FLASH
GO FASTER
WHITEPAPER
BLOCKCHAIN TECHNOLOGY FOR COMMUNITIES

Abstract

FLASH is a public Blockchain operated for the benefit of every person in the world. It is a platform built to enable consumers, merchants, and developers to harness this powerful technology in every day moments.

We have created a transparent and fair playing field for all people; Limited only by your imagination, FLASH invites you experience this feature rich Ecosystem of next-generation Blockchain technology. Bringing together our core values with the values of the community, we are proud to present this next phase in the evolution of FLASH.

May 1, 2018

Contents

Introduction

The fundamental principle of FLASH is that all work or contributions to the network should be valued by the community in an objective fashion. Allowing the free market process to function creates a mechanism whereby all forms of work can be reduced to a common denominator -- FLASH.

Any form of work be it valuable time, a user's work and attention, a special skill set such as developing tools, various forms of energy (ie. processing) and settlement can be valued in real time based on the market supply and demand for that specific work/contribution.

The ways to contribute to the community are only limited by one's imagination and the community's collective valuation of it. At its core FLASH is simple to understand, completely transparent and distributed governance with no complicated rules and conditions that are hard to work with. We put the trust in the community and the technology. Because FLASH has a very simple and fast settlement system along with many tools for community members, it will have many use cases.

FLASH has no defined value; it can be freely exchanged by community members who decide independently the disposition of their private property.

Legal Compliance

FLASH, like Bitcoin, was designed from the ground up to be legally compliant. Originally based on a fork of Litecoin, the technology was developed in Canada by Flashnet Tech, Inc. with support from donors who received Counterparty assets called FLASHPRE and MEGAFLASH. No US donorswere allowed. A community member in Vietnam then mined the entire supply of FLASH and they were given to an anonymous community member to distribute. The entire pre-mined supply of FLASH coins were airdropped for free to community members and people who tested FLASH. The coin had no value when it was distributed, nor is there any defined value to FLASH.

To date, ~818m coins have been claimed and circulate, while ~82m FLASH remain unclaimed as of May 1, 2018. Following the completion of the first version of the FLASH software and distribution of FLASHPRE, MEGAFLASH and FLASH coins, Flashnet Tech was shut down and the Third Millennium Foundation for Economics and Culture was created in Liechtenstein to support community coins and promote the FLASH technology. FLASH is a protocol maintained by the community. It is not any kind of legal entity or company of any kind, it has no employees, contracts and no standing of any kind. It's an idea in the public domain.

The FLASH Philosophy and Applications

The FLASH blockchain is run as a not for profit, public service by the FLASH community. No commercial activity is conducted in the operation of the FLASH blockchain, as the design goal was to make using FLASH nearly free. FLASH is ideal for use in any country, its original design was for developing countries, where fast and cheap transactions are necessary. Commercial activity is conducted inside applications that utilize the blockchain, which can have wallet functions, or they directly access the blockchain or the API. Economic incentives can be created by applications that use the FLASH blockchain, building fees directly into the apps when transactions or other services are conducted.

Applications that are being built with FLASH by various app developers include:

- Media and gaming. FLASH is ideal to reward content creators and players in far flung locations.

- Personal Exchange (aka Human ATM). Developing countries do not have the banking infrastructure of developed countries. This creates vast opportunities for the Personal Exchange, a very simple and fast way to trade crypto coins in person or over the internet anywhere it's legal.

- Remittance. Because FLASH is a low-cost coin, as soon as exchanges are created in different countries, it becomes possible to do rapid and low-cost remittances, either via exchanges or Personal Exchange.

- FLASH Web Wallet. The fastest and most convenient way to get started with FLASH; there is nothing to install, just a quick signup via email. The FLASH Web Wallet will soon include a social send feature.

- FLASH Android Wallet. Android has 75% worldwide market share and leads in developing countries by a wide margin. A wallet with simple send and receive, 25 languages and Personal Exchange toolset including client discovery, interactive map (using location services) and chat.

- (future) Circumvention. This will use FLASH nodes to relay transactions, messages and files via a distributed network. FLASH will be the gas.

- (future) DEX Blockchain Marketplace. FLASH can be used as a signaling plane to announce offers to buy or sell cryptocurrencies in a decentralized manner.

FLASH Architecture Overview

FLASH IS A PREMINED, DECENTRALIZED BLOCKCHAIN, BASED ON THE ORIGINAL BITCOIN/LITECOIN BLOCKCHAIN. IT HAS BEEN OPTIMIZED FOR SEVERAL VERY SPECIFIC USE CASES:

1. Compatibility with BTC/LTC which makes FLASH very easy to add to new exchanges (only a few hours) and port to open source software tools such as the Trezor hardware wallet.

2. High performance. Transactions can settle in less than 2 seconds, but usually under 5 seconds and coins can be immediately re-sent to another community member. Final permanent settlement happens in under 2 minutes.

3. Highly secure. A number of security flaws with the original bitcoin blockchain have been patched. The platform has been enhanced with Curve25519 Elliptic Curve encryption, which is now standard on BTC/LTC.

4. Commercial scale throughput capability. Delegating the mining allows much greater capacity of the system. Real world testing of actual live net throughput will be forthcoming.

5. Low cost and environmentally friendly. Removing nearly all of the expensive mining removes a lot of the cost most proof-of-work (PoW) cryptocoins add through inflation or high transaction fees.

6. Designed for communities, it is a community coin, issued and used by a community, it is not a utility or security token. The FLASH blockchain is nearly free, while commercial users are free to use this blockchain to build applications using FLASH and charge fees, sending FLASH directly into the app developer's or partner's wallets.

7. A web wallet with easy to use features including Send using email or public address, transaction history, request FLASH, QR code support, contacts, 2 Factor Authentication (2FA), merchant tools and key recovery.

8. Third party wallets including Qt, Coinomi, CoinPayments, ETHOS and Android wallets are supported. Support for the Trezor hardware wallet is expected soon.

In order to reach these critical metrics for wide scale use by millions of community members the FLASH platform uses a model with delegated governance and minimal mining. The platform leverages both the existing distributed database technology with blockchain technology characteristics, decentralized control, immutability and creation and movement of digital assets.

FLASH Web Wallet Overview

The FLASH web wallet system builds upon three tier application platforms. Like most standard systems. we have User Interface, Communications, and Business Logic / Storage tiers.

The User Interface Tier enables the end user or application to interact with the FLASH blockchain. The FLASH web wallet platform leverages HTML5, CSS3 and JavaScript on the browsers, with no extensions, enabling seamless cross browser support. We have developed and adopted all technologies that enable JavaScript to do what C/C++ and Java are able to do. In addition, the FLASH System uses Twitter Bootstrap to provide a responsive web framework that works on any device. The Communication Tier enables a secure tunnel between the User Interface Tier and the Business Logic / Storage Tier without the need for OpenSSL. The secure protocols used by the FLASH web wallet includes:

Our work in the area of communication also means that we are an early adopter of new secure protocols that run over JavaScript using HTML5.

The Business Logic / Storage Tier enables all of the transaction flows, business logic, and networking systems to be stored and operated within this layer. This tier is responsible for executing proprietary algorithms to store in the distributed database, which power FLASH.

- The Key Exchange Node is a horizontal cluster of servers that provide key exchange or key lookup for every transaction on the FLASH System, like an index that runs alongside the blockchain to look things up faster. In the near future, these nodes will be decentralized. Due to the nature of cryptographic key pairs on the Bitcoin transactional system, every transaction requires a public wallet address lookup. Key exchange needs to act as the public-address registrar. The Key Node also allows currency exchange, messages, public escrow and other information flow among peer-to-peer wallets. The message engine has enabled logging and notification of all activities on the wallet communication using Sendgrid email notification. The Key Node uses NodeJS, ZeroMQ, Redis, and MySQL and is hosted by various community members for the benefit of the community.

