

Decentralized storage of versioned BIDS datasets with IPFS, datalad and Ceramic

Contact Details

Full Name: Kinshuk Kashyap

Email: kinshukkashyap.me@gmail.com, kinshuk.181it222@nitk.edu.in

Location: India, Vadodara

Portfolio: [LinkedIn](#), [GitHub](#), [Twitter](#)

Discord ID: 551330344310407179

Neurostars: <https://neurostars.org/u/kinshuk/summary>

Pursuing B.Tech., Computer Science & Engineering at NIT Karnataka, Surathkal - Class of 2022

Mentor: Shady El Damaty, PhD - shady.el.damaty@opscientia.com

Overview

The goal of this Google Summer of Code project proposal is to expand Opscibay, a scientific data management software, to *include support for peer-to-peer file storage protocols as part of a larger open-source pipeline for GDPR-compliant dataflows powered by self-sovereign decentralized identity indices*.

The field of neuroimaging has generated hundreds of petabytes of data illustrating the functioning of the nervous system in various modalities, contrasts, and scales. The accumulation of this large and diverse data has resulted in major digital logistical challenges for neuroscience researchers including 1) data interoperability, 2) bandwidth limitations for sharing massive datasets, 3) provenance and versioning, 4) redundancy and accessibility. The emergence of the Brain Imaging Dataset Structure (BIDS) specification has significantly lowered the barrier to interoperability, however, the ability to easily share versioned datasets, ensure authenticity, maintain dataset availability, and manage permissions remains an outstanding problem.

Current cloud-based solutions involve monolithic and expensive solutions that may act as a single point of failure - where if the provider (like AWS) becomes inaccessible, the dataset becomes inaccessible. This monolithic property also allows easy censoring (location-based or otherwise). Furthermore, several simultaneous downloads lead to bandwidth limits, making downloads of large datasets time-consuming, expensive, and ultimately impractical.

The Interplanetary File System protocol is an emerging web standard that offers a unified open-source service for *peer-to-peer sharing of datasets*. On IPFS, multiple peers may share components of a dataset simultaneously, offering greater bandwidth. Data is immutable, and easily verifiable by checking the hash of the Merkle root. This enables provenance tracking of

datasets, verification of authenticity, the potential for open access, and a tool against censorship. Datasets stored on IPFS can be anonymized and permissioned, available as read-only by those with sufficient privileges, by linking the individual records to research participant or data curator DIDs, which can be revoked at any time by the owner of the dataset.

In this project, a special IPFS remote for git-annex will be developed and integrated with datalad to enable intuitive query and retrieval of neuroimaging data stored on peer-to-peer networks. First, unit tests will be performed to fix a bug in the git-annex-ipfs special remote file and assessed against the performance of traditional cloud storage services. Next, we will test a pipeline for creating a versioned BIDS dataset with datalad stored on IPFS in public and encrypted formats. The ability for participants and researchers to sign dataset versions to ensure authenticity and enable permissions management will also be explored with DID standards. Lastly, documentation, walk-throughs, and supplemental guidance will be synthesized to enable smooth integration with front-end and UX development.

Implementation Plan (Project in Detail)

Minimum Deliverables

Datalad and git-annex-ipfs-remote

The [datalad](#) package is a command-line utility that enables provenance and version control of neuroimaging datasets containing binary files. Datalad also includes powerful metadata querying and general-purpose functionality for accessing datasets stored in a variety of web standards.

A *datalad* dataset is generally composed of a ‘*superdataset*’ and its ‘*subdatasets*’. The binary file versioning method is built on top of git-annex which handles git operations on large files by maintaining pointers in the work-tree and keeping an internal record of remotes that may be used to fetch the files. The top-level (versioned) dataset only contains information on where to fetch each smaller piece from, very much in the fashion of a *.git* submodule installation.

The best feature of datalad is the ability to download lightweight datasets containing only metadata, and then deciding which content to pull for specific analyses later. For example, on running the `datalad clone` command on the superdataset (hosted on, say, Github) only the meta-structure and fetching information is downloaded.

