5538a4509d9daa99e01ca5912cda3e98a7f79ca01248c2bde16

Signatures
Bismuth uses PKCS1 v1.5 Signature algorithm. Both public key and signature are sent with every transaction, and validated upon reception.

Multiple Address Schemes

In order to expand Bismuth's capabilities and footprint, the Bismuth Foundation had planned to support several alternative cryptographic primitives. This was on the roadmap since a while and a reality since July 2019. Bismuth nodes do now also support a new ECDSA cryptographic primitive as well as a new addresses scheme, while retaining RSA for coinbase operations and keeping full compatibility between the two schemes.

ECDSA is used by most existing crypto-platforms, and has allowed for more efficient and swift operations, such as rapid signing times and smaller signatures. While integrating with Bismuth, the core developers took care to follow as much as possible the current BIP standard as to guarantee an optimum compatibility, which enables BIS to integrate seamlessly with the existing architecture built around the Bitcoin ecosystem. It will introduce a new address format beginning with the “Bis1” prefix, allowing for consistent and nicer-looking Bismuth addresses. With ECDSA, Bismuth becomes paper wallet compatible, as well as seed-word compatible. Integration with existing hardware wallet solutions such as “Trezor” and “Ledger” become simplified, and last but not least, efficient mobile wallet applications for Bismuth become a reality.

It’s up to the user
The Bismuth Foundation wants to give users the choice of which scheme to use, whether it be the older RSA or newer ECDSA. Both algorithms have their respective strengths and weaknesses, and it should be up to the preference of the users and application developers which one they prefer to use. Both will coexist within the Bismuth protocol, and be compatible but this additional facet of modularity should appeal to everyone. Bismuth is the only platform to offer this level of choice, and offers a high level of security- if at one point in the future a backdoor was found for ECDSA, Bismuth would be the most unaffected as it uses RSA for all coinbase operations, and users could immediately fallback to it for all activity, without complicated chain swap process. The same cannot be said for most other existing crypto-platforms, in which ECDSA is used for all operations.

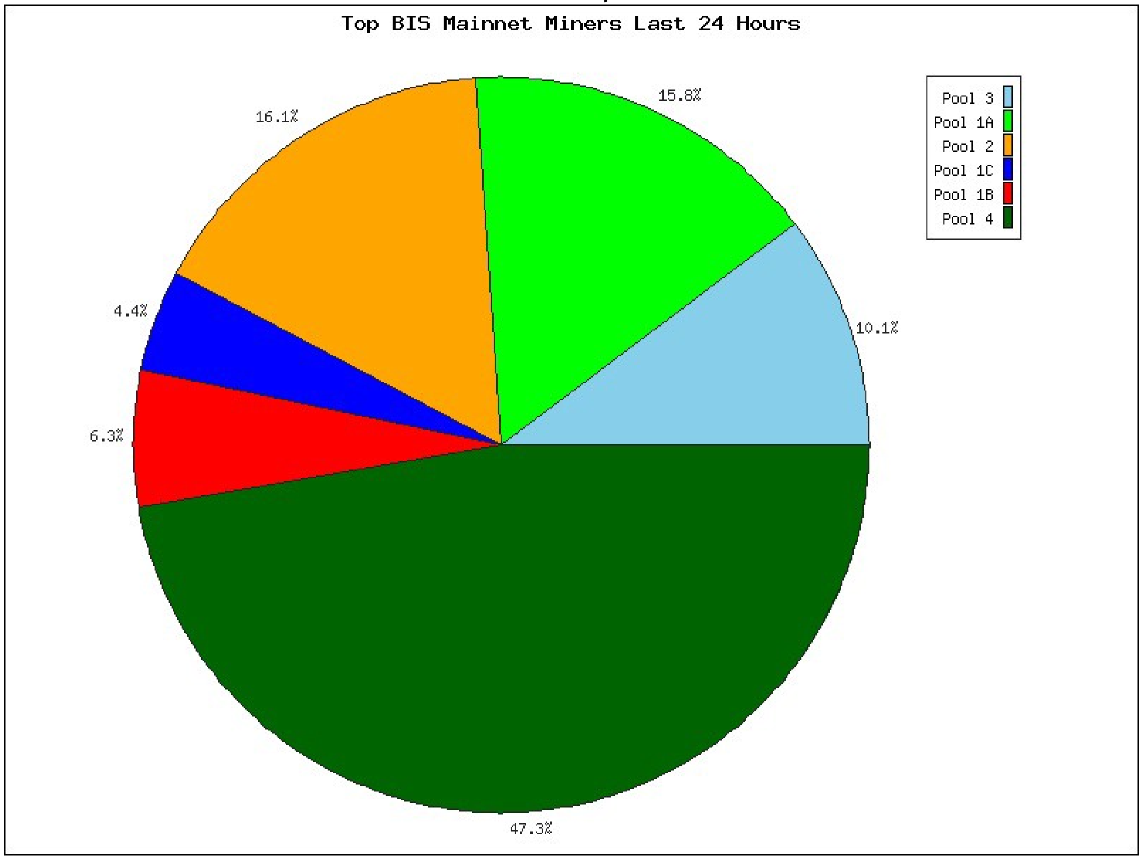
Future proof
The introduction of multi-scheme addresses will help advancing Bismuth in many aspects, whether it be expanding its footprint and presence in hardware devices, or boosting its capabilities in regards to address generation and formats. It is a step forward in both scheme modernization and in establishing a new standard of multi-scheme addresses. At some point, options can even be extended beyond RSA and ECDSA. The code handling the signatures and addresses is modular and extendable. Although undocumented, the ed25519 cryptographic primitive is also supported by current nodes for instance. More schemes could be added later on, to address the possible threats quantum computing could pose to any of the existing ones.

