Factom Ledger by Consensus

Contributors: Paul Snow, Brian Deery, Peter Kirby, and David Johnston

Technical Advisors: Adam Stradling, Steven Sprague, Jack Lu, Jay Berg, and Shawn Wilkinson

Reviewers: Vitalik Buterin, Charles Hoskinson, Peter Todd and Marv Schneider

Version 1.0 of the Consensus Paper
Release Date: January 17, 2015
www.Factom.org

Factom is a general purpose data layer, secured by the Bitcoin Blockchain. Users can create their own “chains” of data, We call each data packet entered simply an Entry. Factom places no restriction on what that data might be. While what Factom does is no more complicated than that, how Factom creates a censorship resistant, distributed and autonomous network for this purpose is, admittedly, complicated. This paper describes the context and the algorithm behind Factom. The reader will be aided by referring to Appendix A, which provides a glossary of terms.

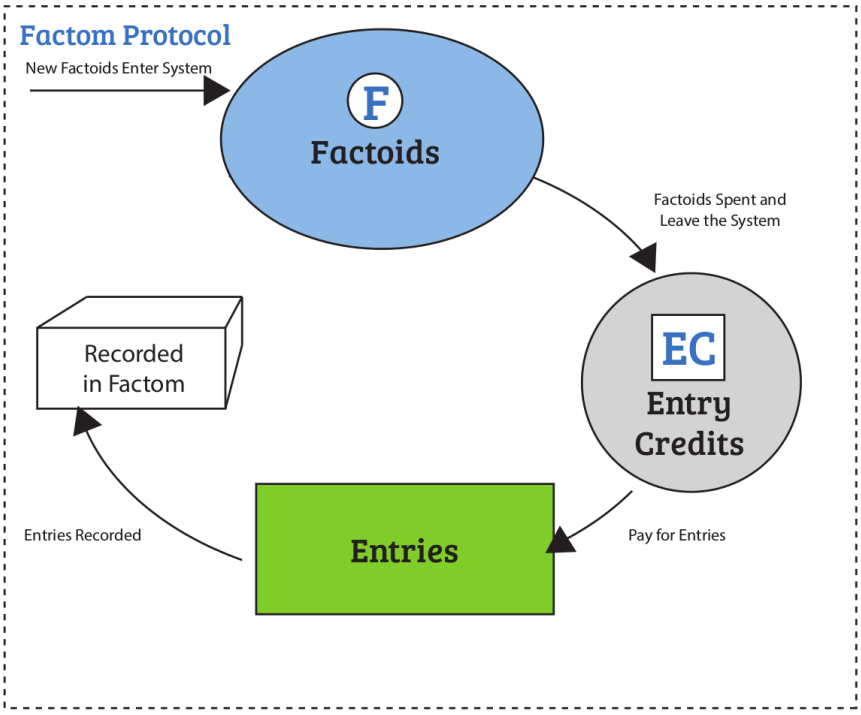
Factom creates a consensus system that ensures Entries are quickly recorded, in the order provided, without centralized control, and without requiring public identities of participants. Factom uses a set of cryptographic protections to ensure the protocol is executed properly. The protocol is also backed up by a user voting system to ensure bad actors are removed from authority where misbehavior cannot be detected in real time from just data within the Factom protocol.

This paper goes over the architecture of Factom, and describes how its components fit together. We discuss a set of attacks, and demonstrate that as long as more than half of the participants are honest, and run the protocol as defined, Factom will perform as needed. Performance slows with increased dishonesty among system actors, however. As actors fail to perform, they are replaced with new actors.

Factoids

Factoids are the token used by Factom creating incentives to support the protocol. Factoids represent the right to obligate the protocol. Anyone holding Factoids can provide access to the protocol by the conversion of Factoids into Entry Credits. Factoids are also payment for work done by the Federated Servers who support the protocol. This system limits spam, since the spammer must purchase the right to publish, forcing there to be a cost to spamming.

Factom Token to Recorded Entry Progression



The implementation of Factoids follows the pattern of cryptocurrencies. Transactions can move value from one address to another via signed transactions. Multi-signature will be supported. Something analogous to UTXO commitments possibly optimized by Patricia-Merkle trees will likely be used as an optimization to support lite clients.

Factoids are tracked on the Factoid Chain in Factom. The Factom servers are capable of differentiating valid from invalid transactions to this Chain, so they can protect it. Only valid Factoid transactions will be placed in the Factoid Chain.

The Factoid chain will be processed by only one Server acting as the leader within a minute. The leader will then move to another server in the next minute, and so forth. Transactions then can be considered processed (at the lowest level of confirmation) upon the confirmation of the leader processing the Factoid chain.

Entry Credits

Factoids are converted into Entry Credits, which are assigned to a public key. This public key can be used to sign Factom Chain Commits, which allow a new Factom Chain to be created, or sign Entry Commits, which will then allow an Entry to be added to a Factom Chain. Entry Credits are tracked and managed on the Entry Credit Chain. The number of Entry Credits required to place an Entry are 1 Entry Credit per 1 KiB (including partial KiB). An Entry that

requires 3 KiB of storage would require 3 Entry Credits. If an Entry Commit pays too few Entry Credits, a note of the problem is posted in the Entry Credit Chain, and the Entry Credits are forfeited. There is a good chance that the Federated server will never see the underpaid transaction, though, since the full nodes will not propagate underpaid Entries.

When an Entry Commit is recorded by the protocol, the obligation to create a specific Entry is created. The obligation to write this Entry lasts for the Entry Credit Timeout Period (ECTP). We are currently considering an ECTP of roughly 24 hours. An Entry Reveal must be done within this period. If not, then after the ECTP, the timeout is noted in the Entry Credit Chain, and the Entry Credits used in the Entry Commit forfeited. This prevents the creation of an ever expanding set of pending Entry Commits that await Entry Reveals that never happen.

Entry Credits are non-refundable, and non-transferable. They are tied to a public key until spent. Unspent Entry Credits might expire over some long period of time (some number of years). We are still considering this. A Merkle tree proving the servers' view of unspent balances will be useful for light clients of Entry Credits.

Factoids are directly converted into Entry Credits. The conversion transaction requires the appropriate signatures (just like any other Factoid transaction) and specifies an Entry Credit public key as the destination, along with other valid outputs. The conversion assigns value to an Entry Credit public key at the current Factoid to Entry Credit exchange rate. The number of Entry Credits per Factoid will be determined by the Federated Servers.

