The notes below describe the approach in designing and writing the initial reference implementation of Handshake. This is not a prescriptive document and should not be used as such. This document's goal is to provide a referenece on the rationale and initial design of the protocol.

Abstract

The foundation for the internet's security has relied upon trusted Certificate Authorities (CAs) which attest that a user is connecting to the correct server or node. This has created a reliance upon a handful of trusted actors, many of whom are for-profit corporations or other actors who may not have long-term incentive towards stewardship of the internet. The net-effect is a "1-of-m multisig" whereby if any one of the trusted CAs fail, the entire security of the internet fails. This failure has occurred and will continue to occur with the trusted-CA design, with catastrophic risks as more and more infrastructure becomes networked.

Many replacements have been proposed to properly secure the internet, but none have been successful. Fundamentally, a trust anchor is needed to provide a secure association between names and servers who claim to be the correct endpoint for those names. There has been a tradeoff between federated nodes (CAs) and a single trusted entity controlling the network (DNSSEC). Handshake is an ongoing project to establish a decentralized network whereby cryptoeconomic incentives are established to coordinate consensus on the association between names and certificates.

This document describes a proposal, operational functionality, and intention to replace centralized trusted internet infrastructure, with a decentralized Certificate Authority and globally unique namespace composed of a decentralized blockchain and cryptographic proofs backed by cryptoeconomic mechanisms. This construction enables the namespaces to point directly to a compact certificate representing a trust anchor which does not rely upon a single trusted authority to create attestations as in the existing federated Certificate Authority model. Handshake builds in compact verifiable proofs to ensure compatibility with embedded and mobile devices, with significant committed merkelized state proof-size and performance improvements.

Further, a method is proposed to achieve decentralized, large-scale community coordination inspired by the free and open source software aesthetic. The free and open source community has provided the most critical contribution towards development of the internet and has produced software which humanity relies upon worldwide. This coordination is achieved by building a decentralized infrastructure backed by a blockchain to support collective agreement on certificates and coordination using direct ownership of a commodity token by those who are most capable of integrating and using the Handshake blockchain, which optimizes for the long-term incentives of the free and open source community. The Handshake project and community is a performance experimenting on replacing the social function of centralized corporations in favor of self-interested gift economies, which achieve coordinated goals.

Project Summary

Many understand the green lock icon on the web browser as meaning the connection is secure and encrypted between themselves and the server identifying as the website. However, the security has always been entrusted in a handful of centralized Certificate Authorities (CAs). These entities are the guardians of the internet and there has been many documented cases of failure[fakecert-fr][fakecert-ir]. As the internet connects more devices and economic infrastructure becomes internetworked, the impact of any one failure dramatically increases.

Historically, a trilemma known as Zooko's Triangle[zooko] was believed to exist where one must pick two of three possible properties consisting of: Human-meaningful names, Decentralized, and Secure. This trilemma reflected the perceived constraint revolving around the notion that it may be necessary for a single point of trust for a consensus around short names (e.g. a website address), otherwise a lack of global consensus around the owner of the name eliminates the name's meaningfulness and security.

Recent innovations with the blockchain has possibly maneuvered around this trilemma by creating a single point of consensus around the association between names and certificates, in a single decentralized blockchain represented by many actors verifying the network[aaron]. Achieving this security property requires not only redesigning the trust anchor in certificate authorities, but also requires deep integration in the naming infrastructure itself.

A blockchain is proposed which optimizes for correcting prior weaknesses around acknowledging stakeholders such as existing top-level domain (TLD) holders and optimizes for decentralization (while still allowing for n-of-m attestations). Users use the native token (coin) to register TLDs which are pinned to a specific certificate as the identity. A committed merkelized proof of all top-level names allow for compact, shareable inclusion and exclusion proofs. This blockchain exists to attempt to resolve the need for a globally unique namespace which is necessary to have an association with unique names and certificates. While it's possible to create a singular centralized globally unique association (DNSSEC), a decentralized system can be resolved by creating a blockchain with its own cryptoeconomic incentives (coin), including name auctions of a unique namespace and block creation. Scarce resources require sybil protection, usually managed by a central trusted authority (CAs, ICANN), but can be resolved by having a blockchain based mechanism for global consensus and resource allocation.

As a blockchain is needed for global consensus on a namespace, there needs to be a choice upon how to allocate resources for this system. The intent behind Handshake is to allocate a representative portion of the resources to the stakeholders which may be potentially contributive towards development and adoption, hence the overwhelming majority of the resources being allocated to the free and open source community. It is possible that this project may spur other projects to allocate the overwhelming majority of economic resources to the free and open source community in the form of an obligation-free distribution. This document describes an emergent mechanism and game whereby decentralized blockchain project developers and investors, in the face of competition from other projects, may have significant self-interested incentives to distribute an overwhelming majority of token/coin ownership to the free and open source community, and ultimately the whole of humanity.

In order to achieve new games of distribution and a new economy predicated upon true gifts over contracted labor, it must be achieved via a self-perpetuating and self-interested mechanism which is game-theoretically sound. One of the principal of this project is sparking a mechanism whereby individuals have self-interested incentives towards creating decentralized projects as well as a wide distribution via gifts. Consequently, its goal is to fulfill the self-interested imperative of a return by the principals initiating, developing, and scaling the project worldwide. However, to scale up, there is a game between projects which distribute ownership of the chain itself to as wide of a set of participants as possible. Handshake aims to distribute 15% of the coins to the individuals and companies responsible for creating the coin (with the developers/organizations/advisors, and early investors split evenly at 7.5% each). This ensures a self-interested game can perpetuate for future projects, and future projects without significant front-loaded development costs may be even lower.

The Handshake project aims to distribute around 70% of the coin supply to open source developers, projects, and non-profits without any contractual expectation of work by the individual free and open source developers.

Fundamentally, the self-interested mechanism requires all developers and users to be receiving coins as an incentive. A summary of the mechanism is as follows: Presume in the future there are three hypothetical projects released which achieve the same goal, let's say it's a decentralized mesh networking blockchain. Two of the three give 90% of its value to the creators of the project. The third gives 85% of the value to FOSS developers and those who put up nodes. It would stand to reason that the third would have significantly greater odds of success. The Handshake mechanism is designed to create a competitive game of asset ownership distribute more to FOSS developers, and perhaps all of humanity.

Much as capitalism creates a competitive game between participants which competitive self-interest reduces the price of goods in non-monopolistic commodity environments, the handshake mechanism is a project exploring a similar concurrent game to maximize ownership for FOSS developers and the public. No single producer reduces the prices of their own good for altruism in capitalist marketplaces, it is done through self-interested competitive incentives, "I make more money when I lower my prices". Similarly, the handshake mechanism is experimenting with a process whereby "I make more money the more is gifted to FOSS developers and the whole of humanity".

Decentralized Certificate Authorities and the Blockchain

The resolution to the trilemma, Zooko's Triangle, is between the relationship with the name and the cryptographic identity of the owner. By making the owner of the name a cryptographic key, one can create a certificate chain of the owner down to the key by creating a signature signed by that owner's key (a chain of custody). This is not possible under the current system as the current owner of the name is not owned by a cryptographic key, but rather trusted records held by custodians with non-cryptographic records of named owners, e.g. the TLD ".com" is held by Verisign.

However, cryptographic attestation of certificate chains for SSL/HTTPS is insufficient in creating decentralized certificate authorities. If there are no canonical points of truth for a decentralized record of the relationship between keys and names, then the record can be disputed. Alice can claim to own the TLD ".example", and Bob can think he's the correct owner. It is uncertain who the true owner of of ".example" could be.

The blockchain creates the ability for canonical ownership records by recording in order which record exists before another (thereby letting one know that Alice registered a name before Bob could possibly register one, and therefore only Alice's is correct). Without this canonicalization, it would not be possible to have confidence that one is talking to the correct owner of a name and hence is fundamental to canonical name resolution.

This canonicalization introduces an interesting dilemma, namely that of the ability to sybil the network. Even with canonical ordering, the problem of namespace allocation remains. A single party could spam the network registering all possible short names in existence, monopolizing the network. This would severely reduce the usefulness as one or a handful of parties monopolize the resources. To correctly mitigate this problem, a currency native to the blockchain is necessary to create a cost function for the names. When a name is auction and sold, the coins are permanently destroyed from the system. Without a cost, there is no cost to spamming and a single party owning everything. A native coin is necessary, as a dollar-pegged coin would depend on an external environment and have a trusted 3rd party operate as a gatekeeper to the system, whereas a native coin would not depend on any single trusted third party.

As a consequence, it then becomes a question of resource allocation as a method of prevention against sybil attacks. Should the majority go to initial developers/investors, majority to miners, or majority to FOSS developers in the early days of the coin? Handshake is an experiment in the possibility that the majority of ownership claimed by the FOSS community is a rational and game theoretically superior strategy to traditional models of corporate growth and development for some project types; those where a decentralized blockchain is ideal.

Consensus

Proof of Work

Proof of work[pow] saw its first use in cryptocurrency with the advent Bitcoin[bitcoin]. Bitcoin's PoW function is a further iteration of a specific proof-of-work construction known as Hashcash[hashcash]. The use of proof-of-work led to the creation of specialized chip hardware intended to optimize this function. While specialized hardware can ensure that a proof-of-work network is protected, we concede that it does have the capacity to enable the existence of hardware monopolies.

However, we submit that this is an acceptable risk due to the benefits proof-of-work offers in the way of SPV. Our protocol is not usable in practice without proper SPV proofs.

There is currently no known sufficiently _decentralized_ proof-of-stake system in production resilient against fraudulent SPV proofs.

Proof of Work Functions

A Hashcash proof-of-work function using SHA3 and blake2b is used. SHA3 is currently under-represented in proof-of-work functions. We find that simplistic nature of Hashcash limits the room for unforeseen optimizations, and that the current lack of SHA3 usage in combination with blake2b in proof-of-work functions creates a more level playing field for hardware manufacturers.

Cuckoo Cycle[cuckoo-1][cuckoo-2] was considered, however, throughout the course of the development of our protocol, we witnessed frequent optimizations to cuckoo cycle mining algorithms by Tromp and other contributors. Given these optimizations and the complexity of the underlying algorithm itself, we began to strongly consider the room for unforeseen optimizations in the mining process.

