Iris

Draft 0.0.1*

A decentralized storage layer for a modern data economy

Tony Riemer
Ideal Labs, 2022

Abstract

Iris is a decentralized network that provides a secure data storage, delivery, and ownership layer for Web 3.0 applications. It is infrastructure for the decentralized web, providing a storage and data exchange which enables the transfer and monetization of access to and ownership of data across chains, smart contracts and participants in the network or connected through a relay chain. Iris provides security, availability, reputation, and governance on top of IPFS, enabling data ownership, access management, and the commodification of latent storage capacity and content delivery. It applies defi concepts to data, reputation, storage capacity and availability to synthesize computation and storage and to represent off-chain assets in an on-chain context.

* The contents of this document may be subject to frequent and radical changes.

1. Overview

Today's data economy is heavily reliant on centralized authorities and third party providers to acquire, store, govern, and secure data. Such platform owners have also developed policies that govern how transactions are facilitated, disputes are mediated, and in some cases who can use or acquire access to their data.

There are some advantages to the existing centralized paradigm. Indeed, enterprises have decades of experience managing data and facilitating transactions and have made incredible advancements through the development of progressively superior data storage innovations. However, centralization has also resulted in the suboptimal situation where central entities have positioned themselves as gatekeepers to data access and have excluded millions of people from the data economy by creating barriers to entry. Additionally, as data is governed by centralized entities, individuals often have few tools at their disposal to control how their own data is used.

To resolve these issues, developers and decentralization enthusiasts have built both decentralized ownership platforms and decentralized storage platforms. However, there is often a lack of synthesis between them. Additionally, there is a general lack of Intellectual Property (IP) protection capabilities or moderation, and the decentralized open themselves to abuse or misuse.

We seek to address the existing shortfalls of the data economy by proposing a decentralized storage layer with a dual network topology to synthesize both data storage and ownership, allowing the application of tried and true DeFi concepts to data.

Built with IPFS [1] and Substrate[2] Iris embeds a private IPFS network within the Substrate runtime, facilitating the emergence of a dual network topology that enables the synthesis of computation and storage. This approach allows for the application of well-known concepts in Decentralized Finance (DeFi) to data. In particular, Iris treats content as an asset class and access as assets. The goal is to provide a simple and intuitive means of monetizing storage capacity as well as monetizing data in a cross-chain environment, which both constitute pivotal elements to the data economy and the transition to web 3.0, all while providing data-owners and consumers with a straightforward, extensible, rule based system through which they can buy, sell, store, and access data in a private and secure way. There are three major layers to the Iris blockchain:

- First, we introduce the idea of **data-assets** and the application of DeFi to data, including extending functionality of data assets and building applications using them
- Second, Iris's **storage system** uses a game-theoretical approach to construct an availability-encouraging mechanism that promotes decentralization of storage nodes
- Lastly, Iris is opinionated via a robust **governance, reputation, and moderation** protocol which provides protections for data owners and consumers as well as safeguards the integrity of the network.

[1] https://docs.ipfs.io/
[2] https://docs.substrate.io/v3/getting-started/overview/

2. Background

2.1 Centralized Storage and Ownership

*A **centralized system** is any system where data flows through a central authority. The central authority may perform validations and/or provide services, typically for a fee. The central authority is also responsible for storing and governing user data, of which it is generally the sole owner and steward.*

In the current state of the web, commonly referred to as "Web 2", centralized authorities constitute the majority of platforms and services. Centralized network architecture and applications make issues such as data management, ownership, and moderation easily achievable, however they also come with drawbacks.

- Data ownership is ultimately at the discretion of the owner of whoever is in control of the database behind the application. Large, centralized systems are more prone to data breaches[3] that can expose user data that leads to negative consequences both for users and the reputation of the platform. Furthermore, a centralized platform may monetize and profit from user-generated data without providing any additional benefits to the users themselves.
- Centralized systems are prone to failure, and seemingly minor issues can lead to catastrophic outages and downtime, such as when FaceBook, Instagram, and WhatsApp simultaneously experienced outages in October 2021[4].
- When a social aspect is added to data (such as applications like YouTube), lack of transparent governance and moderation is frequently an issue for users.
- Centralized systems silo data within their own private data stores, leading to a lack of means to easily share data between systems and to the rise of the subscription-based services with exclusive rights to data where users have to compromise which services they choose to use based on personal finances and not only preference.

2.2 NFTs

The non-fungible tokens (NFT) is the currently accepted decentralized solution to cryptographically proving ownership of something. Generally, an NFT stores the location of some data hosted on some server which the holder of the NFT then "owns". However, there is no cryptographically-verifiable relationship between the on-chain ownership and the off-chain storage. For example, an NFT that encodes the location of some off-chain data, such as the CID of a JPEG and the IPFS gateway where it can be accessed, has no relationship with the actual storage system where the data is stored. The ownership is protected by the blockchain, but the access to and availability of the underlying data is entirely independent. In scenarios where the ownership of an NFT is sufficient for some proposed functionality this poses no issues, however, in most scenarios which rely on the off-chain data, there are a variety of potential shortcomings to the approach taken above. Additionally, the completely unmoderated and unregulated approach taken by existing decentralized storage platforms opens the platform to major abuse and IP theft. Generally, there is a lack of reputation, governance, and

[3] https://www.upguard.com/blog/biggest-data-breaches

[4] https://www.theguardian.com/technology/2021/oct/05/facebook-outage-what-went-wrong-and-why-did-it-take-so-long-to-fix

moderation and a massive proliferation of scams into the market which could potentially devolve into a sea of dead links in the future.

2.3 Tech Stack Overview

We provide a brief overview of some of the key technologies involved in this paper.

2.3.1 IPFS

IPFS is a distributed system for storing and accessing files, websites, applications, and data [1]. IPFS uses the concept of the multiaddress and CID to identify data and who has it using a distributed hash table (DHT). IPFS can be thought of as the “hard drive for web3", though its capabilities are much further reaching.

2.3.2 Substrate

Substrate is a blockchain framework developed by Paritytech. Blockchains built with substrate can easily plug into a relay chain, such as Polkadot, and participate in a cross chain environment. Additionally, substrate allows developers the ability to easily customize any aspect of the blockchain runtime, execute offchain tasks, and enables forkless runtime upgrades. For further information, refer to the official substrate documentation at https://docs.substrate.io/.

2.3.3 Trusted Execution Environment

Several node roles (proxy, moderator) within Iris are dependent on the usage of a trusted execution environment. A trusted execution environment is an isolated execution environment running on a processor that ensures confidentiality and integrity of execution. Specifically, we use Intel SGX [5] to securely execute offchain services that are run as part of Iris. The usage of a trusted execution environment places hardware requirements on potential nodes: they must be intel processors. Most desktop and mobile devices with 6th generation Intel Core CPUs support SGX[6].

2.3.4 Zero Knowledge Proofs

Zero-knowledge proofs enable proving mathematical statements while maintaining the confidentiality of supporting data. This can serve as a privacy-enhancing cryptographic tool in a wide range of applications, but its usability is dependent on secure, practical and interoperable deployments [9]. Specifically, Iris uses a type of zero knowledge proof called a “non-interactive zero-knowledge proof” in which interactions consist of a single message from the prover to the verifier.

2.3.5 IRIS

The common approach taken today by many solutions is to add data to an external storage provider, habitually IPFS (most often through a pinning service), and to encode the resulting hosted location within a smart contract to associate it with some asset. In the case of IPFS, the user adds a file to IPFS and submits a transaction containing the CID using the blockchain network (e.g. Ethereum). While this approach solves issues pertaining to decentralization of ownership, it fails to solve the problems inherent to centralized storage, such as data availability, existence, and ownership and does not form a cryptographically verifiable relationship between data ownership and data availability or existence.

---

[5] https://www.intel.com/content/dam/develop/external/us/en/documents/overview-of-intel-sgx-enclave-637284.pdf

[6] more in depth SGX hardware requirements are available at: https://github.com/ayeks/SGX-hardware

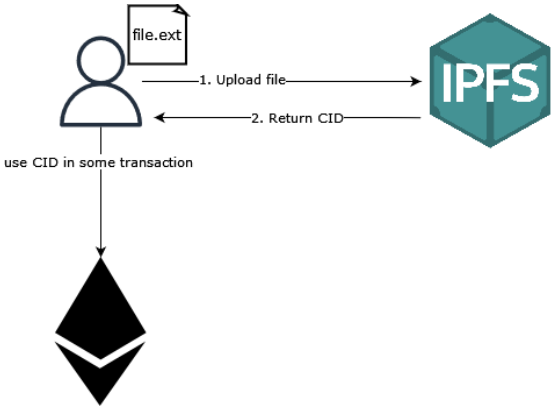

Fig. 1 Using external storage

Iris embeds a storage layer within the blockchain runtime, a novel approach that ensures a cryptographic relationship between data ownership and data storage . Our solution acts as an intermediary between users and IPFS, allowing for additional logic and validations that do not exist in IPFS. As opposed to figure 1 above, Iris first sends commands to the blockchain, where input data can be verified and logic applied, such as building models for data ownership and access controls. Commands are added to a queue, which can be executed by authorized network participants (proxy, moderator, and storage nodes), and the results are published on chain along with a zero knowledge proof.