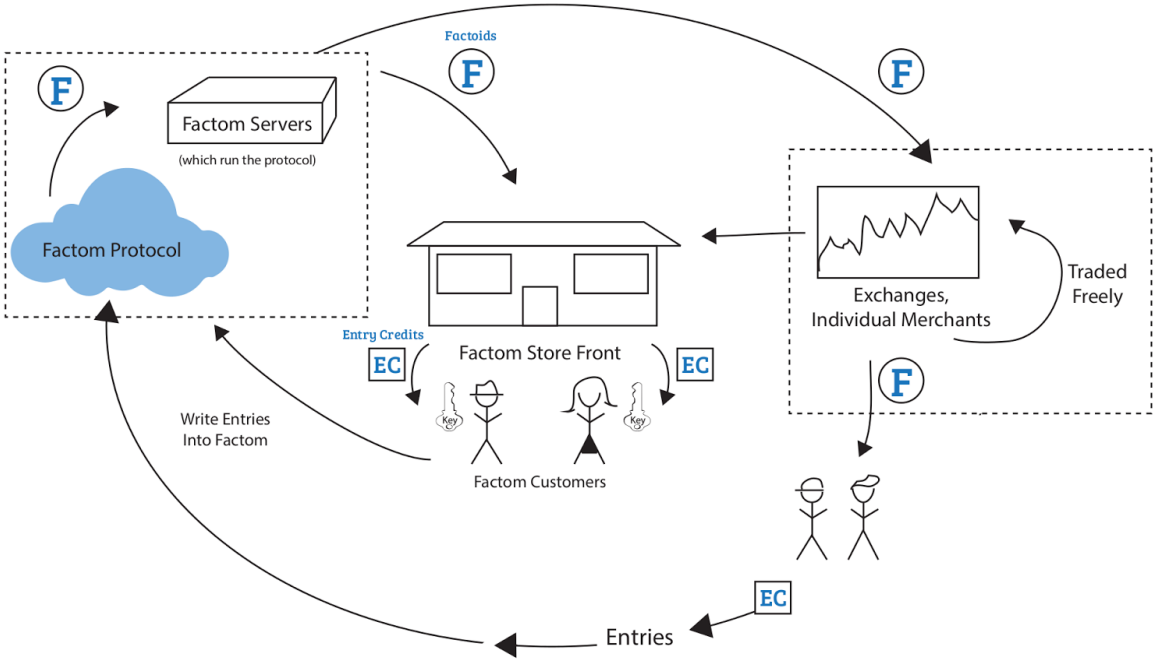
Entry Credits are managed on an Entry Credit Chain in Factom. This Chain is a Protected Chain, much like the Factoid Chain. Invalid entries are not allowed into these chains, a rule enforced by the protocol. Valid entries can be validated by checking signatures, and the audit rules of these chains.

Although Entry Credits can only come from Factoids, different parties can own the different private keys. This allows a storefront to be set up which holds the Factoids, and customers pay by some other method for their Entry Credit keys to be charged.

The Ethereum project has a similar model of value flow. Their tradable Ether is converted to Gas at a variable exchange rate set by the system. The Gas is used up as contracts are executed. Ethereum miners are paid with a fixed amount of Ether, and do not get to recapture the Gas value expended by users.

Factom and Ethereum are somewhat analogous in the multiple token model: Factoids=Ether and Entry Credits=Gas.

Factom Value Cycle



Public Keys in Factom

Users have public keys to sign Factoid transactions, Entry Commitments, and Chain Commitments. If they wish to have a vote in the system, they will create an identity and associate public keys holding Entry Credits with the identity. Factom Servers must have an identity including known Bitcoin keys to post anchors in the Bitcoin blockchain. We will have one public key format for the Entry Credit Chain, and another for the Factoid Chain, to avoid confusion at the user interface.

Identities in Factom

Various entities must establish, maintain, and secure identities within Factom in order to interact. The interactions direct who gets to define consensus. Users of Factom need to establish an identity if they wish to vote on a Factom Server, and potentially receive rewards. The Factom Federated Servers as well as the Factom Audit Servers must establish their identities. The Factom Audit Servers are those servers "waiting in the wings" to become Federated Servers. They have support, but not enough to displace a server in the Federated Server pool.

An identity is constructed within Factom as a Chain. This Chain contains a set of Entries which defines one or more of the following hierarchies:

A List of Prioritized Entry Credit Keys
- A list of cold storage keys
- A list of off line keys
- A list of backup keys
- A list of hot keys

A List of Prioritized Factoid Keys
- A list of cold storage keys
- A list of off line keys
- A list of backup keys
- A list of hot keys

A List of Prioritized Bitcoin Public Keys
- A list of cold storage keys
- A list of off line keys
- A list of backup keys
- A list of hot keys

The first Entry in an Identity Chain must define a hierarchy of Entry Credit Public Keys. And these keys must be used to sign the next entry. Entries can replace prior Keys, but must be signed by a valid Entry Credit Public Key from the current (at that point in the chain) Entry Credit Public Key.

Entries that are not signed by a Entry Credit Public Key in the hierarchy of keys for that identity are ignored as far as the consensus algorithm is concerned. They may, however, be useful for other purposes.

Other hierarchies are possible and useful for other Applications. There is no limit on what can be constructed and secured in an identity. Factom and other Applications simply parse an identity Chain for the information they need.

Even our own identity structure may change to incorporate other, more secure technologies and algorithms in the future.

The hot keys are used directly in Applications to represent an identity. Unfortunately, systems can be compromised. Backup keys allow the list of hot keys to be updated to exclude compromised keys. Additionally, off line keys can be used to replace backup keys. The cold storage keys are an identity's last line of defense.

Identities used for Supporting the Federated Servers

Federated Servers and Audit Servers (those looking to become Federated Servers) are elected by users of Factom, i.e. those that buy Entry Credits. To vote, a user needs to set up an identity to hold the Keys used to purchase Entry Credits. The identity of a user can hold the references to public keys they use to place Entries into Factom.

The user can then lend their support to particular Federated Servers, or lend their support to a proxy who can distribute the support.

The need to make the Federated Servers answerable to the parties that actually use the protocol is essential to ensure proper behavior and service under situations where the protocol alone cannot identify misbehavior. For example, patterns of censorship may be evident from the behavior of servers, yet the servers are not breaking the protocol.

