Sigma Network

Daniel Lubarov
daniel@lubarov.com

DRAFT Version 0.1
September 11, 2018

Abstract

We introduce Sigma Network, a cryptocurrency protocol which provides strong privacy guarantees using zero knowledge proofs. In contrast to existing privacy-oriented protocols, our blockchain design is based on proof of stake, which opens the door to a number of improvements.

We introduce a novel, leaderless BFT consensus algorithm to quickly finalize each block, thereby eliminating forks. Under normal conditions, our algorithm achieves consensus after just two rounds of one-way communication. We prove the algorithm's safety under a 2/3 honest majority assumption, and prove its liveness under a partial synchrony assumption.

Our protocol is designed with the expectation that each validator node will be run by multiple servers. We divide blocks into small shards, and our peer discovery protocol matches peers with overlapping shard ranges. This allows transaction volume to scale without any practical limits.

To support users who lack the resources to process all transactions, we propose a unique light client model. Instead of verifying all transactions in a block, light clients verify that BFT consensus was achieved. Under our honest majority assumption, light clients enjoy the same security as full validator nodes, and since vote counts are not affected by transaction volume, this model allows the network to scale without increasing computational requirements for ordinary users.

While existing cryptocurrencies have generated a great deal of interest and investment, fiat currencies remain ubiquitous in day-to-day life. Sigma Network's unique combination of privacy, scalability and low latency make it a strong candidate for adoption in everyday commerce.

Contents

1 Background

The success of Bitcoin demonstrates a strong interest in decentralized currencies, where monetary policies are determined by a community of users rather than government decree. Bitcoin's purported anonymity contributed to its success, but a growing body of research shows that accounts can often be deanonymized based on their public activity [1] [2].

Some newer cryptocurrencies improve upon Bitcoin's privacy. Monero [3] uses ring signatures to obfuscate a transaction's source of funds, and Diffie-Hellman exchanges to hide its destination. Similarly, Zcash [4] uses zk-SNARKs to prove that a transaction has a valid source of funds without revealing what that source is. Zcash's privacy guarantee is stronger by comparison, since its shielded transactions leak no information at all, whereas Monero uses rings of size 7 by default.

While Zcash and Monero offer strong privacy, both protocols are designed to be executed by individual computers, so their transaction volume is limited by the capabilities of a single machine. Both protocols have sufficient capacity for the current transaction volume, but their limited scalability precludes their adoption in mainstream commerce.

Another drawback of Bitcoin, Zcash and Monero is that they rely on proof of work for consensus. The security model fundamentally requires that mining be expensive, as this creates an economic barrier to 51% attacks. At the time of writing, Bitcoin mining is funded by about $4 billion worth of block rewards per year, which causes an inflation rate of about 4%.

Yet another challenge is latency. Bitcoin has an average block time of 10 minutes, and [5] recommends waiting for six confirmations before accepting a transaction as final. Some newer cryptocurrencies such as Tendermint [6] and Algorand [7] provide quick finality using BFT consensus algorithms, but they are subject to the same privacy issues as Bitcoin.

Finally, most existing cryptocurrencies do not offer post-quantum security. This includes Zcash and Monero, both of which rely on the hardness of the discrete logarithm problem for transaction soundness. A universal quantum computer could compute discrete logarithms in polynomial time using a variant of Shor's algorithm. To date, Shor's algorithm has been demonstrated only with very small factorization problems [8]. Still, the potential future impact of quantum computers is a cause for concern.

Our design aims to address all of the aforementioned challenges. It offers a strong privacy guarantee using zero knowledge proofs, similar to Zcash, but using a post-quantum secure proof system based on multiparty computation. Our blockchain protocol is based on proof of stake, so security is inexpensive compared to proof of work blockchains. We use use a novel BFT consensus algorithm to quickly finalize each block, thereby ensuring immutability and eliminating forks.

To allow transaction volume to scale, we use a simple sharding mechanism which lets each validator divide their communications among multiple servers.

2 Protocol overview

Sigma Network organizes protocol state into a blockchain structure. We use a BFT algorithm, described in section 5, to commit a single (possibly empty) block at each height. Because exactly one block is committed at each height, committed blocks form a path. And because all valid blocks at the same height must share the same committed parent, valid blocks form a caterpillar tree.

