Peer-to-Peer Distributed File Storage and Sharing on a Blockchain

Raymund Alksninis
Ali Khan
Matthew Kyawmyint
Tobias Lindberg
Tiffany Yuen

May 31, 2016

Abstract

A peer-to-peer distributed file storage and file sharing, utilizing blockchain technology to maintain a ledger of files and who stores the file fragments. Utilizing end-to-end encryption to maintain the user's, hereby called the storer's, privacy, it breaks down the documents into fragments and redundantly stores it among storage providers, hereby the farmers, in order to guarantee file integrity and retrievability. To retrieve the document from the network, the storer issues a request, along with a proof of the ability to decrypt the document, to all farmers who hold a fragment of the document and combines them into the original document. The storer is responsible to periodically to request proof of storage from the pool of farmers who hold a fragment and pay for the storage of the file via microtransactions for every proof of storage.

1 Introduction

With online file storage and online file sharing becoming more and more popular, security of files and the capacity of computers become more of a concern. To protect the security and integrity of users' documents, and at the same time striving to minimize the amount of storage needed in users' own computers, we would like to introduce a method of file storage and sharing using blockchains, which combines the ideology from Nakamoto's Bitcoin: A Peer-to-Peer electronic Cash System [1] and Shawn Wilkinson's Storj A Peer-to-Peer Cloud Storage Network [2].

Using the idea of encrypting files to public keys and private keys, and decrypting using both keys, storers can distribute the encrypted fragments of files (described in section 2) to different farmers, who are willing to store the fragments in their unused hard drive space (described in section 3). When the storers want to request the fragments back, they can check the integrity of the

fragments in a secure manner using hexdigest proof (described in sections 5 and 6) to avoid any potential malware and/or viruses added by the farmers to the fragments before combining the fragments to a whole document as intended. The storers would pay a fee based on the size of the document they want to store, and the farmers who hold a fragment would split that fee based on how many fragments they hold. This would create a self-regulating network where farmers would move from large pools to smaller ones in order to gain a larger share of the file fees.

2 Encryption of Files

When a user wishes to store a document on the network, the system splits the document into parts and then the user encrypts each individual piece, which we define as fragments. Each fragment's size is capped at 64 MB so to make it difficult to determine the nature of the contents stored in each fragment. In addition, a manageable size of 64 MB allows easy distribution through the network and quicker uptime when it comes to verifying the contents of the fragment.

To encrypt each individual fragment of the document, the user encrypts each part individually with a public and private key pairing. The standard encryption across the system will be RSA encryption using the SHA-256 hashing algorithm. The pair of keys is generated by the user, who then uses one to encrypt their fragments with RSA. This key is published to the blockchain along with other information about the document being stored, which is discussed further in section 4. The other key, which will be used to decrypt the fragments returned by the network, is kept by the user. This allows the storer to have control over who can access and decrypt the documents that they put on the network.

3 Distribution of Storage

Fragments will be stored amongst the farmer nodes within the network. The farmers would share fragments amongst themselves in order to join as many farming pools to split as many fees as possible. To protect against a loss of data, multiple farmers would be holding the same fragments, ensuring that, even in the event of a loss of a node, a fragment will not be lost once stored on the network. Potential redundancy will be strengthened as more farmers join the network, because storers are able to store copies of fragments amongst more unique nodes. This also means that any particular farmers' downtime will be less impactful to the availability and retrievability of a file at any given time.

4 The Blockchain

The blockchain will maintain a record and map of what files are stored on the network and where their pieces are located. Upon creation and distribution of a

document and its fragments, the blockchain will be updated with the documents information. This will include the root hash of a Merkle tree built from the fragments of the document, as well as the encryption key that was generated by the user, both of which can be used as unique identifiers of the document. The network locations of the farmer nodes where each fragment (and their redundant copies) are stored is also saved here.

