Sia: Simple Decentralized Storage

David Vorick
Nebulous Inc.
david@nebulouslabs.com

Luke Champine
Nebulous Inc.
luke@nebulouslabs.com

November 29, 2014

Abstract

The authors introduce Sia, a platform for decentralized storage. Sia enables the formation of storage contracts between peers. Contracts are agreements between a storage provider and their client, defining what data will be stored and at what price. They require the storage provider to prove, at regular intervals, that they are still storing their client's data.

Contracts are stored in a blockchain, making them publicly auditable. In this respect, Sia can be viewed as a Bitcoin derivative that includes support for such contracts. Sia will initially be implemented as an altcoin, and later financially connected to Bitcoin via a two-way peg.

1 Introduction

Sia is a decentralized cloud storage platform that intends to compete with existing storage solutions, at both the P2P and enterprise level. Instead of renting storage from a centralized provider, peers on Sia rent storage from each other. Sia itself stores only the storage contracts formed between parties, defining the terms of their arrangement. A blockchain, similar to Bitcoin [1, 12], is used for this purpose.

By forming a contract, a storage provider (also known as a *host*) agrees to store a client's data, and to periodically submit proof of their continued storage until the contract expires. The host is compensated for every proof they submit, and penalized for missing a proof. Since these proofs are publicly verifiable (and are publicly available in the blockchain), network consensus can be used to automatically enforce storage contracts. Importantly, this means that clients do not need to personally verify storage proofs; they can simply upload their file and let the network do the rest.

We acknowledge that storing data on a single untrusted host guarantees little in the way of availability, bandwidth, or general quality of service. Instead, we recommend storing data redundantly across multiple hosts. In particular, the use of erasure codes can enable high availability without excessive redundancy.

Sia will initially be implemented as a blockchain-based altcoin. Future support for a two-way peg with Bitcoin is planned, as discussed in "Enabling Blockchain Innovations with Pegged Sidechains" [5]. The Sia protocol largely resembles Bitcoin except for the changes noted below.

2 General Structure

Sia's primary departure from Bitcoin lies in its transactions. Bitcoin uses a scripting system to enable a range of transaction types, such as pay-to-public-key-hash and pay-to-script-hash. Sia opts instead to use an M–of–N multi-signature scheme for all transactions, eschewing the scripting system entirely. This reduces complexity and attack surface.

Sia also extends transactions to enable the creation and enforcement of storage contracts. Three extensions are used to accomplish this: contracts, proofs, and contract updates. Contracts declare the intention of a host to store a file with a certain size and hash. They define the regularity with which a host must submit storage proofs. Once established, contracts can be modified later via contract updates. The specifics of these transaction types are defined in sections 4 and 5.

3 Transactions

A transaction contains the following fields:

Field	Description
Version	Protocol version number
Arbitrary Data	Used for metadata or otherwise
Miner Fee	Reward given to miner
Inputs	Incoming funds
Outputs	Outgoing funds (optional)
File Contract	See: File Contracts (optional)
Storage Proof	See: Proof of Storage (optional)
Signatures	Signatures from each input

3.1 Inputs and Outputs

An output comprises a volume of coins. Each output has an associated identifier, which is derived from the transaction that the output appeared in. The ID of output i in transaction t is defined as:

$$H(t||\text{“output”}||i)$$

where H is a cryptographic hashing function, and “output” is a string literal. The block reward and miner fees have special output IDs, given by:

$$H(H(\text{Block Header})||\text{“blockreward”})$$

Every input must come from a prior output, so an input is simply an output ID.

Inputs and outputs are also paired with a set of *spend conditions*. Inputs contain the spend conditions themselves, while outputs contain their Merkle root hash [2].

3.2 Spend Conditions

Spend conditions are properties that must be met before coins are “unlocked” and can be spent. The spend conditions include a time lock and a set of public keys, and the number of signatures required. An output cannot be spent until the time lock has expired and enough of the specified keys have added their signature.

