Arching Kaos Infochain Whitepaper

Kaotisk Hund

November 13, 2021

Abstract

Contents

Chapter 1

Introduction

The following is a document on how an info-chain is may be constructed and on how it could be analyzed. Along the way, Arching Kaos Info-chain is introduced as a proof of concept.

Chapter 2

Arching Kaos

The words "Arching Kaos" are referring to the Asset used on Stellar Network, a decentralized block-chain. There is a project called "Arching Kaos" which is issuing and distributing the Asset. It's main cause is to create a distributed and decentralized web-radio station.

Chapter 3

Infochain

3.1 Introduction

The word "Infochain" is used as reference to the chain of information we are producing. The last decade, block chain technology covered many aspects of financial and data exchange requirements. Many applications have been developed based on crypto-currency networks. Along side, smart contracts are a part of the many possibilities someone can use.

3.2 Block chain technologies

To make a chain, most of these technologies have a reference into every block of data to the previous valid block. This is the block-chain. Nodes of the network validate each block so there is one block-chain.

3.3 How is Infochain any different

Firstly, we build our "chain" on top of a block-chain. We use the transactions to pass information, so on top of a block and add information. Now, we already are in a meta-block-chain. For now, we are not validating transactions as the rest of block chain technologies do. We add the previous "block" but anyone can reference to that as well. Their inserts won't be discarded by anyone. Multiple types of blocks can co-exist as well. Anything can be "posted" on the chain. It can be a fork of it, a whole different chain, or a "main" place. People can use encryption as well and post encrypted messages as well.

3.4 Rendering

There is also the need to read these information. To do that, an open protocol has been developed, since the use of such a chain is possible to be used for far too many cases. Different or bundled application can be developed to read the chain and present it to the public through internet. To do that we create one protocol, a predefined set of rules on how the information will be formatted and structured.

3.5 How do we do that?

We make use of IPFS to store information in a way that we can retrieve them and validate them, serialize them and present them.

IPFS is creating a CID for each file or folder is added to its repository. Furthermore, while connected to any network with other instances of IPFS running, the same information can be retrieved using the same CID.

We want to be descriptive on what we want to do, while we maintain a series or chain of information.

Chapter 4

Protocol

To build a protocol we need to know what it's requirements are. By specifications of the memo text length on Stellar Network transactions, we know that the maximum length of an alphanumerical string is 28 bytes. We also know from specifications of IPFS that its CIDv0 is 46 bytes. Our first concern is to publish the IPFS CID to the Stellar network.

How things need to be done in a way that works and is reliable in the sense of what we want to achieve. In particular, what we do is encode an IPFS CID, split it and add its sliced parts into memo text in transactions of an asset **??** on Stellar Network.

4.1 Minimum transactions

To send the CID (46bytes) using a memo (28bytes),we need 2 transactions, this means that we can use the memo field twice ($28bytes * 2transactions = 56bytes$). Now, the CID $46bytes/2parts = 23bytes$ fits into the memos. There is a difference though $28bytes - 23bytes = 5bytes$ and a total of $56bytes - 46bytes = 10bytes$. These 10 bytes will be used to fill the gaps in a way that is recognizable to help us make assumptions about each transaction's memo later.

4.2 Encoding

Since we have 10bytes more, we can make use of them in a way that we can tell which part of the CID we want, appears in the transaction memo.

We split the 46bytes string into 11,12,12 and 11bytes
$CID_a = CID[1 - 11]$
$CID_b = CID[12 - 23]$
$CID_c = CID[24 - 35]$
$CID_d = CID[36 - 46]$
in sequence for CID.

Then, we append or prep-end the fitting amount of dots in each part to reach the maximum of 14bytes in order to reach maximum length of the transaction memo, so $DOTS_3$ = "..." and $DOTS_2$ = "..". For the first 2 parts we add prefixes and for the last ones we append:
$Memo_{PartA} = DOTS_3 + CID_a$
$Memo_{PartB} = DOTS_2 + CID_b$
$Memo_{PartC} = CID_c + DOTS_2$
$Memo_{PartD} = CID_d + DOTS_3$

4.2.1 Referencing

We also want to make sure that what we read is unique. By sending 2 transactions the CID can not be retrieved in sequence. We have a first part and a second one. Maybe congestion could result to many first parts in sequence and second ones mixed later in the chain.