Identities as used by the Federated Servers

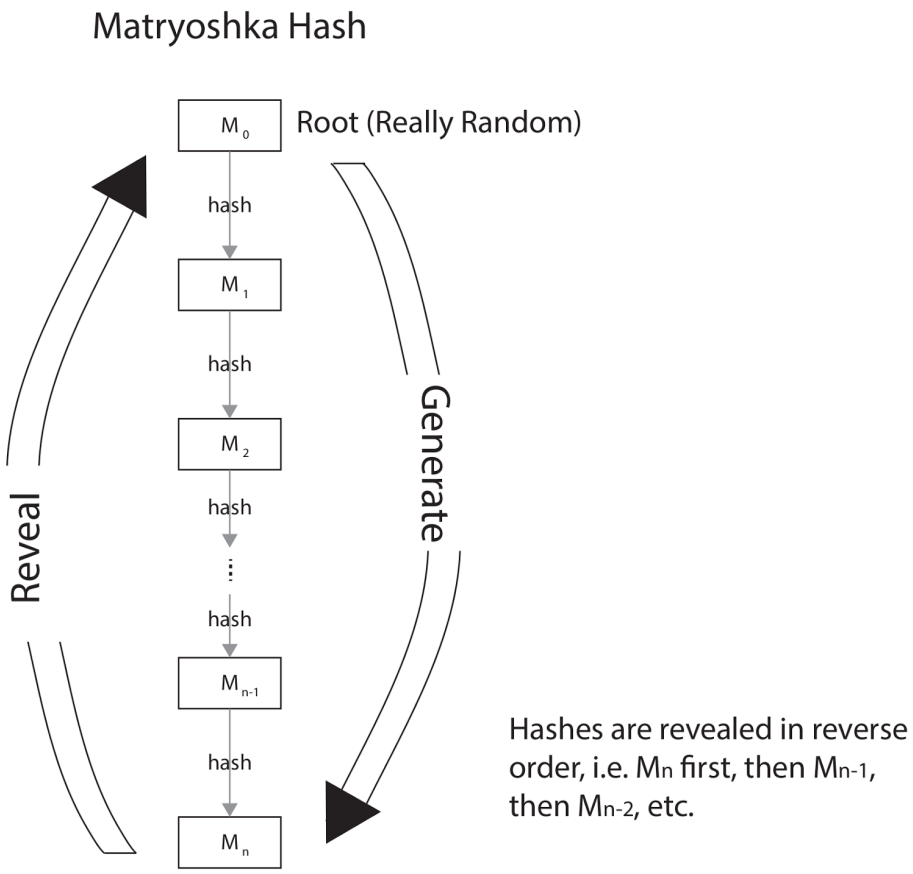
Bitcoin public keys are used by servers to post hashes into the Bitcoin blockchain. They cannot be used for general transactions. If any of these keys are used in a spend transaction which moves BTC without an OP_RETURN, or with an invalid OP_RETURN format from Factom's specification, then it is immediately considered invalidated. If an attacker compromises a key, it will be automatically revoked if the attacker attempts to take the Bitcoin balances. Such a theft should be insignificant, as the only Bitcoin required by a server would be the modest amount required to drop anchors.

In addition to invalidating keys simply by invalid use by an attacker, compromised keys will generally be replaced using the higher priority keys as defined in the identity to toss the compromised keys and replace them with new keys.

Every server maintains a Matryoshka hash chain (M-hash) in its identity. The server starts with some secret value that they hash some large number of times **r** (in the order of 100k). The last result $\mathbf{m_r}$ is recorded as part of their identity. The server can later reveal successive preimages, (i.e. $\mathbf{m_{r-1}}$ $\mathbf{m_{r-2}}$ etc.) Take an M-hash where the series **M** is defined:

$\mathbf{M} = \mathbf{m_1}\ \mathbf{m_2} \ldots \mathbf{m_q}\ \mathbf{m_{q+1}} \ldots \mathbf{m_r}$
where $h(\mathbf{m_q}) = \mathbf{m_{q+1}}$ for any $\mathbf{m_q}$ and $\mathbf{m_{q+1}}$ in **M**

For a given revealed M-hash, $\mathbf{m_q}$ cannot be predicted but can be validated after the reveal because the hash of **q, h**(**q**) = **q**+1 for any **q** and **q**+1 in **r.** This system allows the servers to inject deterministic, but unpredictable entropy into the selection process. It will prevent a single server from adjusting the state in order to influence the next state. As long as two servers in the majority maintain secrecy, the next state cannot be predicted.



Deterministic Random Seed

Factom generates the next system state with randomness pulled from the Federated servers. Each of the Federated Servers reveals the next Matryoshka hash in their M-hash sequence at the end of the minute. These are added together to produce the Random Seed for that

minute. As long as at least one reverse hash sequence is unknown to each Federated server, the Random Seed is unknown. The sequence cannot be easily manipulated (see Attacks section).

Identities as used by User Applications

The concept of identities can extend beyond the internal governance of Factom. Some privileged position relevant to some Application can homestead a Chain recording a hierarchy of public keys. After creating the identity, they can advertise it. Other clients in the Application can rely on information that the identity supplies. If an identity needs to update some information about itself, it can sign and publish it by writing Entries on their identity Chain. If a separate identity “B” needs to provide an endorsement for the original identity “A”, then they can write said endorsement, sign it, and place in identity A’s chain. If A needs to approve, A can enter their signature of said endorsement. Or B can send the signed endorsement to A, then A can sign, and make a single post with the endorsement and both signatures.

Identities can be placed into authorized lists, removed from authorized lists, etc. Lists can be private via encryption, or validated with Merkle roots (so proof of inclusion does not require exposure of the list), etc. They can be stored on Factom, or off Factom. Much more could be written on the possibilities of user identity management to support applications built upon Factom.

The Factom Network

The Factom protocol defines a series of communication rounds. Every minute a majority of Federated servers come to consensus. Every 10 minutes one of the Federated servers in the majority places a hash into the Bitcoin blockchain to secure the collective data. Factom uses a gossip based P2P network similar to Bitcoin’s. The network is flood fill, so all full nodes should see all valid transactions that are broadcast. The P2P network is also randomly connected. If a Federated server does not have a packet broadcast over the P2P network every few seconds, they are replaced by an Audit server as determined by a majority of Federated servers. This assumes partial synchrony as described by Dwork Lynch Stockmeyer[1].

There are some assumptions made about the Factom system.

- All nodes that validate anchors can see the historical Bitcoin blockchain, and are connected to the Bitcoin network
- Nodes validating anchors watch for and validate Factom Anchor posts, and look for invalid uses of Server keys.
- The Bitcoin blockchain is not substantially forked.

- The Bitcoin P2P network is accessible, and that the Bitcoin P2P network is not segmented.
- Propagation times for transactions on the Bitcoin network are within a few seconds.
- Bitcoin miners are not substantially censoring OP_RETURN, Factom anchors.
- The Factom P2P network is not segmented beyond a point where a majority can communicate.
- Propagation times on the Factom P2P network are limited to a few seconds between a majority of Federated servers.
- A majority of Federated servers are honest.
- A subset of the Federated servers greater than 2 maintain the secrecy of their Matryoshka hash chain.

Full Node Definition

The Factom Network consists of full nodes creating a peer to peer mesh network. A full node maintains all Directory Blocks, all Entry Blocks, the Factoid Chain, the Entry Credit Chains, and server identity Chains. Notice this list does not include the Entries themselves. With this data, they can validate submitted Entry Commits before relaying them. They can also reject invalid Entry Commits. The same for Factoid transactions, and Entry Reveals.

Factom's primary mission is to package data in a censorship resistant fashion while also resisting spam. It is not primarily intended to be a data storage mechanism. The bulk data is shared via peers on a Distributed Hash Table (DHT). The full nodes can and for the foreseeable future will host the Entry data, but in order to lower the system requirements for running a full node, total bulk data storage of Entry data will not be required in the future.

Message Definitions

There are 5 primary types of messages in the Factom P2P network. Server messages are covered in the Server Fault Message (SFM) section below. Of the remaining 4 messages, two message types pay for, then create a new Chain. The other two pay for, then create new Entries, placed into a previously created Chain. The messages are broadcast over the P2P network and are placed in assorted Chains for long term recording in Factom.