The blockchain is updated whenever a new document is distributed to the network, whenever a farmer grabs a new fragment and joins a new pool, as well as when a farmer fails to provide a proof of storage. The latest block in the blockchain should then represent the most recent changes in the status of the network and its files. The blockchain as a whole should describe all documents currently being stored and where their fragments are within the network.

5 Monitoring of the Network

To make this a reliable service where data can be stored for longer periods, there is a need to be able to ensure that farmers live up to their part of the agreement: to actually store the data in the format it was distributed to them, as well as having it available for retrieval for the storer. We approach this issue in a similar way to [2], where a heartbeat is performed regularly or irregularly.

The heartbeat itself is a request sent out by the storer which is answered by the farmers to confirm that they still hold on to their piece of data. However, we need to come up with a way of doing this so that it is confirmed that each farmer stores his/her fragment at the current time, i.e. requesting only a hexdigest of the pieces of encrypted data would not prevent him/her from only storing that hexdigest and deleting the actual data.

Our workaround for that concern is the following: At the time of distribution, the storer divides the full document into pieces, as previously described. By adding a number of time stamps, such as the dates for the upcoming year (or simply a counter), to the end of the encrypted pieces of the document, this creates 'time unique hexdigests'. These hexdigests need now to be stored but that will demand significantly less storage space, assuming that the original document is large.

There is now no way for a farmer to pretend to hold on to his/her fragment because of the non-continuous property of the hash function used, i.e. increasing only the time stamp or counter at the end of the hash input should completely change the whole hexdigest. So on the farmer's side at each heartbeat, he/she needs to add the current time stamp or counter value to his/her fragment before responding with the hexdigest of the combination.

On the storer's side, each farmer's response has to be checked against the stored list of time unique hexdigests. If a farmer responds with an incorrect hexdigest or does not respond at all to a request, this is interpreted as if that farmer no longer store his/her fragment. If all the farmers of the document in question respond correctly, the storer can be ensured that the process proceeds according to plan.

Another issue a storer could possibly run into is if his/her list of stored hexdigests appears to be insufficient for the time he/she wishes to store the file. A concrete example would be if hashes are precomputed for a chosen heartbeat frequency and a storage time of three years. At the end of this period the storer realizes that he/she needs another two years of storage. One solution would be for the storer to retrieve the whole file from the network (in accordance with section 6), decrypt the fragments and precompute a new set of time unique hexdigests. This way, the farmers are not affected since they can hold on to their piece of data the whole time.

5.1 Market Based Regulation

Our idea, differentiating this service from Storj ([2]), is to let the network of farmers collectively but anonymously determine the sizes of the subnetworks corresponding for each stored document. That is, each file on the network will be stored in the form of fragments by a number of farmers. These farmers make up the pool for that file. And since a farmer typically store fragments from many different documents they will be part of many such pools.

With the storer setting the reward per heartbeat, the price for the storer will not depend on the number of pool members. This will incentivize farmers to join smaller pools where this reward is shared only among a smaller number of farmers and consequently even out the redundancy level between all the files stored on the whole network. For storers that wish to ensure a higher level of redundancy, they can offer a larger total reward for the pool, which will make the pool more attractive to a larger number of farmers.

6 File Retrieval

With each document now successfully encrypted, stored, and maintained on the network, it is important for storers or other users to be able to retrieve and access the complete document.

The blockchain holds the locations of storage for each fragment of a document, so it is trivial for a user to obtain the necessary information to make requests for all the fragments needed to assemble a document. After acquiring all the node addresses of the farmers, the user would send out requests for their fragment of the document, associated with a particular documents identifying hash, and the farmer would respond by transferring their encrypted fragment to the user. With single fragments being stored in multiple locations on the network, it should be highly unlikely that a fragment would be impossible to retrieve, even in the event of one or more farmers failing to respond to the request.

There is nothing inherently insecure with any user being able to make these requests for the fragments as they are all encrypted, so the data the user receives would be useless to them if they do not possess the private key with which they could decrypt the document. However, to guarantee to the storers that only

correctly privileged users will be able to obtain the files, as well as to prevent any malicious users from flooding the network with essentially useless requests and tying up farmers with constant file transfers, we look to prevent anyone who cannot properly decrypt the fragments from making transfer requests for them from a farmer.