Mining Algorithm

The mainnet of the Bismuth crypto-currency project was launched on May 1st, 2017. The mining algorithm was based on sha224 and is briefly described here http://dx.doi.org/10.4173/mic.2017.4.1. In the beginning there were only CPU miners, but after less than 6 months the first GPU miners appeared and shortly afterwards the first GPU mining pools. Bismuth was listed on the Cryptopia exchange during October 2017, and that led to a large increase in new accounts on the network. By January 2018 the number of Bismuth accounts had quadrupled compared to before the exchange listing.

Since Bismuth had a relatively simple mining algorithm requiring very little memory on the GPUs, the network was vulnerable to a 51% attack by a large FPGA or ASIC mining operation. The core development team was well aware of this threat, but decided to work on other issues instead, such as general network stability improvements, in addition to new functions and features. The introduction of Hypernodes and the side-chain was one example.

During August and September 2018 it became increasingly evident that an FPGA miner had been developed, and that this mining operation was approaching a 51% portion of the overall mining power in the network. The figure below shows the hashrate distribution of the different pools at that time:



The presence of a large FPGA miner was identified by several independent channels: Bismuth network monitoring pages, pools, miners themselves reporting anomalies, regular exchange dumps, internal core team research on FPGA capabilities as well as work with FPGA developers.

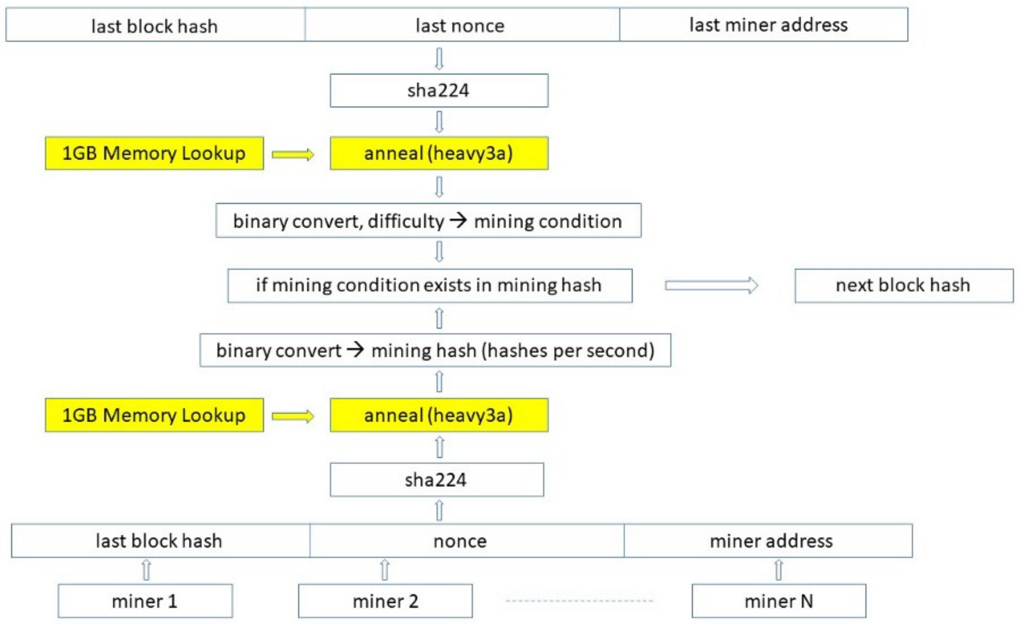
The FPGA operation was alternating between mining on his own account and using Pool 4 in the figure above. After the Hypernodes had been successfully launched, the core dev team had to act swiftly, and an evolution of the mining algorithm moved to the top of the priority list, even though this had not been placed on the roadmap which was published a few months earlier. The modified mining algorithm was developed and tested on a private testnet in record time during September 2018. It took less than 3 weeks from the first conceptual ideas until launch of the new mining algorithm. Even with this rapid pace of development, the exchanges and the pools were given 1 week's notice and time to update their nodes.

What made the FPGA mining so efficient was the fact that the legacy Bismuth Mining algorithm required only processing power, but no memory. After careful research and tests, one of the core developers, EggdraSyl, came up with a slight change to the current mining algorithm that:

- Is memory hard.
- Would block or penalize the specific FPGA miner a lot.
- Still is fast to verify on nodes.
- Needs only minimal change to current GPU miners, so it can be implemented quickly by pools.

The "Bismuth Heavy 3" mining algorithm was born, and would be used after the fork.

On October 8, 2018 - block height 854,660 - the new and novel Heavy3 mining algorithm was introduced on the Bismuth mainnet. Previously the Bismuth mining algorithm was computationally expensive, but required little memory. In order to make the new mining algorithm more resistant to FPGAs and ASICs, a requirement to hold a 1GB random binary file in memory was introduced, as illustrated with the yellow boxes in the chart below:



Bismuth Heavy3
The idea behind the "Heavy3" algorithm designed by EggdraSyl is both simple and effective: It requires a read from a random offset in a fixed lookup table, for each tested nonce.