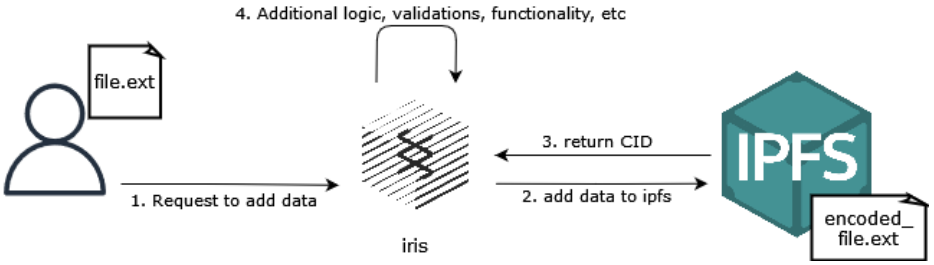

Fig. 2 Embedded storage

Since blockchains are state-management systems where all nodes agree based on consensus, we can construct a state to achieve much of the functionality of centralized content hosting and delivery systems but in a decentralized way. With the IPFS integration into our runtime, we define ownership and access controls to data as well as provide a means to monetize both data, storage capacity, and data availability provided to the network.

All data is stored off-chain within an embedded IPFS network, which exists as a secondary network to the blockchain network. Actual file data exists within the storage layer and is never exposed on-chain. This provides two important features that are required for the data economy: Files remain editable and removable (required for timely information like real-time location or unstructured data which require regular updates) and we do not bloat the network traffic by attaching files to transactions.

Iris treats data as an on-chain asset class with access controlled by assets minted from an owned class. Both content creators as well as network validators are rewarded for publishing and storing data, enabling an incentivized and decentralized pinning service on top of an embedded IPFS swarm. At its

core, we are delivering a permissioned, private and decentralized IPFS pinning service which enables data owners to monetize data access and to determine custom access rules

3.The Iris Network

In this section we describe the architecture of the Iris network, including the blockchain and storage layers. We discuss several of the key components of the Iris runtime and explain how each of the pieces fit together to enable a secure, reusable, storage layer for web3 applications.

3.1 Architecture

Iris's unique dual network topology enables the synthesis of storage and computation, allowing Iris to take full advantage of IPFS while retaining the privacy and security of a blockchain. By embedding IPFS within the substrate runtime and allowing pass-through wasm[7] calls via substrate's off-chain worker, Iris is able to control access to the embedded IPFS network using on-chain mechanics and incentives while data storage is managed entirely off-chain within a private, embedded, and decentralized IPFS network.

The primary network, or **blockchain layer**, is formed by the blockchain participants themselves. The blockchain layer uses a nominated proof of stake consensus mechanism to enable consensus. It provides the capability to treat both data and its availability (i.e. storage capacity, uptime, bandwidth) as on-chain assets. That is, it lets us track asset ownership and requests to IPFS on-chain while data itself is never exposed to the runtime. In addition, it allows the network to secure data through a novel **proxy re-encryption mechanism**, enable **governance protocols**, **a privacy-preserving reputation and feedback system**, **decentralized moderation**, and **availability-encouraging storage protocols**, among many other features.

The secondary network, or **storage layer**, is formed by a subset of nodes within the blockchain layer which both:

1) Run the embedded IPFS node
2) Are authorized to connect to a bootstrap node within the network

In general, any authorized storage node in the network also acts as a bootstrap node, thus allowing authorized nodes to proxy requests from the primary network (the blockchain) and interact with data in the secondary, embedded network (IPFS). Using **zero knowledge proofs**, the results of proxied IPFS requests are submitted and verified on-chain.

Iris is an opinionated network and imposes rules on which data is eligible for inclusion into the network. As such, proxy nodes and moderator nodes, selected by a proxy routing service and moderator routing service, respectively, act as the basis for executing these checks when data is injected into or ejected from the network.

[7] WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications. See: https://webassembly.org/

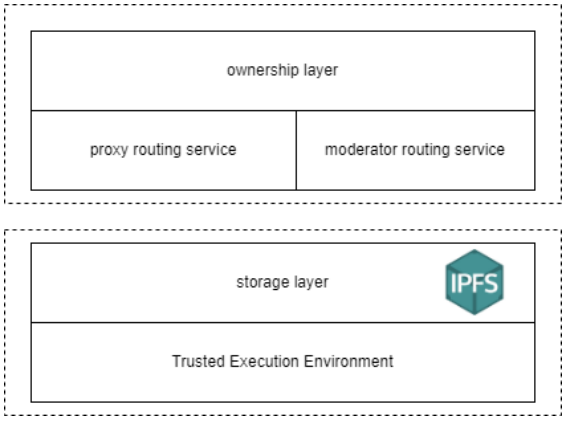

Fig. 3 High level architecture of an Iris node

Further, as Iris is a substrate-based blockchain, it benefits from many of the modules that already exist within Substrate, just as the usage of Grandpa and Aura[8] for block authoring and finality, as well as adapting several already defined and understood paradigms, such as the session, assets, and balances modules.

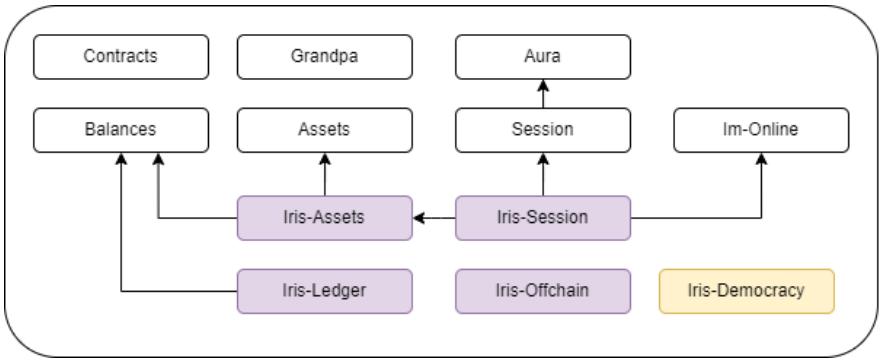

Fig 4. The Iris runtime as substrate modules

3.2 Iris Network Participants

There are six distinct, but not exclusive, roles that nodes can take on within the Iris network. Each role is unique in its needs and capabilities, and many, if not most nodes, will generally take on multiple roles. Fig. 4 details the architecture of each node type.

Data owner

Data owners are responsible for adding data and creating, managing and minting asset classes and assets. Additionally, they set their own business logic for the assets they create and to which data consumers must adhere to. This role is also responsible for renting storage space within the network in order to maintain the availability of their data. For the scope of this paper, data owners are the only nodes subject to reputation scores. In general, data owners never interact with the storage layer directly and must rely on proxy nodes and storage nodes.

[8] https://docs.substrate.io/v3/advanced/consensus/

Data consumer

The consumer role's capabilities are the most limited and simple of the four roles. Data consumers are responsible for retrieving data from Iris when they hold an asset minted from some owned asset class and for submitting feedback scores to fuel Iris's reputation system. Consumers implicitly agree to abide by the rules that the data owner determined, the enforcement of which is automatic and non-optional.

Validator/Nominator

Network validators are responsible for finalizing blocks in the nominated proof of stake[9] consensus mechanism. Nominators are responsible for nominating validator nodes.

Storage Node

A storage node is any node that has promised and proved a minimum amount of storage to the network (100 mb), a minimum ba. Storage nodes earn rewards for providing storage and availability to the network. The functions of storage nodes are more deeply discussed in section 4. Storage nodes are the only nodes capable of storing data in the storage layer (i.e. synonymous with 'ipfs pin CID').

Proxy Node

The proxy node is a democratically elected node responsible for enabling ingestion and ejection of data into the embedded IPFS network. This node runs specialized software off-chain within a **TEE** in order to verify the incoming data. This could include general validations such as malware and virus scanning or more specialized use cases, such as image recognition for IP protection. Proxy nodes are the only nodes capable of ingestion of data into the storage layer (i.e. 'ipfs add <data>') and ejection of data from the storage layer (i.e. 'ipfs get <CID>'). Further, proxy nodes provide the foundation for the network to enable (invisible) proxy re-encryption by acting as a third party that re-encrypts data when desired. Proxy nodes act as the gateways to Iris, and as such, the number of proxy nodes is directly correlated with the bandwidth of the network.

Moderator Node

The moderator node is a democratically elected node who must fulfill the role of a moderator. The moderator role is heavily discussed in section 5.4.

Below, we present the two basic architectures a node can take. The basic Iris node uses on-chain commands to use an off-chain worker which can call the embedded IPFS node. The proxy node is the only node with unique architecture due to the fact that it must execute commands within a TEE.

[9] https://research.web3.foundation/en/latest/polkadot/NPoS/index.html

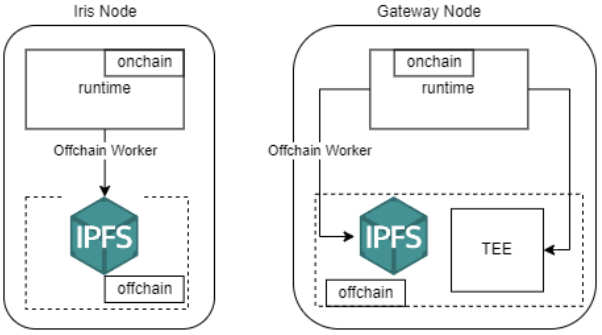

Fig. 4 The two basic node architectures

3.3 Data ownership as asset classes

Iris applies concepts from decentralized finance (DeFi) to data. In one context, Iris operates by allowing nodes to create their own tokens within Iris. These tokens are minted from a data-asset class, which is like a template that a node creates in the network and which they have admin access to and acts as the on-chain representation of their data stored in the embedded IPFS network.