In order to achieve continuity, we need to reference in each transaction parts from other memos.

So, we split the transaction memo in two parts ($28bytes/2parts = 14bytes$). In two transactions we can have 4 parts. Each part can be used as a reference to find the rest of the parts.

4.2.2 Multiplexing the CID

The CID can fit in that 56byte string provided by the minimum length ($56 > 46$). There is a difference though $56bytes - 46bytes = 10bytes$ which we can use to make some formatting. Also, 56 can be divided with 14 to give as 4 parts. Then can use each part of it, add another next to it and send it as transaction. One way to do is for parts A, B, C and D: AB, AC, AD where here is the original message ABCD. We make use of the 10bytes filled with dots, making unique additions for every part, instead of leaving blank the last 10 bytes of the D part. So we do as follows, from CID:

We have 4 parts of 14bytes. We can now mix them like this:
$Transaction_1 = Memo_{PartA} + Memo_{PartB}$
$Transaction_2 = Memo_{PartA} + Memo_{PartC}$
$Transaction_3 = Memo_{PartA} + Memo_{PartD}$

Because of splitting a unique message doesn't necessarily provide unique messages. Two or more CIDs can share some parts of the CID. To overcome this, we add two more parts, referencing B part this time, including the parts that been referenced only once ($Memo_{PartC}$, $Memo_{PartD}$):
$Transaction_4 = Memo_{PartB} + Memo_{PartC}$
$Transaction_5 = Memo_{PartB} + Memo_{PartD}$

In total now we have 5 memos to send. Because of what we did, splitting 1 IPFS CID into 5 transaction memos, we will send them by moving 0.2 of the asset for each transaction while for all of them it would be paid 1 asset unit

($\frac{1}{5} = 0.2$). In this case, we assume that this one pair of transactions made the containing message (CID) an asset called ZBLOCK4.4.1. However, payments can vary, which means that different amounts can be decoded differently as well.

4.3 Decoding

To decode, we gather transaction memos and start scanning for possible ZBLOCK CIDs encoded in the way described above.

If we find a similar formatted transaction memo matching either AB, AC, AD, BC or BD memos expected, we try to find the rest parts in the transactions. After that we have a list of IPFS CIDs that can be further analyzed.

We can now say that we added the CID, but what would be a useful way to add into IPFS that we can them information further more? We need a structure.

4.4 Structure

In order to be prepared for the chaos of information it can be dumped into the chain in any case, certain requirements can be accomplished according to the needs that can be raised up. So we should specify a prototype of how things look like in the parts we are going to read.

Two procedures are coming into place. We need to craft messages and we need to read them. Encapsulation of information is used. The

Our first concern is to publish useful information that can be decoded easily. The prototype is a way to make an example usage.

CIDs published in the asset's transactions should contain JSON formatted messages. We will refer to this as ZBLOCK.

4.4.1 ZBLOCK

This IPFS CID is called ZBLOCK (z block <- ze block <- the block). It should be a JSON object with two IPFS CIDs. ZBLOCK consists of a BLOCK and its detached signature in JSON format.

BLOCK

A BLOCK contains action, data, detach signature of data, previous CID and public GPG key of the publisher.

ACTION An action.

Can be literally anything! A convinient way to use it is refering to an object or a subject and to a verb using URL like scheme: e.g.: mixtape/add

DATA A data JSON object relative to the action.

DATA should be the place to be creative. It's rendered based on the action that is mentioned in the BLOCK. Can be literally anything, but is specified by the ACTION's handler. We use IPFS CID to reference to it.

DETACH The detached signature of the DATA JSON object.

GPG is used to sign crypto-graphically the DATA JSON file. We added directly, not through IPFS. Although we can do that as well.

KEY The GPG key of the signer/contributor of both the block and data.

It should be the same to be very sure about who contributed. There is no implementation of the checking for now since, but the tool to add new blocks require it.

PREVIOUS The previous ZBLOCK.

This is done if we want to go back to previous transactions but we don't have all the transactions. If the chain is maintained we can walk back to it through IPFS with out the need of reading more transactions from Stellar network.

4.4.2 BLOCK_SIGNATURE

This is the IPFS CID of a detached PGP/GPG signature of the BLOCK.