This concept can be applied to any other mining algorithm as an additional layer to protect against a similar attack. If the matching algorithm uses hashcash or not - bismuth does not - is irrelevant. The final hash state that is tested is a vector of 32 bits words. Since it is a hash result, it can be considered as a random vector, it can contain anything, and you can not reverse the process – this is a hash core property – The lookup table also contains random data. For each nonce, the extra step is applying a XOR transform to the hash output, given a random vector from the lookup table, with the index begin determined by the hash itself, therefore at a random, non predictable, location. The result – xor'd hash state – is considered as the input vector to the difficulty matching function.

- This transformation does not affect the probability of finding a good candidate.
- It does not change the hashing algorithm itself.
- It does not change the difficulty matching algorithm either.
- It requires reading of about 8 words from a random index for every tried nonce.
- The miner has to keep a copy of the whole lookup table in memory at all time.

This is then a generic tweak that can be applied instantly to any other crypto.

Mid and Long Term Considerations
The core team is still in favor of FPGAs and – why not – dedicated ASICs hardware for Bismuth.

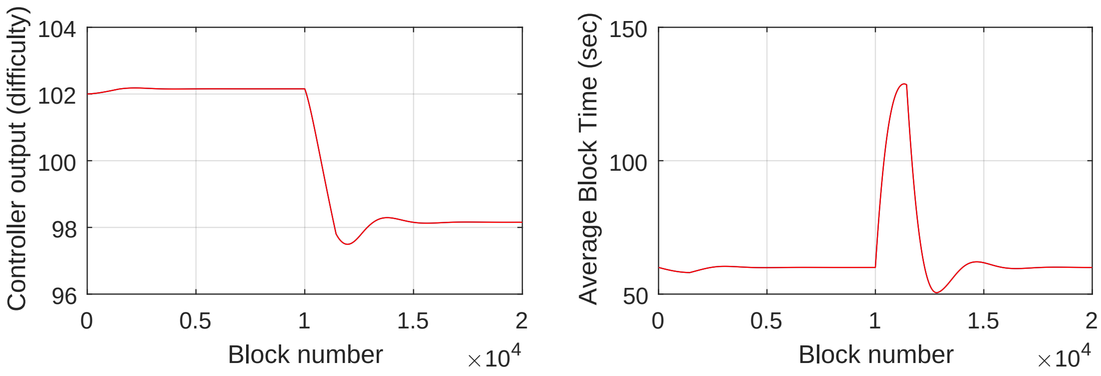
- This is the only way a PoW coin can protect its network. Pure GPU coins always are at the mercy of a nicehash or similarly rented hash attack.
- Hash/Watt of FPGAs and ASICs is way better than GPUs, so you have better efficiency and more hash, a more secure network with more resources needed to take over.

This is only true if the mining equipment is largely available. It is not when a single operation has thousands of custom proprietary hardware.

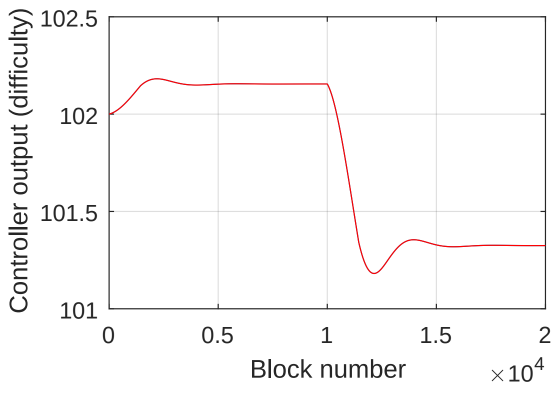
Next step could be the introducing *several mining channels*, so that everyone has a fair chance to mine, reach profitability, and contribute to the network safety (CPUs, GPUs, FPGAs, ASICs). This would also allow for faster algorithm changes should a similar situation arise again.

Chain Security

The Bismuth blockchain uses a feedback control strategy to calculate the difficulty adjustment in the mining process, see doi:10.4173/mic.2017.4.1 in the MIC journal for more details. In combination with this feedback controller Bismuth uses the longest chain rule to determine consensus. How does such an approach compare with the alternative approach of deciding consensus based on total hashwork (ie. selecting the chain which has the most hashwork in it)? To answer this question, the Simulink model shown in Figure 7 in the MIC paper is used. Consider the following scenario: 1) The difficulty level is stable at diff=102 and the 24 hour average blocktime is stable at 60 seconds. 2) At block number 10,000 a large pool with 25% of the total hashpower decides to break off on it's own chain to try to mine a longer chain than the rest of the network. There will now be two competing chains: 1) The chain breaking off with 25% of the original hashpower and 2) The main network which will in this case get reduced to 75% of the original hashpower at block number 10,000.



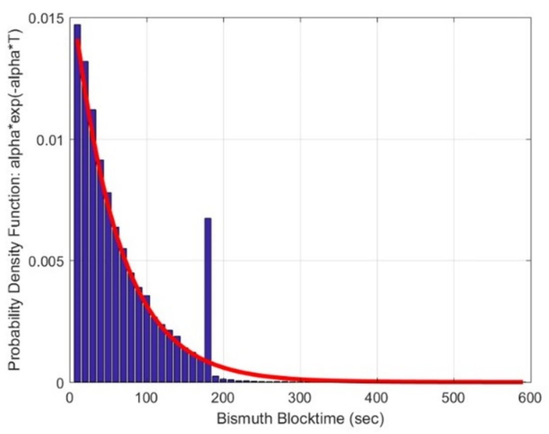
The figure above shows how the feedback controller and the difficulty adjustment react for the 25% hashpower chain. The difficulty drops from 102 down to 98, while the average blocktime increases to almost 130 seconds, before it comes back down again and settles at 60 seconds. The total accumulated time to generate 20,000 blocks in this example is 14.87 days.