3.3.1 Asset Classes

An asset class is a unique, non-fungible asset that exists on-chain. At a minimum, an asset class consists of a unique identifier, the *asset id*, an *owner and admin node* who is capable of managing the asset, and a set of *metadata* that defines the asset class's symbol or name. Admins of an asset class have the ability to burn or destroy, transfer ownership of, and to mint assets from the asset class. The existence of an asset class does not directly correlate to the availability of the underlying data, which must be managed by data owners themselves via interactions with the storage layer.

3.3.2 Composable Access Rules

Iris enables data owners to determine rules that consumers must follow when requesting their data from the network. These rules could include: limited use access tokens, minimum token requirements, time sensitive tokens, and many other use cases. We accomplish this through a set of smart contracts called composable access rules (CARs). They are composable in the sense that the data owner may specify any number of rules, each of which must be validated when a consumer requests data. In general, each rule stipulates a condition which, when met, results in the consumer being blocked from accessing data or in their token(s) being burned. If the token is not burned by the end of execution of each CAR, then the consumer can proceed to fetch the data. In general, the greater the number of CARs, the more costly it will be for consumers to access the data, since they must pay for each one to be executed.

More information on composable access rules is provided in section 3.5.

3.3.3 Provisioning Data Access Assets

Authorization to access data is equivalent with ownership of assets minted from an asset class which satisfy the composable access rules which the asset class has registered. To provision access to other users, admin nodes mint assets from their asset classes, which can then be distributed or sold to other nodes in the network. Each of these assets are functionally their own token and can be traded on a decentralized exchange for such data assets. These tokens form the basis for allowing access to data and building decentralized applications on Iris.

3.4 Data Spaces

A **data space** is a user-defined configuration and rule set that allows users to group data together into *curated* collections. More explicitly, the data space lays the foundation to enable user-defined moderation rules within the network and provides a means to securely associate disparate data sets with one another (i.e. under an 'organization'). All data in Iris exists within the context of a **space**, however, there is no limit to the number of spaces a data owner may associate their data with.

There are three major components to a data space that a user may define. The creator defines metadata (such as name, description, etc.), an inclusion policy, and a moderation policy. Inclusion policies may be either public or private. A private inclusion policy implies only nodes authorized by the owner of the space may associate data with the space. A data space's moderation policy consists of a moderator-to-storage-capacity-increase (MSCI) ratio and a collection of moderation rules, which includes both allowed file types and other specific checks and balances that each file added to the space must pass (such as text and image recognition checks or other machine learning models). The MSCI ratio implies that for some static amount of data to be added to the network within a static amount of time, there must be a minimum number of available moderators. This is to safeguard the space to ensure that data integrity is preserved and to ensure that space-inclusion finalization duration has a lower bound. Other moderation rules, which ultimately determine if the data may be associated with the space, are carried out by moderator nodes (the role of the moderator node is discussed more thoroughly in section 5.4).

3.5 Proxy Re-Encryption

Proxy Re-encryption (PRE) is a scheme through which a third party (the proxy) is able to alter the ciphertext of some encrypted data so that it can be decrypted by an authorized party. Iris leverages **proxy re-encryption** to secure data, allowing a data-owner to encrypt data without knowing the identity of a potential consumer, and for consumers to receive encrypted data without exposing or sharing their secret keys. Specifically, Iris uses a **unidirectional proxy re-encryption** mechanism as outlined in [10]. Data owners encrypt their data offchain using their secret key prior to ingestion. When an authorized consumer requests data, a proxy node re-encrypts the ciphertext for the consumer and delivers it offchain, after which the data owner can decrypt the data using their secret keys.

Iris uses two levels of encryption: a first-level encryption that cannot be re-encrypted by the proxy and a second-level encryption that can be re-encrypted by the proxy and then decrypted by delegatees. That is, a user Alice uses her secret key to encrypt data, the proxy node uses a proxy key to re-encrypt it for another user Bob, and Bob uses his secret key to decrypt it. Though our approach does not fully expose a data owner's secret key to the proxy node, data owners implicitly delegate to consumers re-encryption abilities. Since we already trust consumers (by ultimately providing access to data offchain), and since Bob could theoretically just transfer the data offchain, we do not see this as an issue.

3.6 Secure Offchain Client

Both data owners and data consumers must run an offchain client in order to receive files from Iris and to stage files so they can be ingested by proxy nodes. The offchain client performs encryption before a proxy node ingests it and decryption after delivery by a proxy node. There are two flows that the client follows: data ingestion and data ejection.

In the first flow, data ingestion, the client verifies that data does not contain anything malicious (e.g. viruses). Next, it encrypts the data with the user's private key and then stages the file in a secure file server to be ingested by proxy nodes when the node requests data ingestion via the blockchain.

The second flow enables nodes to receive files from Iris. The client listens for incoming data from proxy nodes. When data is retrieved, it is decrypted, verified, and finally made available for download to the node's local machine.

3.7 Data Workflows

Here we discuss, at a high level, how data ingestion and ejection occurs in the network.

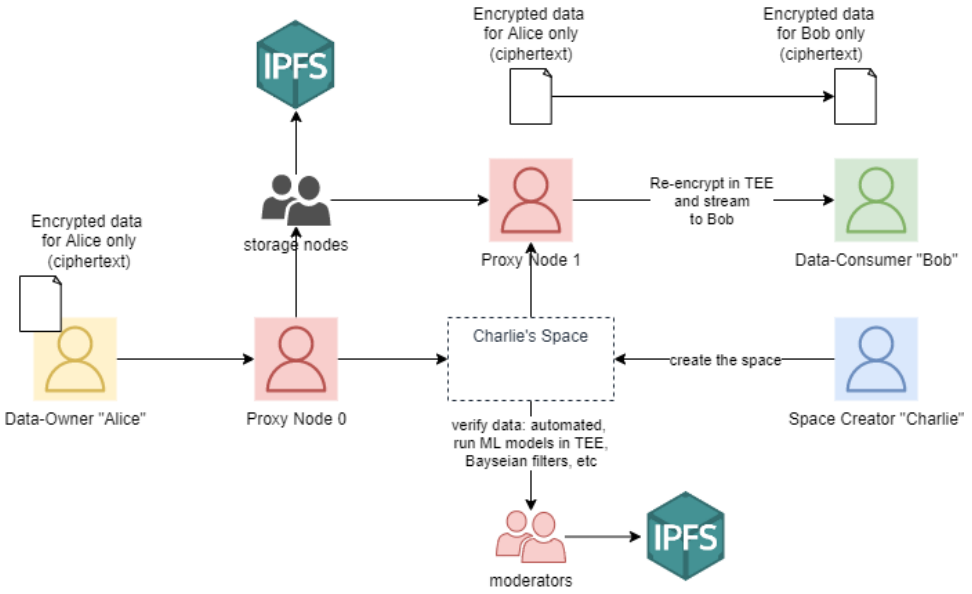

Fig. 5 E2E data workflow

In figure 5 we present a high-level end to end workflow of data ingestion into a space and data ejection from it. Assuming the existence of some space created by a node "Charlie", Alice is able to ingest her data into Iris through a proxy node and associate her data with Charlie's space. The space is subject to moderation rules determined by Charlie when the space was created (e.g. the file must be a .jpg or .png and can not be an image of a horse), as well as any changes reached by consensus by the participants of the space (i.e. those nodes who own assets minted from the space asset class). When a node Bob has access to the data, Bob fetches the data (this is done independently of the space) through a proxy node (that re-encrypts the ciphertext for Bob) and delivers it to him through the offchain client.

3.7.1 Data Ingestion Workflow

Data ingestion is synonymous with asset class creation. Each unique set of data ingested into Iris results in the creation of a new asset class, and with it, the potentiality to mint new assets, which act very much the same as a token, such as an ERC20, though with the possibility of easily extending its behavior and introducing enhanced functionality to your data.

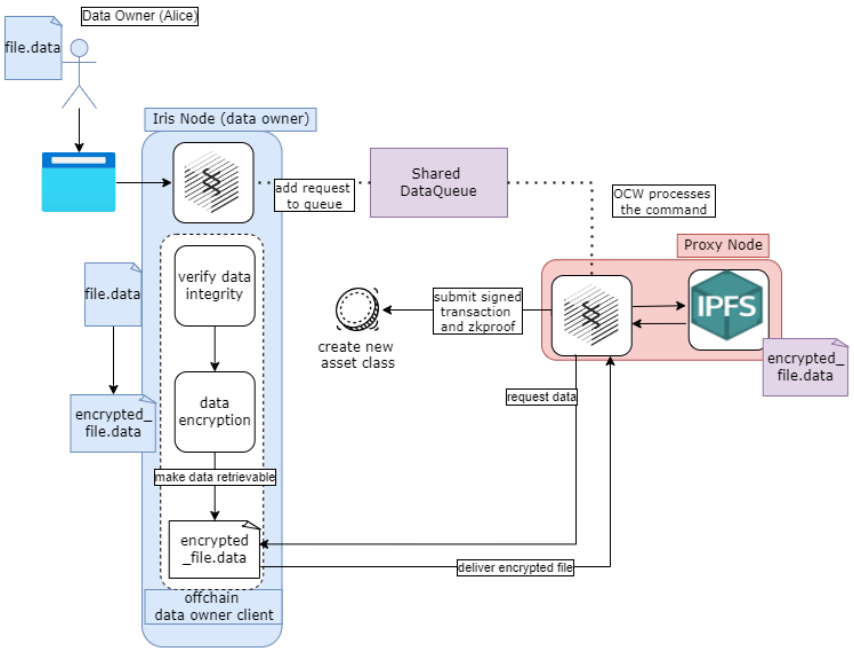

Fig 6. Data ingestion with a Proxy node

The data owner node, Alice, holds some data, say *file.data*. To begin the ingestion process, Alice uses the offchain client (see section: 3.5) to encrypt the file with her private key, creating the ciphertext *ecrypted_file.data*. Finally, Alice moves the file to the offchain client's file server. After the file is available, Alice uses her node to send a request to add data to the IPFS network, supplying her identity which is mapped to the unique identifier of her file server. A proxy matching service chooses a proxy node, which retrieves her file and adds it to the embedded IPFS network. Lastly, the proxy node submits a zk-proof to prove they added the file and creates a new asset class on behalf of Alice.