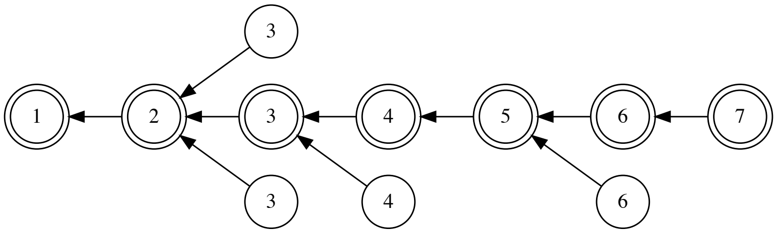

Figure 1: A possible composition of valid blocks. The numbers represent block heights, and the double circles represent committed blocks.

This structure has several benefits. Since blocks are committed shortly after they have propagated, transactions are confirmed very quickly, with a strong guarantee of irreversibility. Additionally, blocks can be generated as quickly as the network is able to propagate them; the BFT process eliminates the need for fixed block timing. Finally, the lack of forks makes client performance very predictable.

A validator must register as a block creator before being eligible to create a block, and must register as a voter before being eligible to vote in the BFT consensus protocol. These registration messages are similar to transactions—they must be included in a block before they are recognized, and they may include transaction fees in order to incentivize inclusion.

Each registration message has an associated score, which is pseudorandomly generated based on some account data. An account must have a certain minimum amount of stake, `DepositAmount`, in order to submit any registrations. The more stake an account has, the more registrations they may submit, each with a unique score.

If more than one validator registers to create the same block, only the one with the highest score is eligible to do so. Similarly, if more than `MaxVoters` register to vote in the same epoch, only the highest-scoring `MaxVoters` are eligible to do so.

Blocks are grouped into *epochs*, each consisting of `EpochSize` blocks. If a validator wishes to participate in epoch i, they must submit their registration during epoch $i-1$. Since there are many blocks in an epoch, validators have many opportunities to have their registration messages included. Censoring registration messages would be difficult, since it only takes a single uncooperating block creator to thwart a censorship attempt by including any pending registrations.

3 Validator registration

In each epoch, validators may register to create blocks or vote in the following epoch. Since the process is similar for both, we describe a generic protocol for event registration.

A validator may register for an event by submitting one or more *tickets*. A validator with a stake of s may submit up to $\lfloor s/\texttt{DepositAmount} \rfloor$ tickets per event, each with a unique index i.

Each ticket has a score, which is computed as

$$R(x, \mathrm{sk}, e, i)$$

where R is a pseudorandom function, x is the random seed of the current epoch, sk is the validator's secret key, e is an event identifier, and i is the ticket index.

A registration message includes a non-interactive zero-knowledge proof (NIZKP) showing that a certain ticket was scored correctly. The event e is a public input to the NIZKP, but sk and i are private, so no account data is revealed. The details of the NIZKP circuit are covered in section 6.2.

At the end of an epoch, tickets are approved or denied based on their ranking. For block creation events, only one ticket is approved at each block height. For voting events, the top `MaxVoters` tickets are approved to vote in the next epoch.

Since ticket scores are public, validators can often judge whether their ticket is likely to be approved based on the scores of any previously submitted tickets. Validators can save on transaction fees by not submitting tickets which are likely to be denied.

4 Entropy collection

When a validator creates a new account, it is important that they not be able to predict its ticket scores. Otherwise, a validator could enumerate possible key pairs until they eventually find one with high-scoring tickets. To prevent such "grinding" attacks, we use a new random seed in each epoch. A validator account created in epoch i cannot submit tickets until epoch $i+2$, so there is a full round of entropy generation in epoch $i+1$ before the account becomes eligible.

The random seed for each epoch is derived from the signatures of block creators. In particular, each block creator submits $\mathrm{R}(\mathrm{sk}, h)$ as their entropy contribution, where R is a pseudorandom function, sk is their secret key and h is the height of the block. This computation is done in zero knowledge; the details are described in section 6.4. At the end of each epoch, the bitwise XOR of these entropy submissions becomes the random seed of the following epoch.

Note that this scheme has the properties of a verifiable random function (VRF), although it does not use a typical VRF construction. In particular, the pseudorandom computation is deterministic, and its correctness is publically verifiable since a zero-knowledge proof is provided. Therefore, the only way for a block creator to influence the entropy stream is to skip their block, forfeiting the block reward.

4.1 Profitability of entropy manipulation

Consider a validator with a stake of p, expressed as a fraction of the money supply that is actively staked. Let b be the number of contiguous blocks at the very end of an epoch which they are eligible to create. By submitting or withholding blocks, the validator can manipulate up to b bits of entropy. In other words, they may choose from among 2^b random seeds for the next epoch.