- The FLASH Web Wallet Application is a horizontal scaling server application that enables all FLASH Wallets to be used from different Web Browsers. The FLASH Wallet Application has most of the Bitcoin Wallet functionalities and key exchange functionalities. It's a virtual online wallet using the HTTP Protocol / Web Sockets. Additionally, Qt wallets with source code have been provided without mining for Windows, Mac and Linux. Exchanges are supported by FLASH.

- The Web Service API Servers pre-process, convert and index all the FLASH transactions from the Web Interface and send them to the blockchain. This significantly speeds up the transactions as each wallet doesn't need to sync all blocks from the blockchain of the local server. All wallets can share the blockchain on the Blockchain API server. This improves the transaction performance significantly due to the reduced network latency for each wallet synchronization. Another very important aspect of the technology is the significant reduction in the possibility of "double spending". All blockchain transactions are indexed, preprocessed and validated through the Gateway Server in a manner similar to Bitcoin.

- The FLASH Blockchain is a fork of Litecoin technology. A number of significant modifications have been made. The settings have been changed in order to speed up the transactions to the blockchain. All the coins have been pre-mined, and the mining has been reset to just above the minimal degree of difficulty factor. Up to 151 staked Delegate nodes will be selected by the FLASH community at large. Each Delegate is required to keep a minimum of 1 million FLASH staked in a Qt wallet. These Delegates elect up to 25 miners and determine the governance of the FLASH blockchain, transaction fees and other matters. The Delegates and Miners share the transaction fees and a pool of donated FLASH from the community. Miners are approved by the Delegates.

- The purpose of the FLASH Blockchain is to replace the traditional transactional database with a network storage database and include an end-to-end security data structure.

- Transaction Fees are currently set to 0.001 FLASH per transaction, these fees may be raised or lowered in the future, depending on voting by the elected Delegate nodes.

Flash Governance and Processing Overview

Abstract

FLASH will implement a new delegate-based consensus model which relies on trusted delegates elected by the community to rapidly reach consensus on the blockchain and ensure its security. Every user will be able to use coins they control to cast votes for delegates. Elected delegates will vote on matters relating to the network such as transaction fees and miner selection. Delegates elect a small set of 25 miners and these miners are the only nodes that are allowed to create new blocks on the blockchain. Rather than using PoS or PoW to secure the chain, a set of rules is enforced by the network to control the ordering of block generation rights which are granted to the set of trusted miners. This new delegate model maintains a high degree of network security while at the same time enabling very high transaction throughput.

It is really simple actually: if >50% of the Elected Miners "vote" that a block is valid, then that block is permanent and can never be undone. The "vote" is cast by the Elected Miner creating a block in the chain that comes after the block in question. The block chain goes like block 1 → 2 → 3 → 4, etc. So if >50% of the Miners build a block in the chain after block 1 then we know block 1 is legit and permanent. No node will ever accept a conflicting blockchain that tries to say block 1 is invalid.

Network Entity Types

There are three types of entities on FLASH's new network:

1. Normal users
2. Delegates
3. Miners

Normal Users

Any FLASH address holding Flashcoin is considered a normal user for the purpose of this discussion. Just like in Bitcoin, there are no on-chain user identifiers, just UTXO's controlled by pseudo-anonymous addresses.

Delegates

Delegates are elected by normal users. Delegates must run for election by staking a minimum of 1,000,000 FLASH and providing information about themselves, including a Delegate ID. Users use the coin they control to vote Delegates into office. Once elected, Delegates can vote on matters relating to the FLASH network. Each Delegate places votes which are weighted based on:

1. The amount of FLASH they've staked
2. The amount of votes they've received
3. How long they've been in office (seniority)

Delegates are elected for a period of 30 days at a time. Votes for Delegates can be cast at any time, but they are only evaluated once every 30 days for the changing of seats. A maximum of 151 Delegates will be allowed. If there are more than 151 nodes running for election as a Delegate, then the top 151 based on the criteria above will be elected.

Delegates must stake their coin to begin running for election, and their coin must remain staked for the entire election period. Third-parties may also stake coins for any given Delegate but those staked coins are locked up for the entire election period and term in office the same as if the Delegate had staked the coins themselves. Delegates can remove themselves from the running for the next election cycle at any time, but their coins must remain staked for the current cycle. Delegates will be automatically removed from office if they become inactive.

Delegates receive a portion of the network transaction fees as compensation for their duties.

Permanent Delegates

Of the 151 Delegates, 50 are Permanent Delegates who do not require voting via elections. These Permanent Delegates are otherwise the same as elected Delegates; they must stake the minimum 1,000,000 FLASH, their vote weight is based on the same factors, they can be Miners, they are compensated the same and have the same requirements except the minimum staking requirement to mine is 1,000,000 FLASH instead of 2,000,000 FLASH required by Delegates. Permanent Delegates do not have to run for election.

Permanent Delegate positions are held initially by those who have contributed to the strength and well-being of the FLASH ecosystem; these positions are transferable.

If a Permanent Delegate does not stake the required minimum of 1,000,000 FLASH, or they become inactive, they maintain their position as Permanent Delegate but cannot act as Delegate until in good standing or until this role is transferred to someone who meets the requirements for good standing.

Miners

Only Delegates can become Miners, and they do so by signaling that they wish to be a Miner and by gaining support from their fellow Delegates in the form of votes and by having staked a minimum of 2,000,000 FLASH. The minimum staking to mine for Permanent Delegates is initially set to 1,000,000 FLASH. Delegates are therefore responsible for voting-in only high-quality and trustworthy miners. The number of Miners is kept low, at a maximum of 25, in order to support high block rates across the network and high transaction throughput. The larger the mining pool is the less efficient it becomes, and FLASH's limited Miner count is an optimal balance of redundancy, security, and performance. Miners receive a portion of the network transaction fees as compensation for their duties.

Transaction Fees

At network bootstrapping there will be a default fee of 0.001 FLASH per kilobyte, with a minimum fee of 0.33 FLASH. The transaction fee rate and minimum are parameters that Delegates can vote to adjust in the future. Transaction fees are not collected per-block by the miners, rather they are collected once per day and distributed to both Miners and Delegates.

Staking Mechanics

In order to run for election a person must create a FLASH address to represent their node. The candidate will then be identified by that address and the actions they take will be signed with the associated private key. To enter the election the candidate can use the Qt wallet to create and broadcast their node's registration form along with their FLASH stake. All registrations and stakes are sent to a special election address from which the network will not allow spending, except to return the staked coins back to the sender at the sender's request. Strict rules are enforced on the acceptance and return of funds to and from the special election address.

Delegates must stake a minimum of 1,000,000 FLASH to register for election. The more coin staked for a Delegate the more easily they can become elected, the more influence they will have on future votes, and the more reward they will receive. Additional stake contributions can be made for a Delegate by any person at any time, however all staked funds will remain locked for the entire election campaign as well as for the subsequent election term(s) if that Delegate is elected.

Voting Mechanics

Every transaction has the opportunity to vote for a Delegate by prepending a zero-value transaction output to that transaction that uses the OP_RETURN opcode followed by new voting related opcodes and data. Voting related OP_RETURN tx outputs must always be the first tx output. The sender of a transaction is able to use this OP_RETURN metadata to specify which delegates get how much of the transaction's voting power.

The Qt wallet will make managing a user's votes as easy as possible by showing which coins in their wallet are voting for which Delegates and which coins are not voting at all, and automatically managing the casting of votes according to the configuration specified by the user. Voting is not required in a transaction, but it is encouraged for the security of the network.

This method where the sender sets the voting metadata has an unfortunate side-effect on the user experience: every time a person receives FLASH they will need to send it back to themselves along with their vote if they want that FLASH to be used to vote in a different way than the sender specified. The Qt wallet can be configured to do this automatically so that it is barely noticeable to the user. This is an annoyance and will generate some redundant transactions as users re-send funds to recast votes, but this method allows for a dramatically more efficient accounting and scaling for the voting system than other methods. The need to resend transactions to recast votes can also be leveraged to combine transactions to reduce the total UTXO set.