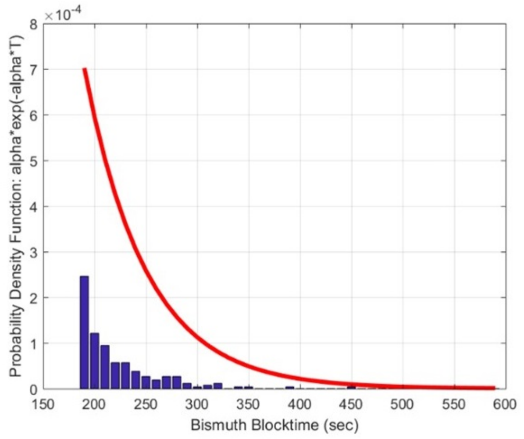


The figure above shows how the feedback controller and the difficulty adjustment react for the other chain with 75% of the hashpower. The difficulty drops from 102 down to 101, while the average blocktime increases to about 71 seconds, before it comes back down again and settles at 60 seconds. The total accumulated time to generate 20,000 blocks in this example is 14.02 days. The reason why the 75% chain generates blocks faster than the 25% chain, is because the overshoot in blocktime (71 seconds vs 130 seconds) is smaller for the chain with the largest hashpower behind it. Since the 75% hashpower chain produces 20,000 blocks in this example faster than the chain with 25% of the hashpower, the 75% chain will also be the longest and the consensus rule currently implemented in Bismuth will select the chain with the most hashwork in it as the winner. As can be seen from this simulation, the chain with the largest amount of remaining hashpower will produce the longest chain in case of a network fork. In other words, Bismuth's implementation of a feedback control algorithm for the difficulty adjustment combined with the longest chain rule achieves the same result as selecting consensus based on the largest total amount of hashwork, a method which is used in other blockchain implementations. Bismuth's unique implementation of difficulty adjustment and longest chain rule can therefore be considered as secure as other consensus implementations based on total hashwork.

Tail Removal Block Validation

The probability density function (PDF) describing the distribution of blocktimes in many crypto-currencies, such as Bitcoin for example, has a long tail, meaning that there is a small, but nonzero, probability that it can take a very long time to generate a new block, even when the computational power of the miners is constant or increasing. Such long blocktimes is a problem for two reasons: 1) Long processing times of transactions is undesirable. Processing times which are many factors larger than the desired average blocktime are seen as negative, 2) a blockchain feedback control algorithm can typically not distinguish between a long tail blocktime and the situation where the computational power of the miners has dropped. Hence, a long tail blocktime will normally cause a fastresponding controller to lower the difficulty when it should not do so, and this behavior can cause unwanted oscillations or instability in the process.

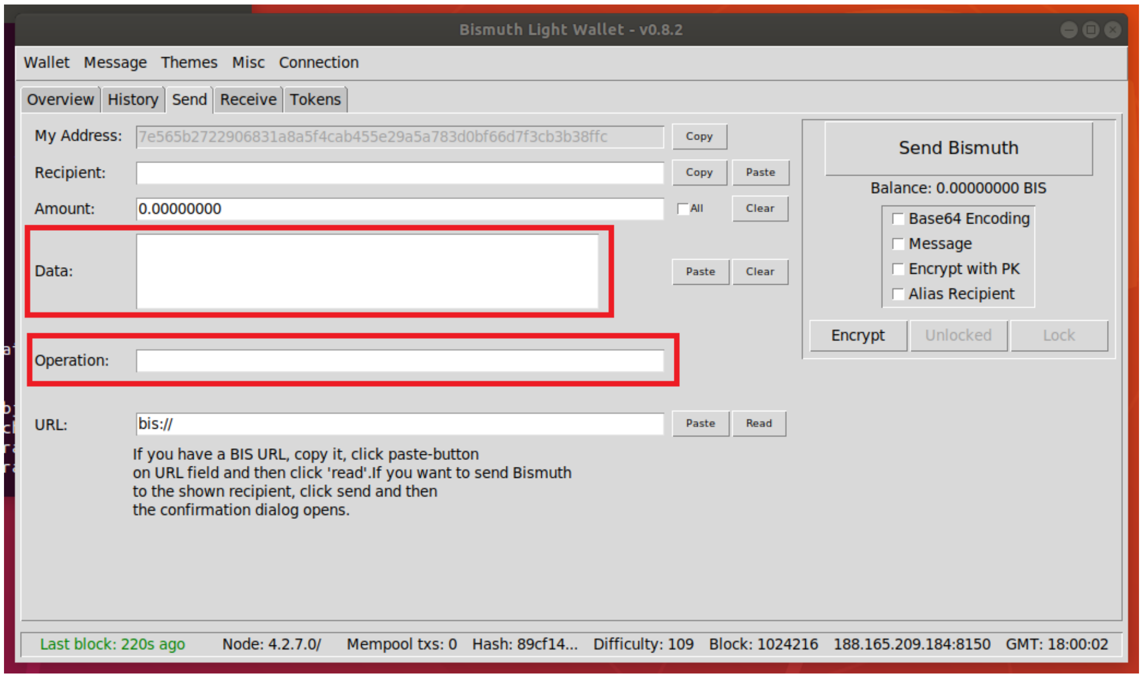




In Bismuth a solution for removing long tail blocks has been implemented and the results are shown in the figures above. The figure on the left shows the PDF without the tail removal code activated (solid red curve) and with tail removal activated (blue bars). The figure on the right shows the same, but zoomed in at blocktimes larger than 180 seconds. As seen from these figures, the probability of long tail blocktimes (larger than 180 seconds) has been significantly reduced in Bismuth, ensuring timely execution of transactions. For more information about the actual implementation, see the journal article: J. Kučera and G. Hovland, "Tail Removal Block Validation: Implementation and Analysis", http://dx.doi.org/10.4173/mic.2018.3.1