Net n be the number of blocks in an epoch. Let r_1 be the validator's representation in the next epoch (that is, the number of blocks they will be eligible to create) which would result from the "default" seed, if no manipulation occurs. Let $r_2, \ldots, r_{2^b}$ be the validator's representation resulting from the alternative seeds which the validator can choose from.

If the validator performs no entropy manipulation, their expected block rewards from the next epoch would simply be np. We will compare this to the expected block rewards from a manipulative validator, who calculates their block rewards for all possible seeds, and choses the seed which results in them receiving the most block rewards. In other words, we want to calculate

$$E\left[\max_{1\leq i\leq 2^b}(r_i)\right]$$

To calculate this expected value, we start by applying Fubini's theorem with the linearity of expectation:

$$\begin{aligned}E\left[\max_{1\leq i\leq 2^b}(r_i)\right] &= \sum_{j=0}^{n} P\left(\max_{1\leq i\leq 2^b}(r_i) > j\right)\\ &= \sum_{j=0}^{n} P(r_1 > j \text{ or } \ldots \text{ or } r_{2^b} > j)\end{aligned}$$

Applying De Morgan's law, this becomes

$$= \sum_{j=0}^{n} (1 - P(r_1 \leq j \text{ and } \ldots \text{ and } r_{2^b} \leq j))$$

Since $r_1, \dots, r_{2^b}$ are independent and identically distributed, we can separate the probabilities and use r_1 in place of the other variables:

$$= \sum_{j=0}^{n} \left(1 - P(r_1 \leq j)^{2^b}\right)$$

Finally, applying the binomial CDF, we have

$$= \sum_{j=0}^{n} \left(1 - \left(\sum_{k=0}^{j} \binom{n}{k} p^k (1-p)^{n-k}\right)^{2^b}\right)$$

where n is the number of blocks in an epoch.

We now have a formula for the validator's expected reward given a particular number of manipulable bits b. But since b varies, we want to calculate the expected reward for all possible values of b, weighted by their probabilities:

$$\sum_{b=0}^{n} P(b) E\left[\max_{1 \leq i \leq 2^b} (r_i)\right]$$

Applying the formula we derived above, along with $P(b) = p^b$, yields the result

$$\sum_{b=0}^{n} \left(p^b \sum_{j=0}^{n} \left(1 - \left(\sum_{k=0}^{j} \binom{n}{k} p^k (1-p)^{n-k}\right)^{2^b}\right)\right)$$

Finally, to compute the expected advantage from manipulation, $E[a]$, we simply divide this quantity by the expected rewards with no manipulation, which is np.

We used a computer program to calculate $E[a]$ for various parameters. The results are shown in Figure 2. Note that the $E[a]$ figures are upper bounds, since we did not account for any missed block rewards resulting from withholding blocks. We also assumed that manipulators can perfectly predict their representation in the next epoch, whereas in reality, they can only estimate it based on their registration ticket scores.

p	n	$E[a]$
20%	100	$\leq 2.71\%$
	1000	$\leq 0.86\%$
	10000	$\leq 0.27\%$
30%	100	$\leq 3.46\%$
	1000	$\leq 1.09\%$
	10000	$\leq 0.34\%$
40%	100	$\leq 4.16\%$
	1000	$\leq 1.31\%$
	10000	$\leq 0.42\%$

Figure 2: The expected advantage resulting from manipulation.

As you can see, the advantage from manipulation diminishes as the epoch size increases. This can be explained by the law of large numbers—the variance of a validator's representation decreases as the number of samples (that is, blocks) increases.

4.2 Alternative designs

There are several possible ways to mitigate entropy manipulation. One option is to slash the funds of validators who withhold blocks. This is essentially the idea behind RANDAO [9], although RANDAO uses a commit-reveal scheme while we use a VRF. This approach would reduce $E[a]$ by making manipulation more costly, although it would also harm validators who accidentally miss their chance to submit a block.

Another option is to use threshold signatures, such as the BLS scheme which is used by Dfinity [10]. In a (t, n)-threshold signature scheme, the shares of at least t of n parties are required to create the group signature. As long as t validators are honest and able to communicate with one another, the result should be impossible to manipulate.

Yet another option is to pass VRF outputs through a verifiable delay function (VDF), such as the trapdoor VDF proposed by [11], and take its output as the next epoch's seed. The idea is that validators would not have sufficient time to compute the seeds which would result from submitting or withholding a block. Under normal circumstances, the network would have moved on before the computation can be done.