3.7.2 Data Ejection Workflow

A node is authorized to fetch data from Iris when they own assets that satisfy the composable access rules specified by the data owner. The owner of some asset (Bob) uses his offchain client to listen for incoming data from proxy nodes. Subsequently, Bob requests data from Iris. The proxy matching service selects some proxy node, which begins by verifying Bob's access to the data (by executing the composable access rules) and then fetches the data that he requested, re-encrypts it so only Bob's secret key can decrypt it, and delivers it to him. Bob then decrypts the data offchain, verifies the security and integrity of the data (e.g. verified correctness of data), after which he downloads it to his local machine.

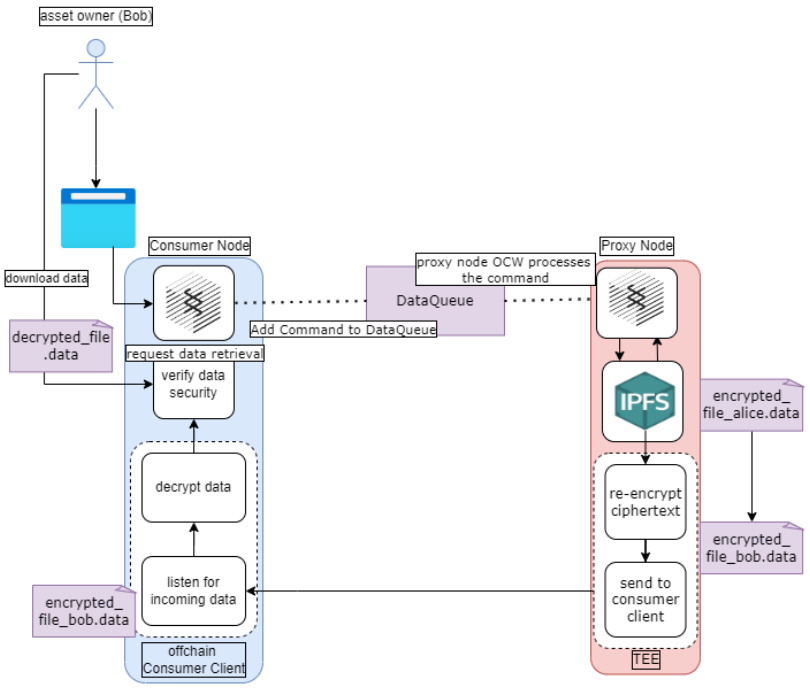

Fig. 7: high level data retrieval

3.8 Dapps and Smart Contracts

3.8.1 Chain Extension

Iris functions are available through a chain extension[10], which is exposed via the contracts API. Iris exposes functionality to mint and transfer assets from a contract, as well as to lock and unlock currency. The specifications followed in the chain extension follow the specifications of the extrinsics [11] exposed by Iris very closely.

3.8.2 Composable Access Rules

A composable access rule is any contract that fits the specification detailed here.

In general, a composable access rule must implement the following functions:

1) Implement an $execute(asset\_id)$ function that produces some boolean output. This function accepts the asset id of an asset class in Iris. It encapsulates the 'access rule' logic.
2) Implement a $register\_asset(asset\_id)$ function. This function accepts the asset id of an asset class in Iris. When executed successfully, it results in the asset id being 'registered' in the rule contract's storage.

With these two functions implemented and the contract deployed to the chain, data owners can then specify, using the contract address, which rules they want to be applied to their data. When a consumer node fetches data from the network, each of the specified rules must be executed by invoking the execute function in each registered contract. The result of contract failure results in the denial of data access.

[10] https://paritytech.github.io/ink-docs/macros-attributes/chain-extension/
[11] https://docs.substrate.io/v3/concepts/extrinsics/

3.8.3 Iris Asset Exchange

The Iris Asset Exchange (IAE) is the initial application that is built and deployed on the iris blockchain. The IAE allows data owners to sell access to potential data consumers. It follows a simple model where a data owner can register a static amount of tokens within the contract, which holds the assets. Consumers purchase data from the contract, which then rewards the data owner accordingly. Functionally, it is a DEX for data-assets and data-asset classes.

4. Storage Network

The goal of the storage network design is to ensure the storage layer is able to remain decentralized as it expands while also retaining high-availability. That is, we want to minimize barriers to enter the network as a storage node while also encouraging nodes to maintain a minimum level of availability. In this section, we first discuss the conditions for becoming a storage node and the on-chain representation of storage space in the network. Then we discuss a modified "stochastic replication game" which minimizes unavailability of data and costs to optimize availability.

4.1 Providing Storage to the Network

Any participant in the network who meets a set of minimum hardware requirements, is capable of becoming a storage node. Storage nodes are able to lease storage and availability to data owners. The network requires a minimum storage capacity of **100 megabytes** for each storage node. We consider the kilobyte to be the minimum unit of storage within the network. Storage space is represented onchain through a token called the OBOL, which is derived from the governance token, IRIS. A single OBOL always represents a single megabyte of storage capacity and acts as the basis for leasing storage space to data owners. With that said, to become a storage node a node must stake enough IRIS to be awarded with 100 OBOL (as determined by the IRIS-OBOL conversion rate in 6.2.1).

To provide storage, nodes stake IRIS tokens which results in newly minted OBOL (minted at a dynamic rate as determined by the IRIS-OBOL conversion rate (see section: 6.2.1)) being awarded to the node who staked the IRIS. The new OBOL is immediately staked in the node's **availability pool**, an on-chain representation of how much storage space the node has available. To prove storage space is available, the node is given a dummy file whose size is comparable to the amount of OBOL they are promising to the network, which they then pin to their embedded IPFS node. For example, staking 10 OBOL in the availability pool means that node is actively pinning 10 OBOL of dummy data. That is, not only do storage nodes promise storage capacity, they also explicitly reserve it.

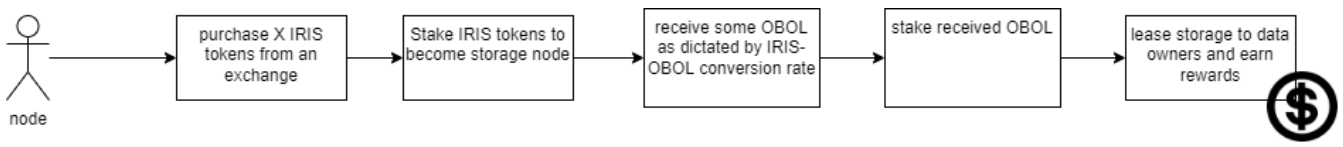

Fig. 8 The process to become a storage provider as a validator

Each storage node has two types of staking pools that it participates in. All of the OBOL that a node owns exists only within these two types of pools. The **availability pool**, specific to an individual node, is the total amount of storage space that the node has available. Secondly, for each data asset **D**, there

exists a **reserved pool for D**. The reserved pool represents the amount of storage space that was reserved for the asset. When a storage node agrees to store data asset, D, with size $\Delta(D)\ mb$, it moves $\Delta(D)$ OBOL from its availability pool and into the reserved pool for D. The reserve pool can allow any number of nodes to stake storage space, where each node stakes the same amount of OBOL, $\Delta(D)$. When any node leases storage space to store a data asset, a new file is generated whose size is equivalent to the OBOL staked in the availability pool, the node pins this file to its IPFS node.

4.2 Storage Price Curve

The cost of renting storage space within the network is a function of how much storage is already leased to data owners and how much storage is requested to be leased in future sessions.

The cost of storage leases, per kilobyte of storage per session, is determined by the below function:

$$p(x) = \frac{\alpha x}{\Delta - x} + p$$

where Δ is the total storage capacity of the network, $x \in [0, \Delta)$, $\alpha > 0$ determines the rate of growth of the price curve, and p is the minimum price of storage.

This function is chosen so that we can satisfy the condition that:

$$p(0) = p,\ p(\Delta) = \infty$$

That is, when there is no leased storage, storage costs reach a minimum, and as storage reaches capacity the cost increases to infinity (when there is no longer any viable storage capacity that can be leased).

The growth constant. α, and the minimum price constant p, are configured to both be 1 at the outset of the network. The values of these parameters can be modified through the governance protocol.