Operation and Data Field

Bismuth has two fields which can be used by dApp developers: the Operation and Data fields. These fields are marked with red rectangles in the light wallet in the figure below:



My Address: 7e565b2722906831a8a5f4cab455e29a5a783d0bf66d7f3cb3b38ffc

Recipient:

Amount: 0.00000000

Data:

Operation:

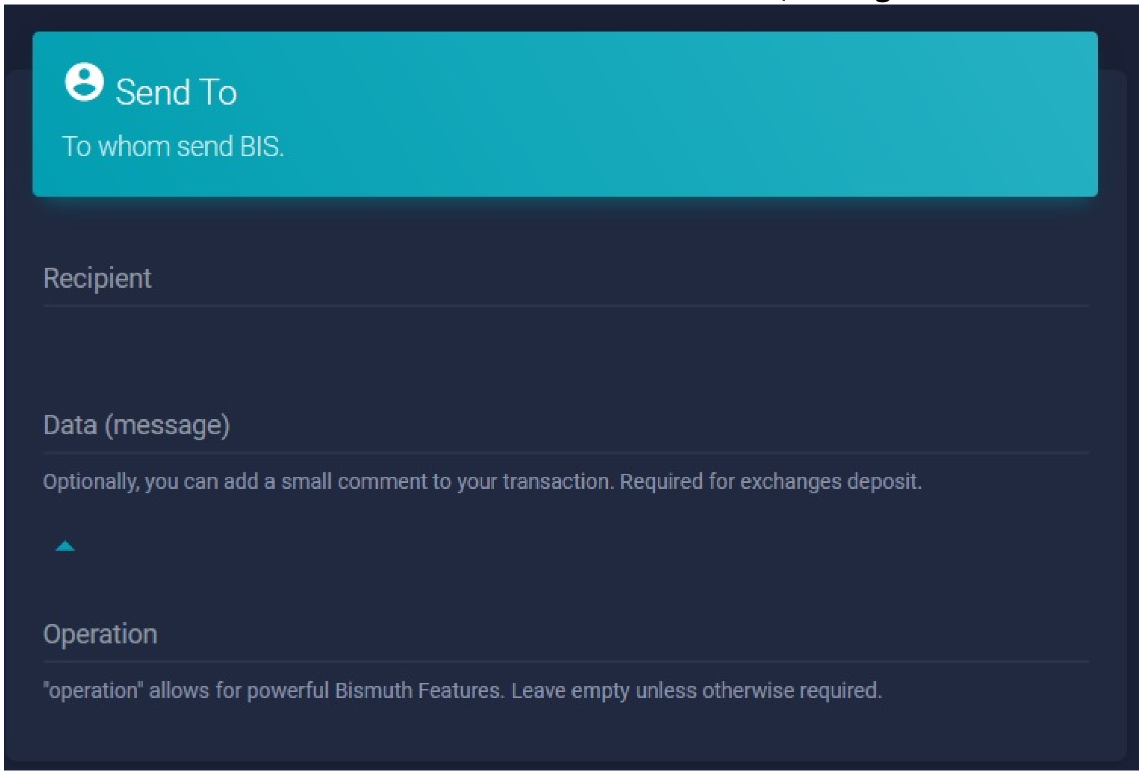
URL: bis://

If you have a BIS URL, copy it, click paste-button on URL field and then click 'read'.If you want to send Bismuth to the shown recipient, click send and then the confirmation dialog opens.

Balance: 0.00000000 BIS

Last block: 220s ago Node: 4.2.7.0/ Mempool txs: 0 Hash: 89cf14... Difficulty: 109 Block: 1024216 188.165.209.184:8150 GMT: 18:00:02

The two fields are also available in the Tornado wallet, see figure below:



To whom send BIS.

Optionally, you can add a small comment to your transaction. Required for exchanges deposit.

"operation" allows for powerful Bismuth Features. Leave empty unless otherwise required.

The operation and data fields can also be used programmatically, for example by using the examples at This Link. A code example from this repository is:

```python
from bismuthclient.bismuthclient import BismuthClient
```

```
if __name__ == "__main__":
    client = BismuthClient(wallet_file='wallet.der')
    if not client.address:
        client.new_wallet()
    client.load_wallet()
    print(f"My address is {client.address}")
    txid = client.send(recipient=client.address, amount=0) # Tries to send
        0 to self
    print(f"Txid is {txid}")
```

This example code sends 0 BIS to yourself. To send BIS to another account, replace client.address with the account address as a string, for example: recipient="9ba0f8ca03439a8b4222b256a5f56f4f563f6d83755f525992fa5daf"

To utilize the operation and datafield, the following example command could be used:

```
txid = client.send(recipient=
    "9ba0f8ca03439a8b4222b256a5f56f4f563f6d83755f525992fa5daf",
        operation='dragg:transfer', data='draggon_adn')
```

The use of operation and datafield, is a major difference from other cryptocurrencies. Because of the real world principle, the Bismuth project acknowledges the need to store data in the chain, and does not make it hard for the user. Instead, Bismuth uses that as meta data that can serve for higher level operations. This is in the pure Bismuth spirit: abstract data, easy to read/write, the node handles without having to know what operation/datafield means and process, but that participating apps can interpret and act upon. The operation/-datafield also allow the separation between the node (base layer, transport/authenticity/immutability/abstract data) and the dApp (use the data, interprets, acts upon, higher level of operations).