To retrieve any file, directory, or subdataset, one can simply use `datalad get [-n] [-r] PATH`, which downloads the relevant files on demand. **Datalad doesn’t currently support using [IPFS](#) as a remote, which is what the first deliverable focuses on.**

The first step is debugging the [git-annex-remote-ipfs](#) BASH script, and testing with sample data; this BASH script will invoke ipfs commands for handling data retrieval, deployment, etc. [\[documentation\]](#)

Once debugged and tested, integration with datalad will be explored so that one can upload datasets to ipfs and download them into any arbitrary number of systems. The workflow design begins with 1) use datalad to upload dataset metadata to a bare Github repository and publish binary files to the ipfs special remote then 2) another user can clone the dataset metadata from Github and retrieve all of the individual files. Note the remote bare repositories on Github will only contain the datalad metadata on how to fetch the actual contents of the dataset.

An example workflow is described below:

```
$ git annex initremote ipfs type=external externaltype=ipfs encryption=none
```

At this point, we've configured IPFS as third-party storage for the annexed contents of the dataset (to be specific, the IPFS content is now a *sibling*). Now, to share the dataset with someone, we need to *publish* it. Let's say we [configure](#) the dataset to publish on Github, we can then run:

```
$ datalad push --to github
```

On the other machine where we want the data,

```
$ datalad clone <github url>
$ datalad siblings -d "local_directory_path" enable -s ipfs
```

Now, we can fetch the whole dataset, or any individual file/set of files:

```
$ datalad get <file_path_inside_dataset_directory>
```

BIDS validation

Datasets published on IPFS are immutable, thus there is a strong imperative to feasible quality control and ensuring compliance with data interoperability standards prior to publishing. A Javascript utility wrapping [bids-validator](#) will take care of validating BIDS-compliant datasets, and will integrate with the datalad pipeline for deployment and pinning to IPFS.

A WebApp frontend will be built using ReactJS. Datalad integration will be handled through datalad extensions (a simple example can be found [here](#)). Sample [BIDS datasets](#) will be used for testing the pipeline, using bids-validator for the heavy lifting of validation. This webapp will be further expanded to support encryption and key sharing, as described next.

Signing datasets for verification

[EIP-2844](#) is a standard that allows Ethereum wallets to support new methods for signing/decrypting data based on [DIDs](#) (Decentralized IDs). Pairing this with the dag-jose IPLD codec, we can deploy authenticated (with signatures) and encrypted data on IPFS. The low-level functionality will be managed using [key-did-provider-ed25519](#) (an implementation of EIP-2844) and [js-did](#) (this allows representing users as DIDs, connected to their ETH wallets). The dag-jose codec allows the creation of linked and signed data structures by creating JSON Web Signatures (JWS).

Signing and deploying objects is as simple as a few lines of code (from the Ceramic documentation):

```
async function addSignedObject(payload) {
  const { jws, linkedBlock } = await did.createDagJWS(payload)
  const jwsCid = await ipfs.dag.put(jws, { format: 'dag-jose', hashAlg:
'sha2-256' })
  await ipfs.block.put(linkedBlock, { cid: jws.link })
  return jwsCid
}
```

This function creates a signed object out of the payload, puts the object on IPFS, and returns the CID. Once we have the DID object, the JWS can be verified using `did.verifyJWS(jws)`. Once the BIDS-validation is done, the data will be signed using keys generated from the person's or organization's DID, connected to their ETH wallet (like MetaMask). This will eventually be extended to support deep identity indices that allow sign-ins with other wallets, Google accounts, etc.

One possible issue might be the webapp not being able to handle very large datasets. A workaround might be to distinguish between datasets that have signatures from each participant in the experiment from those that have a single batch signature for the dataset. This will have to be investigated with testing, once the basic prototype is ready.

Version Control

Using datalad, the datasets can be changed, and the patch pushed to IPFS (since data deployed on IPFS is immutable, we cannot change previous data). The patch will be signed, and the hash will be put on-chain.

An additional feature could be a history explorer that shows the entire history of the dataset. This won't be included in the minimum deliverables but can be explored if time permits.