We fear that an unforeseen optimization, if kept secret after its eventual discovery, could lead to even harsher monopolistic conditions among hardware manufacturers. Furthermore, we find that the Cuckoo Cycle verification and mining algorithms are lacking in the area of formal academic analysis. If economic incentives are created to optimize Cuckoo Cycle, we expect that graph theorists and other experts will be involved with the creation of optimized mining algorithms and hardware.

We see these as unreconcilable issues, as they impede the ability to properly choose Cuckoo Cycle parameters for a blockchain. Cuckoo Cycle parameters themselves are difficult to adjust on the consensus layer, and perhaps cannot be dynamically adjusted safely.

Difficulty Adjustment

Given the prevalence of edge cases such as _timewarp attacks_[timewarp] and stalling during abrupt hashpower changes on the Bitcoin retargetting algorithm, we sought alternatives.

We examined DigiByte's DigiShield[digishield-1][digishield-2], MegaCoin's Kimoto Gravity Well[kimoto-1][kimoto-2], and DarkCoin's Dark Gravity Wave[dgw], as potential retargetting algorithms for our protocol.

Due to the uncertainty of exactly how much mining power will enter the network upon launch, and, initially, the added uncertainty of a new PoW algorithm, we desired an approach which would retarget on every block. DigiShield seems to perform especially well in the case that mining power abruptly enters or exits the network.

The formulation of the _Zcash_ retargetting algorithm[zcash-1][zcash-2][zcash-3] is also of particular relevance, given the similarities of the protocols' respective PoW functions. Zcash has had success with their variation of DigiShield, and as such, our protocol's retargetting is more-or-less a faithful reimplementation of the Zcash algorithm.

Unspent Transaction Outputs (UTXOs)

The UTXO-based blockchain, also introduced by Bitcoin, transfers money from one party to another using a series of _transaction outputs_. Transaction outputs inherently enforce _order_ of transactions within a _block_. Order-enforcement in particular is necessary for our protocol to function with the utmost security.

Our naming system requires on-chain smart-contract-like behavior. It deals in outputs which need to update a global state. This is atypical of UTXO systems. But as such, we require that these operations occur in a predictable order.

This order-preservation mechanism is especially necessary for maintaining the transaction _mempool_ state in a predictable manner. This model ensures that block assembly is a fast and simple process.

Naming History

The history of naming has been profoundly effective, exploratory, and has had many skilled teams and projects. From the beginning, the DNS system and SSL/CAs have been elegant and the Certificate Authority system's existence since the mid 1990s to now has been a testament to its resilience, with the hard work of thousands of individuals and organizations. Since then, there have been many other attempts to replace, upgrade, or distribute this system.

The pioneers of naming cryptocurrencies include Namecoin[namecoin-1], ENS[ens-1][ens-2], and Blockstack[blockstack-1] (among others).

Namecoin's model requires a user to run a fully validating node in order to securely resolve domain names. Although Namecoin was the first cryptocurrency project to attempt to implement a DNS bridge[namecoin-2] for a cryptocurrency naming protocol, the protocol itself is lacking in the area of SPV.

The _Ethereum Name Service_[ens-3][ens-4][ens-5] lends itself to a bit of centralized control as the ENS root[ens-6][ens-7] is centrally maintained by a select group of signatories. As is the case with Namecoin, there is no easy avenue for _compact_ proofs on ENS.

Of our three predecessors, Blockstack came the closest to providing easy-to-verify compact proofs for names.

SPV name verification in the Blockstack _SNV_ protocol first involves retrieving a state root (also known as the _consensus hash_) from a _trusted node_ and securely requesting the name record from an unstrusted node. Unfortunately, one can not verify that the name record served is the most recent revision[blockstack-2]. Furthermore, requesting the state root from a trusted node leads to a similar construction as Namecoin, wherein one must run a fully validating node in order to securely retrieve name data.

This full node requirement, which all naming predecessors are encumbered by in some form, may be one of the primary hurdles to widespread adoption of a naming cryptocurrency. To allow for SPV name resolution in the absence of a trusted full node, provability of names must be an inherent feature of the protocol.

There has also been prior work on alternative root zones. The most significant example of an alternate root zone is OpenNIC[opennic]. This is a proposed alternative to ICANN via an alternate namespace of unused names. This shifts a singular root source of truth to a federated model where a namespace is controlled by different entities, however, does not remove trust in those organizations. It does not create additive security, as the trust is partitioned according to TLD to a single root zone.

The Convergence Project[convergence], initially proposed by Moxie Marlinspike, created a system of certificate pinning and attestation. Notaries would attest to the endpoint's certificate. This system created an additive federated model of trust. This provides a significant improvement to the existing single CA trust model. However, there is no canonical trust anchor, which means that two people can have a different view on the correct certificate in adversarial or faulty notaries.

Certificate Transparency[ct] provides a method to ensure committed proofs of name records and significant increases the security of the system. It creates a method of accountability for failure and rapid detection. This is primarily a detection mechanism in the event of failure of CAs.

Provable Data Sets

In order to be free of the full node requirement, our protocol requires an _authenticated data structure_. In particular, we require a data structure which can efficiently map keys to values. In order for our protocol to be a suitable replacement for the DNS root zone, speed and size were our primary concerns.

In our benchmarks of various data structures, we found that while many of them are indeed performant, they have an unacceptably large proof size, frequently exceeding 1-3 kilobytes. This led us to do further research.

Our strict requirements include minimal storage, exceptional performance on SSDs, small proof size, and _history independence_, wherein order of insertion has no effect on the final state of the tree. The latter requirement is something which every re-balancing data structure inherently lacks.

These requirements severely reduced our options. We examined in particular the Merkelized Base-16 Trie[ept-1][ept-2] used in Ethereum[ethereum], and the Sparse Merkle Tree[smt-1] used in Google's certificate transparency project[smt-2].

We found that while Ethereum's base-16 trie was performant, the proof size was not suitable for our protocol. The storage requirements were also excessive.

We further discovered that Google's Sparse Merkle Tree was unsuitable in terms of performance, as each insertion requires a large number of database lookups without heavy caching, as well as several rounds of hashing for each item inserted. A typical insertion of 5,000 leaves required at least 1.2 million rounds of hashing in our benchmarks, as well as a significant number of database reinsertions.

As a result of this research, we consider authenticated data structures implemented, or intended to be implemented, on top of existing data stores to be inherently flawed in terms of scalability.

FFMT (Flat-File Merkle Tree)

We devise an authenticated data structure, which unlike the Ethereum Trie or Sparse Merkle Tree, is intended for storage in flat files. This removes the overhead of database lookups entirely by making the data structure its own database implementation.

By storing a merkelized data structure in a series of append-only files, we are able to provide traditional database features such as snapshotting, atomicity, and crash consistency. Given our requirements, a trie is the obvious choice as a backing data structure.

The initial implementation of our FFMT is a simple base-2 merkelized trie, and results in a construction which shares many similarities to earlier work done by Bram Cohen[merkleset].

Later iterations of our data structure began storing colliding prefix bits on internal nodes, resulting in a base-2 merkelized radix tree similar to the _Merklix Tree_[merklix-1][merklix-2] proposed by Amaury Séchet.

As the backbone of our FFMT, we propose one of the two aforementioned data structures.

Our initial FFMT implementation resulted in over a 50x speedup over Ethereum's Base-16 Trie and over a 500x speedup over Google's Sparse Merkle Tree. We also found that proof sizes are comparable to compressed Sparse Merkle Tree proofs and roughly four times smaller than base-16 trie proofs.

The FFMT storage requirements, while steeper than those of the Sparse Merkle Tree, are still much smaller than the Ethereum base-16 trie. We benchmarked insertions of 50 million 300-byte leaves into our FFMT in batches of 500 with a periodic commission of 44,000 values to the tree. Our benchmarks were run on a high-end but consumer-grade laptop, containing a Intel Core i7-7500U 2.70GHz and an NVMe PCIe SSD. Near peak capacity, the 500-value insertions themselves averaged roughly 100-150ms, with a commission time averaging 400-600ms for 44,000 leaves. Note that the committing of the tree involves a call to `fsync(2)`.

These insertion and commission times are acceptable with 5 minute blocks. We add the extra fail-safe of limiting any tree updates to a maximum of 600 per block, giving us a predictable worst-case insertion complexity.

Description

Typical of any trie-like structure, the FFMT follows a path down each key in order to find the target leaf node.

Insertion

We start with a simple insertion of value `a` with a key of `0000`. It becomes the root of the tree.

fig. 1

```
Map:
  0000 = a

Tree:
       a
```

We insert value `b` with a key of `1100`. The tree grows down and is now 1 level deep. Note that we only traversed right once despite the 3 extra bits in the key.

fig. 2

```
Map:
  0000 = a
  1100 = b

Tree:
       R
      / \
     /   \
    a     b
```

The following step is important as to how our simplified FFMT handles key collisions. We insert value `c` with a key of `1101`. It has a three bit collision with leaf `b` which has a key of `1100`. In order to maintain a proper key path within the tree, we grow the subtree down and add _null_ (or _dead-end_) nodes (represented by `x`) as children of internal nodes. Dead-end nodes are more tangibly represented by a sentinel hash of all zero bits.

This differs from the Merklix-like version of our tree, which would store bit collisions within a single parent internal node.

fig. 3

```
Map:
  0000 = a
  1100 = b
  1101 = c

Tree:
       R
      / \
     /   \
    a    /\
        /  \
       x   /\
          /  \
         /\   x
        /  \
       b    c
```

We add value `d` with a key of `1000`. It is free to consume one of the null nodes.

fig. 4

```
Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
    a    /\
        /  \
       d   /\
          /  \
         /\   x
        /  \
       b    c
```

Adding value `e` with a key of `1001` results in further growing.

fig. 5

```
Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d
  1001 = e

Tree:
           R
          / \
         /   \
        a    /\
            /  \
           /    \
          /      \
         /        \
        /\        /\
       /  \      /  \
      /\   x    /\   x
     /  \      /  \
    d    e    b    c
```

Removal

Removal may seem non-intuitive when dead-end nodes are present in the subtree. All previously executed subtree growing must be un-done.

fig. 6

```
Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
    a    /\
        /  \
       d   /\
          /  \
         /\   x
        /  \
       b    c
```

If we were to remove leaf `d` from the above tree, we must replace it with a dead-end node.