For a visual representation, we choose a = 1, p = 0, and delta = 100:

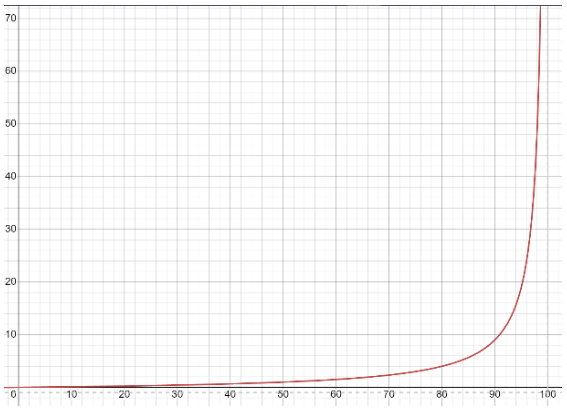

Fig. 9 A storage space rental price curve

4.2.1 Calculating Actual Storage Cost

During any given session, the cost associated renting 1 kb of storage space should be equivalent across all nodes. To accomplish this, we leverage the storage price curve to determine an *actual storage cost.*

Suppose during some session N there is already S kb of storage space leased to nodes. Let $\{b_0,..., b_m\}$ be a collection of storage space that some nodes $\{n_0,..., n_m\}$ have requested to lease in the subsequent

session N+1, respectively. Then the actual storage cost to lease a single kilobyte of storage space in session N+1 is given by:

$$\sum_{x=S}^{S+\sum_{i=0}^{m} b_i} p(x) \bullet \left(\sum_{i=0}^{m} b_i\right)^{-1}$$

The actual storage cost calculation implies that nodes adding smaller amounts of data end up "subsidizing" nodes that are adding large amounts of data in a given session. The system is only fair when all data added during a session is of equal size.

4.3 Mathematical Model of the Storage System

In a decentralized storage system, availability and replication are of the utmost importance. The storage system should be able to meet a minimum level of unavailability while minimizing cost incurred by data owners and minimizing the impacts to the system if storage providers become uncooperative or unresponsive. In the following, we describe an availability-encouraging mechanism for creating ad-hoc availability groups. Additionally, it leads to an increasingly decentralized system as individual storage capacity and availability offered by nodes is increased in a fair way.

4.3.1 Assumption and Notation

We make the following assumptions about the desires of data owners and storage nodes:

- A data owner node wants to **minimize data unavailability** while also **minimizing costs** associated with storing data
- A storage node wants to **maximize the rewards** they receive for storing data

Let the set of storage providers in a session l be defined by:

$$\delta_l = \{\delta_{i,l}\}_{i=0}^{N}$$

where $\delta_{k,l} = (a_{k,l}, \Delta_{k,l}, \gamma_{k,l})$ with:

- $a_{k,l}$ being the availability during the session l ($a_{k,l} \in \{0, 1\}$, with 0 indicating unavailability and 1 indicating availability)
- $\Delta_{k,l}$ being the available storage capacity of the node during the session. This implies that agent $\delta_{k,l}$ has $\Delta_{k,l}$ OBOL locked in the 'available space' pool, $A(k)$.
- $\gamma_{k,l}$ being the total reserved capacity of the node during the session (e.g. occupied from storage deals made during previous sessions). This implies that for node $\delta_{k,l}$, $\gamma_{k,l}$ OBOL is locked in the 'reserved space' pool for any number of data assets (allocated among some pools $\{R_{D_1}, \ldots, R_{D_m}\}$ for some data assets $\{D_1, \ldots, D_m\}$).

Assume that the storage providers are ordered in terms of availability. That is, assume that δ_l has greater unavailability (or is at most as available as) than δ_{l+1}. For the time being we will not consider bandwidth of individual nodes. We also assume that the embedded IPFS network implies that any node is capable of replicating the data available in any other node.

When convenient, we will refer to a storage provider by its index (i.e. k instead of $\delta_{k,l}$).

We also define a function $e_k(l,\ l+m)$ that estimates the probability that node k will remain online m sessions after session l. This probability is self-reported by nodes. Our mechanism incentivizes nodes to report truthful availability estimates.

For each storage node, let $A(k)$ be an amount of OBOL that node k has staked in the **availability pool** A, which is the on-chain representation of the total amount of available storage space. Similarly, for any data asset D, define $R_D(k)$ be the amount of OBOL that node k has staked in the **reserved pool for asset D**. Any node that agrees to store the data asset D submits $\Delta(D)$ OBOL to $R_D(k)$. Note that if nodes k and j both provide availability for D, then $R_D(j) = R_D(k)$, so we can also consider R_D as a membership function into the 'availability group' for asset D.

4.3.2 The Stochastic Replication Game

Our data replication scheme functions similarly to the stochastic replication game as proposed by Rdazca et al [5], though with some significant alterations. In our modified version of the game, we do not necessarily assume that mutual replication is the desire for any node to replicate data. Instead, in place of mutual replication the requested storage node may accept the option to receive rewards from the renting of the storage space.

The **Stochastic Replication Game** (SRG) is defined as an extensive game in which:

- the set of players is equal to the set of agents;
- the set of terminal histories contains list of sets ($\{r_l(i,\ 0\ \vee\ 1,\ j)\}$), i.e., sets of replication proposals (denoted by $r_l(i,\ 1,\ j)$) or withdrawals of previous proposals ($r_l(i,\ 0,\ j)$), possible only when $\exists l'\colon r_{l'}(i,\ 1,\ k)$), made by agents (i) to other agents (j) in round l;in each round, for each agent, the number of active replication proposals does not exceed the agent's storage capacity; all terminal histories end with an empty set ∅ (i.e. nothing is stored at the genesis state);
- the player function $P(h) = \{p_i\}$, i.e., after each history h all players can make proposals;
- Each player minimizes the expected unavailability of its data computed as a product of unavailabilities of players who propose replicating the player's data (and who do not withdraw their proposals in subsequent rounds). We denote by R_j the replication set of j (after a terminal history), i.e., $R_j = \{i\colon \exists l_1\colon r_{l_1}(j,\ 1,\ i)\ \wedge\ !\exists\, l_2 > l_1 : r_{l_2}(j,\ 0,\ 1)\}$. The pay-off is

  $$d(i) = u_i \prod_{j\colon i \in R_j} u_j .$$

4.3.3 T-MAN Protocol

Our protocol makes use of the T-MAN protocol as defined by Jelasity et al [4].

The T-MAN protocol manifests as two threads which each storage provider runs, one active which searches for new agents based on some desired topology, and one passive which accepts incoming messages from agents and gossips with them. We use this to build an overlay topology over the set of all storage providers. We will rely on this topology to build groups of candidate nodes that other storage providers can rely on to maximize data availability.

4.3.4 Matching nodes for maximum availability

By itself the SRG is insufficient to accomplish the goals of the network. Data owners may not want to participate in storage at all, or they might want to have more data in the network than they are personally capable of storing and replicating with other agents in the network. Further, storage nodes will almost always require some type of (guaranteed) reward for storing data over any period of time. The intended experience for a data owner is to be able to specify a minimum data availability threshold, min/max numbers of concurrent replicators that you are willing to lease OBOL from, and a maximum price per session they are willing to pay.

The intention is for highly available nodes to 'help out' weakly available nodes. Nodes with greater storage capacity can provide availability to nodes with limited availability to gain rewards. The greater degree to which they provide availability, the greater the reward.

In the following, when we refer to a 'node' with no other qualifier, we mean storage node.

Let s_m, $s_r > 0$. For any node *i* and data asset *D*, we consider four distinct sets of nodes (or views):

1. The **replica set** (for a data asset D), $replica_D(i)$**:** The set of storage nodes (excluding self) which have previously agreed to replicate some data asset D within the current session.
2. The **taboo set,** $taboo(i)$: A set of storage nodes who have rejected replication proposals.
3. The **metric set,** $metric(i)$: A set of nodes that have been scored (by a scoring function) and meet some minimum score. It has size $s_m < 0$. This set is maintained and updated by the T-MAN protocol.
4. The **random set**, $rand(i)$: A set of randomly chosen storage nodes, with maximum size $s_r > 0$. This set is maintained and updated by the T-MAN protocol.

Each storage node $\delta_k \in \delta$ maintains two lists of replicator candidates refreshed by T-MAN, $metric(i)$ and $rand(i)$ as defined in points 3 and 4 above.

Let $score(i, j)$ be the score function (see: section 4.3.4) which provides a score for node j relative to node i.

Let p be some node that owns the data asset D. Assume that p wants to store their data with some minimum unavailability τ over a number of sessions l_D. Suppose that we begin in session l.

To begin, p chooses at random from the subset G of δ defined by

$$G := \{\delta_k \in \delta : \Delta_k > \Delta(D) \ \wedge \ e_k(l, l + m) \geq \tau \cdot \epsilon\}$$

for some $\epsilon \in (0, 1]$ and $0 < m < l_D$ as chosen by p (ϵ is the maximum expected probability that the chosen node will be online in m sessions). The value ϵ is specified so as to choose a node that will be online long enough to form an availability group.

The chosen node, say node $i \in G$, undergoes the following process (which creates the same conditions as the SRG in section 4.1) in order to form an ad-hoc availability group. There are two major phases. First we use the T-MAN protocol to identify a set of candidate replicators. Secondly, we request that the highest quality nodes replicate the data D in exchange for compensation (on-chain rewards) as a share of the fees paid by the data owner for the storage rental terms.