While these alternative schemes are academically interesting, they would complicate the protocol, and we consider them unnecessary based on the results in Figure 2. Particularly with a large epoch size of 10,000 or more, the potential advantages from manipulation are very small.

5 BFT consensus algorithm

In everyday commerce, transaction latency is of paramount importance. While some users may be willing to accept transactions without a guarantee of finality, others require such a guarantee in order to eliminate counterparty risk. For this reason, we use a Byzantine fault tolerant consensus algorithm to finalize each block as soon as it has been propagated through the network.

Note that our algorithm is special purpose; we leverage certain properties of our blockchain design in order to guarantee convergence without relying on leaders. The algorithm would not work with arbitrary values; such a generalization would invalidate our proof of liveness.

We assume that over 2/3 of validators are honest, meaning that they vote as prescribed below. In particular, given a set of $3t + 1$ validators, we assume that at least $2t + 1$ of them are honest, while at most t are not.

We further assume that the same $2t + 1$ validators are connected with partial synchrony. Given some bounded delay Δ, we assume that the system will alternate between periods of synchrony, where all messages are propagated among this $2t+1$

validator set, and periods of asynchrony, where messages are lost or delayed beyond Δ.

Voting takes place over a series of rounds. For the moment, we will assume that some rounds are wholly synchronous, meaning that all votes are propagated among this $2t + 1$ validator set before any of them consider the round to have terminated. Because of timing issues, this does not immediately follow from our partial network synchrony assumption, but we will justify this stronger assumption in section 5.4.

5.1 Definitions

We define two types of votes. $\langle \texttt{PREPARE}, \nu, b, r \rangle$ signifies that some validator ν votes to prepare block b in round r. $\langle \texttt{COMMIT}, \nu, b, r \rangle$ signifies that ν votes to commit b in round r. Both messages are accompanied by signatures to ensure authenticity. Note that validators may vote for an empty block; we denote this with $b = \emptyset$.

Next, we define the predicate $\text{prepared}(b, r)$ to be true if and only if b has received at least $2t + 1$ votes of either kind in round r. We define $\text{committed}(b, r)$ to be true if and only if b has received at least $2t + 1$ `COMMIT` votes in round r.

Further, we define $\text{prepared}_\nu(b, r)$ to be true if and only if ν knows $\text{prepared}(b, r)$ to be true based on the votes that ν has observed. Similarly, we define $\text{committed}_\nu(b, r)$ to be true if and only if ν knows $\text{committed}(b, r)$ to be true.

5.2 Voting rules

In each round r, each honest validator ν votes as follows.

1. If $\text{prepared}_\nu(a, s)$ for some block a and round number s, then let s, a be the highest such round number and the associated block.
   (a) If $s = r - 1$, then ν votes $\langle \texttt{COMMIT}, \nu, a, r \rangle$.
   (b) Otherwise, ν votes $\langle \texttt{PREPARE}, \nu, a, r \rangle$.
2. Otherwise, let B be the set of valid blocks that ν has received for this block height.
   (a) If B is a singleton set $\{b\}$, then ν votes $\langle \texttt{PREPARE}, \nu, b, r \rangle$.
   (b) Otherwise, ν votes $\langle \texttt{PREPARE}, \nu, \emptyset, r \rangle$.

Each validator ν halts as soon as $\text{committed}_\nu(b, r)$ for any b, r. ν will then accept b as the final, irreversible block for the current block height.

5.3 Proving safety and liveness

Lemma 1. committed(b, r) *implies* prepared(b, r).

Proof. This follows immediately from our definitions. committed(b, r) entails that b received at least $2t + 1$ `COMMIT` votes. This also satisfies the definition of prepared(b, r), which counts votes of either kind. □

Lemma 2. *At most one block will become prepared at each round number.*

Proof. Assume the opposite: $\exists a, b, r$ such that prepared(a, r) and prepared(b, r). Then both a and b received $2t+1$ or more votes in round r, for a total of $4t+2$ or more votes. Since there are only $3t+1$ validators, at least $t+1$ of them must have voted for both a and b, which violates our assumption that at most t validators are dishonest. □

Lemma 3. $\nexists a, b, r, s$ *such that* committed(a, r)*,* prepared(b, s)*, and* $s > r$*.*

Equivalently, if some block is committed, no block may be prepared in a later round.