Voting done by Delegates on matters relating to the network are handled differently than user votes. When a Delegate votes they send a transaction to a special voting address with data attached indicating what they are voting on and what their vote is. No Flashcoin need be sent to vote, however normal network transaction fees do apply. A Delegate's vote can come from any address, but the vote data must be signed by that Delegate's private key.

Mining Mechanics

Miners are elected by Delegates, and only Elected Delegates can become a Miner. Delegates vote for Miners with their vote-weight which is derived from coin stake, user votes, and seniority, and they spread their vote-weight over as many or as few miners as they like. The top 25 Miners, scored by accumulated vote-weight in the ballots, become the Elected Miners. If an Elected Miner is identified as a bad actor, or as an unreliable node, then Delegates can update their votes and remove that bad Miner in 1 block.

Miners must maintain an accurate system clock and have a low-latency and high-bandwidth network connection in order to effectively participate as a Miner. The system is tuned to generate a block every 5 seconds, however when the network is idle then no blocks need be emitted. Clock skew in the block timestamp will have a maximum allowance of +4 seconds in the future, and no block can have a timestamp earlier than the block before it. When a Miner creates a block they place their Delegate ID in the coinbase transaction for identification purposes, along with a signature to prove it is them.

There is no block reward for Miners to claim, no per-block transaction fees to claim, and therefore no incentive to mine blocks as fast as possible. Block difficulty is fixed at a very low value so that there is no hashpower arms race; any modern CPU can hash a block in under a second.

To secure the blockchain strict rules are enforced to control the order in which Elected Miners are allowed to create blocks. Each Elected Miner is assigned a single 5 second window in which they can create a block, if a block is needed, and then the next Miner in the list gets their own 5 second window. This ordered timeslot assignment prevents conflicts over who will mine the next block, thereby reducing the amount of bandwidth and processing that is usually wasted in high block-rate networks as conflicting chains are passed around and evaluated. This timeslot assignment also ensures that every Elected Miner has a fair chance to create a block.

To assign timeslots to Miners a function similar to the following JavaScript function will be used:

```
function canItMine(minerPosition, timestamp){ return ( (timestamp % 25) == minerPosition); }
```

There is a list of currently Elected Miners that is maintained by each node, and it is updated as Miners are voted in and out. A Miner's "position" is the index in the Elected Miner array that corresponds to that Miner. When a Miner is checking if it can mine right now it passes its own position into the "canItMine" function along with the current system time in unix format. When a node is evaluating a block for correctness before accepting it then it passes the position of the Miner who mined that block into this function along with the timestamp from that block.

Because each block creation asserts that the creator believes the chain of blocks before it are true and accurate, each block mined is effectively a vote by the Miner on what the correct current chain is. At any given time, there is a Consensus Height, a block height at which consensus has been achieved by >50% of the Elected Miners, and this Consensus Height increases as new blocks are mined. For any given block *B*, if >50% of the Miners in the Elected Miner pool have each created a block in the chain above *B* then consensus has been achieved for *B* and all transactions in and before block *B* are guaranteed to be final. The Consensus Height, then, is at *B*.

All transactions included at or below the Consensus Height are considered final, and all transactions above the Consensus Height are in the process of being finalized. "Confirmations" are no longer a measurement of how many blocks have been built on top of a given transaction's block, instead "confirmations" is a measure of how many Elected Miners have mined a block on top of that transaction's block, divided by how many Miners are needed to achieve consensus. Therefore "confirmations" is effectively the percentage of consensus a given transaction or block has achieved. Given a transaction *T* which was included in block *B*, if 20 blocks were built on top of *B* by 10 unique Miners from the 25-Miner pool of Elected Miners, then *T* and all other transactions in *B* have a consensus percentage of:

10 *(miners' blocks)* / 25 *(in pool)* / 0.5 *(for 50% consensus)* = 0.8 consensus
= 80 'confirmations'

Once a transaction has reached or exceeded 100 "confirmations" then it is final and there is no risk of it being rolled back in a double-spend. All calculations and readouts for "confirmations" will be capped at 100 to reduce confusion.

As long as an attacker is not able to compromise more than 50% of the Elected Miner pool then double spending of a given transaction is impossible once the Consensus Height has reached or exceeded that transaction's block. If all of the Elected Miner pool remains online and mining blocks then transactions will be finalized within approximately 65 seconds (>50% * 25 Miners * 5 second blocks = 65). All valid transactions will be included in blocks within 10 seconds (5 second blocks * 2 = 10), and if all parties involved are trustworthy then that transaction may be actionable by the recipient with confirmation by just a single Miner (5s avg, 10s max). For guaranteed finality when dealing with untrusted parties the recipient should wait for 100 "confirmations", which should take about 65 seconds. For comparison, pure Proof-of-Work and Proof-of-Stake systems *never* provide guaranteed transaction finality without centralized checkpointing.

Blockchain Fork Resolution

As described above, there is a Consensus Height (*CH*) which increases as new blocks are mined. No node ever accepts a blockchain fork that goes deeper than the current *CH*; the CH is effectively a dynamic checkpoint that is voted into existence by the Miners. When there are two competing forks above the *CH* then the fork with the most unique miners' participation wins. If, for example, fork *F1* is 10 blocks long but only 2 unique Miners made those 10 blocks, while fork *F2* is 8 blocks long with each block being mined by a unique Miner (8 unique Miners) then *F2* wins. If both forks have the same number of unique Miners participating, then the longer fork wins.

If an Elected Miner is evaluating two forks and that Miner is able to mine a new block on either fork, then it will do so prior to comparing the forks.

On initial blockchain sync a node may be poisoned by a malicious peer with an invalid blockchain, and the rule that states that reorgs cannot be deeper that the current *CH* may prevent the node from ever finding the true blockchain. If this happens then the node will need to first connect only to a node on the correct blockchain, and after it finds a true *CH* then the node can openly connect to any peer.

Votes by and for Delegates are only counted once the transaction that cast them has reached 100 confirmations (full consensus), with the exception of Miner selection. When a Delegate

changes its vote for Miners those votes are counted instantly and if the Elected Miner pool is changed as a result then that change applies to the next block.

Network Idling

In the earlier years of any cryptocurrency network there may be long periods of zero transaction activity. As layer-2 adoption grows that will also decrease the level of on-chain activity required for layer-2 supporting networks. In order to avoid the senseless creation of empty or useless blocks the new FLASH network supports idling. All incentives for Miners to mine blocks as fast as possible have been removed, and nChainWork no longer exists, so the network can simply stop creating blocks when there is no longer a need for them. A good deal of bandwidth and index space is saved by idling, and it also keeps blockchain syncing as efficient as possible.

In order for the network to idle without negatively affecting users, the Miners must continue mining blocks at the normal 5 second rate until there are no transactions above the Consensus Height. If the network is idle and then a single transaction is broadcasted then the Miners will begin mining blocks immediately, and they will continue for approximately 13 blocks until that transaction has reached full consensus, and then the Miners will go idle again until the next time a valid transaction is broadcasted.

Miner and Delegate Reward Mechanics

Every day will start with a transaction that distributes earnings to Miners and Delegates. Earnings are derived from network transaction fees - all fees over the last day are added up, split in half so that 50% goes to Miners and 50% goes to Delegates, and then they are allocated as described below.

The 50% for Delegates is allocated based on each Delegate's vote-weight at the time of the reward tx's creation. The greater the vote-weight of a Delegate the greater their share of the reward.

The 50% for Miners is allocated equally across Elected Miners.

Foundation Rewards

The Third Millennium Foundation pledges to donate 32,000 FLASH to the Miners and Delegates every day in the form of transaction fees. This will be done by using an automated process to send one transaction a day from the FLASH Foundation's wallet with a 32,000 transaction fee. These donations will continue until SOME_EXPIRATION_DATE.

Voting Related Opcodes

The voting system requires sending signed voting metadata on the blockchain for all users to see. In order to maintain backwards compatibility with all tools built for Bitcoin, this metadata is sent using the standard OP_RETURN Bitcoin opcode in the script of a zero-value tx output.