Private Contracts

As described in the section above, Bismuth has two fields which can be leveraged by extra protocols: the "data" and "operation" fields. These fields can be utilized in different ways to create private contracts:

1) Private as in private data, ie. encrypted messages.
2) Private as in non public contract code.
3) Private as in private, untraceable recipient by using abstract transactions and encrypted messages.

Abstract Transactions
One of BIS strengths is to allow abstract transactions. These are transactions

with 0 BIS involved, and data that is only understandable by the dApps which participate in that protocol. The ”operation” field is used as a kind of a ”command” operator. The convention is to use strings formated as ”class:operation” to allow for easy classification, like a kind of namespace. The openfield then holds the associated (short) data. Developers can define their own operations, but have to make sure that their protocol does not use already used namespaces, see This Link for an up-to-date list of existing protocols.

BIS transaction fees are fixed and depend only on the openfield length in byte. Fee = 0.01 + len(openfield/100000). On-chain storage is not encouraged and could be restricted in the future. Developers should limit the payload to the minimum and use side chains or dApp to dApp channels to store real data.

Hypernodes and Sidechain

When so many different masternode coins are released, it can be useful to give a few hints at what makes the Bismuth Hypernodes so innovative.

Bismuth Hypernodes as a Tech lab and Bismuth abilities proof
As Bismuth grows and the team gathers more and more experience, some weaknesses of the nodes are more visible. It is not always possible to test or change the design on a running blockchain, used by many people and organizations, but it is easy to use the Hypernodes as a lab to field test new technologies, libraries and algorithms. The Bismuth Hypernodes will include several new technology layers that could be used later on, on the core code.

Hypernodes also demonstrate the ease of integration and leverage of Bismuth abstract transactions. The Hypernodes are a good example of what can be done with the openness and abstraction layer that Bismuth provides.

Network Value
The goal of the Bismuth Hypernodes is to provide added value to the network. Some basic masternodes implementations are merely staking or check for a ”ping” answer.

The Bismuth Hypernodes do operate on an entire different layer. They use a chain on their own, with no currency but metrics - Key Performance Indicators (KPIs) - instead. Both chains are loosely coupled and operate in an almost independent manner, with very different rules. The Hypernodes operate on a Proof of Stake (PoS) chain, with no mining and no competition between the Hypernodes. They observe and store their KPIs in the PoS chain. Since the Hypernodes are not part of the PoW chain, they do not add extra attack vectors.

Eligible Hypernodes are derived from immutable info stored in the PoW chain. Quality index and bad behavior from both PoS and PoW nodes is recorded, immutable also, in the PoS chain for later action.

The independence of the two chains ensures:

a/ It is way harder to manipulate the network, since it requires to forge both chains in very different ways.

b/ Bad actors and cheating attempts are recorded in an independent and immutable way. You cannot cheat unnoticed.

Bismuth may be the first crypto-currency to come with an integrated but independent KPIs dedicated side chain that monitors the network and ensures actors fair play.

Bismuth Hybrid PoW/PoS
The protocol used by Bismuth PoW/PoS is a hybrid, two-layers approach. It uses strength from both PoW and PoS and tries to avoid the pitfalls of other hybrid approaches. Rather than being integral part of the main consensus, the PoS layer acts as an impartial observer of the PoW chain, and its metrics can then be used to act upon the core cause of the issues – the bad actors.

PoS watches over the PoW actors, PoW decides who can be PoS actor. It is like an Ourobouros, each chain having some control over the other one. In terms of security, it is a big improvement, since both chains would have to be attacked at the same time and in a coherent manner to gain some advantage. Since the mechanisms of each chain are so different, this is an epic task.

Future use of such side chains
The loosely coupled two layer approach of the Hypernodes has several advantages, as well as many future uses. Potential ones are:

- There is no incentive, on the contrary, in forging more blocks or blocking others from forging theirs to take their turn.
- For now, there is only a single PoS chain, the Hypernodes with their metrics. But any number of PoS chains can be added to the Bismuth Network, without overloading the main PoW chain. You can see them as side chains, and they give a lot of flexibility.
- Since the chains are so loosely coupled, the very same protocol can be used by other currencies. Almost any currency that is PoW only could add a Bismuth-like Hypernode layer and benefit from the security of this hybrid approach, as well as flexible side or child chains.

So, Bismuth continues to explore new territories, with practical and field tested code. The goal is for the Hypernodes code to become a framework for easy to run side chains. Every such chain can have its own rules, block time, fees (or no fees), storage, and still be secured by the main BIS chain

Some peeks into the technology
From a simple proof of concept, the Bismuth code grew into a full featured node

and client code base. Some new technology had to be left over in order to keep some compatibility with the current network. The Hypernodes are not constrained by that, and so can use a more modern approach from the start.