Chain Create, added to the Entry Credit Chain includes:

- 1 byte Version
- 8 byte timestamp
- Hash of the ChainID
- Hash of (ChainID + Entry Hash)
- Entry Hash

- Number of Entry Credits to be used
- Entry Credit Public Key Signature
- Entry Credit Public Key

Entry Commit, added to the Entry Credit Chain includes:

- 1 byte version
- 6 byte timestamp
- 32 bytes Hash of the Entry
- 1 byte Number of Entry Credits to be used
- 64 bytes Entry Credit Public Key Signature
- 32 bytes Entry Credit Public Key

Chain Reveal provides:

- The first Entry that hashes to the Entry Hash in the Chain Create
    - The first Entry must include the Chain Name, which hashes to the ChainID in the Chain Create

Entry Reveal just includes the Entry. It gets added to the chain specified in the Entry.

- Entry

The Entry includes:

- 1 byte Version
- ChainID
- (Required for a Chain Create Entry) Chain Name
- (optional) List of external keys
- (optional) block of binary data

Confirmations

Every data element processed by a Federated Server results in a Confirmation Message. This Message is propagated through the Factom Node Network, and serves as a promise that an Entry will be recorded. The message also promises the order in which it will be recorded. The Confirmation Message is placed in the Process List of the Federated Server.

The confirmation has the following information in it:

- Directory Block Height: 6 bytes
- ChainID of Server Identity
- Height: 4 byte index of this Confirmation Message in the Process List (per server)
- Affirmation value (one of:)
    - Entry Commit: SHA256 hash of the Commit Message
    - Entry Reveal: The Entry Hash

    - Entry Chain Create Commit: SHA256 hash of the Commit Message
    - Chain Create Reveal: SHA256 hash of the Reveal Message
    - Factom Transaction: SHA256 hash of the transaction
    - End of Minute Marker: Value of 0 instead of a hash
- Serial Hash (one of:)
    - SHA256 hash of (default Serial Hash + Block Height + Height + Hash), if the first Entry in the process list
    - SHA256 hash of (the previous Serial Hash + Block Height + Height + this entry's Hash)
- Server Signature

Proof of Use

Support is provided by Factom users that choose to set up an identity which they use to document their use of Factom. The public keys associated with their identity get a weight based on a diminishing scale. The first day the vote has full weight. At the end of each day, the weight of the vote drops 1/180th of the full vote. After 180 days (6 months) the Entry Credit purchase has no effect on the server selection process. The vote weight decay process is linear between day zero and 180.

Some observations:

- We will track purchases with 180 day slots, and age every transaction a day at about 0:00 UTC. The block ending at a time which is scheduled to match that day end is the last block to collect votes in. The "End Minute" message from the designated Server 1 coordinates the block closing.
- Votes of users most recently buying Entry Credits are worth the most.
- Proof of Use can be delegated via a proxy signed by one of the hot keys placed in the identity of the user indicating another identity.
- We had considered also including in the actual writing of Entries into consideration for voting, but have since concluded that this adds much more processing (writing Entries will be much more common than buying Entry Credits), and doesn't do much if anything to help the incentive structure. This is a change from the earlier Factom White Paper.

Timeout Period (TOP)

The timeout period is one by which all servers are required to respond. Right now we are considering 10 seconds. Propagation times over Bitcoin at the 50 percent level are pretty uniformly below 2 seconds.[2] We are looking at scaling TOP in response to network slowdowns to prevent a surge in network congestion causing faults in Factom that are not really faults, then scaling back to normal as network response improves.

Server Heartbeat

A server heartbeat is the broadcast of a valid Factom message, or a special heartbeat message (that includes the time) if there is nothing a server has to communicate to the Factom network.

Server Fault Message (SFM)

Server Fault Messages are issued for any material breach of the consensus algorithm. Examples of SFMs issued by a server A on some other server B include the following:

- If server A does not receive a heartbeat or a valid message from Server B within the TOP.
- If server B is responsible for placing an anchor into the Bitcoin blockchain and fails to do so.
- Server B places an invalid anchor.
- A Bitcoin key of Server B is used in an invalid way on Bitcoin. (Registered Bitcoin Keys of servers must only be used to place anchors)
- Server B fails to close their process list within TOP.
- Server B posts an invalid Factoid Transaction.
- Server B issues a Entry Reveal Confirmation where A has no matching Entry within TOP.
- Server B issues a Confirmation that does not validate against B's process list

Server Fault Messages can be repaired with a Server Fault Repair Message (SFRM). If a server A sees a majority of servers has issued a SFM on server B, then Server A issues a Server Reject Message (SRM) on Server B, and a Server Add Message (SAM) on the highest ranking Audit Server. SFMs are either cleared by the server issuing them, or they time out.

Federated Servers and Audit Servers

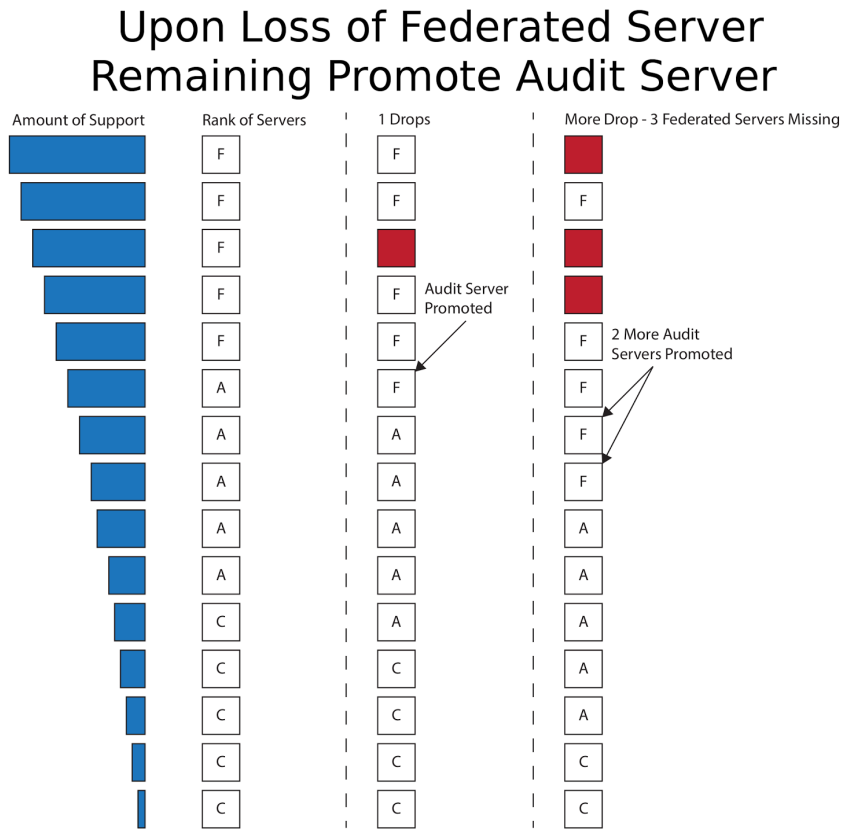
Factom is run by a set of **n** Federated Servers. Current thought is to set n to a number no less than 16, with a schedule to scale up with adoption. We will have **n** Audit Servers as well.