The spend conditions are hashed into a Merkle tree, using the time lock, the number of signatures required, and the public keys as leaves. The root hash of this tree is used as the address to which the coins are sent. In order to spend the coins, the spend conditions corresponding to the address hash must be provided. The use of a Merkle tree allows parties to selectively reveal information in the spend conditions. For example, the time lock can be revealed without revealing the number of public keys or the number of signatures required.

It should be noted that the time lock and number of signatures have low entropy, making their hashes vulnerable to brute-forcing. This could be resolved by adding a random nonce to these fields, increasing their entropy at the cost of space efficiency.

3.3 Signatures

Each input in a transaction must be signed. The cryptographic signature itself is paired with an input ID, a time lock, and a set of flags indicating which parts of the transaction have been signed. The input ID indicates which input the signature is being applied to. The time lock specifies when the signature becomes valid. Any subset of fields in the transaction can be signed, with the exception of the signature itself (as this would be impossible). There is also a flag to indicate that the whole transaction should be signed, except for the signatures. This allows for more nuanced transaction schemes.

The actual data being signed, then, is a concatenation of the time lock, input ID, flags, and every flagged field. Every such signature in the transaction must be valid for the transaction to be accepted.

4 File Contracts

A file contract is an agreement between a storage provider and their client. At the core of a file contract is the file's Merkle root hash. To construct this hash, the file is split into segments of constant size and hashed into a Merkle tree. The root hash, along with the total size of the file, can be used to verify storage proofs.

File contracts also specify a duration, challenge frequency, and payout parameters, including the reward for a valid proof, the reward for an invalid or missing proof, and the maximum number of proofs that can be missed. The challenge frequency specifies how often a storage proof must be submitted, and creates discrete *challenge windows* during which a host must submit storage proofs (one proof per window). Submitting a valid proof during the challenge window triggers an automatic payment to the "valid proof" address (presumably the host). If, at the end of the challenge window, no valid proof has been submitted, coins are instead sent to the "missed proof" address (likely an unspendable address in order to disincentivize DoS attacks; see section 7.1). Contracts define a maximum number of proofs that can be missed; if this number is exceeded, the contract becomes invalid.

If the contract is still valid at the end of the contract duration, it *successfully terminates* and any remaining coins are sent to the valid proof address. Conversely, if the contract funds are exhausted before the duration elapses, or if the maximum number of missed proofs is exceeded, the contract *unsuccessfully terminates* and any remaining coins are sent to the missed proof address.

Completing or missing a proof results in a new transaction output belonging to the recipient specified in the contract. The output ID of a proof depends on the contract ID, defined as:

$$H(\text{transaction}||\text{“contract”}||i)$$

where i is the index of the contract within the transaction. The output ID of the proof can then be determined from:

$$H(\text{contract ID}||\text{outcome}||W_i)$$

Where W_i is the window index, i.e. the number of windows that have elapsed since the contract was formed. The outcome is a string literal: either "validproof" and "missedproof", corresponding to the validity of the proof.

The output ID of a contract termination is defined as:

$$H(\text{contract ID}||\text{outcome})$$

Where outcome has the potential values "successfultermination" and "unsucessfultermination", corresponding to the termination status of the contract.

File contracts are also created with a list of "edit conditions," analogous to the spend conditions of a transaction. If the edit conditions are fulfilled, the contract may be modified. Any of the values can be modified, including the contract funds, file hash, and output addresses. As these modifications can affect the validity of subsequent storage proofs, contract edits must specify a future challenge window at which they will become effective.

Theoretically, peers could create "micro-edit channels" to facilitate frequent edits; see discussion of micropayment channels, section 7.3.

5 Proof of Storage

Storage proof transactions are periodically submitted in order to fulfill file contracts. Each storage proof targets a specific file contract. A storage proof does not need to have any inputs or outputs; only a contract ID and the proof data are required.

