FROSTDAO: Collective Ownership of wealth using FROST

Rahim Klabér

May 15, 2023

Abstract

1 Introduction

In the years leading to the 2008 financial crisis, banks engaged in excessive risk-taking for the goal of profit. They invested in risky loans using their depositors' funds and were bailed out by the government when they failed. Recently, the financial system has once again been put to the test with the failures of multiple banks. In their simplest form, banks act as a middleman between depositors and borrowers. They pool the deposits together and give out loans to other parties while taking a cut of the fee. More importantly, banks act as gateways to today's financial world. Without a bank, a person cannot easily invest, pay online or get loans. However, how can the banks be trusted given their history?

Bitcoin emerged as an alternative to the global financial system. It gave individuals an alternative to banks and allowed normal people to securely send money to each other using the Internet. Many thought Bitcoin could be an alternative to the current financial system. However, Bitcoin has remained a tool for speculation. Real-world use is impractical due to high transaction fees, low throughput, and because it is hard to use correctly by non-technical individuals.

TODO: I don't like the bitcoin part :thinking:

Bitcoin is lacking in several areas to become an alternative to the financial system. Some of which are:

- high throughput
- cheap payments
- privacy
- collective ownership of wealth

High throughput and cheap payments are actively being worked on with the development of the Lightning Network. The Lightning Network is a protocol that lives on top of Bitcoin. It allows for cheap and fast payment by batching transactions and settling them on Bitcoin at a later point in time.

There are a number of Bitcoin mixing services to enable more privacy. These services mix the funds of different users by sending them to newly created wallets.

While collective ownership of wealth on Bitcoin is possible with current tools, it is impractical due to high fees and low scalability. Collective ownership would allow for a group of individuals to truly be their own bank, where each individual is part owner. Money can be pooled and invested. This process is transparent and a majority of the participants need to agree for anything to be done.

In this paper, we contribute to the goal of making Bitcoin an alternative to the financial system. We describe and partially implement a critical primitive for the collective ownership of wealth using Bitcoin. Using this primitive, individuals can create shared Bitcoin accounts with hundreds of others. We achieve this without any overhead to transaction size. Our system can be used by anyone and is compatible with existing Bitcoin tools and services such as the Lightning Network and various mixing services.

2 Background

2.1 DAO

A Decentralized Autonomous Organization (DAO) is a collectively-owned, blockchain-governed organization[cite]. DAOs are leaderless organizations where deci-

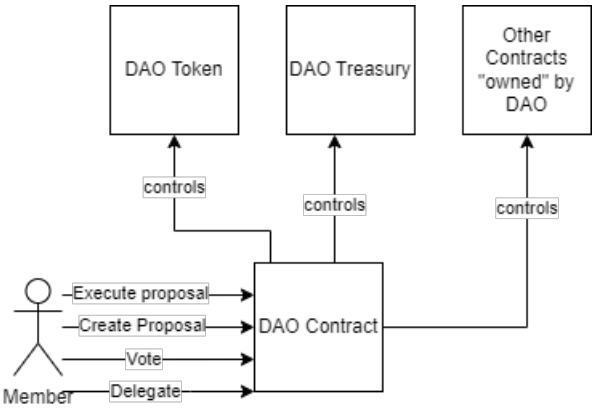

Figure 1: Traditional DAO architecture.

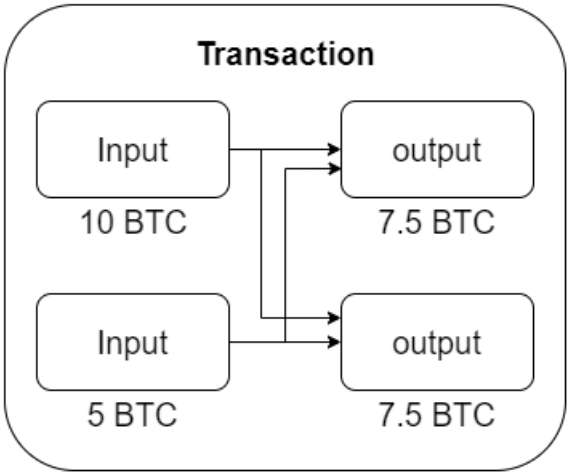

Figure 2: Simplified Bitcoin transaction. Shows inputs with Bitcoin values being consumed to create new inputs.

sions must be made through voting on proposals that are created by its members. Traditionally, DAO consists of a number of components. First, the DAO smart contract, which handles proposals and voting. Second, a DAO token represents a state in the DAO. Third, the DAO treasury contract holds the assets of the DAO. Lastly, various contracts or applications are governed by the DAO.

The DAO smart contract is responsible for registering proposals and allowing members to vote. Proposals include instructions that are executed by the smart contract if the proposal receives enough votes. Often, creating proposals requires a certain amount of voting power.

The DAO token is used to show membership in the DAO and represent voting power within the DAO. The DAO controls the DAO token smart contract, which can be used to issue more tokens or remove existing tokens from circulation.