Some observations:

- All Federated servers and Audit servers must have a Factom identity, with a hierarchy of Bitcoin addresses, hierarchy of Factoid addresses, an address to receive payment of Factoids.
- The bitcoin addresses of the Federated servers are used to place the anchors in the Bitcoin blockchain
- The Factoid addresses are used to manage and secure the identity chain
- Users that wish to vote for Federated servers do so through a Factom Identity. Users can allow others to proxy their vote. Their vote is weighted by their proof of use.
- The top **n** servers by vote are the Federated Servers

- The Audit servers are the next **n** servers by vote after the Federated servers
- All servers must post a heartbeat. (Publishing a heartbeat is required for an Audit server to be eligible to be promoted to the Federated server pool)
- If a Federated server does not post a heartbeat within the TOP, then the Federated Servers will issue a SFM on that server.
- Audit servers that fail to post heartbeats at least 10x TOP are not considered for promotion.
- If a majority of the servers send a SFM message on a Federated server, then that server is dropped from the Federated server pool. Its position in the Federated server pool is replaced by the highest ranking Audit server.

Servers which have accumulated support but are below the ranking of Audit server are called Candidate servers. A Candidate server is automatically considered to be in the Audit server pool as Audit servers ahead of them are promoted to Federated servers.



The Consensus Algorithm

Notation

h(x) is assumed to be the hash of x.

The Federated Servers are Documented in the Federated Server Pool Chain

User votes are calculated and the Federated servers and Audit servers are elected
The identities of the servers are recorded in order in the Server chain.

Server Messages (all types) are recorded in the Server Fault Chain

Record server faults as they are thrown

Order the Federated Servers

Each of the Federated servers broadcasts their Matryoshka hash.
All nodes then compute the Random Seed
The Random Seed specifies an order for the Federated Servers by identities. The first server in that specified order is Server 1, followed by Server 2, to Server **n**.
rs = the Random Seed
Scid = the ChainID of the server's identity
We compute h(**rs**+**Scid**) for each Federated server, then sort.
Server 1 is the first in the sorted list,
…
Server **n** is the last in the sorted list

Chains are Assigned to Servers via the Random Seed

rs = the Random Seed
cs = the ChainID
Server 1 handles all ChainIDs where h((**rs**+**cs**))%**n**+1 = 1
Server 2 handles all ChainIDs where h((**rs**+**cs**))%**n**+1 = 2
…
Server **n** handles all ChainIDs where h((**rs**+**cs**))%**n**+1 = **n**

Entry Credits are processed by Servers determined via the Random Seed

rs = the Random Seed
ecs = the public key signing the Entry Credit
Server 1 handles all Entry Credits where h((**rs**+**ecs**))%**n**+1 = 1
Server 2 handles all Entry Credits where h((**rs**+**ecs**))%**n**+1 = 2
…
Server **n** handles all Entry Credits where h((**rs**+**ecs**))%**n**+1 = **n**

Factoids are processed by Servers determined via the Random Seed

rs = the Random Seed
ftxh = hash of the transaction
Server 1 handles all Factoid Transactions where h((**rs**+**ftxh**))%**n**+1 = 1
Server 2 handles all Factoid Transactions where h((**rs**+**ftxh**))%**n**+1 = 2

…
Server **n** handles all Factoid Transactions where h((**rs**+**ftxh**))%**n**+1 = **n**

Factom Full Node Network Messaging

1. At the Beginning of a minute block:
    a. Add the minute separator to all process lists
2. Entry Commits submitted are Valid Entry Commits if they validate against the Entry Credit Chain. (check if a commit is paid for)
3. Valid Entry Commits are added to the Entry Commit List
4. Valid Entry Commits are forwarded
5. Entry Reveals are validated against the Entry Commit List.
6. Valid Entry Reveals are added to the Entry Reveal List
7. Valid Entry Reveals are forwarded
8. Every confirmation from the Federated Servers is placed in a process list for that server.
9. At the End of a minute block:
    a. If any Entry Commits and/or Reveals in a Federated Server's process list is unknown to the node, make requests to neighbors or the DHT.
    b. The process list for each server is executed in order (i.e. Server 1, ... Server **n**)
    c. Executing the process lists constructs the Directory Block and the Entry Blocks.
        i. The execution of a Valid Entry Commit marks it as received in the Entry Commit List
        ii. The execution of a Valid Entry Reveal marks it as received in the Entry Reveal list.
        iii. Entry Commits and Entry Reveals that match and are revealed are removed from the Entry Commit List and the Entry Reveal List
10. At the end of 10 minutes
    a. We have all the Entries, and the Entry Blocks, with each minute separated within the Entry Blocks.
    b. Build the Directory Block header, containing the Merkle root and Serial hash of the previous Directory Block.
    c. Continue building the Directory Block with the ChainIDs of all the Chains that had Entries over the last 10 minutes (sorted by ChainID).
    d. Compute the serial hash of the Directory Block, and have the majority of Federated Servers sign it. Append the signatures to the Directory Block.
    e. Create a Merkle root of the Directory Block and signatures.
    f. Validate that Server #1 for the last minute records the Merkle root into the Bitcoin blockchain with one of their hot wallet bitcoin addresses.

g. Clear all process lists

Federated Server Processing

At the beginning of a minute block,

1. Add minute separators to all process lists.
2. Use the deterministic seed (DS) from the previous block to reorder the servers
3. Use the Random Deterministic Seed (RDS) to distribute ChainIDs across the servers
4. Use the RDS to distribute Entry Credit Chains across the servers
5. Maybe use the RDS to distribute the Factoids
6. Clear process lists, add minute separator to each list.
7. Add all entries in the Entry Commit List that now are this server's responsibility to the head of the stream of messages from the Factom Network (so they can be handled immediately.)

To accept a stream of messages from the Factom Network

1. Validate messages. Throw away invalid ones.
2. Messages that involve ChainIDs, Entry Credit chains, and Factoids to be processed by this server, do:
    a. Insert Entry Commits into this server's process list and issue Entry Commit Confirmation
    b. Insert Entry Reveals into the process list and issue Entry Reveal Confirmation
    c. Insert Factoid Transactions into the process list and Issue Factoid Transaction Confirmation
3. Messages not handled by 2.
    a. Add Entry Commits to the Entry Commit List
    b. Add Entry Reveals to the Entry Reveal List
    c. Add Confirmations from other servers into their respective process lists.

At the end of the minute

1. Send out End of Process List (EPL) message (includes the server's Matryoshka Hash)
2. Continue handling messages in 3. above until all End of Process List messages are received for all servers.
3. Iterate all Process Lists, building the chains in the order specified, from Server 1 through Server **n**.

Rinse and Repeat for 10 minutes.