Bitcoin tools do not parse OP_RETURN data, nor is this data stored in memory, only on disk. FLASH implements several new voting opcodes which are only used after an OP_RETURN opcode, and these new opcodes tell the parser how to interpret the metadata provided. List of new opcodes:

- **OP_REG_DG:** Delegate registration and information updates
- **OP_STAKE_DG:** Provide additional stake for Delegate
- **OP_VOTE_DG:** User vote for Delegate
- **OP_DG_VOTE:** Delegate vote on a network related matter
- **OP_DG_SIGN:** Used in coinbase tx to prove block creator' s identity

These new opcodes must be used in a zero-value txout script immediately after an OP_RETURN opcode, and this zero-value output must be the first txout (position 0) in a transaction. Only the first txout is parsed for voting opcodes, and only if that first txout starts with OP_RETURN.

Special Election Address

This is a valid FLASH address which has special rules for both receiving and sending transactions. Coins being staked for Delegates are sent to this address, as is Delegate registration information. All transactions into and out of this address must pay the normal network transaction fees. Nobody will ever know the private key for this address, and even if it were somehow derived, the network only allows coins controlled by this address to be returned to the specified return address or the original sender, so they cannot be stolen.

Receiving rules:

- There can be one or many tx inputs. If there is more than one input then a return address must be specified in the voting metadata.
- The first OP_REG_DG for a given delegateID must send at least 1,000,000 FLASH to this address. Any additional stake must be at least 100,000 FLASH.
- If there is one tx output then it must be an OP_REG_DG operation that is updating an existing Delegate
- If there are two tx outputs then the second must send the coins being staked to the special election address and the first output must be an OP_STAKE_DG operation adding to an existing Delegate's stake or an OP_REG_DG operation that is registering a new Delegate. Operation-specific metadata is provided after the opcode.

Sending rules:

- Anyone can spend these coins without the special election address' privkey
- All transactions received by this address have a delegateID that they are being staked for. That delegateID must not be a currently Elected Delegate; coins staked for the Delegate can only be reclaimed if that Delegate did not win the election or they have completed their withdrawal from office.

- For each tx input being spent the output address must match the return address in the input's corresponding metadata, or if not provided then the output address must match the original sending address (return to sender).

Special Voting Address

This is a valid FLASH address which has special rules for receiving transactions, and rejects all sending transactions. When a Delegate casts a vote they send a transaction to this address with the vote data and they pay a transaction fee for sending the tx. No coin is ever deposited in this address, it is simply used to make it easier to account for votes. Monitoring this address will effectively monitor all votes by Delegates.

Receiving Rules:

- There can only be one output tx which uses the OP_RETURN opcode followed by the OP_DG_VOTE opcode and data
- The OP_DG_VOTE data must validate successfully

Sending Rules:

- No transactions sent to this address can ever be spent

The opcode OP_DG_VOTE is explained in a later section, as are the details relating to how a Delegate casts votes and how the votes are counted.

Election Registration Form

To register for election as a Delegate a transaction with at least 1,000,000 FLASH must be sent to the special election address, along with an OP_RETURN txout with the opcode OP_REG_DG which includes hex-encoded serialized JSON data like:

```
{
        delegatePubkey: [full pubkey],
        infoVersion: [number],
        displayName: [string],
        enabled: [boolean],
        mining: [boolean],
        auditURL: [URL],
        contacts: [  // optional
                {
                        type: [email/chat/IRC/URL/whatever],
                        address: [address]
                },
                {
                        type: [email/chat/IRC/URL/whatever],
```

```
                address: [address]
            }
        ],
        website: [URL]  // optional,
        registeredTime: [int],  // timestamp of first ever registration
}
```

The registration JSON document must be less than 4Kb when encoded. An example of the script in a txout[0] which is registering a Delegate:

```
OP_RETURN OP_REG_DG [regData] [delegateSig]
```

A Delegate can update this form at any time by simply providing a new registration form with the updated data and an incremented 'infoVersion' field. No additional stake is required for updates to this form as long as enough coin is already staked for this Delegate to meet the minimum requirement of 1,000,000 FLASH.

Explanation of the fields:

- **delegatePubkey:** This is the full pubkey that identifies the Delegate. All future actions taken by this Delegate will be signed with this pubkey's corresponding private key.
- **infoVersion:** This is a number that is incremented with every update of this Delegate's registration form to ensure proper state. Without this any delays in tx replication or chain reorgs might cause an older version to overwrite a newer version.
- **displayName:** This is intended to be a human-readable string that will appear in various user interfaces for identifying this Delegate, in addition to the delegateID.
- **enabled:** If set to true then this delegateID will be eligible for election. If set to false then it will not be. When an Elected Delegate wishes to withdraw from their duties then they must set this to false and await the end of the current election term, and then they can reclaim their staked coin.
- **mining:** When set to true then this Delegate is signalling that they want to be a miner, so they become eligible for election as a Miner. If set to false then this Delegate is not eligible. If this Delegate is already an Elected Miner they can set this field to false to withdraw from mining. If registering as a potential Miner then a minimum of 2,000,000 FLASH must be staked.
- **auditURL:** This is the URL which exposes this Delegate's audit interface. There is a new API command which when queried returns information that can be used to prove this node is alive and to measure certain performance characteristics. It is recommended that a service like memcached be used when hosting this URL to shield the node from excessive queries. Auditing is explained in further detail in a later section of this document.

- **contacts:** This is a JSON array which contains a list of contact objects. Each object specifies the type of contact address being provided and the address itself. This is intended to provide a way for the community to contact the owner of this Delegate. This is optional; however the community is unlikely to vote for a Delegate that they know nothing about.
- **website:** This is a URL linking to a website that represents this Delegate or the organization that owns the Delegate. This is optional; however the community is unlikely to vote for a Delegate that they know nothing about.
- **delegateSig:** This registration form can be sent from any address, it doesn't have to be from the Delegate directly, however the 'regData' JSON object must be signed with the private key corresponding to the delegatePubkey, and that signature must be placed in this field.

When the registration data is received the delegateID is generated by converting the delegatePubkey into a FLASH address.

Staking for Delegates

Any address may contribute stake to any Delegate, however no coins may be staked for any Delegate that has not registered, and the first registration must include at least a 1,000,000 FLASH stake. The first registration stake must be included with the registration transaction, but further stake contributions can be provided by sending the coins to the special election address with the OP_STAKE_DG opcode. The staking transaction must have two outputs with the second sending the coin to be staked to the special election address (minimum of 100,000 FLASH), and the first output must have zero value and use the OP_RETURN opcode followed by the OP_STAKE_DG opcode followed by the delegateID being staked for, and optionally followed by a return address for the staked funds. If no return address is specified then the funds can only be returned to the sender.
Example:

OP_RETURN OP_STAKE_DG [delegateID] [returnAddress]

It should be noted that staked coins will not be able to be returned until the Delegate they are being staked for has withdrawn from office and has ended their current election term. To stake coins for a Delegate is to surrender control of them for as long as that Delegate is running for election and serving their term(s).

Voting for Delegates

Normal users vote for Delegates by prepending a zero-value tx output to their transaction which uses the OP_RETURN opcode followed by the OP_VOTE_DG opcode, followed by the list of delegateID's to vote for with each transaction output. Votes cast by the tx's inputs are destroyed and votes are created with the new tx outputs. The OP_VOTE_DG opcode expects a

simple list of delegateID's, one for each txout (not counting the first txout which contains this metadata) and the entirety of each tx output's coin amount is used to vote for the Delegate described by this array. Example:

OP_RETURN OP_VOTE_DG [delegateID1] [delegateID2] [delegateID3]

The first delegateID in the OP_VOTE_DG list receives the vote of txout[1], the second delegateID in the list recieves the vote of txout[2], and so on. If there are more txouts than delegateID's provided, then the txouts that are missing a corresponding delegateID are simply not counted in the vote. If a delegateID provided in the OP_VOTE_DG list is not valid then the entire transaction is rejected.