5.1 Algorithm

Hosts prove their storage by providing a segment of the original file and a list of hashes from the file's Merkle tree. This information is sufficient to prove that the segment came from the original file. Because proofs are submitted to the blockchain, anyone can verify their validity or invalidity. Each storage proof uses a randomly selected segment. The random seed for challenge window W_i is given by:

$$H(\text{contract ID}||H(B_{i-1}))$$

where B_{i-1} is the block immediately prior to the beginning of W_i.

If the host is consistently able to demonstrate possession of a random segment, then they are very likely storing the whole file. A host storing only 50% of the file will be unable to complete approximately 50% of the proofs.

5.2 Block Withholding Attacks

The random number generator is subject to manipulation via block withholding attacks, in which the attacker withholds blocks until they find one that will produce a favorable random number. However, the attacker has only one chance to manipulate the random number for a particular challenge. Furthermore, withholding a block to manipulate the random number will cost the attacker the block reward.

If an attacker is able to mine 50% of the blocks, then 50% of the challenges can be manipulated. Nevertheless, the remaining 50% are still random, so the attacker will still fail some storage proofs. Specifically, they will fail half as many as they would without the withholding attack.

To protect against such attacks, clients can specify a high challenge frequency and large penalties for missing proofs. These precautions should be sufficient to deter any financially-motivated attacker that controls less than 50% of the network's hashing power. Regardless, clients are advised to plan around potential Byzantine attacks, which may not be financially motivated.

5.3 Closed Window Attacks

Hosts can only complete a storage proof if their proof transaction makes it into the blockchain. Miners could maliciously exclude storage proofs from blocks, depriving themselves of transaction fees but forcing a penalty on hosts. Alternatively, miners could extort hosts by requiring large fees to include storage proofs, knowing that they are more important than the average transaction. This is termed a *closed window attack*, because the malicious miner has artificially "closed the window."

The defense for this is to use a large window size. Hosts can reasonably assume that some percentage of miners will include their proofs in return for a transaction fee. Because hosts consent to all file contracts, they are free to reject any contract that they feel leaves them vulnerable to closed window attacks.

6 Arbitrary Transaction Data

Each transaction has an arbitrary data field which can be used for any type of information. Nodes will be required to store the arbitrary data if it is signed by any signature in the transaction. Nodes will initially accept up to 64 KB of arbitrary data per block.

This arbitrary data provides hosts and clients with a decentralized way to organize themselves. It can be used to advertise available space or files seeking a host, or to create a decentralized file tracker.

Arbitrary data could also be used to implement other types of soft forks. This would be done by creating an "anyone-can-spend" output but with restrictions specified in the arbitrary data. Miners that understand the restrictions can block any transaction that spends the output without satisfying the necessary stipulations. Naive nodes will stay synchronized without needing to be able to parse the arbitrary data.

7 Storage Ecosystem

Sia relies on an ecosystem that facilitates decentralized storage. Storage providers can use the arbitrary data field to announce themselves to the network. This can be done using standardized template that clients will be able to read. Clients can use these announcements to create a database of potential hosts, and form contracts with only those they trust.

7.1 Host Protections

A contract requires consent from both the storage provider and their client, allowing the provider to reject unfavorable terms or unwanted (e.g. illegal) files. The provider may also refuse to sign a contract until the entire file has been uploaded to them.

Contract terms give storage providers some flexibility. They can advertise themselves as minimally reliable, offering a low price and a agreeing to minimal penalties for losing files; or they can advertise themselves as highly reliable, offering a higher price and agreeing to harsher penalties for losing files. An efficient market will optimize storage strategies.

Hosts are vulnerable to denial of service attacks, which could prevent them from submitting storage proofs or transferring files. It is the responsibility of the host to protect themselves from such attacks.