Removing leaf `d` (we must replace with a dead-end):

fig. 7

```
```

```
Map:
  0000 = a
  1100 = b
  1101 = c

Tree:
       R
      / \
     /   \
    a    /\
        /  \
       x   /\
          /  \
         /\   x
        /  \
       b    c
```

Removing leaf `c` (shrink the subtree):

fig. 8

```
Map:
  0000 = a
  1100 = b

Tree:
       R
      / \
     /   \
    a     b
```

Removing leaf `b` (`a` becomes the root):

fig. 9

```
Map:
  0000 = a

Tree:
       a
```

With our final removal, we are back in the initial state.

Proofs

Our FFMT proof is similar to a standard merkle tree proof with some extra caveats. Leaf hashes are computed as:

```
HASH(0x00 || 256-bit-key || HASH(value))
```

Where `||` denotes concatenation.

It is important to have the full key as part of the preimage. If a non-existence proof is necessary, the full preimage must be sent to prove that the node is a leaf which contains a different key with a colliding path. If the key path stops at one of the dead-end nodes, no preimage is necessary. Any dead-end nodes up the subtree can be compressed, as they are redundant zero-hashes.

If we were asked to prove the existence or non-existence of key `1110`, with our original tree of:

fig. 10

```
Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
    a    /\
        /  \
       d   /\
          /  \
         /\   x
        /  \
       b    c
```

Key `1110` does not exist in this case, so we must provide the hashes of nodes `a`, `d`, the parent hash of `b` and `c`, and finally a dead-end node `x`. The fact that a final leaf node was a dead-end node proves non-existence.

fig. 11

```
Proving non-existence for: 1110

Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
   (a)   /\
        /  \
      (d)  /\
          /  \
        (/\) [x]
        /  \
       b    c
```

The dead-end node `x` can be compressed into a single bit, since it is a zero-hash.

Proving non-existence for key `0100` is more difficult. Node `a` has a key of `0000`. In this case, we must provide the parent node's hash for `d` and its right sibling, as well as `a` and its original key `0000`. This makes the non-existence proof larger because we have to provide the full preimage, thus proving that node `a` is indeed a leaf, but that it has a different key than the one requested. Due to our hashing of values before computing the leaf hash, the full preimage is a constant size of 64 bytes, rather than it being the size of the key in addition to the full value size.

fig. 12

```
Proving non-existence for: 0100

Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
   [a]  (/\)
         /  \
       d   /\
          /  \
         /\   x
        /  \
       b    c
```

We need only send the preimage for `a` (the value hash of `a` itself and its key `0000`). Sending its hash would be a redundant 32 bytes.

An existence proof is rather straight-forward. Proving leaf `c` (`1101`), we would send the leaf hashes of `a`, and `d`, with one dead-end node, and finally the sibling of `c`: `b`. The leaf hash of `c` is not transmitted, only its value (`c`). The full preimage is known on the other side, allowing us to compute `HASH(0x00 || 1101 || HASH("c"))` to re-create the leaf hash.

fig. 13

```
Proving existence for: 1101 (c)

Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Tree:
       R
      / \
     /   \
   (a)   /\
         /  \
      (d)  /\
          /  \
         /\  (x) <-- compressed
        /  \
      (b)  [c]
```

Our goal is to keep the proof size under 1 kilobyte, at least for the first several years.

With 50,000,000 leaves in the tree, the average depth of any given key path down the tree should be around 27 or 28 (due the inevitable key prefix collisions). This results in a proof size slightly over 800 bytes, pushing a 1-2ms proof creation time on our previously mentioned hardware.

Disk Optimization

Due to the sheer number of nodes, a flat-file store is necessary. The amount of database lookups would be overwhelming for a data store such as _LevelDB_. Our FFMT is much simpler than the Ethereum Base-16 Trie in that we need only store two nodes: internal nodes and leaves.

Internal nodes are stored as:

fig. 14

```c
struct internal_node_s {
  uint8_t left_hash[32];
  uint16_t left_file;
  uint32_t left_position;
  uint8_t right_hash[32];
  uint16_t right_file;
  uint32_t right_position;
} internal_node;
```

Leaf nodes are stored as:

fig. 15

```c
struct leaf_node_s {
  uint8_t key[32];
  uint16_t value_file;
  uint32_t value_position;
  uint16_t value_size;
} leaf_node;
```

The leaf data itself is stored at `value_position` in `value_file`.

We store the tree in a series of append-only files, with a particularly large write buffer used to batch all insertions with a minimal amount of writes.

Atomicity with a parent database can be achieved by calling `fsync(2)` after every commission and inserting the best root hash and file position into the database.

Because our database is append-only, traditional crash consistency can also be achieved by writing a metadata root on every commit. This metadata root contains a pointer to the latest tree root, a pointer to the previous metadata root, and a 20 byte checksum. On boot, the database can parse up the files in reverse order to find the last intact state.

Compaction

The FFMT database can be compacted periodically through user intervention, though it's a rather expensive operation. In our benchmarks, 1,136 commits of 44,000 300-byte leaves each (50,000,000 leaves total), resulted in a database size of 49GB. This could be compacted to use approximately 20GB of storage.

Collision Attacks

It goes without saying that it is most definitely possible for an attacker to grind a key in order to create bit collisions. Currently, the Bitcoin network produces 72-80 bit collisions on block hashes. In the worst case, that would delve 72-80 levels deep in our tree, but while storage increases, the rounds of hashing are far less than that of the sparse merkle tree.

With small modifications, our initial FFMT implementation was able to be converted to a base-2 merkelized radix tree, or Merklix tree. We found that while the Merklix tree offers better DoS protection, in practice, it does not seem to have significant performance or storage benefits over the simplified base-2 trie described above.

We see these potential modifications as a trade-off between space efficiency and simplicity. The radix tree modifications to the trie result in a slight increase in complexity as far as the tree's implementation and proof verification are concerned. The most unfortunate aspect of these modifications to be the requirement of variable sized nodes when stored on disk. Unlike the simplified trie, the radix tree must store a variable number of prefix bits on each internal node.

We hope to observe how each of these trees behave on future iterations of the Handshake testnet in order to better determine the proper data structure for our protocol.

Naming Markets

Similar to ENS, our naming protocol seeks to determine the true market value of names before allowing registration. In particular, our system requires an auction system for names. In order to prevent _price sniping_, we implement a _blind second-price auction_, also known as a _Vickrey Auction_.

In a Vickrey Auction, a participant is only aware of their own bid. The bids are revealed at the end of the auction when a winner is chosen. The winner pays the second highest bid instead of his or her own.

This auction structure was first described by William Vickrey[vickrey]. Vickrey argues that the result of a simple non-blind auction is virtually identical to a blind second-price auction. In an auction where bids are public, bidders will announce their bids until the _second highest price_ is reached. At this point, only one bidder will remain who is willing to pay, and he or she ends up paying the second highest price. We agree with Vickrey's analysis, and our conclusion is that if we are to do a blind auction, it is only logical to make it a second-price auction.

However, a Vickrey Auction system requires that we are capable of executing rather complex consensus-layer smart contract behavior on top of a UTXO set. This kind of functionality rarely exists in the UTXO-based world. Our initial implementation sought to add dynamic functionality at the _transaction_ level. This approach was unmanageable. Whatever dynamic behavior that is added must occur at the output level.

Covenants

Bitcoin Covenants, first explored by Maxwell[maxwell-1][maxwell-2] and later formally described by Möser et al[covenants], are a form of smart contracts that exist on a UTXO-based blockchain such as Bitcoin. The word "covenant" itself refers to a legally binding covenant, in which a party agrees to refrain from or participate in a certain action in the future. Covenants, at their most fundamental level, restrict the path of money as it passes from output to output. Once money enters a covenant, it is _locked_ into a specific path and may not under any circumstances deviate from said path. Actors who have the ability to create and sign transactions must create them according to the covenant's state.

In order for covenants on Bitcoin to restrict the path of money, there are several wildly different mechanisms currently in thought. Bitcoin-NG[bitcoin-ng] proposes consensus-level covenants in the form of a new bitcoin script opcode, `OP_CHECKOUTPUTVERIFY`. The widely discussed counterpart to consensus covenants is cryptographic covenants, which are executed via cryptographic trickery. This trickery ranges from novel usage of ECDSA key recovery combined with a special kind of transaction signature hashing, to the usage of SNARKs.

Although never enabled on Bitcoin due to fungibility and AML/KYC surveillance concerns, we believe that the core idea of covenants is the proper framework for implementing complex smart contracts in conjunction with a UTXO set.

The approach elected for our protocol is most similar to the consensus-level covenants proposed by Bitcoin-NG.

Our construction is a deeply consensus-level covenant, and differs from the earlier proposals, which required layer-two blockchain monitoring in order for dynamic behavior, such as asset ownership, to be achieved. Instead of a layer-two node determining these details, the blockchain itself maintains the state of these assets. In our case, the assets in question are _names_.

Output Structuring

In a UTXO-based blockchain, the typical transaction output consists of a _locking script_, or _predicate_, combined with an output value.

A typical bitcoin output exists as a struct of:

fig. 16

```c
struct output_s {
  int64_t value;
  uint8_t script[];
} output;
```

With `script` being the locking script; the predicate which locks up money for redemption. We add a new field called `covenant`:

fig. 17

```c
struct output_s {
  int64_t value;
  uint8_t script[];
  struct covenant_s {
    uint8_t action;
    uint8_t arguments[][];
  } covenant;
} output;
```

The money can still be locked up by the predicate just the same. However, when money is sent into a covenant, it limits where the output may be redeemed _to_.

Due to the fact that the covenant behavior is only prescribed at the consensus level, this construction should be resistant to fungibility attacks in comparison to other covenants proposals.

Auction System

Using our generic consensus-level covenant system, we are able to implement almost any kind of smart contract on the blockchain layer.

In a normal UTXO system, the order of inputs and outputs has almost no meaning. We enforce positional requirements of inputs and outputs when covenants are used.

The behavior of our covenants is prescribed with consensus code in the implementation of the blockchain itself. Our system is generic enough that new covenant types can be _soft-forked_ into the protocol later.

We prescribe a covenant type known as `BID`, which enters a bid into the system, associated with a name and with its corresponding output. The bid itself carries with it some _arguments_: namely, the name a participant is bidding on and the _blind value_. The blind value is the digest of the participant's _bid value_ concatenated with a 256 bit nonce.