Proof. Suppose committed(a, r). Then by definition, $2t + 1$ validators must have voted $\langle \texttt{COMMIT}, \nu, a, r \rangle$, and by our honesty assumption, at least $t+1$ of these votes must have come from honest validators. For each of these validators ν, we know prepared$_\nu(a, r-1)$, otherwise ν would not have followed rule 1a.

By the logic of rule 1, these $t + 1$ honest validators will continue voting for a until a different block becomes prepared. This leaves $(3t + 1) - (t + 1) = 2t$ remaining validators who may vote for a different block. Since $2t$ is one short of a quorum, a different block will never gain enough votes to become prepared. □

Theorem 1 (Safety)**.** *At most one block will become committed.*

Proof. Assume the opposite: $\exists a, b, r, s$ such that committed(a, r) and committed(b, s). Lemma 2 implies $r \neq s$. Without loss of generality, assume $r < s$. Lemma 1 implies prepared(b, s), which contradicts Lemma 3. □

Theorem 2 (Liveness)**.** *It is always possible for a block to become committed.*

Proof. Let r be the current round. Let B be the set of valid blocks which have propagated to at least $2t+1$ honest validators before they begin round r. Consider these possible protocol states:

1. $\nexists b, s$ such that prepared(b, s), and $|B| = 0$.
2. $\nexists b, s$ such that prepared(b, s), and $|B| = 1$.
3. $\nexists b, s$ such that prepared(b, s), and $|B| \geq 2$.

4. $\exists b, s$ such that prepared(b, s), but all such $s < r - 1$.

5. $\exists b$ such that prepared$(b, r - 1)$.

6. $\exists b, s$ such that committed(b, s).

We will show that in a synchronous round, the protocol will always progress to a state with a higher number. In an asynchronous round, it is possible for the protocol to regress from state 5 to state 4, but it will cycle between 4 and 5 until there is eventually a transition from 5 to 6.

If the current state is 1, then validators will vote in accordance with rule 2. If a valid block has been released but not fully propagated, it will be propagated this round, advancing us to state 2. If not, all honest validators will vote to prepare $\emptyset$ as per rule 2b, advancing us to round 5.

If the current state is 2, then the logic is similar. If two or more valid blocks have been released but only one has fully propagated, then a second block will be propagated this round, advancing us to state 3. If not, the 2/3 of honest validators who have seen one block will vote for that block as per rule 2a, advancing us to round 5.

If the current state is 3, then the honest 2/3 of validators will vote for $\emptyset$ as per rule 2b, advancing us to round 5.

If the current state is 4, then honest validators will vote $\langle \texttt{PREPARE}, \nu, b, r \rangle$ in accordance with rule 1b. Thus upon the completion of round r, we have prepared(b, r), which puts us in state 5 for round $r + 1$.

If the current state is 5, then honest validators will vote $\langle \texttt{COMMIT}, \nu, a, r \rangle$ in accordance with rule 1a, resulting in a successful commit, thereby advancing us to state 6 which is terminal. □

5.4 Round timing

In our protocol, the timing of rounds is based on when certain messages are received. Since different validators may receive the same message at different times, this introduces an element of subjectivity. However, these differences in perceived round timing are bounded by Δ during periods of synchrony.

Once a validator ν receives a valid block for some height, they consider that moment the beginning of the first round. If `BlockTimeout` elapses and ν hasn't received any valid block, they will consider the moment of the timeout to be the beginning of the first round.

Subsequently, ν considers a round r to have terminated (and round $r + 1$ to have begun) when one of the following conditions have been met:

1. 2Δ has elapsed since the round begun. This allows Δ for votes to propagate, and Δ to account for differences in validators' perceptions of r's start time.

So if the network was synchronous during this period, then all votes in round r have had sufficient time to propagate.

2. At least $t + 1$ votes have been cast in round $r + 1$. Our honesty assumption implies that at least one of these votes must be from an honest validator, so one of the other two termination conditions must have been met.

3. $\text{prepared}_\nu(b, r)$ for any block b. In this case, Lemma 2 implies that no other block can become prepared in r.[1] Note that this termination condition is not strictly necessary—we could wait for the timeout—but terminating early helps to minimize latency.

[TODO: This part is unfinished.]

5.5 Incentives

Sigma Network offers neither rewards for voting, nor penalties for breaking the rules. Validators already have an incentive to download vote data, since they must be prepared to create a block in the future when it is their turn. This is in contrast to protocols such as Casper FFG [12], which punish validators who violate a rule by slashing their funds.