Chapter 5

Properties

5.1 Ownership

There is no direct ownership, the database that is created in this system can be attached to the transaction sender or the GPG key of them. However, the correct word is "contributor". The contributor is the Stellar address, packed with the GPG key.

5.1.1 Privacy policy

Your data is yours. Whatever you publish unencrypted, is everyones'.

Therefore, it's public information, that cannot be undone. Anyone can use it to refer to you.

In order to avoid your information go stolen, you should use GPG/PGP encryption, or whichever other you trust.

Me, Kaotisk Hund, as the internet figure or the real-life caster of this system, I am responsible for bringing this information and tools to you. That been said, every user of the algorithm is responsible of how they use it.

Malicious software, illegal activities, copywriting violations are NOT welcome. But there is no guarantee that all the users or some of them will not break this.

Each user has a 5-key identity. Stellar address, IPFS id, SSB id, CJDNS ip, GPG key. Whatever someone says, it like their fingerprint everywhere. The reason for this, is to manage a monetizing place (stellar address) with a file-hosting service (IPFS id) with a social profile (SSB id), a networking address for hosting (CJDNS ip) and an encryption/signing key (GPG key).

Possibly, someone can generate everything, retry stuff and try attack the network. Or simply try to make a half thing to get hands on the funds as it's been proven to be a method of attack on assets and tokens on other block-chain networks.

I don't know how to prevent that, or what should I do to keep everyone off the funds. 6.3.1.

Chapter 6

Appendix

6.1 Testing

Someone needs an amount of asset to make a transaction. Read 6.3.1 for info on this.

During the development and research common tools were used. Testing was done on Fedora Linux 34 workstation environment. The tools were run using:

1. Python
2. Bash
3. Javascript
4. Firefox

For Python, the installation on a local environment of stellar-sdk was used.

6.2 Examples

6.2.1 Encoding of IPFS CID

We have the following IPFS CID of an empty file

QmbFMke1KXqnYyBBWxB74N4c5SBnJMVAiMNRcGu6x1AwQH

We mix it with dots in this way

...QmbFMke1KXq..nYyBBWxB74N4c5SBnJMVAiMN..RcGu6x1AwQH...

6.2.2 Split encoded text to 4 parts

A ...QmbFMke1KXq

B ..nYyBBWxB74N4

C c5SBnJMVAiMN..

D RcGu6x1AwQH...

6.2.3 Mix its parts

AB ...QmbFMke1KXq..nYyBBWxB74N4

AC ...QmbFMke1KXqc5SBnJMVAiMN..

AD ...QmbFMke1KXqRcGu6x1AwQH...

BC ..nYyBBWxB74N4c5SBnJMVAiMN..

BD ..nYyBBWxB74N4RcGu6x1AwQH...

We send each the 5 strings: send tx 1 2 3 4 5

6.2.4 Example BLOCK

```
"BLOCK":{
    "action":"something/do",
    "data":"DATA_JSON_OBJECT_IPFS_CID",
    "detached":"INLINE_DETACHED_SIGNATURE_OF_DATA",
    "key":"GPG_PUBLIC_KEY",
    "previous":"PREVIOUS_ZBLOCK"
}
```

6.3 Notes

6.3.1 Asset Distribution

Started with issuing 10,000 ARCHINGKAOS.

Current price for one (1) transfer and store of information, using Arching Kaos Infochain is 1 ARCHINGKAOS.

Supposely, after someone acquires 1 ARCHINGKAOS, it's possible to reuse it by sending either to theirselves, or recycling it on their wallets, while transfering and posting content.

Theoritically, when someone learns something, they don't forget and they can do the learnt thing, again and again.

Because of that and due to the trust issues that governing our relations, the distribution of the asset is something very important.

The ultimate goal here is for everyone to be able to trust each other. That would mean that on earth, peace have been achieved and actions of people are based on their hearts with compassion, intuition and love.

The point is not to just give away to everyone, neither to restrict within a closed community of the “awesome”. The point is to maintain a caring community and expand it. Sky is the limit, maybe there is no limit at all!

So, since 2019, the start of the project over the internet called Arching Kaos, there were and some still are some contributors that put trust in the project along its subprojects: The contributors! Each one of them, will get 1 ARCHINGKAOS to start with.