Async IO
Async / Await, use of co-routines, is a big strength of modern python. This allows to write efficient and easy to read async code. No more need to spawn hundreds of threads of processes, handle locks and hard to debug race conditions. No need to fight with several concurrent access to a single database file. No more infamous Global Interpreter Lock (GIL) limitations. Within the Hypernodes, Async is used to its full extent. The core of a Hypernode is a Tornado Server and client, with callbacks using async calls themselves. This is both a big plus for the performance of the node itself (it barely uses any resource under load) as well as from a safety point of view.

Protobuf
Google Protobuf is a serialization protocol that is both fast and efficient. It also has available bindings for almost any language, so it was chosen as the low level exchange format. Instead of a verbose text encoding of the structures, the Hypernodes will use protobuf.

Pros:

- fast, low overhead.
- Small size of the packets.
- more validity controls.
- cross OS and language compatible.

Cryptographic primitives

Hash
The Hypernodes make use of modern blake2 cryptographic primitives. They are fast, safe, and have a variable output length.

Keys and signature
Hypernodes addresses use classic ECDSA cryptographic curves.

Chain coupling
How do the two layers interact ? How does PoS (Hypernodes) use PoW (nodes)? Each Hypernode runs along with a classic Bismuth (PoW) node. It has read access to the PoW ledger and node status, like peers, consensus, block height. This is a read only access. The Hypernode is just an observer. From this data, the Hypernode is able to:

A/ Get a safe, immutable, shared list of valid Hypernodes. At each round start, the Hypernodes need a common list of theirs to decide of the round jurors. This list is extracted from the PoW chain, from a checkpoint in the past with enough confirmations to be stable. It is composed of the valid registrations of the Hypernodes owners. This start list cannot be manipulated by the PoS layer.

B/ Collect metrics over the PoW peers. Each time a peer connects to the PoW node, it leaks much info: version, block height, ip, ping time, consensus state, connectivity status ... The same for specific actions like rollbacks. Even a failed connection attempt is something worth recording. Those are metrics the Hypernode can collect and just write in the PoS chain. Once aggregated over a round by all the Hypernodes, these metrics can be used to evaluate the state of the PoW participants, and then compile lists of «good» and «bad» peers for instance.

How does PoW (nodes) use PoS (Hypernode)?
In the same way, the PoW layer can use data provided by the Hypernodes:

- Dynamic list of bad and good peers.
- More reliable info about the current net height.

This will allow the nodes to avoid bad rep peers - swarms of fake nodes in the cloud or fake nodes targeting miner nodes - as well as old nodes that are stuck on a bad block, not maintained or on an old version. So, instead of being integral part of the PoW process - that would only add complexity as well as attack vectors - the PoS chain rather gives access to impartial extra data that is used to qualify the PoW and PoS actors.

The metrics - KPIs
Many metrics are available here to act upon. The role of the Hypernodes is precisely to collect them all, so the team can analyze and decide which one is to be used, and how. The metrics themselves can evolve with time. So can the active metrics and their corresponding trigger. The Bismuth developers believe this will be hand crafted at start, then evolve into governance parameters. Hypernodes and/or nodes owner will likely be able to vote upon various triggers and levels in order to mitigate observed bad behavior. The precise metrics and how they can be used will be detailed in another – evolving - document.

Hypernodes payouts
With usual masternodes mechanism, the forging masternode gets a reward for each block it forges. This means there is an incentive to cheat, since if you get more blocks or deny other blocks, you can hope for a higher reward. With the Bismuth Hypernodes, such an attempt would play against you, since the reward is not directly related to the blocks you forge, and everyone would see

you cheat and record it in the PoS ledger, forever. Also, with the Bismuth scheme, every Hypernode owner is paid on a regular basis. No randomness involved. Every finished round leads to a payout for every active Hypernode. Hypernode payouts can be handled by several means. In the first steps, they will be handled by a private contract so it can be tuned and the team makes sure it is safe.

The process is something like this:

- There is a given reward amount for the period (can be fixed or a governance parameter).
- The «active» Hypernodes for the period share the reward. Active means the Hypernode was in the list of valid Hypernodes, has collected and sent metrics, has interacted with peers, may have forged some blocks if it was a juror.
- The rewards should be proportional to the collateral amount. If you have one Hypernode with twice the collateral as another one, it will get twice its reward.
- The private contract computes every reward and pays them.
- The details – all computation inputs - are public and stored in the PoS chain for everyone to verify.
- The algorithm is public too.

Governance
Since KPIs and levels will likely be used to trigger bans or actions against some actors, it is natural to ask for a governance process. At launch, given the experimental nature of the project however, this will not be possible. The team will manually handle the analysis process, in order to have a good overview of the meaningful KPIs and their usage. Later on, the filters could become autonomous and various parameters could be voted upon by Hypernodes owners.

Hyperblock Compression

Bismuth uses a dual database system for both redundancy and speed. Besides the standard full ledger database, hyperblocks are a compressed version of the same data, which contain only the last 15000 blocks and only sums of outstanding balances greater than zero for all preceding transactions. In some scenarios, hyperblock balances are compared to ledger data for discrepancy detection. Also, the hyperblock database is loaded into the memory on node startup, limiting hard drive access to minimize system load and increase database reaction speed and availability.

Penalty System

Every node in the Bismuth consensus system tracks the behaviour of all connected clients. Penalties are applied to all nodes that enforce chain switching through a single block rollback, one half of the penalty is removed for being honest and delivering the agnostically longest chain block. Nodes that are too far in the future for local consensus are banned automatically. This set of rules requires an attacker to not only own more than half of the computing power, it also requires him to be connected to all nodes in the system, attack them all at the same time and setup a majority of nodes. Even then, the attack becomes progressively harder with every attacked block.