Voting by Delegates

Delegates vote by sending a transaction to the special voting address. Use of this address makes it easy to account for how and when Delegates vote, and this address has special rules as described in the Special Voting Address section above. The transaction casting the vote does so by having a single tx output that uses the OP_RETURN opcode followed by the OP_DG_VOTE opcode and data describing the vote.

Votes are described using a JSON object which is passed to the OP_DG_VOTE opcode. Each key in the voteData object represents a ballot item, and each value indicates the vote itself. Each value in the voteData JSON object has a maximum size limit of 2Kb and the entire voteData object has a max size limit of 100Kb. Given the 900M coin supply and the 1M minimum Delegate registration stake there can never be more than 900 Delegates in the running, and therefore the mapDelegateState variable that holds voting information can never be more than approximately 90Mb. In practice it ought to be an order of magnitude smaller.

The opcode OP_DG_VOTE expects three arguments:

OP_RETURN OP_DG_VOTE [delegateID] [voteData] [voteSig]

The 'voteSig' argument is a signature of the 'voteData' argument, signed with the private key that corresponds to the delegateID. The 'voteData' argument is a hex-encoded serialized JSON document as follows:

```
{
        voteHeight: 12527,
        minTxFee: 30000000,  // satoshis
        txFeePerByte: 110, // satoshis
        miners: [
                delegateID3: 50,  // 50% of vote-weight
                delegateID14: 25,  // 25% of vote-weight
                delegateID31: 25,  // 25% of vote-weight
        ]
}
```

Explanation of the keys and values:

- **key:** The key in this JSON object represents the ballot item for which this vote is being cast. Each time a tx is sent to update the voteData the update is considered incremental - it is applied on top of the existing voteData for that Delegate, overwriting any existing values for given keys. Setting a key's value to 'null' will delete the key and the value, thereby withdrawing the vote.
- **value:** This is the value of the vote being cast. The value can be no more than 2Kb in size. New values for pre-existing keys will overwrite the old value.

Any transaction using the OP_DG_VOTE opcode with a JSON document that differs from the format described above, or which violates any of the constraints described, is considered invalid and is discarded. The limitations on the size of the vote data prevents DoS and spam attacks, and the overall structure of the voting document allows great flexibility for future ballot items. OP_DG_VOTE transactions must pay the same per-byte transaction fee as every other tx on the network.

The special key 'voteHeight' in the voteData is incremented each time a Delegate updates its vote. This allows the network to apply these vote updates to mapDelegateState in the correct order. If a Delegate attempts to provide an update with voteHeight 123, for example, without previously providing update with height 122, then the update for 123 is rejected. If a second update for 123 is received then it is rejected. If a different txid that claims to be update 123 is included in a block after a reorg then the first update 123 is reversed and the second update 123 is applied; the mapDelegateState data must be kept in sync with the validated votes in the blockchain.

Vote Accounting and Indexing

This voting system has been carefully designed to be as efficient as possible, to add as little overhead on the blockchain and node processing requirements as possible, in order to facilitate the highest transactions-per-second rate possible. In addition to performance considerations,

it is ideal to maintain full compatibility with all Bitcoin related tools, so this voting system has been designed to maintain that backwards compatibility.

Every vote cast, every stake made, and every Delegate registration is performed via a tx output in the first output position which uses the OP_RETURN opcode followed by a new voting opcode that is unique to FLASH, followed by the voting metadata. Any Bitcoin-based script interpreter will not parse the data following the OP_RETURN opcode, but the FLASH script parser will continue reading the script if there is a voting opcode immediately after the OP_RETURN opcode. Requiring that this voting metadata be in the first tx output position (txout[0]) assures efficient lookups when discovering old votes that are being destroyed by new tx outputs. It also increases OP_RETURN script parsing by skipping the parsing of any OP_RETURN data that is not in the txout[0] position.

To facilitate accounting and indexing in the core code some new variables and structs will be created:

- 'Delegate' class: This holds information about a Delegate and methods for polling and interacting with Delegates
- 'mapDelegateState' : This is a delegateID-indexed map of Delegate objects
- 'VoteState' struct: this holds the current ruleset, the result of the vote, derived from mapDelegateState
- 'PendingVote' struct: This is a struct that contains information about a vote that is pending full consensus before being counted. Attributes include blockheight, txid, and interpreted vote data.
- 'mapPendingVotes' : This is a txid-indexed map of PendingVote structs, used for accessing PendingVotes by txid. If a tx is removed due to chain reorg then that tx is removed from this map as well as the multimap mentioned below.
- 'mmPendingVotesByBlock' : This is a blockheight-indexed multimap of vectors of txid's. When the Consensus Height is updated this multimap is scanned to find all PendingVotes that are now fully confirmed and then their votes are counted. Once all transactions in a discovered vector are counted, all corresponding txid's are removed from mapPendingVotes and then the vector is removed from this mmPendingVotesByBlock.

Given a max coin supply of 900M and a minimum Delegate registration stake amount of 1M, there will never be more than 900 entries in the mapDelegateState variable. When a Delegate reclaims their stake their entry in mapDelegateState will be deleted.

Upon receiving a valid Delegate registration transaction each node will check to see if this new delegateID already exists in mapDelegateState, and will add a new struct for it there if it does not exist. When receiving valid registration updates each node will overwrite the given

delegateID's entry in mapDelegateState with the new struct if that the new struct has a higher infoVersion number than the existing entry.

As each new block is accepted into the blockchain each transaction in the block is checked for voting metadata in txout[0], and if present it will be validated and noted in mapPendingVotes and mmPendingVotesByBlock. Votes are not counted until the Consensus Height has reached or surpassed that transaction's blockheight. Every tx has inputs and outputs; any votes previously cast by the inputs must be subtracted from mapDelegateState and any votes cast by the outputs must be added.

Each time the Consensus Height changes, which might happen with every new block, mmPendingVotesByBlock is iterated and each entry with a blockheight at or below the new Consensus Height is processed and removed from both the multimap and the map. Processing mmPendingVotesByBlock means to fetch each transaction in the vector being processed, parse the voting metadata, and update mapDelegateState to count the new votes. If mapDelegateState is changed after processing the new Consensus Height change then all votes in mapDelegateState are re-counted and the global VoteState struct is updated to index the current active ruleset. Future blocks will be validated against this updated ruleset. In this way a running total is maintained for all votes in mapDelegateState, updated with each Consensus Height change, and the result of the vote is indexed in structVoteState so that all parameters can be easily and efficiently looked up.

Note: changes to a Delegate's Miner vote is processed instantly when received by a node, but all other changes must wait for full finalization by allowing the Consensus Height to reach that vote's block.

Calculating Vote Results

The keys in the voteData object are arbitrary however there are several keys which will be utilized by default when the FLASH network is launched. Other keys will be ignored. Each key in voteData can use a different method for determining the outcome of the vote, and that method is hardcoded in the FLASH governance code. Here are explanations of the voting method for each of the voteData keys supported at launch:

- **voteHeight:** This is a number that is incremented each time a Delegate updates their vote. Sequentially ordering vote updates assures that the updates are applied to mapDelegateState in the correct order.
- **minTxFee:** This is the minimum transaction fee allowed for any tx, denominated in satoshis. The outcome of the vote is the weighted median of all Delegate votes, weighted with their voteWeight.

- **txFeePerB:** This is the tx fee per byte of the transaction size, denominated in satoshis per byte. The outcome of the vote is the weighted median of all Delegate votes, weighted with their voteWeight.

- **miners:** The outcome of this vote is an array of the top 25 Miner delegateID's, sorted by cumulative voteWeight from every Delegate. The voteWeight contribution from a Delegate is that Delegate's total voteWeight multiplied by the percentage that the Delegate gave to this Miner.