The DAO treasury holds the assets of the DAO. This can include DAO tokens, but also other tokens on the Blockchain. The treasury is used to pay members who contribute to the DAO and is used to pay for other expenses.

A DAO may have control over other smart contracts. This enables the DAO members to control and make changes to those contracts when they deem it necessary. For example, AAVE, a blockchain lending platform, is controlled by a DAO that can adjust the risk parameters of the lending smart contracts.

DAOs often have a small number of members who are active and a large number who do not actively participate. This means that a large portion of power within the DAO is inactive, which can lead to not enough participation and proposals failing. A solution to this problem is delegation, members can delegate their voting power to other members who they think are trustworthy. Effectively delegating temporarily gives someone your voting power.

2.2 Bitcoin

Compared to newer Blockchains like Ethereum, Bitcoin uses an interesting transaction model. Bitcoin transactions consist of a number of inputs and outputs. Each input is an output of another transaction.

Outputs have a Bitcoin value attached to them, which can be used to pay other Bitcoin accounts. Outputs can be created by ”spending” an input which renders that input unusable anymore. Each input has a small program that must be satisfied to spend the input. This allows for some interesting constructions. Figure 2 shows a simplified view of a Bitcoin transaction

As mentioned previously, Bitcoin inputs include a program that must be satisfied to spend the input. When attempting to spend an input, the user provides inputs to satisfy the input program. When doing a normal payment the input to the program is a signature proving that the

user owns the address that is allowed to spend the input.

The spending rules can allow for more than just sending payments. One example related to the problem statement of this paper is requiring an input to be approved by multiple users to be spent. This would allow a basic DAO to exist on Bitcoin where the execution of proposals creates outputs with arbitrary rules. However, this solution only works for extremely small DAOs as Bitcoin transactions shave a size limit and Bitcoin fees are dependent on the size of a transaction. Therefore, even for small DAOs this solution is not optimal as the fees would be high.

2.3 Threshold Signatures

Threshold signatures Schemes is a method for creating signatures where multiple users are required to collaborate to create a signature. A Threshold signature scheme has 2 parameters. The number of participants n and the threshold t where $t \leq n$. Only t participants are required to create a valid signature.

To create a Threshold signature scheme a private key must be split up and each member must have a share of the private key. This can be done by relying on a trusted party that would generate a private key and split it up for each of the members. This does not work for peer-2-peer systems as there is no trusted party. Instead, a distributed key generation protocol is used, which generates the key shares in a distributed fashion such that no party learns the actual private key. Compared to using a trusted dealer, distributed key generation is much costlier and takes multiple network rounds.

To create a signature t participants need to sign the message. The t signed messages can then be combined to create the final signature.

2.3.1 Flexible Round-Optimized Schnorr Threshold Signatures (FROST)

FROST is a Threshold signature scheme that is able to create signatures that are compatible with Bitcoin. FROST consists of two protocols, one for key generation and one for signing.

The key generations protocol consists of 2 rounds, which both require every participant to broadcast a message to every other participant. The first round is particularly problematic as the size of the message depends on the number of participants.

The signing protocol is much lighter than key generation. It also consists of two rounds. However, the first round does not require knowledge of the unsigned data. Therefore, the output of the first round can be batched, resulting in a much quicker signing protocol. In the second round of signing, each participant signs the data with their own key share. The signatures are then combined to create the final signature.

2.4 IPV8

IPV8 is a peer-2-peer networking library. It allows for the creation of applications that do not require a central server. One example, is MusicDAO[cite], a Spotify alternative aiming to give a more significant share of the revenue to music artists.

IPV8 works by keeping a list of peers that are periodically checked for liveness. Peers are introduced to new peers which they can add to their list to keep track of. IPv8 is particularly useful because it supports hole-punch and therefore works behind WIFI, where peer-2-peer applications normally wouldn't work.

On top of the networking layer, IPV8 has the concept of Communities. These can be seen as protocols that live on top of IPV8. Members of a Community can communicate with each other using Community specific messages which are handled in Community specific ways. For example, A Community named Torrent Community could implement torrent functionality.

3 System Architecture

3.1 System Design

4 Implementation

4.1 Bitcoin specifics

5 Evaluation

TODO: some sort of intro?

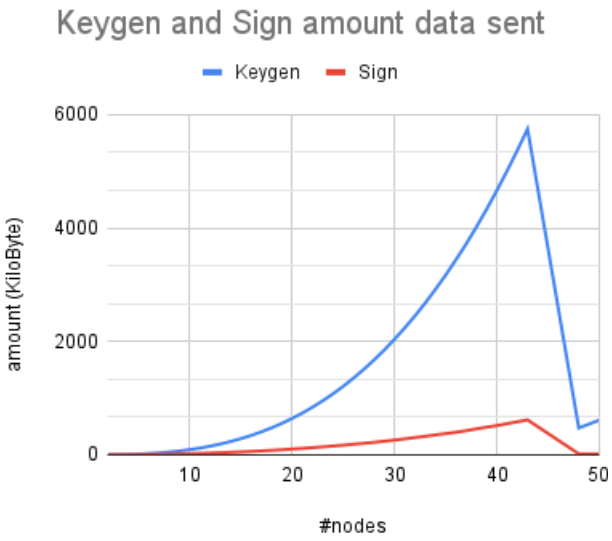

Figure 3: TODO: Remember to rerun. It timed out at around 40 nodes.Amount of data in Kilobytes sent during Key generation and Signing