Additional Deliverables

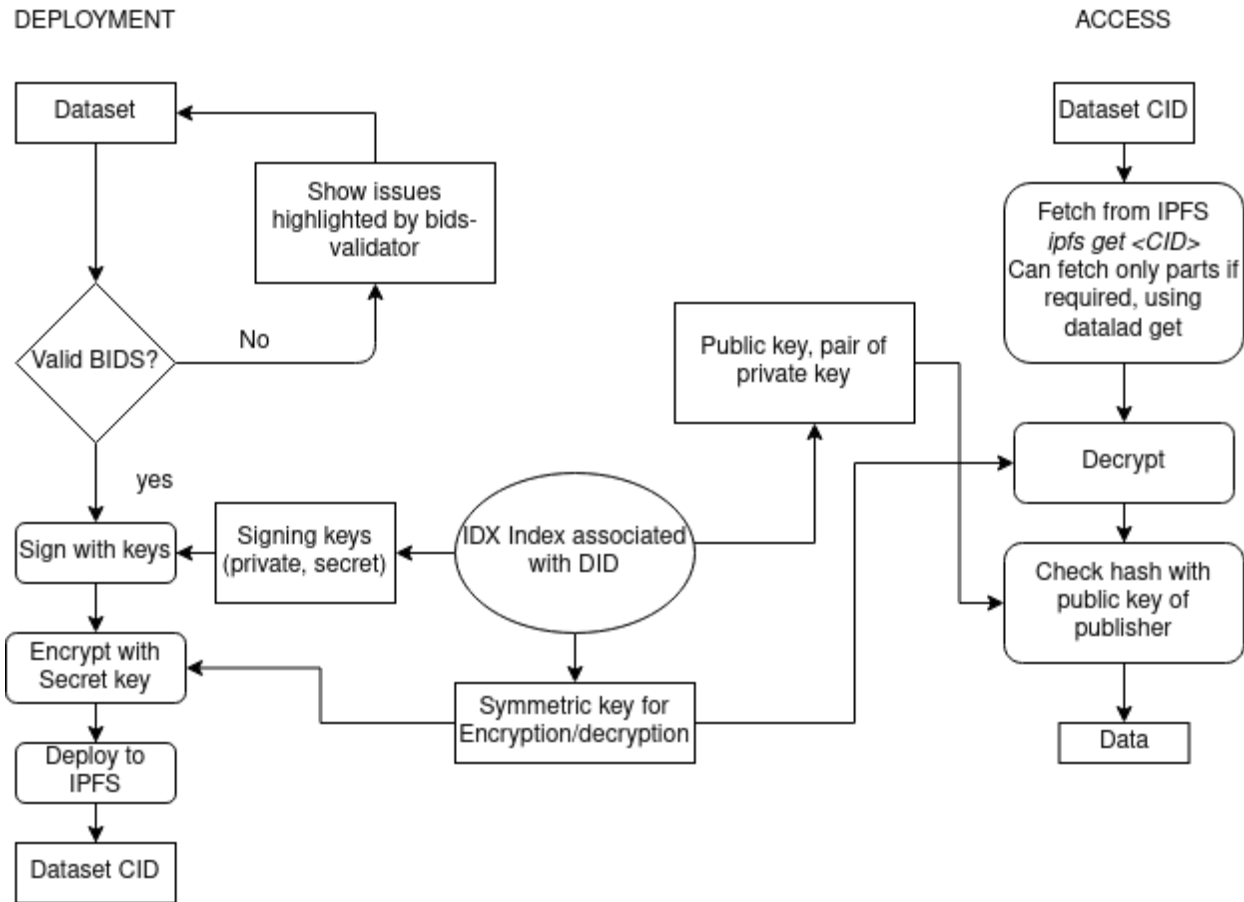
Encrypted datasets

Encryption is provided by ceramic libraries, similar to signing. Once deployed, the data cannot be read without having access to the keys used for encryption. Authentication through key sharing can be implemented using IDX and associated libraries. [IDX](#) lets us associate pieces of information to particular DIDs, through key-value mappings. This property can be used to store secure secrets, in this case, the encryption key.

[Ceramic](#) uses a document-based storage system, where each DID is associated with an identity index. This is the key-value map that links the DID with different documents (for example: documents for profile information, public keys, etc.). To store our encrypted secret using Ceramic and IDX, we need two things:

- A JSON schema for the encrypted payload (JWE). Publishing this schema returns a schema URL, which is referenced by all documents using this schema.
- An IDX definition to uniquely identify the encrypted secret in the user's IDX index

With this generic JWE schema, we need to create a unique definition for the secret data we want to store using the `idx definition:create` command. Multiple definitions can use the same schema, but each definition functions as a unique key in the user's IDX index. This returns a Document ID, which can be used to upload and download the document.



Overview of intended background processes in the webapp

Further details for key distribution to allow access to other DIDs will be explored during the course of the project, as I get more familiar with the libraries.

Dataset Discovery

Once the dataset is deployed to IPFS, we need a way for other people to discover these datasets. We can create an IDX schema that will keep a list of all the IPFS-deployed datasets that belong to an entity. Datasets with open access permissions would reveal them on the front-end explorer. This can be part of a dashboard that lists entities and their associated datasets. Metadata harvested from the BIDS specification and datalad fields can be used to construct rich querying options.