Coinbase Transaction and Delegate Signatures

Every block mined will have a coinbase transaction with 0 inputs, similar to most Bitcoin-derived cryptocurrencies, however FLASH's coinbase transaction will have a zero-value txout[0] that uses the OP_RETURN and OP_DG_SIGN opcodes as follows:

OP_RETURN OP_DG_SIGN [delegateID] [delegateSigOfBlockhash]

This txout[0] is proof that the given delegateID was the creator of the block.

Once every day the coinbase transaction will contain additional outputs to distribute the previous day's rewards to Miners and Delegates. These rewards will be distributed as described in the "Miner and Delegate Reward Mechanics" section above. As each day progresses and tx fees are paid, those fees are burned, they are temporarily removed from the coin supply. The first block of every day recreates those burned coins and distributes them to the Miners and Delegates.

Audit Interface and URL

There will be a new API command called **getauditpoint** which can be used by the community to verify that this Delegate is online with a blockchain that is in-sync. This command takes no arguments. When users are choosing a Delegate(s) to vote for from a list, that list ought to show the sync-state of each Delegate so that the user has information to inform their vote. The response from the getauditpoint API call will follow the standard Bitcoin API format and might look like this:

```
{
        error: null,
        result: {
                auditPoint: {
                        delegateID: [pubkey],
                        blockHeight: [chain-tip height],
                        blockHash: [chain-tip blockhash],
                        mempoolSz: [number],
```

```
                timestamp: [current time]
            },
            delegateSig: [base64 signature with privkey]
        },
        id: 0
}
```

Each call of this API will generate a fresh signed message with the node's current timestamp and chain-tip information. Delegates that are currently elected or are running for election need to make this information publicly accessible. It is recommended to use a separate caching proxy in front of this API call to protect against DoS and spam attacks, and to hide the Delegate's true IP from the public, if desired.

Delegate Info Interface

There will be new API command called **getdelegateinfo** which returns a JSON object revealing information about every Delegate, both elected and running. The results of this API command will be used for dashboard displays on community websites and in the Qt wallet. This command has one optional argument: delegateID. Specifying a delegateID will filter the results to only return voting status for the specified Delegate, otherwise data for all Delegates will be returned. The data provided in this API command's response is derived from the mapDelegateState variable. This information can be used to see information about all Delegates, including which delegates are voting in which direction and with how much influence. Example:

```
{
        error: null,
        result: {
            delegateCount: 326,
            votesCast: 235486723.12345678,
            totalVotesPossible: 900000000.00000000,
            totalVoteWeight: 789437432323.12345678,
            delegates: {
                "Sojgf3w40FSj9fw92jgFSmbZAdqT2" : {  // key is delegateID
                    infoVersion: 123,
                    displayName:  "This Delegate" ,
                    enabled: true,
                    mining: true,
                    auditURL:  "http://1.2.3.4:80/audit" ,
                    contacts: [
                        {
                            type:  "email" ,
```

```
                    address:  "contact@delegate1.com"
                }
            ],
            website:  "http://delegate1.com" ,
            registeredTime: 1567234635883,
            lastDGVoteTime: 164264529832,
            votesRcvd: 1324523,
            stake: 2000000,
            timeInOffice: 1356542,
            voteWeight: 5673561,
            voteData: {
                minTxFee: 33000000,  // satoshis
                txFeePerByte: 100,  // satoshis per byte
                miners: [
                    delegateID3: 50,  // 50% of vote-weight
                    delegateID14: 25,  // 25% of vote-weight
                    delegateID31: 25,  // 25% of vote-weight
                ]
            }
    },
      "USgs39sdVnkdpa30SDmOP353zvc4" : {  // key is delegateID
            infoVersion: 234,
            displayName:  "Another Delegate" ,
            enabled: true,
            mining: true,
            auditURL:  "http://2.3.4.5:80/audit" ,
            contacts: [
                {
                    type:  "email" ,
                    address:  "contact@delegate2.com"
                }
            ],
            website:  "http://delegate2.com" ,
            registeredTime: 165234635812,
            lastDGVoteTime: 164264523825,
            votesRcvd: 526272,
            stake: 1000000,
            timeInOffice: 356548,
            voteWeight: 2673564,
            voteData: {
                minTxFee: 33000000,  // satoshis
```

```
                                txFeePerByte: 100,  // satoshis per byte
                                miners: [
                                        delegateID4: 20,  // 20% of vote-weight
                                        delegateID63: 65,  // 65% of vote-weight
                                        delegateID17: 15,  // 15% of vote-weight
                                ]
                        }
                },
        }
},
id: 0
}
```

Vote State Interface

There will be new API command called **getvotestate** which returns a JSON object revealing the current state of all ballot items as accounted for in the structVoteState variable. This is the law for the current block. Example:

```
{
        error: null,
        result: {
                minTxFee: 33000000,
                txFeePerByte: 100,
                miners: { // map of elected miners
                        delegateID4: true,
                        delegateID63: true,
                        delegateID17: true
                }
        },
        id: 0
}
```

This API command and structVoteState are intended to only show the result of all votes as of the previous block's closing, which makes these results the ruleset that will be applied to the next block generated. If more details are needed about weights and Delegates then the **getdelegateinfo** API command should be used instead.

Voting User Interface

Both the Qt wallet and the CLI will provide interfaces for voting. The wallets will have configuration parameters that can be set to control automated voting with coins held in the wallet, as well as interfaces that show the current state of this wallet's voting power. Because

votes are cast as coins are spent, not as they are held, the automated voting will allow the user to set a threshold level for the amount of coin in the wallet that is not voting as specified, and when that threshold is exceeded then the automation will send that coin back into the wallet with the correct voting preference set. The voting automation will by default combine all transactions into one output when resending to self, unless the user has opted out of tx combining.

Miner and Delegate Statistics

Data can be collect for both Miners and Delegates from their published auditURL's for monitoring how well synced they are. All Delegates and Miners should rapidly sync the current chain tip and maintain roughly identical mempool tx count. The auditURL provides a signed message showing this information from each Delegate and Miner.

Additional Miner statistics can be collected from the blockchain itself, by analyzing the coinbase transaction to identify which Miners are participating properly. Elected Miners should never miss their block generation timeslot unless the network is idling, and if a Miner misses too high a percentage of their timeslots then they should be replaced with a more reliable Miner.

By gathering the necessary information and making it accessible to all users it becomes possible for the community to make informed decisions about Delegate and Miner elections, and well-informed decisions will lead to an optimally efficient network.

FLASH Web Wallet - Account Structure + Key Generation, Storage and Recovery

ACCOUNTS IN THE FLASH WEB WALLET ARE STORED IN A CENTRAL AUTHENTICATION SERVER (CAS).