7.2 Client Protections

Clients can use erasure codes, such as regenerating codes [4], to safeguard against hosts going offline. These codes typically operate by splitting a file into n pieces, such that the file can be recovered from any subset of m unique pieces. (The values of n and m vary based on the specific erasure code and redundancy factor.) Each piece is then encrypted and stored across many hosts. This allows a client to attain high file availability even if the average network reliability is low. As an extreme example, if only 10 out of 100 pieces are needed to recover the file, then the client is actually relying on the 10 most reliable hosts, rather than the average reliability. Availability can be further improved by rehosting file pieces whose hosts have gone offline. Other metrics benefit from this strategy as well; the client can reduce latency by downloading from the closest 10 hosts, or increase download speed by downloading from the 10 fastest hosts. These downloads can be run in parallel to maximize available bandwidth.

7.3 Uptime Incentives

The storage proofs contain no mechanism to enforce constant uptime. There are also no provisions that require hosts to transfer files to clients upon request. One might expect, then, to see hosts holding their clients' files hostage and demanding exorbitant fees to download them. However, this attack is mitigated through the use of erasure codes, as described in section 7.2. The strategy gives clients the freedom to ignore uncooperative hosts and work only with those that are cooperative. As a result, power shifts from the host to the client, and the "download fee" becomes an "upload incentive."

In this scenario, clients offer a reward for being sent a file, and hosts must compete to provide the best quality of service. Clients may request a file at any time, which incentivizes hosts to maximize uptime in order to collect as many rewards as possible. Clients can also incentivize greater throughput and lower latency via proportionally larger rewards. Clients could even perform random "checkups" that reward hosts simply for being online, even if they do not wish to download anything. However, we reiterate that uptime incentives are not part of the Sia protocol; they are entirely dependent on client behavior.

Payment for downloads is expected to be offered through preexisting micropayment channels [11]. Micropayment channels allow clients to make many consecutive small payments with minimal latency and blockchain bloat. Hosts could transfer a small segment of the file and wait to receive a micropayment before proceeding. The use of many consecutive payments allows each party to minimize the risk of being cheated. Micropayments are small enough and fast enough that payments could be made every few seconds without having any major effect on throughput.

7.4 Basic Reputation System

Clients need a reliable method for picking quality hosts. Analyzing their history is insufficient, because the history could be spoofed. A host could repeatedly form contracts with itself, agreeing to store large "fake" files, such as a file containing only zeros. It would be trivial to perform storage proofs on such data without actually storing anything.

To mitigate this Sybil attack, clients can require that hosts that announce themselves in the arbitrary data section also include a large volume of time locked coins. If 10 coins are time locked 14 days into the future, then the host can be said to have created a lock valued at 140 coin-days. By favoring hosts that have created high-value locks, clients can mitigate the risk of Sybil attacks, as valuable locks are not trivial to create.

Each client can choose their own equation for picking hosts, and can use a large number of factors, including price, lock value, volume of storage being offered, and the penalties hosts are willing to pay for losing files. More complex systems, such as those that use human review or other metrics, could be implemented out-of-band in a more centralized setting.

8 Siafunds

Sia is a product of Nebulous Incorporated. Nebulous is a for-profit company, and Sia is intended to become a primary source of income for the company. Currency premining is not a stable source of income, as it requires creating a new currency and tethering the company's revenue to the currency's increasing value. When the company needs to spend money, it must trade away portions of its source of income. Additionally, premining means that one entity has control over a large volume of the currency, and therefore potentially large and disruptive control over the market.

Instead, Nebulous intends to generate revenue from Sia in a manner proportional to the value added by Sia, as determined by the value of the contracts set up between clients and hosts. This is accomplished by imposing a fee on all contracts. When a contract is created, 3.9% of the contract fund is removed and distributed to the holders of *siafunds*. Nebulous Inc. will initially hold approx. 88% of the siafunds, and the early crowd-fund backers of Sia will hold the rest.