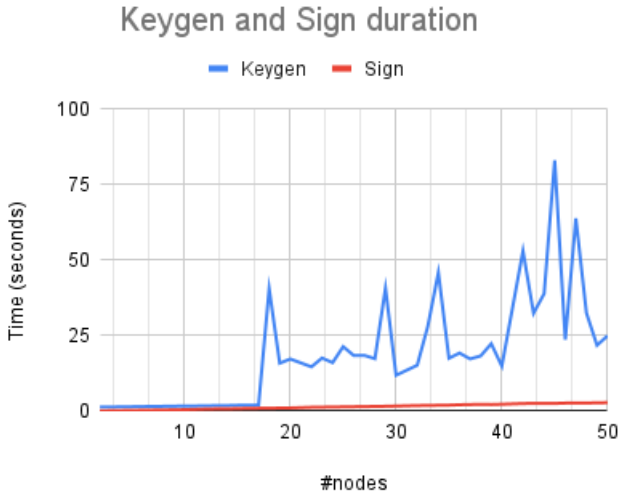

Figure 4: Duration of Key generation and Signing.

Class / Package	Line coverage	Lines of code
FrostManager	93%	404
SchnorrAgent	94%	106
FrostCommunity	65%	141
FrostViewModel	0%	156
ui	0%	980

Table 1: Code coverage of the FROSTDAO application.

5.1 Experiment Setup

We will run the experiments on a Windows 10 PC with 32GB of RAM and a Ryzen 7 3700x CPU that has 8 cores and 16 threads. We modified the code responsible for communication and signing to the work in a Desktop environment with a Java virtual machine. This included compiling the native code to work on Windows. Our experiments will be run within one application that is responsible for creating the individual nodes, that will represent a participant in the DAO. Each node is an IPV8 node that runs the entire IPV8 stack. However, since all of the nodes are on the same PC network latency is not a factor. We limited the number of nodes in the experiment to 50, as we ran into problems with more than 50 nodes. TODO: mention our configuration? So the timeout we used etc.

In our experiments, we are interested in the performance of the FROST algorithm as this is the most expensive part of our system. We measure the performance in two ways. First, we measure the time it takes to Key generation or to create a signature. Note that in the case of signing, we are doing the 2-round procedure and not the optimized 1-round version. Second, we measure the amount of data that is sent when signing. This is important as we want the system to be usable on mobile devices. We further investigate by introducing artificial delays to simulate potential network delays and we introduce random packet drop to investigate performance in a more real-life scenario.

5.2 Experiment results

TODO: maybe we can chec the amount of retries? Figure 3 shows the amount of data sent during Key generation and signing. We notice that Key generation scales exponentially in the amount of data sent. This is expected as the size of messages sent during Key generation depends

on the number of participants. While the graph paints a bad picture, keep in mind that this is the total data sent and not data sent by one node. This still means that each node sent and received around 150KB of data. Even this low amount of data can be problematic as everything is sent using UDP packets and may therefore be dropped without warning, resulting in even more data being sent. In contrast to Key generation, the signing protocol requires significantly less data to be sent. This is expected as each signing operation requires a constant amount of data per participant.

Figure 4 shows the duration of the Key generation and signing protocols. Both Key generation and signing have an extremely low duration up to 18 nodes, after which the duration of Key generation increase dramatically. This is because, at this point, the size of messages sent during Key generation is no longer small enough such that the UDP packet is delivered reliably. Attempting to use UDP packets at this point will result in them getting dropped. The dramatic increase in duration is due to EVA, IPV8 TFTP protocol for sending larger amounts of data. This protocol splits the data into chunks and sends each chunk via UDP while using acknowledgment to make sure that each chunk is delivered. EVA does not send the data immediately and instead schedules transfers in the future which results in a large spike in duration.

TODO: havent done this yet

Figure x shows the duration of the Key generation and signing protocols with varying artificial delays. In the case of Key generation, we expect the results to be similar to the results when no artificial latency was introduced. This is because the overhead of EVA is much more significant than the overhead of delays. Additionally, even though there is a delay, many nodes will send messages concurrently and therefore the number of nodes does not influence the increase in duration. The impact on signing will be more significant, but it will not scale as the number of nodes grows.

Figure x shows the duration of the Key generation and signing protocols with varying levels of packet drop. We expect that the duration of both protocols will be significantly increased. This is because our message acknowledgment system has a high timeout and will therefore wait a long time for an acknowledgment. A single packet drop will result in delays in the order of multiple seconds. The high timeouts are to accommodate the usage of EVA and while it can be improved, this would lead to more complex code.

6 Conclusion