The safety of our protocol lies in the difficulty of coordinating $t + 1$ voters and convincing them to participate in a joint attack. We consider this unlikely, since validators are directly invested in the success of the currency, and a successful 1/3 attack would likely have a major impact on the value of their investment.

[TODO: Consider having validators expose their nullifiers so that, in the event of a successful 1/3 attack, the community could coordinate a hard fork to blacklist the attacker. This seems almost as effective as Casper FFG's automatic penalties, since in in the event of a successful 1/3 attack, an emergency hard fork would be required anyway to sort out the conflicting finalized checkpoints.]

5.6 Related work

Our consensus algorithm is similar to that of Tendermint [6], the main difference being that we have eliminated the propose step. In the general problem of BFT consensus, proposals play an important role in guaranteeing eventual convergence. Without proposals, voters may be split among multiple candidate values and possibly never converge, particularly if malicious voters impede convergence by voting

[1] It is possible that $\text{committed}(b, r)$, and ν may not become aware of it until more votes are received. For example, if ν received one vote to prepare b and $2t$ votes to commit it, they would terminate the round based on this rule. If ν later received one more vote to commit b, then they would learn that b has been committed. In cases like this, however, there is no harm in ν terminating the round early, since the commitment would have succeeded regardless.

for minority values. In the context of our blockchain design, however, the logic in rule 2 ensures convergence, as we showed in the liveness proof.

Under normal conditions, our algorithm transitions from state 1 to state 5, then to state 6. So finality is typically reached after just two rounds of communication, whereas other BFT consensus algorithms require a minimum of three rounds.

Another advantage of our leaderless voting rules is that soft forks require 67% support, rather than 34% support, as in proposal-based protocols like Tendermint. We discuss this further in section 8.1.

Our timing mechanics also differ from other BFT algorithms. While Tendermint and Algorand allocate a fixed period of time for each round of voting, our protocol terminates rounds early in certain cases. This speeds up the consensus process, particularly in the common case where a single block is announced, becomes prepared in the first round, and becomes committed in the second round.

6 Zero knowledge proofs of validity

We use zero knowledge proofs to demonstrate the validity of transactions, as well as other messages such as voter registration. In particular, we use the proof system described in [13], which gives reasonable proof times and sizes for circuits with relatively small numbers of AND gates. The proof system, hash functions, and the other cryptographic primitives we use are all parameterized for 128 bit classical security.

[The proof system is subject to change. Need to evaluate zk-STARKs which purportedly were improved to 80kb for a Zcash-like circuit? Also consider using zk-SNARKs for now, and transitioning to something else later if/when Shor's algorithm becomes more of a threat.]

The proofs described below all deal with *notes*. A note is a message $\langle \mathrm{pk}, n, s \rangle$, where pk is the public key of the user who owns the note, n is the number of coins which the note represents, and s is a salt which is pseudorandomly generated by the user to sent the note.

6.1 Spend circuit

Each transaction has one or two *source* notes and zero, one or two *destination* notes. The source notes are consumed by the transaction, while the destination notes are newly created. In order for a transaction to be valid, the combined amounts of the source notes must equal the combined amounts of the destination notes plus the transaction fee.

The spend circuit proves that a transaction is valid. We use a design similar to Zcash's `POUR` circuit. The circuit consumes one or two notes and produces one or two new notes. This lets users split one account into two, or combine two accounts into one, all without revealing which type of transaction they are performing.

To ensure that a note cannot be consumed twice, a “nullifier” associated with the note is exposed in the output layer. Validators keep track of all previous nullifiers, and reject any transactions whose nullifiers are already present in the nullifier set. An outsider cannot tell which nullifiers are associated with which commitments, but the circuit guarantees that a unique nullifier is produced for each commitment. Thus, we are able to prevent double spending without sacrificing privacy.

Zcash stores all commitments in a Merkle tree of depth 64, using SHA-256 as the hash function. To validate that the commitments being consumed are actually present in the blockchain, the `POUR` circuit includes two Merkle proofs going all the way up to the Merkle root.

The SHA-256 circuit Zcash uses has 27,904 AND gates, and the entire `POUR` circuit has 4,109,330 AND gates.

[TODO: Explain our approach in detail. TLDR: We use LowMC-256 [14] + Davies-Meyer as the hash function. We use Merkle trees with a depth of 5, giving us an AND count of 20k, and proof sizes well under 100kb. Our privacy guarantee is not as great as Zcash’s, but better than Monero’s; our default ring size is 32 compared to their default of 7. This is all tentative and subject to change.]