The value tied to the output itself must be greater than or equal to the bid value (although, the blockchain has no way of verifying this initially). Once entered into the `BID` covenant, the value may no longer be redeemed to a normal output. The value associated with this output is called the _lockup value_.

The first participant to enter an opening bid initiates the _bidding period_, wherein other participants are free to join in the bidding.

fig. 18

```
             TX #1 (txid=f3ce)

 Input #0          |  Output #0
   ...             |    covenant_type=BID
                   |    covenant_items={name, blind}
                   |    address=0d1a
                   |    value=15
```

After the bidding period has ended, a _reveal period_ is automatically initiated by the blockchain. Any participant who entered a bid during the bidding period now has a limited amount of time to _reveal_ their bid. This is accomplished by revealing their blind value's full preimage in a `REVEAL` covenant.

fig. 19

```
             TX #2 (txid=c1d3)

 Input #0          |  Output #0
   prev_txid=f3ce  |    covenant_type=REVEAL
   prev_index=0    |    covenant_items={name, nonce}
                   |    address=0d1a
                   |    value=5
                   |
                   |  Output #1
                   |    covenant_type=NONE
                   |    covenant_items={}
                   |    address=c0a8
                   |    value=10
                   |
```

The full preimage includes the participant's 256 bit nonce, which was kept secret up until this point, as well as their bid value. At this point, the `REVEAL` output's value must be equal to the participant's bid value. The remainder of the lockup value can be taken as change. In the case of _fig. 19, the bid had a value of 5 coins, with the 15 coin lockup value successfully concealing the true value of the bid. This participant is able to immediately redeem 10 coins as change.

Once the reveal period has ended, a winner is chosen. This winner is able to redeem their `REVEAL` output to a `REGISTER` covenant. The `REGISTER` output must have a value equal to the second highest bid, or in the case of only one bid, the participant's own bid value. We call this value the _name value_. Similar to the `REVEAL` covenant, the remainder of the bid value can be taken as change.

fig. 20

```
             TX #3 (txid=a7be)
```

```
 Input #0           |  Output #0
   prev_txid=c1d3   |    covenant_type=REGISTER
   prev_index=0     |    covenant_items={name, name_data}
                    |    address=0d1a
                    |    value=3
                    |
                    |  Output #1
                    |    covenant_type=NONE
                    |    covenant_items={}
                    |    address=b1c9
                    |    value=2
                    |
```

Once entered into the `REGISTER` covenant, the name value can _never_ be redeemed normally and cannot be used for transfer of value or for regular payments. It is effectively _burned_ from the system by its inability to leave the covenant's path.

However, in the case that a participant loses, their funds can exit the covenant path with a `REDEEM` output.

fig. 21

```
             TX #3 (txid=c0c1)

 Input #0           |  Output #0
   prev_txid=c1d3   |    covenant_type=REDEEM
   prev_index=0     |    covenant_items={name}
                    |    address=0d1a
                    |    value=5
```

The `REGISTER` output allows for a second parameter known as _name data_. The name data by consensus standards is a 512 byte blob with no required format. By policy standards it should be in a format akin to the DNS message format[rfc1035].

fig. 22

```
             TX #4 (txid=cc1e)

 Input #0           |  Output #0
   prev_txid=a7be   |    covenant_type=UPDATE
   prev_index=0     |    covenant_items={name, [name_data], block_hash}
                    |    address=0d1a
                    |    value=3
```

Once a name is registered, a one-year timeout is initiated before a name renewal is required. Renewals and updates to the name data are achieved through the `UPDATE` covenant action.

The `UPDATE` covenant is similar to the `REGISTER` covenant, but it accepts a third argument, _block hash_. In order to refresh the renewal timer, the owner of the name is required to provide a recent block hash (one that occurred on the main-chain within the past 6 months). We require this to prevent an owner from pre-signing many thousands of years worth of renewals. A renewal should amount to a proof that the owner is still in possession of his or her private key.

Throughout this entire process, the address must remain the same as the one provided in the original bid output. If a change in ownership is desired, the output must be redeemed to a `TRANSFER` covenant. The `TRANSFER` covenant has parameters which require the owner to commit to the address they intend to change ownership to after a 48-hour delay. After 48 hours worth of blocks, the owner can redeem the `TRANSFER` output to a `FINALIZE` output.

fig. 23

```
             TX #5 (txid=0b17)

 Input #0           |  Output #0
   prev_txid=cc1e   |    covenant_type=TRANSFER
   prev_index=0     |    covenant_items={name, address=fe13}
                    |    address=0d1a
                    |    value=3
```

fig. 24

```
             TX #6 (txid=11a3)

 Input #0           |  Output #0
   prev_txid=0b17   |    covenant_type=FINALIZE
   prev_index=0     |    covenant_items={name, name_data}
                    |    address=fe13
                    |    value=3
```

The 48 hour delay mentioned before is necessary in order to dis-incentivize theft of names. During the delay, an owner may redeem the `TRANSFER` output to a `REVOKE` output. The `REVOKE` output renders the name's output forever unspendable, and puts the name back up for bidding.

fig. 25

```
             TX #6 (txid=d1da)

 Input #0           |  Output #0
   prev_txid=0b17   |    covenant_type=REVOKE
   prev_index=0     |    covenant_items={name}
                    |    address=0d1a
                    |    value=3
```

For increased security throughout this process, we define a new script opcode: `OP_PUSHTYPETOSTACK`. This particular opcode makes use of _transaction introspection_ in order to push the covenant type of the _next_ output to the script execution stack. This allows for a name owner to assign a _hot key_ and a _cold key_. The former key is used for updates to name data, while the latter is intended to be used only for transfers and revocations.

An example script utilizing this feature may look something like what is displayed in fig. 26.

fig. 26

```
OP_7
OP_PUSHTYPETOSTACK
OP_GREATERTHANOREQUAL
OP_IF
[cold-key]
OP_ELSE
[hot-key]
OP_ENDIF
OP_CHECKSIG
```

More advanced script code can also be used for example by requiring transfers and revocations to require signatures from multiple parties.

A participant can assign this script to a pay-to-scripthash address and use it when they enter their initial bid, or perhaps later transfer their name to it.

We intend for scripts to be the primary mechanism for robust security of name ownership. In contrast, we intend for the `REVOKE` output to be a last resort on the part of the name owner. It exists primarily to make the for-profit theft of names all but impossible.

Commission

Name data is periodically committed to our authenticated tree at regular intervals. Because our tree is implemented as a series of append-only files, a commission interval is required to prevent history bloat, which may otherwise require the user of the software to compact their history regularly.

Names and name data is batch inserted into the tree four times per day on a six-hour interval on average (defined by blockheight). This means that the _time-to-live_ for any resource on the blockchain is at least six hours in practice.

Context Optimizations

In cryptocurrency implementation, there is a notion of _contextual_ and _non-contextual_ validation. Non-contextual verification functions perform basic sanity checks on transaction data before executing any resource-intensive code such as database lookups or elliptic curve operations. This is done for logic separation as well as for a measure for denial-of-service prevention.

In contrast, contextual verification is only executed once the majority of the network state, such as UTXOs, is readily available.

In the implementation of our protocol we found it was easier to separate blockchain validation into three logical categories: non-contextual, contextual, and super-contextual.

We define _super-contextual_ verification as the validation functions which execute only once a _global state_ is readily available. Our global state is the state of the auctions and names. Whereas UTXOs are localized to a specific transaction, auction and name state is globally accessible by any transaction.

Our prescription of covenant types is specifically designed to make super-contextual verification easier and more performant.

Our system deals with a number of transaction locktimes which are enforced by the covenants in a transaction. One particular example of our difficulty with this was a number of edge cases we discovered in our implementation of the transaction mempool. Because a blockchain reorganization can cause a change in height of the main chain, a reorganization has the potential to invalidate hundreds or even thousands of transactions in our mempool. Bitcoin also has these issues when dealing with transaction locktimes and spends from coinbases (which are required to have a maturity of 100 blocks). These edge cases are much more severe in our system, and initially required a full revalidation of the entire mempool whenever a reorganization occurred.

Typically, cryptocurrency mempool implementations do not hold any UTXOs in memory, and optimize for only the state required to assemble a block. By designing our specified covenant types with some redundant data, and by separating super-contextual verification from contextual verification, we were able to optimize this process.

In order to validate covenant-related locktimes, our implementation requires no access to contextual information such as UTXOs. It only needs access to the global state, and non-contextual data such as the outputs on the transactions themselves.

This separation of logic enforced by a deliberate design, among other things, allows us to avoid having to store an in-memory UTXO cache specifically for the mempool.

We believe that having small amounts of redundant data in the covenant parameter vectors, along with some redundant covenant types, also helps with implementation of wallets, particularly SPV wallets.

Naming Architecture

The naming system of the internet is DNS[rfc1035]. DNS currently operates with a multi-layer model. Operating systems typically expose a _stub resolver_. A stub resolver has no recursive capabilities, and is only capable of sending simple DNS message queries to remote nameservers. They are intended to be pointed at _recursive servers_. Recursive servers perform full DNS iteration on behalf of stub resolvers by traversing each _zone_'s nameserver until an _answer_ is received. A zone's nameserver is referred to as an _authoritative server_. Authoritative servers are capable of serving their own records, but also sending _referrals_ to _child zones_.

Recursive servers are typically public and maintained by either internet service providers or other organizations, such as Google, Cloudflare, or OpenDNS.

Currently, recursive DNS resolvers, such as Google's Public DNS[google], hit a number of root servers[root] maintained by various entities. These root servers serve the _root zone_. The root zone is collection of _top-level domains_ (TLDs). The information necessary to resolve these TLDs is stored in a root _zone file_ distributed and maintained by IANA[iana], a branch of ICANN[icann]. ICANN currently acts as a gatekeeper as to which domains are allowed an entry in the root zone file.

A zone file, in its most general definition, can be thought of as a database which houses all the information necessary to serve the domain name records in a given _zone_. In the case of the root zone, the domain names are TLDs such as `com`, `net`, and `org`, and the zone file itself is quite literally a plain text file[internic] in the RFC 1035[rfc1035] presentation format.

Provability

The current method for proving DNS is known as _DNSSEC_[dnssec]. DNSSEC includes cryptographic signatures of DNS resource records in every DNS message. This prevents an attacker from altering any DNS record as it is in flight.