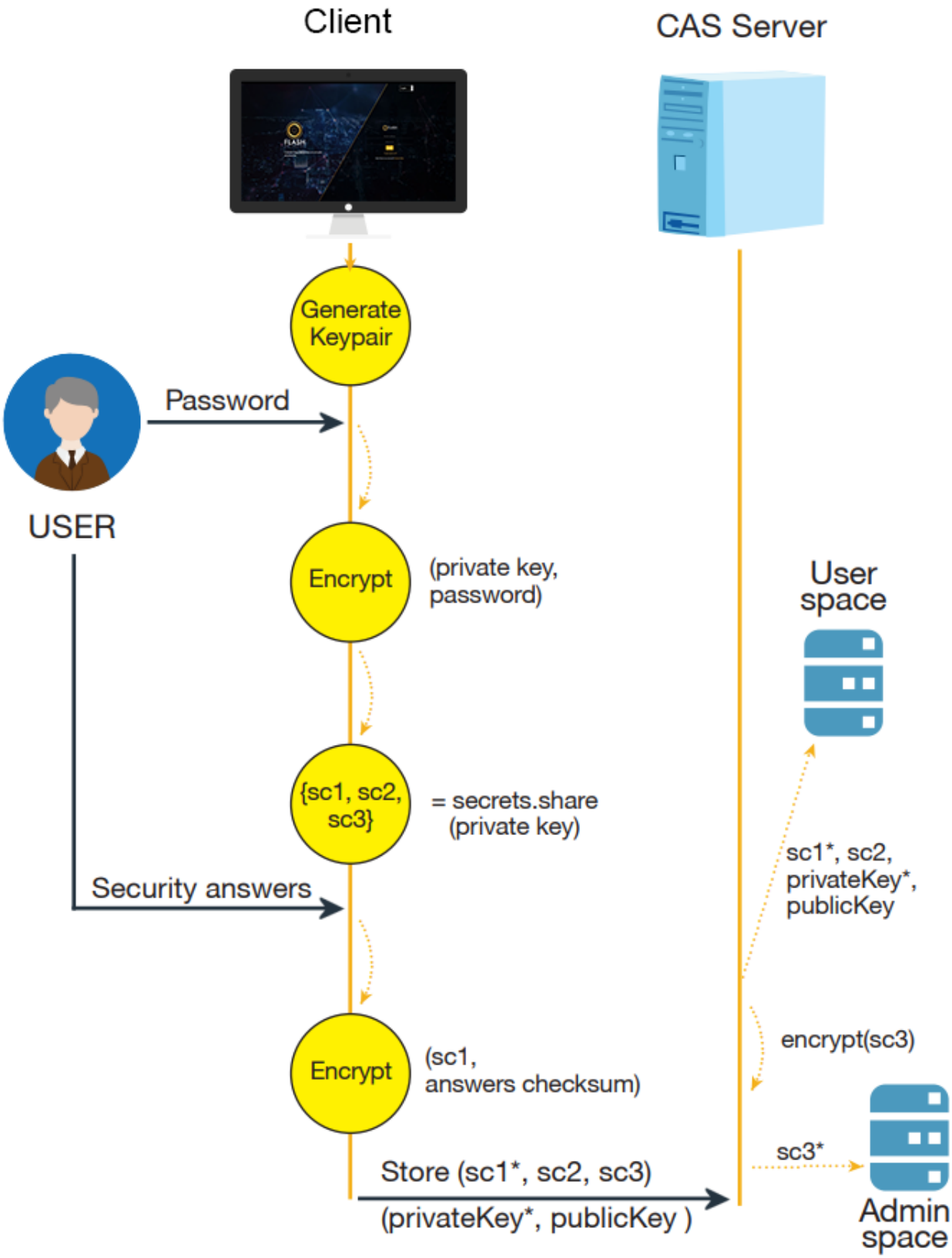
Each CAS account has the following fields:

- **id:** account ID.
- **email:** account's email
- **role:** used for authorization, e.g: USER or ADMIN
- **privateKey:** EC crypto private key (encrypted by user's password)
- **publicKey:** EC crypto public key.
- **sc1:** user's share used in recovery procedure (encrypted by user's security answers).
- **sc2:** server's share used in recovery procedure
- **sc3:** administrator's share used in case the user lost sc1 and cannot recover himself.

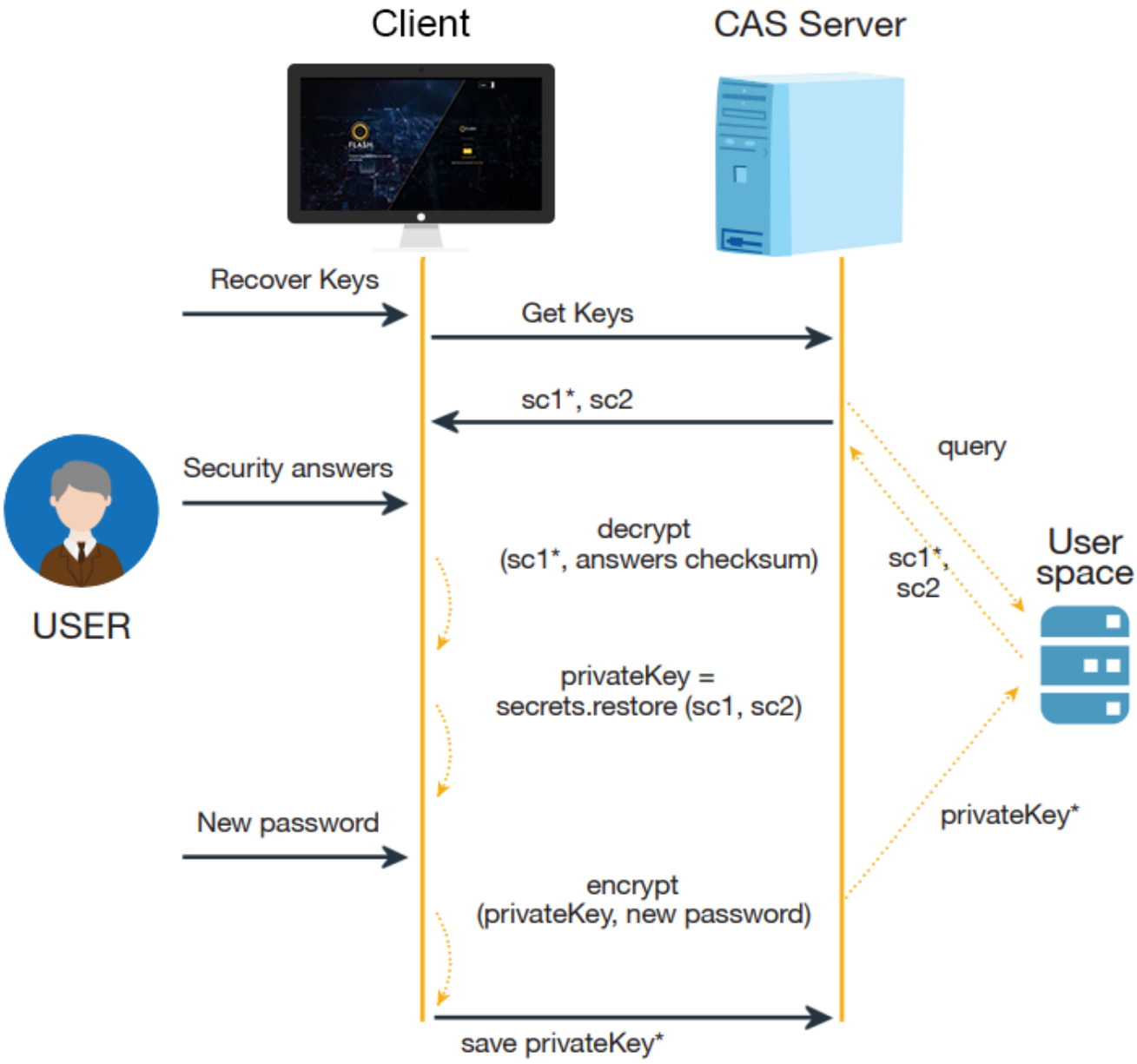
In addition user's profiles are stored in Flashcoin Key Server. The information includes: display name, avatar, country… which varies from app to app.

FLASH Web Wallet Key Generation, Storage and Recovery

Key generation at signup	EC crypto keypair is generated at client side when signing up. The private key is then encrypted by user’ s password. Recovery keys are also generated from the private key, which is a tuple (sc1, sc2, sc3). After user answers the security questions, the responses are then used to encrypt sc1.



The server stores the following: the encrypted private key, public key, sc1 encrypted, security questions, sc2, sc3 (which is then encrypted separately by the admin)



Key Recovery

The recovery process is automatically triggered by the user. After the email is verified, the client receives sc1 encrypted and sc2, security questions. By answering the security questions correctly, the answer is used to decrypt sc1. From sc1 and sc2, the private key is restored. User then needs to provide a new password and start the process of protecting and storing keys following the same as above.

Users can also choose a super-secure mode where the server stores a unique sc. When the recovery mode is activated, user must provide his/ her share to combine with the server's share. If user lost the sc1 (given in the sign-up process) then no one can recover his/her password. Therefore, the compulsory participation of user in the recovery process ensures the security of user's share as well as the password.