At the end of 10 minutes

1. We have all the Entries, and the Entry Blocks, with each minute separated within the Entry Blocks.
2. Build the Directory Block header, containing the Merkle root and Serial hash of the previous Directory Block.
3. Continue building the Directory Block with the ChainIDs of all the Chains that had Entries over the last 10 minutes (sorted by ChainID).
4. Compute the serial hash of the Directory Block, and have the majority of Federated Servers sign it. Append the signatures to the Directory Block.
5. Create a Merkle root of the Directory Block and signatures.
6. Server #1 for the last minute is delegated to record the Merkle root into the Bitcoin blockchain with one of their hot wallet bitcoin addresses.
7. Clear all the process lists

Finding the Consensus Head

Finding the consensus head in Bitcoin is easy to do automatically. A node connecting to the system will ask around to find potential candidates which have the highest proof of work. The client can scan backwards through time to see if the blockchain originates with the correct genesis block. If it does, then it can scan forward to make sure that all the rules were followed between the genesis block and the longest candidate chain. If two valid blockchains are found, then the one the software chooses to recognize is the one with the most Proof of Work (POW). Factom relies on Bitcoin's Proof of Work, so a full Factom client will need to find Bitcoin's head to fully validate Factom's history.

Two Stage Scanning

Finding the consensus head of Factom can be done in a similar fashion to Bitcoin. A Factom client will need to bootstrap into the Factom P2P flood fill network, and know the Factom genesis block. Once they have a connection to the network, they can ask around from several nodes to see if there is general agreement on the Factom head. They would also find nodes on the DHT network by talking to the flood fill peers.

An assumption is that honest nodes will be well connected and will be presenting the same head to newcomers. An attacker may conduct a sybil attack and present a false head, but they would also need to be numerous. If the attackers presented many different fake heads, then each fake head would have fewer nodes attesting to it. It would be in a sybil group's interest to present only one fake head, to a victim, lowering the number of false heads to rule out.

Backwards Scanning

The client will then successively download the Directory Block headers going backwards in time from a potential head. If the chain of Directory Blocks leads back to the known genesis block, then they can start validating forwards. If the Directory Block chain does not lead to the genesis block, it can be ignored.

Forwards scanning

Once candidate Directory Block (DB) chain has been selected, a client will scan forwards, checking each DB to see if the rules were followed. There are various levels of assurance that a client can get as they scan forwards from the genesis block. The DB header will specify the identities which will be allowed to sign the next block. The identities also specify which Bitcoin addresses an anchor may be found under for the next DB.

The client will examine the Bitcoin blockchain to see if there is an anchor placed by one of the authorized addresses in Bitcoin. If the anchor is timestamped soon after the data was purported to have been published, then the client can know the data is the correct age. This eliminates an attack where the loss of private keys allows the creation of false histories.

Limiting the Bitcoin addresses where the next anchor can be found under eliminates the possibility that the majority can hide a fork. If two potentially valid anchors exist for the next block, then the client will suspect a fork by the majority. If the majority were acting correctly, then they would react as described in the section “A Federated Server Outside the Majority Places an Anchor”.

This has the drawback that if a majority cannot be formed, Factom stops creating new valid blocks. If Federated servers slowly drop off the network, then Audit servers will be promoted as Federated servers disappear. A majority of Federated servers is needed to promote, though, so a sudden loss of the majority of Federated servers will fail inelegantly. Although sudden loss of the majority will seize up Factom in the present, clients scanning through history will not see an unwarranted change in authority.

Criticisms and Attacks

Factom is a Centralized Service

No, Factom is a decentralized protocol. Membership in the Federated Server Pool is based on both performance and community support. The Federated Server Pool orders the transactions in all the Chains.

The confusion about centralization comes from the fact that in any one minute, a particular server from the Federated Server pool is responsible for ordering Entries into a given Chain. That responsibility shifts to another server in the next minute, and so forth. No one server has control of a particular Chain over time. Over 10 minutes, 10 different servers have the responsibility of ordering Entries for a chain.

The Federated Server pool is made up of the servers with the greatest support from actual users of the Factom protocol. Servers can schedule downtime, which will not penalize them. Communication failures or algorithm failures result in dropping servers out of the Federated Server Pool for one future election cycle, plus the remainder of the current cycle.

Federated Servers are backed up by Audit Servers. It is the Audit Servers that step up when Federated Servers leave the Federated Server Pool.

The Federated Server Pool is of a fixed size, at least 16 in number. (The exact number is still under consideration). The Audit Server Pool is the same size. Rewards are paid out at a fixed rate to the Federated Servers and the Audit Servers.

The decentralization of Factom compares well with Bitcoin as only 4 or 5 parties make up a sort of “voting block” with over 50 percent of the hashing power building blocks. Even at just 16 servers, it would take 9 parties to comprise more than 50 percent of control over Factom, independent of how support may be distributed. Furthermore, even a few non-censoring nodes in Factom will ensure all Entries are recorded on their Chains.

Forks of Factom may go Undetected

Factom follows an algorithm (See the section: Finding the Consensus Head) for setting anchors that insures that forks are always detectable. As long as a majority of the servers are honest, only one path will be valid. If a majority conspire to fork Factom, both forks are none the less detectable, insuring that the community will be aware of the breach.

Reordering Chain Entries

Once a client has submitted an Entry Reveal to a Chain, the server handling that Chain sends out a confirmation for that Entry. The client receiving that confirmation has a level of certainty that the Entry will be recorded. If the client has received all the confirmations from that server, the client can construct the entire process list for that server. Since each confirmation has a serial hash covering all the previous Entries plus the new entry, then the order and structure of the process list is known, not just to the client, but to all the Federated Servers and Audit servers alike.

If the Federated Server attempts to change the process list, it cannot do so in a way that changes the process list on all the other Federated Servers; Once a serial hash has been broadcast, any attempt to add Entries that do not include a confirmation in their history will fail to validate. If the server waits until the end of the 10 minutes to drop an Entry, such a server will not be able to sign the Directory Block at the end of the 10 minutes; there is no valid way to communicate changes to process lists. If a server cannot provide a valid signature for the Directory Block, such a server will be faulted and removed from the Federated Server Pool, and its process list as broadcast will be applied and the dropped Entry will be included in the dataset.

Once a Merkle root has been recorded into the Bitcoin blockchain, the contents cannot be changed.

Withholding Deterministic Seed

A Federated server can delay broadcasting their End of Process List (EPL) message until all the other servers have broadcast their EPLs. The withholding server can then ascertain what the next state would be if they broadcast their EPL with 100% certainty. If they withheld their EPL, the majority of the network would time them out by sending Server Fault Messages

(SFM) after a few seconds. The highest ranking Audit server would be promoted to Federated server. The next system state would now have a higher than 0% chance of being the attacker's desired state.

When a Federated server issues a SFM after it has closed its process list, it will publish an updated M-hash. This will prevent collusion between the withholding server and the highest priority Audit server from predicting with 100% certainty the results of the attack.

Mitigations to this attack are the timeout period. Once the withholding server performs this attack, the majority will remove it from consensus for an extended period of time, on the order of hours. This will make the attack only "work" at the cost of the attacker being tossed from the Federated Server pool. The attack is public. If a Federated server repeatedly pretends to lose connectivity only at this critical moment, then it is obvious they are performing this attack. Evidence can be presented to the community to convince users to pull support from the server.

Censorship of a User

A Server can attempt to censor a particular user so they cannot place their Entries into their Chains.