6.2 Registration circuit

[TODO: Explain the circuit.]

[TODO: Add a graph to show the structure.]

6.3 Vote circuit

[TODO: This should just be a simple signature? Preimage and message in the input layer, postimage in the output layer?]

[TODO: Add a graph to show the structure.]

6.4 Block creation circuit

[TODO: This should just be a simple signature to show that a block was endorsed by the eligible block creator.]

6.5 Performance

[TODO: Add average case proof sizes, plus proof times and verification times.]

7 Communication, sharding and light clients

Most blockchain protocols were designed to be run by individual computers, which severely limits their capacity. With its current maximum block size of 1 megabyte, Bitcoin supports a theoretical maximum of 7 transactions per second. This is a far cry from global electronic payment volume; Visa's payment network, for example, has been stress tested with up to 47,000 transactions per second [15].

Our protocol, on the other hand, is designed to be executed by a scalable cluster of machines. At the same time, we provide a strong security guarantee for light clients, so that users can safely send and receive payments with minimal computing resources.

7.1 Sharded block structure

Each block b is broken up into n shards: $b_1, b_2, ...b_n$. Validators are free to shard transactions in any manner they choose, as long as the size of each shard does not exceed a specified maximum.

The mempool, which contains pending transactions and validator registrations, is also sharded. Regular transactions are sharded based on the nullifier of the first transaction being spent, and validator registrations are sharded based on the nullifier of the validator account. This scheme ensures that a single transaction cannot be added to multiple mempool shards, which helps mitigate mempool spam.

When a newly started node is discovering peers, it advertises the range of shards that it processes and searches for peers with overlapping ranges. Nodes should ignore any peers whose shard ranges do not overlap with their own.

7.2 Suggested implementation strategy

Validators are free to use any implementation strategy. They could, for example, run a full node a single server, with a high-capacity network interface and many CPUs for parallel proof verification. However, the protocol design lends itself to the following suggested implementation.

Consider a validator with $k + 1$ servers. One server is designated as the coordinator, while the k remaining servers are designated as workers. The entire shard interval $[1 \mathrel{..} n]$ is divided into k non-overlapping intervals, and each worker is assigned one.

When the coordinator discovers a new, valid block header, it instructs each worker to download and verify the block shards assigned to it. When a worker has finished verifying a shard, it reports the validity back to the coordinator. If all shards are reported as valid, the coordinator will vote to prepare the block, as per rule 2 in the Consensus section. If one or more shards are invalid, the coordinator will vote to prepare the empty block $\emptyset$.

When a validator is eligible to create a block, each worker generates a block for each of the shards assigned to it, drawing transactions from the mempool shards assigned to them. Once a worker has finished preparing a block shard, it computes a hash of the message and reports that to the coordinator. One the coordinator has received a hash for every block shard, it broadcasts a block announcement message which contains these hashes along with other block headers.

7.3 Light clients

[TODO: Needs a lot of work.]

Light clients download block headers, which contain the Merkle roots of commitment trees, along with all vote data.

To send a payment, a light client will broadcast it to the network and then ask a full node to provide a Merkle proof that it was included in some committed block.

Since commitment votes attest to the validity of an entire block, light clients enjoy the same security as full nodes without having to verify any transactions.

7.4 Comparison to payment channel systems

In "eventual consensus" systems such as Bitcoin, verifying the validity of a block requires downloading all of the transactions therein. To minimize on-chain transactions, the Lightning Network [16] establishes payment channels to enable bidirectional off-chain transactions. Settlement is done on-chain, but is intentionally delayed so that if one party attempts to settle using an old channel state, the counterparty has time to intervene by uploading the latest channel state. Similar payment channel systems have been proposed for Ethereum, Zcash and others.

Poon et al. argue that limiting on-chain transactions is important because it allows individuals with limited resources to verify each block. However, directly verifying each transaction is only one of several ways for a client to validate a block. Another possible strategy, for example, would be to probabilistically verify a proof of a block's validity. The PCP theorem [17] implies that this can be done by examining a constant number of bits in the proof, so block sizes would have no impact on verification difficulty.

In our system, the voting rules prescribe that validators should only vote for block which they know to be valid. We assume that an invalid block will never be committed, since doing so would require that a 2/3 supermajority break the rules by voting for an invalid block. Thus if a light client downloads vote data and verifies that a block has been committed, they have implicitly verified the validity of its transactions as well.