In order to avoid having to distribute every domain name's public keys to every DNS resolver, a _chain of trust_ is verified starting with the root zone as the trust anchor. This means IANA's public keys must be stored by any participant of this network, and limitations must also be placed in the notion that the _root key-signing keys_[anchors] never become compromised[signing]. Furthermore, in order for a domain to be considered secure, IANA acts a gatekeeper for security for top non-auction name registrations only, signing off on keys in order to add them to the trust chain.

Handshake Architecture

The Handshake naming protocol differs from its predecessors in that it has no concept of _namespacing_ or subdomains at the consensus layer. Its role is _not_ to replace all of DNS, but to replace the root zone file and the root servers.

The goal is to maintain our own root zone file in a decentralized manner, making the root zone uncensorable, permissionless, and free of centralized gatekeepers such as ICANN and Certificate Authorities. In our protocol, every _full node_ peer on the network acts as a root server, serving a _provable_ version of the root zone file. Our blockchain is essentially a larger, but distributed, zone file, permissionless, which any participant has the right to add an entry in.

Proof of Work as a Trust Anchor

Proof-of-work is an interesting mechanism, as it is one of the few mechanisms known which is completely immune to _man-in-the-middle_ attacks. Because of this, a proof-of-work blockchain is able to act as a decentralized trust anchor for anything we may need to prove. In our case, we are able to use it as a permissionless mechanism to "sign off" on child DNS keys.

DNS currently has a feature for storing fingerprints of a child zone's DNS keys in its parent zone. Because the root zone in our protocol is a blockchain, the protocol itself can act as a trust anchor for these keys, allowing anyone to prove not only their name on on the blockchain, but any subdomain they may have as well. This involves simply committing one's key fingerprints to a record on the blockchain.

Even after the trust chain has been validated all the way down each zone via regular DNS, a user of this system still ends up with similar security to a blockchain. This is because the trust anchor is, in fact, a blockchain.

Compatibility

In contrast to other naming projects, our goal is to work with DNS, not against it. We intend to transparently provide an alternative to existing centralized systems, all while requiring zero or minimal intervention from most users.

DNS is a very mature piece of internet architecture. Several tools we need are already built. For example, it is already possible to store SSH fingerprints in DNS[sshfp]. This feature is currently supported in OpenSSH[openssh]. This allows one to verify SSH fingerprints in a decentralized way without installing any extra software beyond the Handshake daemon.

DNS also has a feature for verifying SSL/TLS certificates[tlsa] by storing a hash of the SubjectPublicKeyInfo in a DNS resource record. Using this feature, one is able to run a recursive DNS resolver locally, with root hints pointed at a blockchain. If this were the case, there would be no reason to mistrust a self-signed certificate as long as a valid DNSSEC chain were present.

We believe these technologies, when used together, can remove the need for Certificate Authorities.

Implementation

Our consensus protocol is usable as a suitable replacement for ICANN root zone servers. An alternative to the ICANN Root Zone is, of course, not a new idea. This avenue has been previously explored by the Open Root Server Network, or _ORSN_[orsn].

In order for the root zone to be replaced transparently, recursive resolvers must point at an authoritative nameserver which serves records committed to the blockchain rather than ICANN's root zonefile. This is a difficult dynamic to work with, as virtually no consumer devices currently ship with a recursive resolver running locally.

There are multiple full DNS implementations include ISC's BIND[bind], as well as LDNS[ldns] and Unbound[unbound] maintained by NLnetLabs[nlnetlabs]. We were inspired by these implementations and created our own full DNS implementation[bns] throughout the course of our research.

Network Bootstrapping

In order to bootstrap the network, all entries in ICANN's existing zonefile are _pre-reserved_ by consensus rules. Names in the list of Alexa top 100,000[alexa] domains are also pre-reserved for further inclusion of existing stakeholders (with deduplication and common words down to above ~80,000 names). The latter names are converted to top-level domains by selecting their first domain name label.

Owners of these reserved names are be able to claim them directly on the blockchain, bypassing the auction process. We aim to migrate existing nameholders to the handshake blockchain in a permissionless manner. To accomplish this, we propose _DNSSEC Ownership Proofs_.

While DNSSEC itself is intended to mitigate man-in-the-middle attacks, we have found that, with some modifications, DNSSEC proofs can be used as secure proofs of name ownership.

Smaller names may be unreliable as it is unknown whether they are the perceived rightful owner of the names. Our project has a sunrise period whereby rightsholders may claim their name.

DNSSEC Ownership Proofs

We propose DNSSEC ownership proofs as a much stricter subset of DNSSEC proofs in that they do not allow for CNAME glue or wildcards. Furthermore, every label must be separated by a zone cut using a typical DS-to-DNSKEY setup for referrals. All zone referrals are retrieved and combined, in aggregate, to produce the final proof.

These proofs must stem from ICANN's _key-signing keys_ (KSKs) to the final ZSK in the target zone. The final _zone-signing key_ (ZSK) must sign a TXT record which commits to the name's desired address on the blockchain. The proof is broadcast to the peer-to-peer network and included by miners in the coinbase transaction of a block. Consensus rules dictate that the miner must create an output for the associated proof, thereby granting the name to the committed address.

By consensus rules, the proofs are verified against ICANN's existing trust anchors (KSK-2010 and KSK-2017). Although KSK-2017 is not currently in use, ICANN did publish the key last year (2017). This allows us to include it in the blockchain's consensus rules from day one.

Relying on only trust anchors for verification results in large proofs (typically ranging between 3 and 10 kilobytes), however, this method allows nameholders who lack an existing DNSSEC setup to upgrade and claim their name in the future.

We design the DNSSEC claim system to be operational for a total of 4 years on the blockchain.

Security Concerns

For a time, ICANN will indirectly be a limited arbiter of this system due to their control of the root trust anchors used for ownership proofs. This raises potential concerns.

During our analysis of the root zone file, we discovered that a significant majority of domains use SHA1 for RSA signatures and key fingerprints. This is unfortunate, as SHA1's security against collision resistance was recently compromised[shattered]. Our consensus rules must disallow for the use of insecure algorithms, like SHA1, even with existing DNSSEC setups.

As a result of this, in order for an RSA-SHA1 nameholder to claim their name on the handshake blockchain, they must upgrade their key-signing key to at least RSA-SHA256 before creating an ownership proof. Unfortunately, to accomplish this, the nameholder must contact their parent zone and request that they sign off on a new key.

With this in mind, we must consider the possibility that ICANN may become uncooperative and refuse to sign a higher security key for an existing nameholder. If this were to happen, RSA-SHA1 root zone names would be unredeemable on the blockchain. To mitigate this attack, our DNSSEC ownership verification algorithm implicitly upgrades RSA-SHA1 keys to RSA-SHA256, allowing a reserved nameholder to publish the same RSA key in their own zone with a differing algorithm field (RSA-SHA256 or RSA-SHA512). This allows the nameholder to bypass ICANN's root zonefile update process when creating the necessary ownership proof.

In addition to the SHA1 vulnerability, we discovered that several major root nameholders use 1024-bit moduli in their RSA zone signing keys. We believe this is the result of BIND's default behavior when generating keys with `dnssec-keygen`. RSA-1024 has long been considered insecure, and while no practical factorization of a 1024 bit modulus has ever been executed, we consider this a weakness, particularly for an economically incentivized system.

If we consider, by way of consensus rules, requiring 2048 bit moduli or higher, we find ourselves in a similar situation to the SHA1 vulnerablity: nameholders may wind up at the mercy of uncooperative parent zone maintainers.

As a final fail-safe against an attack by uncooperative entities, we allow RSA-1024 and offer a versionbit-activated soft-fork. This soft-fork is responsible for hardening the consensus RSA implementation by requiring at least 2048 bit moduli in ownership proofs. In the case that 1024 bit RSA is compromised, we turn to miner consensus to resolve the issue. This allows the blockchain to support RSA-1024 until a practical attack is demonstrated against it.

As a necessary effect of the activation of this fork, all names which were originally redeemed with RSA-1024 will be immediately revoked until redeemed again with a stronger key. This construction requires us to place a 6 month locktime on trasfers for names redeemed with RSA-1024.

The final risk we have considered, and perhaps the most major, is ICANN's key revocation process. ICANN's stated KSK-2017 rollover plan involves setting the `REVOKE` bit on KSK-2010. Unfortunately, publishing a revoked key does very little to truly invalidate old states. A clever attacker can withhold revocation key records and signatures, serving only older states. Because of this, DNSSEC's revocation mechanism is all but useless to our blockchain.

Our primary concern is that ICANN may decide to revoke KSK-2010 at some point by publishing its corresponding private key. To deal with this issue, a final consensus rule can be added to _disable_ KSK-2010 once a separate proof is published which demonstrates that KSK-2017 is now active. This proof would include the KSK-2017 signature of ICANN's DNSKEY RRset. We are convinced that ICANN will not be able to ethically justify publishing KSK-2010 before the rollover to KSK-2017 is complete. As such, we find that this mitigation provides adequate security against an attack of this sort.

Economic Incentives for DNSSEC implementation

DNSSEC has rather sparse support, with only a handful top-level and second-level domains supporting it to its fullest extent[nameandshame]. Those that do implement DNSSEC often implement it _insecurely_ (by way of SHA1 or RSA-1024).

We find this to be a major impediment to the adoption of a system which bases its security on a validating recursive resolver. To incentivize proper implementation of secure DNSSEC, reserved names may only be redeemed on the blockchain by a proper DNSSEC setup. We hope that this adds a great deal of security to the existing root zone and to a fair majority of the Alexa top 100,000.

To further increase incentives, the blockchain attaches a coin reward to the redemption of any reserved name. The value attached to the name is weighted according to how many child zones are present in the zone.

SPV Name Resolution

Our SPV implementation's architecture consists of the following 4 components:

1. A peer-to-peer layer for synchronizing block headers and verifying name tree proofs.

2. An authoritative nameserver which translates on-chain resources to DNS responses, depending on the request. This nameserver behaves as if it were a root server, serving zone `.`.

3. A recursive server which sets its root hints and trust anchors to its own authoritative root server, rather than ICANN's.

