Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

на тему: **«Легкі децентралізовані системи зберігання інформації»**

«Light decentralized information storage systems»

Виконав: студент денної форми навчання
спеціальності 113 Прикладна математика
Белоченко Олексій Євгенович

Керівник: канд. тех. наук, доц. Мазурок І. Є.

Рецензент: канд. фіз.-мат. наук, доц. Вербіцький В.В.

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від «_____» _____ р.
Завідувач кафедри
_____ _____

Захищено на засіданні ЕК № _____
Протокол № ____ від «____» ____ р.
Оцінка _____ / _____ / _____
Голова ЕК
_____ _____

Одеса "— 2023 р.

CONTENTS

INTRODUCTION

This research aims to devise a framework for a human-centric, ethical, distributed data storage method that facilitates interaction with web applications. This pursuit involved a deep dive into three distributed storage technologies and strategies, each employing unique storage mechanisms like distribution across high uptime nodes, hybrid storage, and decentralized peer-to-peer networks.

As data volume escalates, so does the demand for secure, efficient storage and management solutions. Still, the sheer volume of data has complicated its organization and retrieval. Recent research indicates that 70% of digital consumers spend nearly 3 hours daily searching for information across various platforms, leading to productivity losses. The centralized nature of these systems also raises data privacy and ownership issues.

Governments in Europe and North America have begun regulating personal data usage and storage. Regulatory measures like the GDPR, CCPA, and PIPEDA aim to protect individual privacy, emphasizing user consent, data security, and transparency. However, the complexity and diversity of data licensing across platforms pose compliance challenges, and these regulations often need to be revised as they fail to provide efficient tools to uphold system dependability.

Tech companies like Meta, Google, and Microsoft have policies guiding personal data aggregation and reuse. While these practices reportedly improve user experiences and technological innovation, concerns persist over individual privacy and data-sharing risks. Moreover, the perception of data control is limited, with 81% of individuals feeling they need more control over data collected by companies.

The accelerated growth of AI models requires collecting and aggregating diverse open data from various platforms, creating a potential for data breaches or covert information gathering. Licensing restrictions further complicate this process.

Concerns exist in the open-source community about using open-source Github repositories for commercial AI model training. Compliance with open-source software licenses, mainly when LLMs generate potentially copyrighted

output, is essential to avoid disputes during product commercialization.

Activists like Professor Tim Davis of Texas A&M University have criticized practices like those of Microsoftś Copilot for using copyrighted code without citation. This sentiment is echoed by developers who have found their open-source code distributed under various licenses within Codex model results.

This research compared three different technological approaches for creating decentralized applications and proposed a new application architecture based on the analysis. The primary goal of this research is to consider a potential system architecture that would allow Internet users to protect, manage, and efficiently redistribute data from one service or application to another while adhering to ethical design principles. The goal is to systematize approaches for a human-centered storage system to store personal and service data associated with the user. Propose a future architecture in which applications can exist separately from data management in a database and return responsibility and control to the user. Demonstrate that such applications can work just as well as existing applications.

The principal issues that can jeopardize current distributed systems can be outlined as follows:

- ◇ **Efficient data storage and retrieval**: The system must be capable of storing diverse data types and performing searches, comparisons, data relevance assessments, and similarity analysis in distributed containers scattered over the network. Rapid data retrieval and delivery to the end client are critical.
- ◇ **Seamless web application interaction**: The system should permit web applications to interact with containers to securely and effortlessly retrieve or save data.
- ◇ **Data usage licenses**: The system should have automatic data management tools such as data usage licenses. These permits, which accompany master data, specify the data use regulations. End systems or applications that do not follow or meet the user license criteria will be denied data access, ensuring data privacy and authentic data ownership. Under applicable data protection legislation, allow users to control and give or revoke consent for data usage and sharing. Maintain an extensive data

access and usage record to give people transparent insights into how their data is used.