I. Use T-MAN protocol to generate $metric(k)$:
   1. Choose at random some storage node j

2. Node i fetches $rand(j)$ and node j fetches $rand(i)$
3. Each node merges the received $rand(x)$ set and chooses at most s_r of the most recently added member of the set. That is, both nodes generate a new rand set, say $rand(i)'$ and $rand(j)'$.
4. Node i selects the s_m highest scoring agents from $rand(i) \cup metric(i)$ to construct a new metric pool $metric(i)'$
5. Node i selects the highest scoring node $k \in metric(i)'$. Both i and k exchange metric pools.
6. Finally, node i selects the s_m highest scoring agents of $metric(i)' \cup metric(k)$ to finally form the new $metric(i)$

II. Now that we have generated a new $metric(i)$ set, node i will select candidate replica nodes from the set and submit storage replication requests. Assume that node i has some maximum number of replicas it desires for a data asset, σ.

1. Node i scores each member of its metric pool that is not part of the current replica or in the taboo set. That is, it scores each $k \in metric(i) \setminus (replica(i) \cup taboo(i))$.
2. If i has less than the maximum number of replicas (σ) for D, then select the highest scoring node j^*.
3. Otherwise, compare with the worst node in $replica(i)$, l, where $l = argmax_{m \in replica(i)} u_m$. Here, u_m is the unavailability of node m. Node i queries the first node that has a higher score than l, j^* (i.e. for which $u_{j^*} < u_l$ and $\Delta_j > \Delta(D)$). If there is no such candidate then we do not change the replica.
4. j^* determines if it wants to accept the replication proposal. In general, this will be determined when the cost associated with the storage lease is at some minimum threshold as determined by j^*.
5. if j^* accepts the replication proposal, it moves $\Delta(D)$ OBOL from the availability pool to the reserved pool for asset D. That is, $A(j^*)$ decreases by $\Delta(D)$ and $R_D(j^*)$ increases by $\Delta(D)$. The node j^* is added to $replica(i)$.
6. if j^* rejects the proposal, add j^* to $taboo(i)$ and look for the next most highly rated node

Assuming that the data owner specifies a minimum and maximum number of replicas, we repeat this process sufficiently many times to at least get to the minimum number.

After l_D sessions elapse, the data owner will no longer submit payments for storage leases and storage nodes remove the data from their IPFS nodes.

4.3.4 The Score Function

The score function determines if a node is relatively 'better' than another one. That is, relative to some node i, we can determine a score for all other nodes and rank them. For some node i, a node is 'better' if it is both more highly available and has more available storage capacity, with availability taking precedence over capacity.

4.3.5 Reward Distribution

Reward distribution for data storage occurs in several phases which we detail below. Assume that the data owner has requested storage for some data asset D over some period of sessions, which we refer to as an era. Below, we detail the flow of currency between participants when storing D:

Phase 0: Pre-Session Activities

The data owner locks a minimum amount of currency required for a node to reasonably believe that they will receive payment over the agreed upon era length. This amount is equivalent to locking an amount equivalent to the m value provided when the data owner provided the ϵ value in the previous section.

Phase 1: Session Start

- Verify session storage participants: $P_s := \{p_{s,0}, p_{s,1}, \ldots, p_{s.N_s}\}$

Phase 2: Session End

- Verify session participants that are still online and still have the data stored: $P_s' = \{q_{s,0}, q_{s,1}, \ldots, q_{s.N_s'}\}$ where $N_s \geq N_s'$

Phase 3: Post-Session

- Distribute 33.33% of the total reward amongst the session participants P_s'.
- Lock 66.67% of the rest of the reward and store network participation in a mapping.
- That is, we distribute one third of the rewards now, and the remaining ⅔ at the end of the specified number of sessions.

Phase 4: Post-Era

- Distribute the remaining locked rewards based on total participation of all participants across all sessions.

5. Governance Protocol

Unlike other decentralized storage solutions, Iris is opinionated. Not all types of data should be eligible to be included in the Iris network. More specifically, the nature of decentralization opens the platform to a wide range of abuse, such as allowing the hosting of illegal materials or intentionally monetizing malicious or misleading content. Further, lack of moderation and reputation leads to a higher exposure to scam-like behavior and IP theft. The main goal of our governance protocol is to provide protections to the users of the network and to maintain integrity and trust of the data within. We do this through a de-anonymization resistant relative reputation system and rotating sets of moderator and proxy nodes which are determined through regular on-chain democratic elections.

5.1 Privacy Preserving Feedback Mechanism

In this section we introduce a general privacy-preserving feedback mechanism. This mechanism is used by both the reputation system and the voting system to ensure that both are privacy-preserving and verifiable. Using an approach detailed in [8], the mechanism allows users to anonymously provide feedback to a decentralized bulletin board from which any other user can reconstruct the reputation of

any other node. We use a combination of homomorphic encryption and decentralized tallying to build cryptograms, which are submitted to the bulletin board (BB) using a non-interactive zero-knoweldge proof (NIZK proof). The resulting score can then be reconstructed by analyzing cryptograms in the bulletin board. We also introduce a new token type, the RepCoin, minted on demand and burned after use. Ownership of a repcoin authorizes users to submit feedback. The mechanism allows users to be safe from deanonymization and doesn't exceed computational complexity of $O(n)$. The introduction of the Repcoin allows for easy protections against several attack vectors.

In the following, we consider three distinct entities:

1) The consumer node
2) The data owner (or retailer) node
3) The decentralized bulletin board

5.1.1 Constructing Feedback

Below we outline a general mechanism for anonymously submitting feedback to a 'bulletin board'. The general idea is that after interacting with some agent in the network, Iris awards a newly minted 'repcoin' that provides authorization to submit feedback to a bulletin board. Users who want to submit feedback to a board first generate public and private key, publish their public key to the board, and finally submit their feedback to the bulletin board as a non-interactive zero-knowledge proof.

Assume that some consumer c owns some number of assets minted from an asset class a owned by some node v. When the asset is accessed by c for the first time, Iris mints a new **RepCoin** which is given to c. After feedback is accepted, the coin is burned. We allow feedback to be within the set $\{0, +1\}$, which we refer to as the **feedback range**.

Let p and q be two large primes such that $p \mid q - 1$ and let $\mathbb{G}_q \subset \mathbb{Z}_p^*$ be a cyclic subgroup with generator g. Let $U = \{1, \ldots, m\}$ be the set of consumers in the network. When convenient, we refer to the consumer by only its index (e.g. i).

The goal is to construct a cryptogram that is submitted to a public bulletin board using a NIZK proof. First, we generate new public and private keys.

1. Generate a secret key by selecting some random $x_i \in \mathbb{Z}_q^*$ and compute the public key

$$X_i = g^{x_i}$$

2. The consumer node publishes the public key X_i to the bulletin board.
3. The consumer computes the restructured key R_i (which will be used to construct the cryptogram):

$$R_i = \prod_{j \in n, j < i} X_j \Big/ \prod_{j \in n, j > i} X_j$$

4. Generate the cryptogram for consumer i for providing feedback for a node j:

$$C_{i,j} = g^{x_{i,j} y_{i,j}} g^{v_{i,j}} = R_k^{x_{i,j}} g^{v_{i,j}}$$

where $y_{i,j} = \sum_{k=1}^{i-1} x_{k,j} - \sum_{k=i+1}^{n} x_{k,j} \; \forall i \in [1,\ldots,n],\ j \in [1,\ldots,n]$ and $v_{i,j}$ is the 'vote' of consumer i for consumer j.

5. The consumer node constructs a non-interactive-ZK-proof (NIZK proof) to verify that their vote falls within the feedback range ($\{0, 1, \ldots, n\}$). The construction of the NIZK proof is the same as described in appendix A.
6. The consumer publishes to the bulletin board:
    a. its identity
    b. the encrypted value of the feedback
    c. The identity of the retailer (data owner)
    d. the (1-out-2) NIZK proof*

* The construction and submission of the NIZK proof is described in the appendix.

Below, we visualize this process. First, users fetch the data associated with some data asset owned by a specific node. We assume that each user's holdings satisfy the composable access rules defined by the data owner. When fetched successfully, Iris mints a new Repcoin which allows the owner the ability to submit feedback to any one specific "bulletin board". A bulletin board is a collection of cryptograms that were all submitted using the same type of Repcoin.

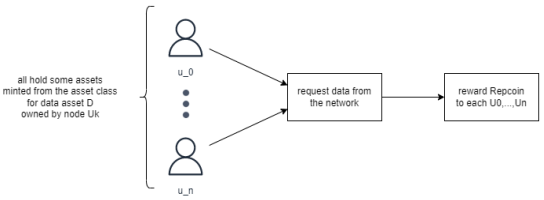

Fig. 9 Phase 1: Interact with the network participant and retrieve a repcoin

Subsequently, once nodes possess a Repcoin, they are then authorized by the blockchain to submit their feedback. Any other user can reconstruct the feedback score regardless of Repcoin holdings.

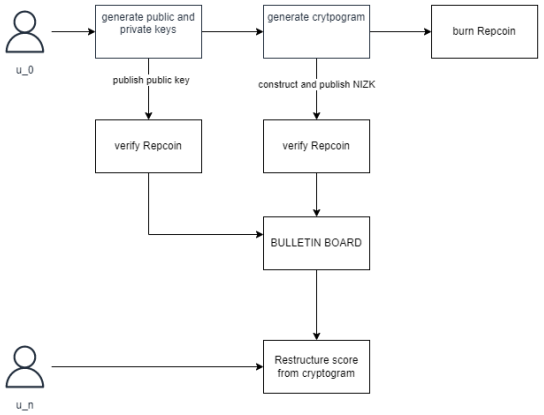

Fig. 10 Phase 2: Use repcoin to submit feedback to a bulletin board

5.1.2 Restructuring of Reputation Score

A node r's *positive score,* θ_r, is given by the product of all cryptograms submitted for that node:

$$\theta_r = \prod_{k=1}^{n} C_{k,r} = \prod_{k=1}^{n} g^{x_{k,j} y_{k,j}} g^{v_{k,j}}$$

In the scenario where there are n choices to vote on, this process is done for each of the bulletin boards to which proofs were submitted.