4. A second non-recursive resolver which resides in the authoritative layer and acts as a fallback to ICANN's system. This is used in the event that a reserved top-level domain has not yet been claimed. Resolutions through ICANN's system only occur if a proper name absence proof was received from a peer.

fig. 27

```
 +-------------+     +------------------+     +----------------------+
 | OS Resolver | --> | Recursive Server | --> | Authoritative Server | --+
 +-------------+     +------------------+     +----------------------+   |
                                                                         |
  +-----------------------------------------------------------------------+
  |
  |    +--------------------+     +---------------+     +-------------+
  +--> | Peer-to-peer Layer | --> | Proof Request | --> | Remote Peer |
       +--------------------+     +---------------+     +-------------+
```

fig. 28

```
 +-------------+     +----------------+     +--------------------+
 | Remote Peer | --> | Proof Response | --> | Peer-to-peer Layer | --+
 +-------------+     +----------------+     +--------------------+   |
                                                                     |
  +-------------------------------------------------------------------+
  |
  |    +----------------------+     +------------------+     +-------------+
  +--> | Authoritative Server | --> | Recursive Server | --> | OS Resolver |
       +----------------------+     +------------------+     +-------------+
```

Our peer-to-peer layer is end-to-end encrypted by default, using a Noise Protocol[noise] handshake, similar to the handshake used by the Lightning Network[lightning]. To further enhance privacy at this layer, a peer resolving a name on one's behalf is only permitted to see the top-level domain (or rather, a hash of the name).

Due to the peer-to-peer encryption, the only plaintext aspect of this architecture resides in the recursive resolver traversing child zones. To improve privacy here, QNAME minimization[qname] can be utilized by the recursive resolver. QNAME minimization is surprisingly non-trivial, because not every domain name label originates from a zone cut, making it non-obvious in how to "trick" an authoritative server into sending a proper nameserver referral. Luckily, validating resolvers such as Unbound currently implement this functionality.

Despite child zones being unencrypted, they can still be validated via DNSSEC. Because the ICANN root zone is replaced with a blockchain, the recursive resolver must also change its trust anchors. Currently, recursive resolvers set their trust anchors to IANA's two DNSKEYs (KSKs) which sign the root zone's _zone signing keys_ (ZSKs). For compatibility purposes with existing resolvers, we propose that an SPV node's authoritative nameserver use a static trust anchor whose private key is known by all.

Of course, signatures at the root zone are no longer necessary due to proof-of-work acting as the security mechanism rather than digital signatures. However, DNSSEC signatures at the root must be included in order for existing DNS implementations to consider the trust chain intact. With the private key publicly known, conforming SPV nodes can create the appropriate RRSIG records in real-time when serving a response. These responses can be cached to avoid repetition of expensive signing operations.

While dummy RRSIGs can be created to fool existing recursive resolvers in to validating a complete trust chain, NX proofs present a different challenge. The most modern non-existence proof verification currently in DNS is NSEC3[nsec3]. NX proofs operate by maintaining a sorted linked list of child zones. When an NX proof is generated, the authoritative server responds with a signed DNS record showing where the requested child zone _should_ reside within the list, thereby proving its non-existence. Unfortunately, NSEC3 obfuscates child zones by hashing labels with the now insecure SHA1. Were NSEC3 to use a more modern hash function, a DNS NX proof at the root zone _could_ be generated by a full node by iterating over our name FFMT. Unfortunately, SPV resolvers would still have trouble generating this dummy proof, due to their lack of ability to arbitrarily seek through the tree.

To work around this difficulty, we generate _dummy_ NSEC3 proofs which imply an empty zone.

Root Zone Fallback

Due to the fact that not all reserved root names will be claimed by nameholders immediately, we propose two solutions for eventual transparent rollover to the new system: _Hardcoded Fallback_, and _Dynamic Fallback_.

Hardcoded fallback involves hard-coding the existing ICANN zonefile into the SPV software. When an absence proof is received by the authoritative server for a pre-existing TLD, the hardcoded fallback is checked for records and returned as a DNS response.

Dynamic fallback is similar, but involves querying ICANN's root servers and validating the response against their trust anchors. This provides better liveness, but bestows more power upon a centralized authority.

In both constructions, once a pre-reserved name is claimed by the proper owner, the software will transparently begin to follow the blockchain instead of either hardcoded records or ICANN's records.

Library Integration

We offer a library integration as an alternative to an SPV resolver or full node. This removes the need for an average user to install extra software, and instead places the responsibility on the developers of software.

We consider this the _lowest security mode_ of our protocol. It is, however, still more secure and more permissionless than the current state of DNS, albeit slightly more centralized than the local recursive resolver model.

Unix-based operating systems, including Mac OSX, configure their OS stub resolver to use nameservers listed in `/etc/resolv.conf`.

The resolv.conf format points to a list of recursive servers for the operating system's stub resolver to make use of. The stub resolver is invoked during a call to `gethostbyname(3)` and `getaddrinfo(3)`.

We propose a new standard OS configuration file, `hns.conf`, residing in `/etc/hns.conf` on Unix, and `%SystemRoot\Drivers\etc\hns.conf` on Windows. This configuration file is reminiscent of the standard resolv.conf, however, its sole purpose is to list nameservers while other options are simply inherited from the regular resolv.conf parsing.

In order to make use of our protocol's resolvers, we require a 33 byte compressed ECDSA public key for each nameserver listing. This key will be used for a _SIG(0)_[sig0] verification of a signed DNS message. DNSSEC is currently insufficient for this purpose, being that it only signs a DNS response's individual records. TSIG[tsig] is also insufficient due to its requirement of symmetric keys. Our SIG(0) usage appends a regular SIG record to the additional section of a DNS message as a pseudo-section. This record signs all data before it on the ECDSA secp256k1 curve.

A user with a local SPV resolver running may point his or her `hns.conf` to `127.0.0.1` (where no SIG(0) verification is necessary). If not present, the hns.conf parser should fall back to a list of hardcoded recursive servers run by respected community members. These entries will be paired with a known ECDSA public key.

Ideally, requests to the public recursive servers should be routed through an _Anycast_[anycast] network in order to provide comparable speed to Google or Cloudflare's public DNS.

This setup is necessary to allow all users to make use of the new root zone. If some users are not willing to run a SPV node, our system falls back to a more centralized model. Note that this centralized model still carries with it a number of improvements over the current ICANN root zone. The root zone is still permissionless, and the data is still authenticated (something which is not present in today's stub resolvers).

The final integration library consists of a simple stub resolver which is aware of our new hns.conf format, as well as capable of verifying SIG(0) records.

Integration

We are convinced that, in order for adoption to be widespread, the SPV daemon and integration library must be runtime-less and written in portable C. The integration library is especially necessary if this protocol is to be used on embedded and IoT devices.

Future Directions

Proof of DNS iteration

To remove the requirement of a local recursive resolver for SPV, a hypothetical _proof of DNS traversal_ could be useful. A proof of this kind would involve aggregating glue records, DNSSEC signatures and keys for each zone, producing a final semi-compact proof. This would allow clients to securely off-load the recursion to untrusted servers. This construction is similar to our DNSSEC ownership proof, which is also an aggregate proof of DNS referrals.

In comparison to a local recursive resolver, this would save on CPU time and bandwidth due to the signature aggregation. It would furthermore reduce memory usage by avoiding the need for a message cache. This would result in massively reduced complexity for SPV resolvers as they also no longer need to maintain a recursive or authoritative root server.

Zone Replication and DNSCurve

Daniel Bernstein's DNSCurve[dnscurve] is an extension to DNS which performs an ECDH and establishes a stream cipher with an authoritative nameserver before exchanging messages. It is backwardly-compatible with DNS as identity keys are encoded as base32 labels in normal NS records.

Unfortunately, DNSCurve adoption is minimal. As mentioned before, the only unencrypted portion of our name resolution protocol occurs during regular DNS iteration through authoritative servers.

If we envision a peer-to-peer network where peers can act as "proxies" for a zone, they would be capable of layering DNSCurve support on top of existing zones' nameservers as a public service. They could then advertise themselves as a DNSCurve enabled proxy for a specific zone on the peer-to-peer layer.

There are many security and incentives questions this raises, but we consider this an idea worth pursuing in the future.

Subdomains

Subdomains are out of scope for this system. Blockchains should be storing the minimal data needed, and domains/subdomains with defined owners should be able to prove the validity of that name out-of-band. If the data is already disclosed in the root zone, then it makes sense for that data to be in a TLD anyway. The load on a chain is the same whether it is a TLD, domain, or subdomain. As there are no efficiencies, any potential use of subdomains on-chain should have a separate name record in the root zone.

Reorg Safety and Name Expiration

It is strongly recommended that client implementations verify whether a chain has a deep reorganization. This can be achieved by having third parties attest to the current blockheight which is widely regarded as non-controversial. This is materially different than existing CA infrastructure, as the security is additive in that everyone is attesting to the same information. Any party can provide this information and attestation (and can even be published/proven on-chain).

In the future, it may be desirable by community consensus changes to have a hybrid proof-of-stake construction similar to the Casper Finality Gadget[casper-finality-gadget] proposed in Ethereum whereby blocks are committed by bonded coins.

It is strongly recommended that certificate pinning[certpinning] is used to associate identity by clients and user agents. This is achievable by pinning the name owner's scripthash to the name itself, so when the ownership record changes, the user must approve this change.

Stakeholders

The principal function behind the Handshake mechanism is maximizing allocation of ownership of tokens, coins, or non-fungible assets towards the most relevant stakeholders. To make effective change, all relevant existing and future stakeholders must be acknowledged. By maximizing correct stakeholder allocation, one maximizes the efficacy of the change. In the case of Handshake, it is the shift from centralized Certificate Authorities and naming, towards a decentralized infrastructure.

All stakeholders are incentivized for development and growth of the project in their own self-interested incentive. In order to do so, many of the individuals require some ownership or value of the coins in order to establish sufficient motivation.

These allocations are this paper's proposed allocations, and the Handshake community will ultimately determine the final allocation in mainnet.

A total of 1,360,000,000 coins are minted in the genesis block to be distributed to relevant stakeholders

Pre-Launch Blockchain Development - 7.5%

This allocation goes to fund development across various stakeholders who have been involved with creation of this project. These coins are used to pay for work prior to mainnet launch and is the only source of development funds. A iterated tit-for-tat game exists whereby there is self-interested benefit for distribution of value ("the more I give away, the more value I accrue") across many projects and development teams emulating this model.

Financial Contributors and Pricing - 7.5%

A total of $10,200,00 USD have been allocated to purchasers to price the initial value of the coins for 7.5% of the total supply, with a total valuation of the initial coin supply at $136,000,000.