To accomplish this we will have farmers ensure that users possess the private decryption key associated with the document. This verification process will need to meet a couple of requirements. First, the farmer must be able to know without a doubt that the requesting user possesses the decryption key. Second, the verification must be relatively computationally simple for the farmer to perform. And third, the storer must not have to relinquish the actual key to the farmer to maintain the integrity of the network, since the farmer could then decrypt what they had stored or a possible eavesdropper on the request could potentially acquire the key themselves.

Our proposed process which would meet all these described requirements is as follows: The user must submit their request with two strings, a simple plain text message of intent to transfer a fragment, and the second being the same message but rsa signed using the same key that is used in the decryption of the fragment being requested. Now the farmer, using the encryption key that is publicly available on the blockchain, can unsign the second message and compare it to the first message. If they are equal, then the farmer can continue through with the transfer of the fragment, else they may just ignore the request.

This should be sufficient to prove that the user holds the required decryption key. It is worth noting that this may still be susceptible to a flood of requests by a malicious user in order to burden farmers and plug up the network. A possible protection against this could be a cooldown period following a request where a farmer could ignore any requests that come quickly following a previous one by the same user. Other possibilities would include having some penalty assessed to the user upon a failed request, though this may require attaching a price to retrieval requests which may deincentivize the file sharing aspect of the network, something we hope to preserve.

7 Reward

A healthy network requires a reliable user base of farmers and storers. Attracting and retaining such users necessitates incentives to keep them engaged and active.

Storers pay for the file size they are storing with a storage fee every heartbeat. All the farmers who own a fragment of the file will have a share of the storage fee, making up a "farmer pool." We accomplish this by setting up microtransactions for storage usage between the storer and each farmer for the service period since the last heartbeat or the initial transaction, whichever is the most recent. If the storer fails to continue payment for too long a period, then the farmer has an interest in dropping the fragment in favor of a different fragment whose owner is more likely to pay. In the opposite direction, farmers would want to join smaller farming pools in order to get a larger fee for their storage space. If too

many farmers drop an old fragment that could potentially still pay out, other farmers would want to join the drastically smaller pool. This way, the market self-regulates and farmers might want to take the risk of joining small, old pools for the larger payouts.

The pricing of storage space will be set by the storer. The storer essentially broadcasts a request to store a file at an advertised fee/fragment, and farmers would take those fragments if it is a competitive price. After that, farmers can exchange fragments within the network in order to balance out their potential earnings.

Unlike Storj [2], the needs of this system does not necessitate a new currency; it can simply use an existing currency such as Bitcoin or something similar to facilitate payments between the storer and the farmer. The micropayments between the storer and each one of its contracted farmers can be automated via a client program, so that the storer is not bogged down by transactions, the number of which would be large due to splitting the file up into fragments and requiring fragment redundancy over the network.

8 File Sharing and Other Uses

Similar networks to ours provide only file storage on the network, that is, only the user who distributes their file is able to retrieve it. Our network allows for retrieval based only on possession of the decryption key associated with a document. Because of this, documents are able to be shared between many users through the sharing of said key. This allows for a flexibility of document access based on the storers discretion. One can keep the key to private files to themselves, perhaps have a collaborative project or a family photo dump and hand the key to several users, or simply publicly publish the key for a file intended to be shared, like a movie.

From this we can foresee a market for the decryption keys, where owners can advertise that they own a key to a particular document on the network, likely in a similar fashion to the proof of ownership described in section 6, and users could bid on or purchase access to the key, which they can then use to retrieve the document's fragments. This would be useful in that it could offset the costs of storage for the storer. There would be a concern over how to prove what a document actually is, meaning it would be considerably trust-based, but that would be something the other users would have to worry about and not something we could enforce.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[2] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, and Vitalik Buterin. Storj a peer-to-peer cloud storage network. 2014.