Siafunds can be sent to other addresses, in the same way that siacoins can be sent to other addresses. They cannot, however, be used to fund contracts or miner fees. When siafunds are transferred to a new address, an additional unspent output is created, containing all of the siacoins that have been earned by the siafunds since their previous transfer. These siacoins are sent to the same address as the siafunds.

9 Economics of Sia

The primary currency of Sia is the siacoin. The supply of siacoins will increase permanently, and all fresh supply will be given to miners as a block subisdy. The first block will have 300,000 coins minted. This number will decrease by 1 coin per block, until a minimum of 30,000 coins per block is reached. Following a target of 10 minutes between blocks, the annual growth in supply is:

Year	1	2	3	4	5	8	20
Growth	90%	39%	21%	11.5%	4.4%	3.2%	2.3%

There are inefficiencies within the Sia incentive scheme. The primary goal of Sia is to provide a blockchain that enforces storage contracts. The mining reward, however, is only indirectly linked to the total value of contracts being created.

The siacoin, especially initially, is likely to have high volatility. Hosts can be adversely affected if the value of the currency shifts mid-contract. As a result, we expect to see hosts increasing the price of long-term contracts as a hedge against volatility. Additionally, hosts can advertise their prices in a more stable currency (like USD) and convert to siacoin immediately before finalizing a contract. Eventually, the use of two-way pegs with other crypto-assets will give hosts additional means to insulate themselves from volatility.

10 Conclusion

Sia is a variant on the Bitcoin protocol that enables decentralized file storage via cryptographic contracts. These contracts can be used to enforce storage agreements between clients and hosts. After agreeing to store a file, a host must regularly submit storage proofs to the network. The host will automatically be compensated for storing the file regardless of the behavior of the client.

Importantly, contracts do not require hosts to transfer files back to their client when requested. Instead, an out-of-band ecosystem must be created to reward hosts for uploading. Clients and hosts must also find a way to coordinate; one mechanism would be the arbitrary data field in the blockchain. Various precautions have been enumerated which mitigate Sybil attacks and the unreliability of hosts.

Siafunds are used as a mechanism of generating revenue for Nebulous Inc., the company responsible for the release and maintenance of Sia. By using Siafunds instead of premining, Nebulous more directly correlates revenue to actual use of the network, and is largely unaffected by market games that malicious entities may play with the network currency. Miners may also derive a part of their block subsidy from siafunds, with similar benefits. Long term, we hope to add support for two-way-pegs with various currencies, which would enable consumers to insulate themselves from the instability of a single currency.

We believe Sia will provide a fertile platform for decentralized cloud storage in trustless environments.

References

[1] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System.*

[2] R.C. Merkle, *Protocols for public key cryptosystems*, In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

[3] Hovav Shacham, Brent Waters, *Compact Proofs of Retrievability*, Proc. of Asiacrypt 2008, vol. 5350, Dec 2008, pp. 90-107.

[4] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar, *Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction.*

[5] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Peolstra, Jorge Timon, Pieter Wuille, *Enabling Blockchain Innovations with Pegged Sidechains.*

[6] Andrew Poelstra, *A Treatise on Altcoins.*

[7] Gavin Andresen, *O(1) Block Propagation*, https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2

[8] Gregory Maxwell, *Deterministic Wallets*, https://bitcointalk.org/index.php?topic=19137.0

[9] etotheipi, Ultimate blockchain compression w/ trust-free lite nodes, https://bitcointalk.org/index.php?topic=88208.0

[10] Gregory Maxwell, *Proof of Storage to make distributed resource consumption costly.* https://bitcointalk.org/index.php?topic=310323.0

[11] Mike Hearn, *Rapidly-adjusted (micro)payments to a pre-determined party*, https://en.bitcoin.it/wiki/Contracts#Example_7:_Rapidly-adjusted_.28micro.29payments_to_a_pre-determined_party

[12] Bitcoin Developer Guide, https://bitcoin.org/en/developer-guide