This is rather difficult to do in Factom. The public keys used to manage Entry Credits can change over time, since the Factom Chains are independent of the system of Entry Credits. If a server wished to censor a user by tracking their public key, the user who suspects censorship or is concerned about censorship could defeat this censorship by using a multiple public keys.

Another way Factom circumvents censorship is that the server responsible for recording the Entry Commits for a public key shifts every minute. Unless all servers are party to the censorship of a public key, eventually the user will be assigned to a server that will record their Entry Commit.

The nature of submitting Entries also makes other modes of payment possible. A user can pay someone to place their Entry Commits. A single public key could represent a large number of individual users. Since the collective payor is only signing hashes, they would not know beforehand what was being paid for. Censoring only one user is nearly impossible when that user pays for Entries with the same public key used by many others.

Since the Entry Commit only includes the hash of the Entry, no server can tell what Chain they will be writing to before the Entry Reveal is broadcast. There is no ability for the servers to censor a particular Chain at payment time.

Once the Entry Reveal is broadcast, there are only two requirements for inclusion. One is that the Entry's hash matches a paid Entry Commit. The other is that the Entry Credits of the

Entry Commit is enough for the data size of the actual Entry. If the server responsible for the Factom Chain specified by the Entry refuses to include the Entry, this does not prevent the inclusion of the Entry into the Factom Chain. The Entry remains in the protocol, and the next server will pick it up and include it into the appropriate Factom Chain. As long as one server is not censoring, the Entry will eventually be recorded.

This censoring attack is very public. The Entry Commit is performed without knowledge of the Entry or destination Chain. (The commit is done with only the Entry hash, and the Entry holds the ChainID). Any systematic delays between the Entry Commits and the Entry Reveals being accepted by a particular server will be cause for suspicion. Those that lend their support to a Server expecting it to maintain the protocol may very well withdraw that support if a server fails to process entries.

If a user does not use a collective payor, then censorship proof changes from being permanently recorded to being ephemeral. The user might want to reuse an Entry Credit key. On the first use of that key, the Entry contents would be unknown. On the second Entry Commit, the servers could surmise what the Entry would be. The servers could censor an Entry Commit from that private key, denying payment. This would create an ephemeral, rather than permanent record of censorship, which is harder to prove later. The user could fund a different Entry Credit key with Factoid balance and try to submit again. That Factoid balance would be associated with the user, so any Entry Credit key funded with that Factoid value could be censored. The censored user could acquire unconnected Factoids to get around the censorship, or mix the value with users who should not be censored.

Censorship attacks, to be successful, need to be implemented by every Federated server. A mere majority can temporarily remove a Federated server which defies censorship, but that is very drastic.

Placing Offensive or Illegal Entries into Factom

Censorship is very hard and perhaps nearly impossible in Factom. No data in the Directory Blocks, nor the Entry Blocks will be plain text. The Entries themselves can contain any information at all, but Entries containing offensive material can be identified and removed from any particular system using Factom. Even though Factom secures such information, that does not mean users are forced to hold the offensive material.

Majority Sending SFMs to Eliminate a Minority

When a specific server gains authority as a Federated server, the majority can issue SFMs removing the authority gained by that server, ejecting it from the system. This breaks the assumption that 50% of Federated servers are honest. Additionally, the attack will be public. If the attacking majority cannot provide evidence that the ejection was justified, then they would lose community support.

The Federated servers would not gain a direct monetary advantage from this attack. When they eject a server in the minority, the network would promote an Audit server to a Federated server. The Audit server would get the payment originally slated for the ejected server. This is different from Bitcoin where if the majority of Bitcoin miners excluded a minority, they would get the balance not claimed by the excluded minority.

Costs of Attacks

In general, if the majority of Federated servers deviate from honest operation, they risk losing their investment in the identity they created. Attacks by Federated servers on users are generally publicly visible. Proof of censorship can be generated. Servers attacking other servers will risk causing the users to lose confidence in the attacking servers, withdrawing support. If a majority manages to gain control and substantially disrupt the network, they would lose all value they had invested building in the network, since users would abandon Factom for a more secure system (potentially a hard fork of Factom rebasing power excluding the attackers).

The incentives of the servers is to please active users instead of currency holders. If an attack were profitable to currency holders while being disadvantageous to users, the users would pull support. The effect would be similar to how community outcry happens when a mining pool gets near 50% of hash power. The difference is that the community would actually be able to do something about it, not just pool members.

Segment P2P Network Isolating Majority

A majority of servers is needed for consensus in Factom. A majority is needed to both confirm blocks as well as promoting Audit servers to Federated servers. If the network is split where a majority cannot communicate, consensus stops. This breaks one of the assumptions. A majority is needed if the Federated server count is even or odd.

A Federated Server Outside the Majority Places an Anchor

A client scanning over the historical Bitcoin blockchain will build a model of consensus deciders at that time. Ideally, only one Federated or Audit server will enter an anchor for each block height every 10 minutes. There is some asynchronicity with the blockchain. Anchors can be placed out of order either in the processing of transactions or in a re-org. To mitigate this issue, anchors are tagged with their height, so they can be reassembled in the correct order. There is a possibility that an anchor is placed by one of these servers which is not agreed upon by the majority. The attacker could anchor a Directory Block not signed by the majority, so it wouldn't be valid, but the client would have to download something from the DHT to determine that. The downloaded data could be malicious somehow (i.e. very large), but it cannot be signed by the majority unless the majority of the Factom Servers were malicious.

To mitigate this attack, the majority of Federated servers will detect this error. All servers see the transaction stream from Bitcoin, so they can react prior to the anchor appearing in any

block. Each honest server will submit the correct anchor for that block height upon detection of the false anchor's posting. All the Federated servers would issue Server Fault Messages (SFM) against the server issuing the invalid anchor, as they have a signed message indicating the server's failure to act within the consensus algorithm. Such SFMs will drop the offending server from the Federated Server Pool.

A future user scanning the blockchain instead of only seeing one potentially valid anchor, would see 10 or more. They would determine which anchor was correct based on majority consensus forcing a correct Directory Block. The minority could continue this, causing higher fees for the network, until a point where a future user interpreting the Directory Blocks would not see the attacker as being in the set of consensus deciders.

Since transactions being included in the Bitcoin blockchain are not guaranteed, then a user scanning a historical blockchain should read ahead some number of blocks to determine if the anchor was refuted. If the assumption holds that the majority of Factom servers are acting honestly, then they will not sign blocks which are malicious. This attack would only make users scanning forward through history look closer to see how well the rules were followed. This attack can be further mitigated by incorporating side channel information on what happened during this period.

Replay Attack of User Data

All data on Factom is public, and data can be injected into the P2P network long after it was created. There are 4 user generated pieces of data which others can resend potentially years later.