5.1.3 Attack Vectors

There are five high-level classifications of attack types that agents may try to perform within the reputation system [6]. Below, we detail the types of attacks and the protections and vulnerabilities that Iris is subjected to.

Self Promotion Attacks

A self-promotion attack manifests as an agent in the network (e.g. data owner node) attempting to fraudulently inflate its own reputation. Since we require that the submitter holds a Repcoin, we can reduce our exposure to this type of attack. More specifically, by requiring that a node submit a review with a Repcoin and also refusing to issue Repcoins for self reviews, we mitigate the risk of self-promotion attacks.

A self-promotion attack can also take the form of a node, or a set of colluding nodes (which are distinct entities from the malicious node), creating multiple identities and submitting real feedback after gaining access to their owned data. In the future, we may build an overlay topology of colluders that interact only with each other or only with specific sets of nodes to attempt to identify colluding nodes and to reduce the weight of their feedback or to reject it altogether.

Whitewashing Attacks

Whitewashing attacks occur in two ways. In the first method, a node with a bad reputation generates a new identity and begins to operate with that identity instead, essentially starting with a fresh reputation. One way to account for this is to perform some type of KYC. We will not pursue this method at this time. Another way is to enforce some type of financial barrier or penalty for creating new identities as a data owner. Clearly, one such barrier is that the data owner must pay to create the on-chain asset classes, mint and sell assets, and rent storage space in the network. Additionally, the network can easily identify duplicate datasets (via proxy nodes inspecting the CID of the data) and reject ingestion when it is identified as a duplicate or a potential whitewashing attack.

The second approach happens when a reputation system relies heavily on historical values of reputation. For example, if a node has built a strong positive reputation, they may be able to engage in reputation-degrading activities for short periods of time with minimal impact on the total reputation score. Attentive design of the score restructuring mechanism can circumvent this issue.

Slandering

Slandering attacks occur when one or more nodes attempt to degrade another node's reputation by submitting negative feedback only. Since negative feedback by a single node generally has little impact on the overall reputation of a node, this type of attack is usually performed by a cohort of

colluders. For example, in a method similar to a self-promotion attack, a node may create several new identities which then provide negative feedback for some competitor.

A protection against slandering is provided by the fact that nodes must first purchase access to some data, and access it at least once, before they are allowed to submit feedback for the data owner. That is, any potential slander also means that the attacker is paying the owner to slander it. Again, similar to a potential protection, we can build an overlay topology to attempt to identify nodes colluding in such an attack.

Orchestrated

Orchestrated attacks occur when a coalition of colluding nodes use multiple attack vectors and change their behavior over time. One such example is the oscillating attack, where a set of nodes separates into two teams, one which performs activities that degrade its reputation and the other which builds positive reputation. The positively rated team attempts to inflate the reputation of the negative team, while the negative team reaps the benefits of reputation-degrading behavior until their reputation is too low. When it reaches this point, the two teams switch behavior and the attack continues. This type of attack can become very complex and difficult to detect, especially if there are many different types of attacks undertaken simultaneously with nodes that have little relation to each other. As such, the network has no specific protections against this type of attack currently.

5.2 Reputation

Reputation is the opinions of some community in relation to some specific member of the community. Reputation allows us to build trust between entities. In a decentralized network, a proper reputation system can also serve as a means to filter out bad content [7]. The goal of our reputation system is to build a community-generated reputation that enables a measurement of trust that some other member will reliably deliver or perform some service where feedback is anonymous and is safe from deanonymization. Additionally, the reputation of a node may lead to potential reward of consequence as a result of changes in reputation (see: moderator nodes).

Submission of a reputation score equates to the one-dimensional case of the scheme laid out above. That is, it is the case where each node providing feedback generates on public/private keypair and uses a single coin to submit a single NIZK to submit feedback within the set $\{0, \ +1\}$.

5.2.1 Restructuring of Reputation Score for a Data Asset

A simple method of calculating the node's overall reputation, taking into account negative votes, can be given by:

$$RE_E = \frac{P_E - N_E}{n+2}$$

where P_E is the number of nodes that provided positive feedback (+1), N_E is the number of nodes that provided negative feedback, and n is the total number of nodes that provided feedback.

5.4 Proxy Nodes

Proxy nodes form the first layer of security for the Iris network as well as allow data owners and consumers to ingest and eject data. They operate in two contexts, as a gateway and as a proxy. As a gateway to data ingestion, they allow data owners to add data to the Iris network and create an asset class while also enforcing a minimal set of data verification rules prior to allowing data to be ingested,

such as ensuring the data does not contain malicious content such as a virus. Conversely, as a proxy, proxy nodes form an ‘invisible’ proxy layer that re-encrypts data for authorized callers when requested. That is, they are the proxy layer in the proxy re-encryption mechanism. After re-encryption, the data is streamed to the authorized caller who requested the re-encryption.

Proxy nodes must meet a minimum hardware requirement, specifically as determined by the ability to execute offchain services within a TEE. As proxy nodes are the layer between IPFS and the external world, they define the bandwidth of the network, and so we require a minimum bandwidth to become a proxy node, which should be at least as much as the minimum allowable bandwidth of the network as a whole. To become a proxy, a node stakes a minimum number of IRIS tokens. When staked, a **proxy routing service** routes requests to ingest data, to re-encrypt data, and to eject data to proxy nodes. To be eligible for rewards, a proxy must be online for a predetermined minimum number of sessions. The proxy routing layer collects fees from callers and subsequently distributes fees to proxy nodes as required, based on uptime and bandwidth.

5.5 Moderation

Within data spaces, moderator nodes, as selected by the **moderator routing service**, autonomously perform data verification tasks offchain within a TEE. Moderator nodes operate autonomously and they are bijective with the set of proxy nodes. That is, any proxy can also be a moderator. Moderators run offchain services that execute and report results based on the moderation policy defined for a data space. That is, moderator nodes are responsible for enforcing the moderation policy within the data spaces. When authorized nodes request data inclusion within a space, moderator nodes are the final barrier to the space. A moderator is responsible for executing any number of machine learning models offchain within a TEE to verify certain properties about data as specified by the creator of the space. For example, a data space moderation policy may stipulate that certain file extensions are not allowed in the space or that certain data properties are not allowed as a result of the outcome of machine learning models, such as “no images of horses”.

6. Token Economics

In this section we discuss the tokens within the Iris blockchain, their functionality, uses, and inflation models.

6.1. Tokens

There are two tokens involved in Iris, IRIS and OBOL.

6.1.1 IRIS

The IRIS token is the governance token in the Iris blockchain. This token powers the proof of stake consensus within the Iris blockchain, the moderation and governance framework, and acts as an entry point for validators to become storage providers through the IRIS-OBOL conversion rate(6.2.1). The IRIS token is inflationary, with new tokens minted as rewards for block validators.

6.1.2 OBOL

The OBOL exists uniquely within the context of Iris and enables the network to treat storage capacity as an on-chain asset. Similar to the IRIS token, there is no cap to the amount of tokens that can be minted, though it is dependent on the amount of IRIS currently available. In contrast to the IRIS

token, the OBOL is minted in real time as needed rather than on a scheduled basis. As mentioned in section 2.5, the ownership of an OBOL requires that the owner provides one kilobyte of storage to the network.

OBOLs are minted when a node stakes at least the minimum quantity of IRIS required to mint 100,000 OBOL (1 OBOL ~ 1kb of storage and the minimum storage required per node is 100 gb), as determined by the IO burn function (see: section 4.2). When burned, the minted OBOL is automatically staked. See section 2.5 for more information on usage of the OBOL and interactions it facilitates between storage providers and data owners.

When a storage provider desires to reclaim or reduce their storage capacity promised to the network, a storage node can unstake their IRIS, which results in the burning of their OBOL.

6.2 Token Conversions

6.2.1 IRIS-OBOL Conversion Rate

The IRIS-OBOL conversion rate is a dynamic rate at which IRIS can be "converted" into OBOL. We cap the total amount of OBOL that can be minted per any block by a value $B_{max} > 0$. At network genesis, we want 1 IRIS token to convert to the minimum number of OBOL that a storage node must hold, 1000 OBOL (which represents 1 megabyte of storage).