Compared to payment channels, this scheme provides a more seamless user experience. The user does not need to think about how many payment channels to create, which hub or hubs to connect with, and how many coins to deposit in each channel. Additionally, payment channels require users to proactively monitor the blockchain for any attempts to commit an old channel state. Our system imposes no such requirement; users are free to go offline for arbitrary periods of time.

Finally, in our system, users have immediate access to the entirety of their funds. In a payment channel system, funds can typically be spent at any time, but there are exceptions. If a direct connection becomes unresponsive, any funds locked in that channel will be temporarily unusable. It is possible to close a channel unilaterally, without the involvement of the other party, but it is a slow process since the other party needs time to contest the action.

8 Attack vectors

Since validators will only accept blocks with commitment votes from over 2/3 of validators, fork-based attacks such as stake bleeding [18] are not possible. Sybil and eclipse attacks are possible, but would only cause a denial of service; any targets isolated from the honest majority would simply stop accepting new blocks.

8.1 Censorship and other soft forks

[TODO: Explain why soft forks require a 67% supermajority. The key point is that with our leaderless BFT algorithm, when honest validators see exactly one valid block for a certain block height, they will repeatedly vote for that block unless they see a conflicting supermajority.]

8.2 34% attacks

[TODO: Add analysis about how much stake a validator would actually need in order to have a good chance of eventually getting a 34% stake in a single round. Maybe a graph with stake on the x axis and log(probability) on the y axis, with a few lines for different validator set sizes. The probability can be calculated with the binomial CDF.]

8.3 Private key markets

[TODO]

8.4 Denial of service

An denial of service (DOS) attack could temporarily prevent new blocks from being committed by forcing 34% of validators offline. Note that this applies to any BFT consensus system; Bracha et al. [19] proved that $\lceil (2n+1)/3 \rceil$ honest voters are necessary to reach agreement if the remaining voters are malicious.

"Eventual consensus" systems such as Bitcoin are somewhat vulnerable to DOS attacks as well. If a DOS attack forced 50% of Bitcoin miners offline, the remaining miners could continue creating blocks, but users should not accept those blocks as final until the network's hashrate returned to normal. This is because users cannot distinguish between an outage and a partition, and in the event of a partition, a fork on the other side of the partition might prevail after the partition is resolved.

It is up to validators to defend against DOS attacks using whatever methods they see fit. Still, we will offer a few recommendations for application layer DOS attacks. Other DOS attacks, such as SYN floods or NTP amplification, are out of scope since there are well-known countermeasures which are not specific to our protocol.

At the application layer, an attacker might [TODO].

Validators should accept a limited number of connections, and should throttle each one.

8.5 Sybil attacks

[TODO]

References

[1] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*, pp. 127–140, ACM, 2013.

[2] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *International Conference on Financial Cryptography and Data Security*, pp. 34–51, Springer, 2013.

[3] N. Van Saberhagen, "Cryptonote v 2. 0," 2013.

[4] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Security and Privacy (SP), 2014 IEEE Symposium on*, pp. 459–474, IEEE, 2014.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[6] J. Kwon, "Tendermint: Consensus without mining," *Retrieved May*, vol. 18, p. 2017, 2014.

[7] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, "Algorand agreement: Super fast and partition resilient byzantine agreement." Cryptology ePrint Archive, Report 2018/377, 2018. `https://eprint.iacr.org/2018/377`.

[8] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'brien, "Experimental realization of shor's quantum factoring algorithm using qubit recycling," *Nature Photonics*, vol. 6, no. 11, p. 773, 2012.

[9] "Randao: Verifiable random number generation." `https://randao.org/whitepaper/Randao_v0.85_en.pdf`, 2017. Accessed: 2018-09-05.

[10] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series consensus system rev. 1," 2018.

[11] B. Wesolowski, "Efficient verifiable delay functions.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 623, 2018.

[12] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[13] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures." Cryptology ePrint Archive, Report 2018/475, 2018. `https://eprint.iacr.org/2018/475`.

[14] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for mpc and fhe," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 430–454, Springer, 2015.

[15] "Stress test prepares visanet for the most wonderful time of the year." `https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html`. Accessed: 2018-07-03.

[16] J. Poon and T. Dryja, "The bitcoin lightning network," *i*, pp. 1–22, 2015.

[17] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *Journal of the ACM (JACM)*, vol. 45, no. 3, pp. 501–555, 1998.

[18] P. Gazi, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," tech. rep., Cryptology ePrint Archive, Report 2018/248, 2018. https://eprint. iacr. org/2018/248, 2018.

[19] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 1985.