After a holder makes use of it, they can bring people into the network, having also the responsibility of who they bring in.

Previous thoughts Another way of managing ARCHINGKAOS would be from a central automaton that could create a “bonding” BLOCK that could much someone’s Stellar address with one generated and managed from the automaton. Think it like this: when we touch, we don’t really touch. So when you are getting ARCHINGKAOS, you are NOT getting ARCHINGKAOS. Someone else is holding them for you. It is like a lottery on a phone game. But instead of getting to play or get paid and go, you just ask for some when you need them to package your stuff.

Further reading Since it’s possible for everyone to reproduce this system into other assets, I guess doing so is the best way to expand the system. Read more on HACKING.

6.3.2 Invites

Since we are building a interconnected Radio station, we need to get more people in this.

We invite people by sending them ARCHINGKAOS. Hitting an invite button and add them with email address or stellar address or something.

Note: You need to know who you are inviting, personally.

6.3.3 Handling

Internally we create a key for us and a key for the invited. We give them their key like this

- alice has access and wants to invite bob!
- alice has pressed invite button
- she puts the public gpg of bob
- we craft a message that using: gpg.public.bob gpg.secret.alice
- we create bob’s address: xlm.bob ak.trustline(xlm.bob) ak.fund(xlm.bob)

- inside funding: t.memo(aki(ipfs(encrypt(new(message)))))
- message is: we enrypt for bob and we sign as alice
- bob can then visit archingkaos with his public key to find messages for them: it's possible for bob to try a lot to get his files if he see an invite, this means that he can spend 1 ARCHINGKAOS to someone else (in form of invite) or make use of it inside the application ArchingKaos
- the application knows that bob and only bob used this kind of asset because to spend it he has to sign the thing so we use bob's secret key to sign the transaction message before we sent this. then we publish the packet of data [xlm, gpg] if someone wants to run ArchingKaos radio they can

In the end, we get a transaction of 4XLMs from the "client", with a REGISTER thing All messages are packed like this, so inside the send packet from IPFS we put two things: {"ipfsmessage","messagesign"} (we have to firstly verify the message) ("type":"ak/register", // the transactions address to the one we create "ipfsgpg", "ipfssignature" }

We reply to by creating an address for them:
{
"type":"ak/user/add",
"akuser":"address that requested to register"
}
[{ "type":"ak/mixtape/add",
"ipfs":"file",
"ipfs-detach":"of the file",
"ipfs-artist",
"ipfs-title"
}, { "ipfs.gpg.signature of the previous object"
}]

6.3.4 Hacking

If you want to hack this you are more than welcome to do so. Instead of only trying to break though, you can contribute your solution to the problem.

You can use this piece of software to maintain your own asset/token.

It is possible for someone to copy this program and use it to roll out another infochain on a different asset.

Things that would matter of research based on my opinion are:

Render all transactions and decide how to perform each BLOCK's action

This would be nice to have and use in general, it's creative subject.

Download latest transactions (like 5-100) and climb rest of the chain through BLOCK's previous IPFS CID

This would interesting on figuring out if and how each BLOCK inherits the previous CIDs correctly. Assuming that anyone could forge the previous' value it would be really valuable to verify in comparison with the whole of transaction history which is always the safest route.

Create subchains

Since our genesis block is an empty file, it would be interesting to start chains with other values as well. For example, a file containing 0 or 1 or other integers, simple words and/or phrazes.

Happy hacking

6.3.5 Contribute

Apart from reading 6.3.4, there is also this one in order to say a couple of stuff.

I would really appreciate any comments, issues, pull requests or whatever in order to make this better.

Since you may have found this on github `https://github.com/kaotisk-hund/arching-kaos-infochain` there are tools to help each other. If you feel doing so, you are more than welcomed to do so.

I worked mostly on how to split and stuck back the messages, so the "core" is developed, though this is done in parts.

Read ROADMAP file to get an idea onto which directions the project is mostly to move towards. Note, that ROADMAP is also incomplete.

6.3.6 Discuss

You can discuss in Keybase where you can find *arching_kaos* team. Channels there are *#general*, *#roadmap*, *#infochain*. There is also IRC client ready to use at `https://irc.arching-kaos.net`.