1. Chain Create. This message is a payment which pays for both a new chain to be created as well as for the the first Entry. It has a timestamp which does two things. First, the timestamp must be close to the present time. This allows full nodes to drop transactions which are too old to be valid. This makes it easy to filter out payments replayed later, intentionally or otherwise. The timestamp also acts as a nonce. Two transactions are not allowed at the same time from the same public key. Multiple transactions only differing by maleated signatures would be treated as the same, since only the timestamp and pubkey are used for uniqueness.
2. Reveal Chain. This data is only valid if there is an unfulfilled Create Chain message. Full nodes will not propagate it without having a recent payment to create the chain. It is not susceptible to a replay attack, because a Chain cannot be created more than once.
3. Entry Commit. This message pays for an Entry. It has a timestamp which limits the period the Commit can be valid. Full nodes do not need to guard against replays beyond a timeout period. The timeout period is probably a day plus or minus. Like the

Chain Create message, there can only be one commit per Entry per private key per timestamp.

4. Entry. These **are susceptible** to replay attack, but only if the Entry has already been paid for. An identical Entry can be paid for twice, and will be inserted into a Chain twice. The system does not care who paid for an Entry, so an attacker can pay for thousands of clones of a target's Entry. If a user's Application is susceptible to a replay attack, then the Entry data should be made unique and ignored if duplicated. One option is to have the Entry contain a signed timestamp or counter, but that would be up to the Application designer, not Factom. If the Entry has not been paid for, then the full nodes will not relay it.

Conclusion

Factom attempts to build a generalized proof of publication system for general data. The proof of a chain of data links back to the Bitcoin Blockchain; users need only to have access to the Directory Blocks and the entry blocks of the chains which they care about. Most of the data in Factom can otherwise be ignored by an application.

We believe Factom will prove to be essential to the construction of much more complex and useful distributed autonomous applications. Factom will also be useful in creating more secure, tamper proof, trusted computer systems.

None of this would be possible without the work of Satoshi Nakamoto, and the many developers that made Bitcoin a reality.

References

[1] http://groups.csail.mit.edu/tds/papers/Lynch/jacm88.pdf
[2] http://bitcoinstats.com/network/propagation/
[3] Information Propagation in the Bitcoin Network
[4] https://ramcloud.stanford.edu/raft.pdf
[5] https://blockchain.info/pools
[6] https://github.com/ethereum/wiki/wiki/Patricia-Tree

Appendix A. Terminology

We assume that the user is familiar with Bitcoin terminology. This appendix defines the terms unique to Factom.

Anchor: The Merkle Root of a Directory Block in Factom that has been recorded into the Bitcoin Blockchain. All the data in Factom can be validated by finding the collection of Anchors in the bitcoin blockchain, validating them, then confirming that all the Directory Blocks in Factom are, in fact, anchored to the Bitcoin Blockchain.

Audit Server: A server running the exact same software as a Federated Server, but without the support required to be part of the Federated Server Pool. If a Federated Server in the Federated Server Pool goes off line, or steps out of the pool for maintenance, the Audit Server with the greatest support joins the Federated Server Pool.

Audit Server Pool: The set of servers running the Federated Server software which publish a heartbeat, and have enough support to join the Audit Server Pool. The Audit Server Pool is of a fixed size.

ChainID: The name of a Factom Chain is hashed to create the ChainID. ChainID's are used in the Directory Block, and anywhere else in the protocol that must refer to a Factom Chain.

Directory Block: This is the list of (ChainID, Entry Block Merkle Root) pairs collected by Factom over a 10 minute period. The Directory Block is sorted by ChainID. Directory Blocks are organized in a chain over time, and include the serial hash of the previous Directory Block.

Entry: A single data submission made by a user. A collection of entries makes up a Factom Chain. There are certain bookkeeping entries required, but there are no restrictions on the content of an Entry. An Entry is restricted to no more than 10K in size, and requires an Entry Credit for every 1K of data. Entry size is so restricted to insure reasonably fast propagation of Entries through the network, as required by the consensus algorithm. The user can string together multiple entries for larger content.

Entry Block: The Entry Hashes of all entries for a particular Factom Chain received over a ten minute period. The Merkle Root for an Entry Block is stored in the Directory Block for that ten minute period.

Entry Commit: The submission of the Entry Hash along with the number of Entry Credits required to pay for the recording of the Entry. The Entry Commit must be signed by a valid Entry Credit Address with a balance of Entry Credits sufficient to cover the Entry Commit.

Entry Credit: Factoids can be converted to Entry Credits, which act like a software license. They are tied to an Entry Credit Address. They are non-transferable, non-refundable, and can only be converted into Entries for the purpose of writing an entry into Factom.

Entry Credit Address: A cryptocurrency address for holding Entry Credits, and authorizing Entry commits

The Entry Credit Chain: A Factom Chain maintained by the protocol that tracks all the Entry Credit balances of all the Entry Credits issued to public keys. This is one of the few protected chains that can only contain entries created by the Factom protocol.

Entry Hash: The hash of an Entry, stored in the Entry Block for the ten minute period when the Entry was revealed.

Entry Reveal: Submission for an Entry that matches a previous Entry Commit. No signature is required, as the hash of the Entry Reveal must match an existing Entry Commit.

Factoid: The token issued by the protocol to the Factom Servers for collecting entries, creating and publishing the updated entry chains, and submitting the anchors to the Bitcoin blockchain.

Factoid Address: A cryptocurrency address for securing Factoids. Only multi-signature transactions are supported, with a single signature being a special case (1 of 1).

Factoid Transaction: Like a Bitcoin transaction, a Factoid transaction moves a balance of Factoids from the control of one Factoid Address to another Factoid Address.

Factom Chain: Entries are written to a Factom Chain. A Factom Chain is composed of the first Entry which provides its Name, which is hashed to give its ChainID. Subsequent Entries must include this ChainID.

Federated Server: A server running the Factom Server software which collects entries and Factoid transactions, validates them and commits them to their respective Factom Chains.

Federated Server Pool: The collection of servers with the highest support that are currently in charge of the protocol. If any Federated Servers leave the pool, they are replaced with the highest ranking Audit Server.

Heartbeat: Federated Servers are responsible for broadcasting a valid message within the Timeout Period (TOP). A special Heartbeat message can be broadcast should there be no valid communications to broadcast. Audit Servers must have a heartbeat broadcast every 10 TOPs.

Process List: Each Federated Server broadcasts confirmations for entries to be added to Factom Chains. These are hashed and ordered, so that all listeners can validate the order and content of the process chain for a server. At the end of a minute, all Process Lists are executed to add entries to the Entry Blocks.

Protected Chain: A Factom Chain which does not allow invalid entries to be entered. At this time, the Factoid Chain and the Entry Credit Chain are the only two protected chains in Factom. They are protected against invalid entries because the protocol requires the entries to be valid in order to properly function.

Random Deterministic Seed (RDS): A Random number generated at the end of each minute with a provably correct value used to order processes in Factom.

Server Reject Message (SRM): A message broadcast by a Federated Server A to indicate A has seen a majority of Federated Servers have issued a SFM on some Server B.

Server Fault Message (SFM): A message broadcast by a Federated Server to indicate a fault with a Federated Server or Audit Server.

Server Fault Repair Message (SFRM): A message broadcast by a Federated Server to indicate a retraction of a previous SRM.

Timeout Period (TOP): The period required for every server to broadcast, or a SFM will be issued, and the Federated Server will be dropped from the Federated Server Pool. Audit Servers must broadcast once every 10x TOP.