FLASH Blockchain

FLASH HAS FORKED THE LITECOIN VERSION OF THE BLOCKCHAIN TO USE AS A DISTRIBUTED NETWORK STORAGE SYSTEM. A NUMBER OF SIGNIFICANT MODIFICATIONS TO THE CODE HAVE BEEN MADE IN ORDER TO ADDRESS THE WEAKNESSES OF FULLY DECENTRALIZED NETWORKS:

- **Network Latency** - Block synchronization among the nodes dramatically slows down the transaction validation (double spending) speed. FLASH is a distributed network with mining nodes limited by the Delegates who vote for the miners. Because of the delegated and limited number of mining nodes, the network latency should be under 400ms (propagation time window).

- **Performance** - Two factors that determine the blockchain performance include Block Synchronization and Block Mining. FLASH uses cache and index servers to synchronize the nodes and reset the mining algorithm to the least difficulty factor. Because we have the trusted network of nodes, there is no need to continue to increase the degree of difficulty of mining for the block verification process. FLASH provides a unique solution to ensure distributed network storage data integrity.

- **Security Risk** - Block mining is vulnerable with a fully open distributed blockchain network via 51% and other attacks. The FLASH Blockchain is not open to the public to mine or manipulate the blockchain by computational advantage. This task is maintained by Miners selected by the Elected Delegates.

End to End Encryption

In order to ensure no single point of weakness, the design of the security system and of all encryption functions have to be done from the web wallet (client node).. As a result, transactions are encrypted by the recipient's public key which are then written into the FLASH blockchain. This methodology protects from intruders obtaining any encrypted data on the FLASH database. In order for any attacker or intruder to decrypt information, they must compromise the system and crack the Elliptic Curve

Cryptography (ECC) algorithms are used on each key. Even if someone had a quantum computer and was able to crack ECC the cost to decrypt a transaction would by far outweigh the possible gain. For an average computer it would take more than 100 Billion years in computation effort, according to most experts. Therefore, the cost of cracking the ECC on each transaction far exceeds the potential return.

Blockchain API

A protocol that empowers Web Application to communicate with the FLASH BlockChain network. All transactions have been indexed at the Blockchain API layer to pre-compute and speed up transaction lookups such as double spending verification and transaction logs.

APPENDIX

Wallet Webservice API

Create Account

Name: create_unverified_account

Description: Create unverified account (need to verify via email)

Request params: name, email, ip, callbackLink, g_recaptcha_response

(Google recaptcha response)

Response: {rc: Number}

Set Password and Verify Email

Name: set_password

Description:

Request params: password, privateKey (encrypted private key),

publicKey, token

Response: {rc: Number}

Get Session Token (sso)

Name: get_session_token

Description:

Request params: idToken, resource

Response: { rc: Number, profile : Object { sessionToken: String } }

Check Session Token (sso)

Name: check_session_token

Description:

Request params: sessionToken, resource

Response: {rc: Number, profile: Object{username: String, email: String} }

Login (sso)

Name:

Description:

Request params: email, password, ip, resource

Response:

Success: {rc: Number, profile: Object{email: String, display_name: String, gender:

String, ...} }

Update Account

Name: update_account

Description: Update user profile

Request params: display_name, gender, profile_pic_url, about, timezone ...

Response: {rc: Number }

Get Profile

Name: get_profile

Description: Get user profile

Request params: {}

Response: {rc: Number, profile: {username: String, email: String, display_

name: String, profile_pic_url: String ⋯} }

Set PIN

Name: set_pin

Description: Set PIN

Request params: pin

Response:

Success: {rc: Number}

Check PIN

Name: check_pin

Description: Check if PIN is correct

Request params: pin

Response:

Success: {rc: Number}

Change PIN

Name: change_pin

Description: Change the PIN

Request params: old_pin, new_pin

Response:

Success: {rc: Number}

Get Contact Details

Name: get_contact_detail_by_email

Description: Get contact details by email

Request params: contact_email

Response:

Success: {rc: Number, profile: {username: String, email: String, display_

name: String, gender: String, profile_pic_url: String, ⋯} }

Get Profile

Name: get_profile

Description: Get user profile

Request params: {}

Response: {rc: Number, profile: {username: String, email: String, display_

name: String, profile_pic_url: String ⋯} }

Get Users

Name: get_users_by_uid

Description: Get users information by user id

Request params: [‘user1’ , ‘user2’ , ⋯]

Response:

Success: {rc: Number, accounts: [account1, account2, ⋯] }

Get Roster

Name: ros_get

Description: Get contact list of a user

Request params: {}

Response:

Success: {rc: Number, roster: {total_subs: Number, subs: [], ⋯}

Roster Operation

Name: ros_op

Description: operate roster, where operation could be REQUEST, APPROVE, REMOVE

Request params: op, from, to

Response:

Success: {rc: Number}

Notification: notify to related users

Create Wallet

Name: create_flash_wallet

Description: Create a new wallet

Request params: idToken, wallet_secret

Response:

Success: {rc: Number, wallet: {passphrase: String, wallet_id: String, address: String }

Search Wallet

Name: search_wallet

Description: Search for wallet by keyword, to send money to

Request params: start, size, term

Response:

Success: {rc: Number, criteria, wallets: [wallet1, wallet2, ..], total_wallets: Number }

Get My Wallets

Name: get_my_wallets

Description: Get my wallets (currently only support 1 wallet)

Request params: {}

Response:

Success: {rc: Number, my_wallets: [], total_wallets: Number}

Add Transaction

Name: add_txn

Description: Push transaction to blockchain and add transaction log

Request params: receiver_id, amount, currency_type, receiver_public_address, transaction_id, memo, request_id, transaction_hex (signed) Response:
Success: {rc: Number, id: String}

Notification: notify to the recipient about the new transaction

Get Transactions Log

Name: get_txns

Description: Get transaction log of current user

Request params: date_from, date_to, order, start, size

Response:

Success: {rc: Number, txns: [tx1, tx2, ...], total_txns: Number}

Get Transaction Log By Id

Name: get_transaction_by_id

Description: Get transaction detail by id

Request params: transaction_id

Response:

Success: {rc: Number, txn: {...} }

Create Unsiged Transaction

Name: create_unsigned_raw_txn

Description: Create a unsigned transaction to be signed by the owner later

Request params: from_address, to_address, amount

Response:

Success {rc: Number, transaction: {...} }

Get Transaction Details

Name: get_transaction_details

Description: Get transaction details from blockchain

Request params: transaction_id

Response:

Success: {rc: Number, transaction: {...} }

Get Balance

Name: get_balance

Description: Get wallet balance from blockchain api

Request params: {}

Response:

Success {rc: Number, balance: Number}

Add Money Request

Name: add_money_request

Description:

Request params: to, amount, note

Response:

Success: {rc: Number, id: String }

Notification: notify to the requested user

Get Money Requests

Name: get_requests

Description:

Request params: date_from, date_to, status, start, size, type

Response:

Success: {rc: Number, money_requests: [req1, req2, ...], total_money_reqs: Number}

Mark Money Request as Accepted

Name: mark_accepted_money_requests

Description:

Request params: receiver_id, request_id, note_processing

Response:

Success {rc: Number}

Mark Money Request as Rejected

Name: mark_rejected_money_requests

Description:

Request params: receiver_id, request_id, note_processing

Response

Success {rc: Number}

Mark Money Request as Cancelled

Name: mark_cancelled_money_requests

Description:

Request params: sender_id, request_id, note_processing

Response

Success {rc: Number}

Mark Money Request as Read

Name: mark_read_money_requests

Description:

Request params: receiver_id, request_ids: Array<{request_id, sender_bare_uid}>
Response

Success {rc: Number}

Blockchain APIs (in progress)

Push transaction to the blockchain

Name: push_transaction

Description: push a transaction raw format (hexa encoding) to the blockchain

Request params: transaction hex

Response: {}

Send token

Name: send_token

Description: send token (coin) to a wallet identified by public address

Request params: to_public_address, amount, message

Response: {}