100% of the dollars raised are being given to non-profits and FOSS projects, and FOSS communities such as hackerspaces. This is effectively a one-way non-destructive "proof-of-burn" on the dollar side to price the coins.

The role of coin purchasers is critical as an initial stakeholder in the growth of the project. The purchasers have been curated to maximize effective change by primarily allocating funds to Venture Capitalists and Token Funds with specialty in the cryptocurrency and decentralized internet ecosystem. Many of these purchasers have been effective in disrupting entire industries and have been involved in large-scale growth of internet services (some even across generations).

The existence of these participants are necessary and fundamental in pricing the tokens, as the distribution event requires real value to be established (a sale of 1% of total initial supply is not credible in pricing the tokens). Additionally, the sale has occurred as close to launch/announcement as possible.

Other projects replicating this mechanism may require greater capital to fund development and/or greater claim to the Pre-Launch Development allocation. This may result in not having a one-way "proof-of-burn", and instead use the capital to fund development of the project.

The role of pricing the coins for distribution is necessary as the coins need to have understood value during the distribution process. While it would ostensibly be ideal to spin-up projects and deploy blockchains without this mechanism, there may be insufficient coordination and ex-ante expectations of value. The role of the high-reputation Venture Capital provides a tastemaker function which provides a signal and Schelling Point for potentially economically and socially valuable projects. These entities are a significant stakeholder in the current ecosystem and a continuing game for project selection and curation may persist as a result ("putting your money where your mouth is").

Free and Open Source Software Developers - 68.0%

The free and open source community is the principal coin owner of the project upon launch. These coins are distributed without any expectation of work. As free and open source software is the principle of giving away code without any direct financial return in exchange, similarly Handshake is about giving away financial value without any expectation of code in exchange.

If the community is interested and Handshake becomes viable in the future, it is possible (but without any obligation whatsoever, contractual or implied) for individuals to have incentive to integrate the functionality into their own software.

Domain Name Holders - 10%

The Handshake blockchain will allocate all TLDs to the rightful holders upon submission of a sufficiently secure DNSSEC proof on the blockchain. Additionally, the Alexa top 100,000 domains were used and filtered (duplicates, generic dictionary names) to over 80,000 non-generic names to be claimable as a TLD via a DNSSEC proof. This way, all currently existing domains can work on Handshake provided it is claimed in a timely manner, as all TLDs can be claimable on Handshake by the owner of the TLD.

In addition to backwards compatibility with the existing domain name system, coins are also provided to incentivize adoption by name holders. This creates an incentive for name registration on Handshake. Unclaimed coins after the claim period will be burned.

Some domain names may not be claimable until secure DNSSEC records (not SHA1 keyhash) are provided by the domain's TLD DNSSEC record, and there may be a delay period before the coins are matured and available to use.

2.5% will be allocated to TLDs to be distributed evenly.

2.5% will be allocated to the top 100 Alexa names.

2.5% will be allocated to the Handshake reserved names (over 80,000 names) to be distributed evenly. These names will also be able to claim a TLD on Handshake.

CA/Naming Corporations and Other Blockchain Projects - 5%

The following corporations and projects are being allocated Handshake coins. They have not acknowledged or accepted the coins at this time of writing. In the event the coins are not claimed or accepted, these coins will not be reallocated and are effectively burned. Some of these allocations may be only redeemable by submitting a DNSSEC proof of their domain to the blockchain to claim coins. There is no contractual expectation for any of these entities to help the Handshake project in any way and is explicitly an obligation-free distribution with no strings attached.

ICANN has been the root namespace for the internet. ICANN (CA, US) is allocated 24,480,000 of the initial coin supply by the Handshake community.

Cloudflare, Inc. (DE, US) is a corporation doing fundamental research for naming, caching, and certificate authorities. They are allocated 6,800,000 of the initial token supply.

Namecoin is a decentralized naming blockchain. 10,200,000 of the initial supply was allocated to leading current and prior Namecoin developers.

Verisign, Inc. (VA, US) is the registrar for .com and .net. They are allocated 6,800,000 of the initial token supply. The .com and .net TLDs on Handshake will be given to Verisign with a DNSSEC proof.

Keybase has been innovating in the naming and certificate authority space. Keybase, Inc. (DE, US) are allocated 0.25% of the total token supply.

Public Internet Registry (VA, US) maintains the .org namespace. They are allocated 3,400,000 of the initial token supply. The .org TLD on Handshake will be given to PIR with a DNSSEC proof to pir.org.

Afilias plc (IE) has been the service provider for the .org namespace. They were allocated 3,400,000 of the initial supply to a DNSSEC proof of afilias.info. Note for both PIR and Afilias, this allocation was made before the proposed sale of the .org namespace.

Brave is a browser which has cryptoeconomic incentives. Handshake allocated 3,400,000 to Brave Software, Inc. (CA, US). This is a distribution to the entity owners itself, and is not an implied distribution to the Basic Attention Token holders.

Namecheap is a large domain registrar and has support cryptocurrency in the past. They were allocated 2,720,000 of the initial supply.

Godaddy (AZ, US) is a large domain registrar and has a Certificate Authority as well. They were allocated 2,720,000 of the initial supply.

Blockstack is a corporation developing a naming blockchain as well as a decentralized internet stack. The Handshake distribution allocated 408,000 of the initial supply to Blockstack Token LLC (NJ, USA) under a DNSSEC proof for blockstack.com. This is to the entity owners itself, and is not implied distribution to the Blockstack Token holders.

ENS (True Names LTD (SG)) is developing an alternate naming root (.eth) using an Ethereum smart contract. They were allocated 136,000 of the initial Handshake coin supply to the entity running ENS (ens.domains).

GnuNet provides PKI infrastructure and provides identity via a DHT. They were allocated 136,000 of the initial coin supply to gnunet.org (may require DNSSEC upgrades on the .org namespace).

Non-Profits and FOSS Projects - 2%

Not to be confused with the $10,200,000 USD given to non-profits and FOSS projects, Handshake coins (HNS) will be given to some of those entities in addition to the USD.

Miners

Historically, many blockchains have distributed 100\% of the total coin supply to Proof-of-Work miners or the overwhelming majority to investors (with none going to free and open source developers directly). These miners capture value by competitively discover lower operating expense costs with electricity or optimizing computation with better hardware.

Miners receive a block reward for validating the chain correctly. There is no promise, implied or explicit of guarantees around future rewards to miners and may be changed upon future technological progress. Additionally, there are no guarantees on expenditure upon continued operation of hardware.

Further Distributions

Further incentivized distributions presupposing the incentives derived from Metcalfe's Law are theoretically possible via a hard-fork but outside the scope of this document, as it would be wrong to be prescriptive on future actions, especially as the future is unknown with the ability to achieve coordination of this activity and programmatic ability to acquire keys.

Coordination

The purpose of this project in addition to developing a decentralized naming system is to perform and demonstrate a method to conduct wide-scale coordination without an end-state of centralized power structures, with mechanisms and incentives for coordination across millions of people without authorities or contractual obligations.

The blockchain allows for cheap verification, but it does not inherently rally people to a change in infrastructure. It is exceptionally costly to build a top-down organization of millions in a hierarchy to replace entrenched interests, hence a ground-up structure is proposed using gift economies, which is only possible with emerging technologies which is capable of accounting value without central authorities (the blockchain), as well as tools for wide-scale coordination across millions (internet free and open source communications software). Handshake is an open performance by all parties worldwide to participate and deliver a new naming system as a gift to wider society.

The Theory of the Firm[natureofthefirm] postulates that organizations exist due to transaction costs, it is often cheaper to conduct transactions within an organization rather than between organizations as intra-organizational goals are aligned, rather than inter-organizational goals, hence the high transaction costs from due diligence with external parties. Firms become large fundamentally due to trust and ensures everyone is rallying to the same cause. Power centralizes into the agents of these firms, we believe we can do better with technology.

The firm model for alignment faces significant issues around externalizing costs ("is it good for society") and principal-agent problems ("is this person acting properly on my behalf"). This requires the principal to always watch the activity of the agent, to ensure the principals' goals are being acted upon. The more trust complexity required by an activity, the greater the likelihood that it is within the boundaries of a firm. Shareholders watch the board who watches the executives who watch the employees, with a hardcoded contractual structure for organizing and ensuring correct action.

The purpose of smart contracts[smartcontracts] is that it allows the principal to also be the agent, as computation can allow one to read the code once and ensure that the code is being followed, significantly reducing potential information costs. However, smart contracts do not solve social coordination inherently, they merely make it cheap for those whom have already agreed with what is being coordinated.

By constructing a capital structure without obligations (gift economies), method is proposed to create immensely valuable social structures with proper social incentives for restructuring the current organizational centers of power within society by participants of this gift economy.

Inability to Verify Actions

It is not possible to ascribe value directly to code. While it is possible for future projects to reward dependent upon actions using peer-to-peer oracles, code is far too subjective. Instead this project proposes removing ongoing verification of reward for code. While future iterations of this design may have ongoing elements of verification/reward via optimistic tit-for-tat mechanics, it is believed that it is possible to build systems without inherent enforceability of labor exchange.

As this is constructed as a blockchain, there is no single entity controlling this system. The system exists due to the collective belief that this system is valuable, the unit of account has value, and there is sufficient desire to improve/maintain this system. Many forms of Commons-Based Peer Production[wealthofnetworks], including Free and Open Source Software, is a Bazaar[bazaar], but we've always had the reward mechanisms be a Cathedral up until now. There have been prior work around reward mechanisms in exchange for production in Commons-Based Crypto Currencies[primaveradefilippi], and much exploration has occurred in understanding the implications of the blockchain having no single owner (hence similarity to the notion of a commons), but payment for free and open source code are primarily within the context of payment for proposed acceptable work in exchange for cryptocurrencies. Instead, this project is about ensuring that free and open source developers are the majority owners of the system without any direct contractual expectation of return or work.

Previously, FOSS developers have been rewarded by working in large companies such as Red Hat Software who are able to evaluate payment towards end goals of financial returns. This construction creates a direct conflict of interest between the values of FOSS software and the values of profit-maximization of the firm. Further, the principal-agent problem persists with firms funding FOSS software, as they are making payments on an ongoing basis towards the end goals of the firm over the best end goals for open source projects.

