# Dexm: A Decentralized Web App Platform on the Blockchain

Antonio Groza, Ruggero Valli
Email: antoniogroza@gmail.com, ruggerovalli@gmail.com

***Abstract*—Traditional web hosting relies on a centralized entity to provide computing power. A decentralized web app platform could allow users to host content at a lower cost, while being resistant to censorship of oppressing governments. In this paper we propose a novel approach using the blockchain to store contracts and balances, and we describe a way to prove that the user accessing a website received a correct version and that he used the host's bandwidth. The generated tokens can be exchanged for coins on the blockchain as a form of payment for serving the user.**

## I. Introduction

Currently all web hosting relies on a centralized company. The security of the system is based on the user trusting the company. It is impossible to publish some permanent content, which can't be taken down because the corporations hosting it have to comply to local law and often they can be forced to remove content simply by threatening legal action against them. In this paper, we propose a way to build a network addressing those problems, that stays secure if the honest nodes have more than 50% +1 of the computing power, without requiring the people that use the content to modify their client.

## II. Files on the blockchain

In this section we define a way to store files on the blockchain that can be accessed. We define contract as a dictionary of filenames and hashes signed by a wallet. To publish files on the network the publisher has to generate a contract signed by a wallet with some balance. Then it will randomly pick nodes to save his content based on their proof of burn[1]. After the data was published, a fixed fee proportional to the file size will be taken each day from the contract's wallet and given to the nodes as a reward for storing the data if they can prove they still have the data via proof of storage[2]

## III. Retrieving static files

When a client needs to retrieve the data, it first of all would download the contract from the blockchain[3] and verify the signature of the website owner. A subset of the nodes that are currently storing the file is chosen pseudo-randomly so that the whole network can agree upon the choice. The client then proceeds to contact these nodes. It will ask each node a chunk of the file. The sum of the chunks will give the complete file. Here is described the process required to prove that the client received the file and that therefore, the nodes have accomplished their jobs and can be paid by the network:

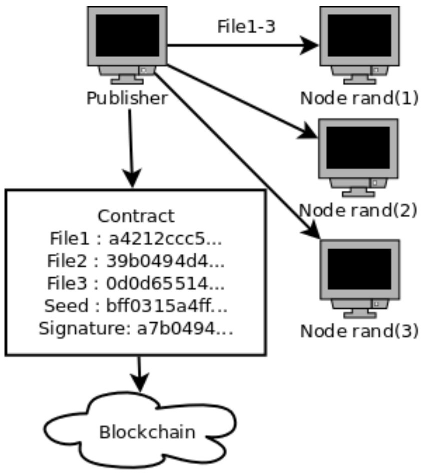

Fig. 1. Illustration of the steps necessary to put a file on the blockchain

- Each chunk is split into very small blocks.
- The first block of each chunk is downloaded for free, in parallel from all the nodes.
- The client then generates a hash between two indexes chosen pseudo-randomly
- To receive the next blocks the client will send to each node the hashes of all the blocks received for each chunk
- Each node will verify the hashes and if they are correct it will sign them
- The node will then send the following block along with the signed hashes
- From the third round each node will require a hash of the block it served signed by at least 50% + 1 of the nodes.

This way, if a node receives said proofs from the client, it can guarantee that a request from a client occurred, because other nodes responded to it too. Therefore, if half of the nodes plus one can confirm that a request occurred and they responded, then they can generate a token to send to the network and obtain their reward.

Additionally:

- The host can not manipulate the content delivered because the other nodes will detect an incorrect hash when they receive the proof of download of said node.
- It is difficult to forge tokens because half plus one of the nodes must agree. And they are chosen pseudo-randomly. This means that if most of the times half

plus one of the nodes are compromised, then the whole network must be compromised too.

## IV. Storing and retrieving secrets

Being personal data on a cloud server, or private information such as email address and credit card number provided to an e-commerce website, we all need to store secrets. We may need them only accessible to us, as with the example of the cloud server, or to a restricted number of people, like the admins of the website. To achieve this we use AES and Diffie-Hellman[4] to agree on the keys.

## V. Dynamic requests

While hosting static files is useful, they are very limited in functionality. To allow dynamic websites, users can publish code as a form of contract on the blockchain, as described in section 2. In this section we define a way to generate a cryptographic **proof of execution**. It demonstrates that the client requested the execution of some code and that it received the correct result. The proof is generated as follows:

- The client fetches the contract and checks the signature as described in section 3.
- The client chooses pseudo-randomly some nodes to execute its request.
- Some of the nodes will respond with the full response, others with the hash of it.
  The redundancy is necessary for the client to check if someone is lying on the result.
  The hashed responses are meant to lessen the network traffic.
- Each node generates the hash of the hashed response plus a secret nonce.
- Then proceeds to send it to every other node involved.
- If 2 nodes have the same hash both have to change it.
- If the client detects a conflict, it warns the network.
- As a consequence, each node has to reveal its nonce.
- It is then easy to determine who lied.
- The rogue node(s) is punished by diminishing his proof of burn.

The secret nonce is necessary to avoid any rogue node to simply copy the hash he got from another node.

## VI. Attacking Dexm

When designing Dexm, security was a priority. In the following section we list some of the attacks that we have hypothesized and we explain why they would be mitigated on a large scale by the network.

- **Sybil attacks**: since involving every node in every event would be a huge waste, it is necessary to reduce the amount of nodes needed to process an event to a small subset of the network. But that poses some security risk because an attacker could control the chosen subset. To mitigate this possibility we choose the nodes pseudo randomly. This way a lucky attacker could get a couple of good events, but over a large amount of requests it won't be able to do any damage.
- **Cheating clients**: When designing Dexm we assumed that clients are always evil, and to address that we have various mitigation. For static content we designed the protocol in such a way that the client has to prove that it received the content, to download the next part. While on dynamic requests, we assume that even if the client lied about integrity, the cost of checking the nonces is negligible.
- **Cheating nodes**: Cheating nodes have the biggest potential to cause problems of all parties involved in a request. Therefore, when serving static files, we have the other nodes check the validity of the hash generated by the client, while on dynamic requests, resilience to rouge nodes is achieved by sharing a hash of the response and a nonce; this way a node is forced to compute the response to generate a valid hash and get paid.

## VII. Conclusion

We have proposed

## References

[1] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J, and Felten, E. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In 36th IEEE Symposium on Security and Privacy (S&P).

[2] David Vorick, Luke Champine, ”Sia: Simple Decentralized Storage” https://www.sia.tech/whitepaper.pdf

[3] Satoshi Nakamoto, ”Bitcoin: A Peer-to-Peer Electronic Cash System” https://bitcoin.org/bitcoin.pdf

[4] Whitfield Diffie and Martin E. Hellman: ”New Directions in Cryptography” https://www-ee.stanford.edu/ hellman/publications/24.pdf