Mirror Blocks

Mirror block technology enables easy implementation of hardcoded contracts like hypernode rewards, offline staking and development rewards. Mirror block storage is on-chain but outside of traditional blocks, denominated by a minus symbol not to interfere with the synchronization process. The mirror blocks are never shared between nodes, instead they are created from the contract execution, optionally dependent on the blockchain data. This approach makes it an integral part of the standard transaction database, streamlining operations like account balance evaluation.

Testnet

The Bismuth testnet consists of a network of nodes with different ip addresses, much like the Bismuth mainnet, but with fewer nodes. To setup a testnet node, the following parameters must be defined in the file config.txt:

```
port=2829
version=testnet
version_allow=testnet
testnet=True
```

The difficulty level for the mining algorithm on testnet is low on purpose. Hence, developers can easily set up a local pool and miner to generate BIS coins on testnet. For this, optipoolware is recommended.

Regnet

To setup a regnet node, the following parameters must be defined in the file config.txt:

```
version=regnet
version_allow=regnet
```

To test that testnet or regnet are running the following command can be used:

```
python3 commands.py statusget
```

The output from regnet should be something like this:

```
Number of arguments: 2 arguments.
Argument List: ['commands.py', 'statusget']
Regtest mode
{"protocolversion": "regnet", "address":
    "6a8b4990784617730af465a0dfcbb87284bca8b2189e02798d0a5a5f",
    "walletversion": "4.2.9", "testnet": false, "blocks": 1, "timeoffset":
    0, "connections": 0, "connections_list": {}, "difficulty": 24.0,
    "threads": 3, "uptime": 131, "consensus": null, "consensus_percent":
    0, "server_timestamp": "1549123577.02", "regnet": true}
```

For regnet, there is no need to set up a miner. The ”generate” command instamines N blocks with current wallet address as miner. For example, you can ”generate” 10 blocks, then you have bis you can test send on your regnet. After that, ”generate” 1 block, and your mempool transactions are instamined.

Regnet is useful because it starts from block height=1 and does not require to sync the blockchain with networked peers. Many features of a new dApp can be tested in the early stage on regnet. Only when distributed networked features need to be tested, does the developer need to switch from regnet to testnet.

Education and Research

Bismuth is an ideal platform for education and research. Examples of research done using Bismuth mainnet and testnet are the following journal articles:

- J. Kučera and G. Hovland, ”Tail Removal Block Validation: Implementation and Analysis”, http://dx.doi.org/10.4173/mic.2018.3.1
- G. Hovland and J. Kučera, Nonlinear Feedback Control and Stability Analysis of a Proof-of-Work Blockchain”, http://dx.doi.org/10.4173/mic.2017.4.1

Regnet would be particularly useful in an educational setting, as each student could run a local regnet on his/her computer, without the need to participate in a distributed network. Many of the basic concepts of blockchain technology could be taught using the Bismuth regnet. Researchers and students who develop novel functions and features using Bismuth, are encouraged to contact the Bismuth core development team on discord, at https://discord.gg/4tB3pYJ. New functions and features could potentially make it into the Bismuth node code, after a period of additional testing by the core team.

Future Prospectives

The goal of the Bismuth project is to make the core node well documented and as streamlined and efficient as possible, and then encourage future extensions and applications building on top of Bismuth to use the plugin system and separate repositories.

The Bismuth developers have a focus on supporting ”crypto standard compliance” and ”ecosystem”, to allow for easier interaction with existing tools, protocols and platforms, and therefore removing ”friction”. New address formats (such as ecdsa), bitcoin like json-rpc server, docker images of bismuth services fall in this category. Another example is the packaged and easy to use ”Bismuth-Client” python module which has proven to be valuable, with several team and third party devs building upon it, and attracting new devs.

Bismuth is already a feature-rich crypto-currency, and this whitepaper has focused on parts which are already implemented, tested and working. As more features are developed in the future, they will be documented and included in updated versions of this whitepaper. The core team has a development roadmap, which is available at this GitHub link.

Summary

This whitepaper provides an outline of the philosophy, pillars and features of the Bismuth crypto-currency. The three pillars of Bismuth are: 1) The real world principle, meaning that the core developer team takes a pragmatic approach when introducing new features and functions, 2) The ”need to store” meaning that you only store on chain what needs to be stored on-chain, and nothing more. Still, following the real world principle, on chain storage is not strongly discouraged and 3) Clear line of trust, meaning that when you have to trust something or someone, you know what/who you have to trust. Bismuth is not presented as trustless, whereas you implicitly trust several layers and people.

Some of the core features of Bismuth have been presented and discussed in some detail. Links and references have been provided for readers seeking more detailed information about the Bismuth crypto-currency and its implementation.

Disclaimer

The information in this whitepaper is for informational purposes only and does not contain any investment or financial advice. Please do your own research before making any investment decisions. None of the information in this document constitutes, or should be relied on as, a suggestion, offer, or other solicitation to engage in, or refrain from engaging in, any purchase, sale, or any other any investment-related activity with respect to any crypto-currency. Cryptocurrency investments are volatile and high risk in nature. Do not invest more than what you can afford to lose.

Document Revision History

- August 13, 2019: v1.2, multiple address schemes, tokens, formatting, updated future prospectives
- May 24, 2019: v1.1, added sections about mirror blocks, chain security, updated coin supply and rewards
- April 3, 2019: v1.0 of whitepaper released