By constructing a mechanism with coin distribution to FOSS developers with no expectation of return, this creates a complimentary system whereby individuals building FOSS software are able to build software for the commons while also having majority ownership and economic benefit of co-creating this system during the development phase, there is simply an incentive to participate and create a valuable system for developers who own the coins. As a result, FOSS developers can do what they believe to be in the best interests of a system without power hierarchies determining what is the best use of resources, in return there is an understanding that the payment will be imprecise for contributors (much like how those using code receive benefits with imprecise payment for the commons).

Comparison with Traditional Capital Coordination Models

Two-sided marketplaces are a fundamental problem in displacement of existing systems. The archetypical example of overcoming two-sided marketplaces is Uber displacing the taxi, which has two sides of riders and drivers -- it is difficult to persuade riders to participate if there's insufficient drivers (long wait times), and difficult to persuade drivers on the platform if there's insufficient riders (low revenue). One of the earliest efforts for resolving this two-sided marketplace problem for internet services was initiated by Paypal, which resolved a side of the marketplace by directly giving money to new users of PayPal.

Overcoming two-sided marketplaces traditionally has required significant capital and involves expectations of future market capture. As a result, these companies have raised significant amounts of capital from venture capital to create the conditions for disruption of existing business models towards more efficient systems with centralized providers being an agent for mass coordination. These venture capital firms have created significant value and have expertise in replacing existing systems and driving the technical, organizational, and wider social coordination of society. This has provided significant social benefit in creating effective change towards social systems, and without which there is insufficient coordination for actors within an ecosystem to create the sufficient change necessary for widespread social benefit.

A replication of this coordination of raising a significant amount of capital towards resolving two-sided marketplaces has been proposed[fatprotocols][fredblog][navalblog] and attempted in the cryptocurrency space, which has historically been known as the emerging phenomena of "ICO" or "Initial Coin Offerings". This uses capital to raise significant sums of value to establish the capital required to develop a platform and raise capital to establish a war-chest to resolve a two-sided marketplace problem. Many of these firms have successfully used significant amounts of capital raised to establish a developer community, market to users, and encourage various service providers by subsidizing one side of the marketplace.

However, while the venture model has been incredibly pro-social and created significant benefits, there are some limitations to this model in traditional venture. Additionally, the model being emulated under the ICO model has raised significant amounts of capital, but there has been insufficient amount of experiments to demonstrate successful models. One of the largest limitations are that the network effects do not accrue to the users who are responsible for developing this system and ensuring its success. Additionally, there is a lack of association between economic stakeholders benefiting from the network effect and those which are responsible for establishing those network effects. Further, there are significant centralization pressures by raising a large amount of funds into a single foundation.

Instead, the Handshake mechanism does not rely upon altruistic actors, it is wholly reliant upon self-interested individuals and stakeholders. One's returns are maximized by giving away value to FOSS developers and wider humanity, and may be an exploration in a different model for development.

Disclaimers

There are no guarantees provided by the Handshake community developers past and present, including continued development and leadership. This document is illustrative and the de-facto community standards are the primary source. All contents of this document is subject to change according to community consensus.

No guarantees are provided with regards to functionality of the naming and auction system, including renewal availability, fees, or block availability in general. Further, no guarantees are provided with coin supply, coin value, or name value. In the event of a worldwide distribution, it is up to the wider community to execute this plan.

Those making unilateral advocacy of where the blockchain should go under the auspicies of being an early developer of the chain, and any rhetoric related should be seen with suspicion. It is up to the community to suggest changes and any code forks are initiated with community consensus and approval.

In the event of deep reorganizations, the community should halt processing and acknowledging new state updates. It is heavily recommended for service providers to have long deposit times and their own internal controls and software verification.

Hard forks are presumed to be possible in this system, there are no guarantees around mining, economics, etc.

Trademark holders are responsible for munging their own renewals and registrations, and project developers do not have the ability to make unilateral changes to the system after the sunrise period. Any changes to the records requires a hard fork and is contingent upon community approval of fullnodes and miners. Any changes after the sunrise period may be proposed to the community as a hardfork, or more likely the updates should be made (without consensus) in the resolver client-side.

No guarantees are provided for transaction formats past one year. Pre-signed transactions should not be presumed to be permanently available. Private keys should be kept in the event transaction formats change.

[pow] http://www.hashcash.org/papers/pvp.pdf
[bitcoin] https://bitcoin.org/bitcoin.pdf
[hashcash] http://www.hashcash.org/papers/hashcash.pdf
[cuckoo-1] https://github.com/tromp/cuckoo
[cuckoo-2] https://github.com/tromp/cuckoo/blob/master/doc/cuckoo.pdf?raw=true
[cuckoo-3] https://github.com/tromp/cuckoo/blob/master/doc/spec
[timewarp] https://bitcointalk.org/index.php?topic=43692.msg521772#msg521772
[digishield-1] https://www.reddit.com/r/Digibyte/comments/213t7b/what_is_digishield_how_it_works_to_retarget/
[digishield-2] https://github.com/digibyte/DigiByteProject/blob/master/src/main.cpp
[kimoto-1] https://github.com/megacoin/megacoin/blob/master/src/main.cpp#L1276
[kimoto-2] https://bitcointalk.org/index.php?topic=240861.msg3040291#msg3040291
[dgw] http://www.darkcoin.io/downloads/DarkcoinWhitepaper.pdf
[zcash-1] https://github.com/zcash/zcash/issues/147
[zcash-2] https://github.com/zcash/zcash/issues/696
[zcash-3] https://github.com/zcash/zcash/issues/998
[namecoin-1] https://namecoin.org/
[namecoin-2] https://github.com/namecoin/ncdns
[ens-1] https://ens.domains/
[ens-2] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-137.md
[ens-3] https://github.com/ensdomains/ens
[ens-4] https://github.com/ethereum/eips/issues/137
[ens-5] https://github.com/ethereum/EIPs/issues/162
[ens-6] https://ens.domains/#section-root
[ens-7] https://docs.ens.domains/en/latest/faq.html#who-will-own-the-ens-rootnode-what-powers-does-that-grant-them
[blockstack-1] https://blockstack.org/whitepaper.pdf
[blockstack-2] https://forum.blockstack.org/t/how-do-lightweight-blockstack-nodes-operate-a-snv-protocol/1017
[ept-1] https://ethereum.github.io/yellowpaper/paper.pdf
[ept-2] https://github.com/ethereum/wiki/wiki/Patricia-Tree
[ethereum] https://ethereum.org/pdfs/EthereumWhitePaper.pdf
[smt-1] https://eprint.iacr.org/2016/683
[smt-2] https://github.com/google/trillian
[merklix-1] https://www.deadalnix.me/2016/09/24/introducing-merklix-tree-as-an-unordered-merkle-tree-on-steroid/
[merklix-2] https://www.deadalnix.me/2016/09/29/using-merklix-tree-to-checkpoint-an-utxo-set/
[merkleset] https://github.com/bramcohen/MerkleSet
[vickrey] https://www.jstor.org/stable/2977633
[maxwell-1] https://bitcointalk.org/index.php?topic=277389.0
[maxwell-2] https://bitcointalk.org/index.php?topic=278122.0
[covenants] https://fc16.ifca.ai/bitcoin/papers/MES16.pdf
[bitcoin-ng] https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-eyal.pdf
[rfc1035] https://www.ietf.org/rfc/rfc1035.txt
[orsn] http://www.orsn.org/en/tech/
[bind] https://www.isc.org/downloads/bind/
[ldns] https://www.nlnetlabs.nl/projects/ldns/about/
[unbound] https://www.unbound.net/
[nlnetlabs] https://www.nlnetlabs.nl/
[bns] https://github.com/chjj/bns
[noise] http://noiseprotocol.org/
[lightning] https://github.com/lightningnetwork/lightning-rfc/blob/master/08-transport.md
[qname] https://tools.ietf.org/html/rfc7816
[dnssec] https://tools.ietf.org/html/rfc4033
[google] https://developers.google.com/speed/public-dns/
[root] https://www.iana.org/domains/root/servers
[iana] https://www.iana.org/
[icann] https://www.icann.org/
[internic] https://www.internic.net/domain/root.zone
[anchors] https://www.iana.org/dnssec/files
[signing] https://www.iana.org/dnssec/ceremonies
[sshfp] https://tools.ietf.org/html/rfc4255
[openssh] https://www.google.com/search?q=openssh%20SSHFP
[tlsa] https://tools.ietf.org/html/rfc6698
[sig0] https://tools.ietf.org/html/rfc2931
[anycast] https://tools.ietf.org/html/rfc4786
[tsig] https://www.ietf.org/rfc/rfc2845.txt
[dnscurve] https://dnscurve.org/
[plasma] http://plasma.io/plasma.pdf
[shattered] https://shattered.io/static/shattered.pdf
[nameandshame] https://dnssec-name-and-shame.com/
[alexa] https://www.alexa.com/topsites
[nsec3] https://tools.ietf.org/html/rfc5155
[btcrelay] http://btcrelay.org/
[opennic] https://wiki.opennic.org/opennic:faq#how_did_opennic_start
[convergence] https://www.youtube.com/watch?v=Z7Wl2FW2TcA
[casper-finality-gadget] https://arxiv.org/abs/1710.09437
[certpinning] https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning
[fakecert-fr] https://www.theregister.co.uk/2013/12/10/french_gov_dodgy_ssl_cert_reprimand/
[fakecert-ir] https://www.computerworld.com/article/2510951/cybercrime-hacking/hackers-spied-on-300-000-iranians-using-fake-google-certificate.html
[zooko] https://web.archive.org/web/20011020191610/http://zooko.com/distnames.html
[aaron] http://www.aaronsw.com/weblog/squarezooko
[natureofthefirm] http://dx.doi.org/10.1111/j.1468-0335.1937.tb00002.x
[smartcontracts] http://journals.uic.edu/ojs/index.php/fm/article/view/548
[wealthofnetworks] http://journals.uic.edu/ojs/index.php/fm/article/view/548#page-60
[bazaar] http://www.catb.org/esr/writings/cathedral-bazaar/
[fredblog] https://blog.coinbase.com/app-coins-and-the-dawn-of-the-decentralized-business-model-8b8c951e734f
[navalblog] https://startupboy.com/2014/03/09/the-bitcoin-model-for-crowdfunding/
[fatprotocols] http://www.usv.com/blog/fat-protocols
[primaveradefilippi] https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2725415