The conversion rate incentivizes storage nodes to increase their storage capacity when we fall under our 'ideal storage rate'. Suppose s_{ideal} is the ideal storage rate, expressed as a ratio of available storage space to total storage capacity. That is, $s_{ideal} = A_{ideal}(\Delta)/\Delta$. When we exceed the ideal storage space ratio we want to incentivize an increase in available storage (or in other words: too much space has been reserved in the network, which will cause the price to quickly increase). If we fall below that rate, we want to reduce the incentive to provide storage. That is, we want the following conditions to be true:

- if $A/\Delta = 0 \Rightarrow \beta_{IRIS->OBOL} = \beta_{max}$ for some $\beta_{max} \geq 1$
  - If there is no available storage, we want to convert IRIS to OBOL at an inflated rate, up to a maximum of β_{max}
- if $A/\Delta = s_{ideal} \Rightarrow \beta_{IRIS->OBOL} = 1$
  - If the network has reached the ideal storage space ratio, we neither incentivize converting IRIS to OBOL or vice versa.
- if $A/\Delta = 1 \Rightarrow \beta_{IRIS->OBOL} = 0$
  - If no storage space has been reserved, we do not incentivize the conversion at all

We use the following function to set the conversion rate:

$$\beta(t) = \frac{c-\beta_{min}}{s_{ideal}} t + \beta_{min}$$

Where c is the ratio of IRIS to OBOL when we reach ideal storage capacity. Setting $c = 1000$, $\beta_{min} = 1$ and $s_{ideal} = 0.70$, we arrive at the following:

$$\beta(t) = \frac{999}{0.70} \cdot t + 1,\ 0 \leq t \leq 1$$

Initial network conditions may make it very difficult to get started using this function. Thus, the ratio described above does not come into effect until the network has a reasonable amount of time to reach sufficient storage capacity so as to reach the ideal ratio. The following table depicts the block number and the IRIS to OBOL ratio that is respected during each block interval. After 600,000 blocks, we default to the ratio described above.

Block number	IRIS-OBOL ratio
0	1:1,000,000
100,000	1:500,000
200,00	1:100,000
300,000	1:50,000
400,000	1:10,000
500,000	1:1,000
600,000	$\beta_{IRIS->OBOL}$

6.2.2 OBOL-IRIS conversion rate

This function enables storage nodes to reduce their storage provided to the network. In essence, to "cash out". This function is simply the inverse of the IRIS-OBOL burn function. That is, the rate is given by $\beta_{OBOL->IRIS} = \beta_{max} - \beta_{IRIS->OBOL}$.

Similarly to the artificially applied burn rate for the first 600,000 blocks for IRIS-OBOL, we follow a similar approach. For the first 600,000 blocks, nodes cannot convert OBOL back to IRIS. After this period of time, we use the default burn function.

Block number	OBL-IRIS ratio
0	0
600,000	$\beta_{OBOL -> IRIS}$

6.3 Inflation Model

6.3.1 Inflation and Interest

Iris enables two avenues which lead to staking tokens, both validators and nominators stake tokens and receive rewards via the consensus protocol. Additionally, storage nodes stake IRIS tokens in order to receive OBOL and provide storage to the network.

For any block b, let $\theta_b^{\ v} = \{\theta_{ib}^{\ v}\}_{i=0}^{n}$ be the amount of tokens staked by validators and nominators, $\theta_b^{\ p} = \{\theta_{ib}^{\ p}\}_{i=0}^{q}$ the amount staked by proxy nodes, and $\theta_b^{\ s} = \{\theta_{ib}^{\ s}\}_{i=0}^{m}$ be the amount of tokens staked by storage nodes, and $|T_b|$ the total token supply. Then the real token staking rate is given by $(\sum_{i=0}^{n} \theta_{ib}^{\ v} + \sum_{i=0}^{m} \theta_{ib}^{\ s} + \sum_{i=0}^{q} \theta_{ib}^{\ p}) / |T_b|$. We can break this into the validator staking rate and the storage node staking rate, given by $x^v = \sum_{i=0}^{n} \theta_{ib}^{\ v}/|T_b|$ and $x^s = \sum_{i=0}^{m} \theta_{ib}^{\ s}/|T_b|$, respectively. Since the rewards given to storage nodes comes directly from data owners, those rewards do not act as a source of inflation and the tokens staked by these nodes, though they are not included in the free balance of IRIS, should not impact the overall inflation rate. In our model, they are considered no different than balances held by any node within their wallet.

The formulation of the interest and inflation model follows the same model proposed for use in polkadot [2]. We set the initial (also the minimum) inflation rate as 0.025, the ideal staking rate (for validators) as 0.30, the ideal inflation rate as 0.2, and the decay rate as 0.05.

In generic terms, we have:

$$I_{NPoS}(x) = I_0 + (I_{NPoS}(x_{ideal}) - I_0)x/0.3 \ for\ 0 < x \leq x_{ideal}$$

$$I_{NPoS}(x) = I_0 + (I_{NPoS}(x_{ideal}) - I_0))2^{\frac{x_{ideal}-x}{d}} \ for\ x_{ideal} < x \leq 1$$

The function determining inflation, as a function of the validator staking rate with our given values, is then given by:

$$I_{NPoS}(x) = 0.025 + 0.116 \cdot x\ ;\ 0 < x \leq 0.3$$

$$I_{NPoS}(x) = 0.025 + 0.116 \cdot 2^{\frac{0.3-x}{0.05}}\ ;\ 0.3 < x \leq 1$$

6.3.2 Validator and Nominator Reward Distribution

Similar to the inflation model, we follow the same approach used by Polkadot. Specifically, for validator actions that are deemed 'payable', we follow the point system:

- 20 points for each validity statement,
- 20 points for each (non-uncle) block produced,
- 2 points (to the block producer) for each reference to a previously unreferenced uncle, and
- 1 point to the producer of each referenced uncle.

With the total for a validator and her nominators given by $\frac{c^e_{\ v}}{c^e} \cdot P^e_{\ NPoS}$, where $c^e_{\ v}$ is the total number of points earned by a validator in era e, c^e the total points of all validators in era e, and $P^e_{\ NPoS}$ is the target payout to all validators (and their nominators). We defer to [2] for a breakdown how distribution works amongst validators and nominators.

7. Use Cases

Below we detail and analyze several use cases that Iris enables. In the first example, we discuss how Iris can impact NFTs. Secondly, we look more broadly at which types of centralized applications Iris can decentralize, and lastly, we consider B2B and middleware cases.

7.1. NFT2.0: Decentralized data marketplaces

Iris can easily be used to build an NFT marketplace that cryptographically ties together data ownership and data availability. For example, a dapp creator may create their own private data space within Iris and allow any user of their dapp to add data to their data space. Subsequently, they can extend the functionality and behavior of data access using smart contracts. For example, many NFT-based gaming applications could take advantage of Iris as a storage layer.

7.2 Decentralized Data Applications

Iris can act as the backbone for several decentralized applications that rely on the storage and sales of data, such as Spotify, Youtube, Netflix, Steam, and other such services. Though Iris does not explicitly provide the infrastructure to produce high-quality real-time streams of data, third party developers may build such functionality independently and leverage Iris as the storage layer. Futuristically, Iris may be connected to many other blockchains via a relay chain, allowing for the transfer of data access (i.e. assets) across multiple chains who may have such functionality available. Additionally, less real-time intensive applications, such as a decentralized version of DropBox or Drive-like applications, are readily and easily achievable.

7.3 B2B and Middleware Solutions

Iris can be leveraged in the B2B sector as well. When deployed as a private network and with an API layer built to interface between businesses and the blockchain, Iris can provide a highly-available, fault tolerant data store that allows businesses to transfer, sell, and monetize their data assets. Along similar lines, Iris can be integrated with already existing centralized solutions to take advantage of decentralized storage and to allow their user base to monetize data produced within their applications.

8. References

[1] https://docs.ipfs.io/

[2] A. Cevallos. and C. Gehrlein, 2021 Token Economics. https://research.web3.foundation/en/latest/polkadot/overview/2-token-economics.html

[3] Häfner, Samuel, Blockchain Platform Design (April 14, 2022). Available at SSRN: https://ssrn.com/abstract=3954773 or http://dx.doi.org/10.2139/ssrn.3954773

[4] M. Jelasity and O. Babaoglu. 2006. T-Man: Gossip-based overlay topology management, In ESOA, Proc. Lecture Notes in Artificial Intelligence 3910 (2006).

[5] Rzadca, Datta, Kreitz, Buchegger, 2015. Game-Theoretic Mechanisms to Increase Data Availability in Decentralized Storage Systems, ACM Transactions on Autonomous and Adaptive Systems, Vol. 1, No. 1, Article 1, Publication date: January 2015.

[6] Hoffman, Zage, Nita-Rotaru, 2007. A Survey of Attack and Defense Techniques for Reputation Systems. ACM Computing Surveys, Vol. V, No. N, Month 2007, 1-34

[7] K. Walsh and E.G. Sirer, 2006. Experience with an Object Reputation System for Peer-to-Peer Filesharing, NSDI ’06: 3rd Symposium on Networked Systems Design & Implementation

[8] Azad, Muhammad & Bag, Samiran & Hao, Feng. (2018). PrivBox: Verifiable decentralized reputation system for the on-line marketplaces. Future Generation Computer Systems. 89. 10.1016/j.future.2018.05.069.

[9] ZKProof. ZKProof Community Reference. Version 0.2. Ed. by D. Benarroch, L. T. A. N. Brandão, E. Tromer. Pub. by zkproof.org. Dec. 2019. Updated versions at https://zkproof.org

[10] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. 2006. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.* 9, 1 (February 2006), 1–30. https://doi.org/10.1145/1127345.1127346

[11] Cramer, Ronald & Shoup, Victor. (2002). Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. LNCS. 2332. 10.1007/3-540-46035-7_4.

[12] Yevgeniy Dodis and Anca Ivan. Proxy cryptography revisited. In Proceedings of the Tenth Network and Distributed System Security Symposium, February 2003

Appendix A.

NIZK Proof Construction