Deliverables and Timeline

Deliverables

1. A functional git-annex-remote-ipfs script, tested with sample data. An opened PR to integrate with datalad, using ipfs as a special remote
2. A Javascript webapp for BIDS validation, followed by deployment to IPFS, making use of work done on datalad in (1)
3. Ability to sign datasets for authentication before deployment, using DIDs. This will be an addition to the webapp in (2)

Additional (Stretch) deliverables

1. Support for encryption and decryption, with key management through IDX secrets.
2. Dashboard for managing version control and key distribution/revocation, as an addition to the existing webapp. This includes the dataset history visual explorer.
3. Integration with the parallel project which focuses on permissioned, anonymized data through IDX, to add IPFS deployment.

Detailed Timeline

I will be starting work before the coding period begins, as explained further in the “Working time and Commitments” section later on. The week 21 June- 27 June is kept light due to personal commitments. Being optimistic, I have included some of the additional deliverables in the timeline (support for encrypted datasets to be particular, since that is a very useful feature to have).

All weeks will begin with an open-hacking office hours meeting with the team on Monday at 12:00 pm UTC.

The planned weekly breakdown is:

Time Frame	Duration	Task
<i>Pre-GSoC</i>	03 May - 16 May	Learn about technical and implementation details of IPFS, Ceramic, datalad, git-annex.
<i>Community Bonding Period</i>	17 May - 23 May	Investigate methods for key sharing for encryption using Ceramic.
	24 May - 30 May	Debug and test git-annex-ipfs-remote script
	31 May - 06 June	datalad integration testing with git-annex-ipfs as remote, work on PR
<i>Phase 1</i>	07 June - 13 June	Begin work on WebApp for BIDS validation and IPFS deployment
	14 June - 20 June	Datalad extension for exposing functionality through REST API, for the webapp
	21 June - 27 June	Test the entire web app workflow so far (this week is kept light)
	28 June - 04 July	Add functionality for signing using DIDs
	05 July - 11 July	Implement versioning support, publishing patches on-chain for verification
<i>Phase 2</i>	12-July - 18 July	Test versioning with multiple patches and larger data
	19 July - 25 July	Work on IDX schema and test definitions for storing keys
	26 July - 01 August	Add Encryption and Decryption functionality to the webapp
	02 August - 08 August	Key distribution to other DIDs for sharing the encrypted data
	09 August - 15 August	Buffer Week

Plan for communication with Mentors

Discord will be the primary mode of communication with my project mentor (on the #opscientia server). Additionally, we will schedule video calls for reviews/presentations as and when required.

Are you interested to further support the project after the end of the fellowship?

I find this project inherently interesting due to the possibility of creating a new paradigm of open scientific datasets. So, I would like to continue being involved with the project after GSoC to work on polishing and iterating the stack, and helping with ongoing efforts to integrate the stack with the Ocean data marketplace.

Candidate Details

Motivation for selecting this project

I personally believe that in fields like neuroscience, where extremely large datasets are becoming the norm, decentralized storage will prove a key factor in open and easy access to data and consequently help accelerate research.

Is this the only project you are interested in?

Yes, this is the only project I'm applying to for GSoC this year.

What makes you a good candidate for the project?

I have been active in the open-source world since I was in high school, having participated in (and been selected as a grand prize winner) in [Google Code-in 2015-16](#), which was the equivalent of GSoC for high school kids. I became interested in neuroscience when I participated in [Neuromatch Academy](#) in 2020, where I had my first experience of interacting with large neuroimaging datasets. In terms of experience with the decentralized science stack, I have been experimenting with the tools as part of hackathons out of intellectual curiosity.

Working time and commitments

If accepted, I will be able to dedicate 25+ hours per week to the project. The only constraint I have is a family function, for which I'll have to take 2 days off (June 21 - 22).

My summer break from college begins on April 30, and college reopens on July 19. I'll begin work on the project in the second week of May, before the official start date. This is to ensure I have plenty of time to finish the project before my classes begin after the summer break. Once my classes have begun, I'll be able to contribute approximately 15-16 hours per week, which would be concentrated around the weekends. In any case, I plan to finish work on the main deliverables much earlier, and only leave the additional goals for the last few weeks.

CV

[My current CV can be found at this link](#)