Most existing systems today need to meet Ethical Design standards, and nearly all of the existing platforms on the market provide the functionality specified above.

Research Objectives and Questions

Research Objectives

- Select three key systems or specifications in distributed (or separate) data storage and perform an in-depth technical analysis of existing systems based on the open-source community.
- Identify essential components and methodologies utilized in system design that benefit the effectiveness of existing solutions.
- Proposed an alternative system based on the open-source distributed data storage system solutions that securely and privately stores and manages various types of data securely and privately.
- Try to approach the standards of Ethical Design in the design of the new system.

Research Questions

- How should the application architecture be designed to be optimal and secure, and effective at accomplishing the tasks at hand?
- What are the key considerations for developing web applications that can easily interact with the distributed data storage system to retrieve or save data?
- What are the proposed solution's performance, security, and privacy characteristics?

Research Methodology

This study compares three technologies for decentralized systems: the InterPlanetary File System (IPFS), the Social Linked Data project (Solid), and the Decentralized Web Node (DWN) by the Identity Foundation. The comparison bases on six key criteria: Design Philosophy, Infrastructure, Authentication and Authorization, Data Control, Decentralization, and Technical Components.

- **Design Philosophy**: The fundamental principles and objectives that guided the development of each technology are explored. This examination provides insights into the intended use and suitability of each technology for specific scenarios.
- **Infrastructure**: This criterion involves assessing the structural design of each technology, whether as a protocol, platform, or node-based system. Such analysis can shed light on the technology's deployment strategies, scalability, and maintenance needs.
- **Authentication and Authorization**: An investigation into how each technology handles access control and user permissions occurs, which directly impacts the security and privacy capabilities of the technology.
- **Data Control**: An evaluation of how each technology allows users or entities to manage their own data is conducted. This aspect is crucial for applications that prioritize data privacy and user data ownership.
- **Decentralization**: Given the focus on decentralized web technologies, an examination of how each technology achieves decentralization is conducted, assessing its robustness, resilience, and potential for peer-to-peer interaction.
- **Technical Components**: The core technical components of each technology are analyzed. This analysis provides a sense of the technology's complexity, flexibility, and compatibility with other systems or technologies.
- **Comparison Methodology**: The method employed in this study is a qualitative comparison based on the descriptive information available for each technology. For each criterion, a contrast and summary of the characteristics of the technologies are provided. This method does not involve numeric scoring or ranking; instead, it provides a narrative

comparison to understand the differences and similarities among the technologies.

- **Reliability Assessment**: To ensure the reliability of the data, authoritative sources for each technology have been used: official documentation for IPFS and DWN, and a reputable encyclopedia for Solid. These sources are typically reliable, being directly related to the organizations developing the technologies or being subject to peer review and fact-checking. However, given the rapidly evolving nature of technology, it is always advisable to consult multiple sources and stay updated with the latest developments.
- **Literature review**: Examine the state-of-the-art open-source solution in distributed data storage systems and web-based data management, as well as any related research on security, privacy, and data ownership.
- **Design and implementation**: Propose a prototype of an alternative human-centered system and web applications that can interact with it. Some submitted implementations will be based on a suitable programming language and technology stack.
- **Conclusion and recommendations**: Conclude the research and make recommendations for further work in the field.

Significance of the study

This study explores novel strategies in architecture and design for future software applications, emphasizing user-centric principles. The focus on data storage provides a fresh perspective on information management and suggests potential advancements in the field.

The outlined architectures form the groundwork for future applications, which are envisioned to operate independently of data control. A demonstrative application was developed based on these architectures, offering a tangible example of this new paradigm's opportunities.

The implications of this study go beyond application development, impacting data privacy and user empowerment. Detaching applications from direct user data control offers potential benefits in terms of improved privacy, security, and control. This innovative approach fosters an environment for applications

prioritizing user needs, promoting a more transparent and ethically-conscious digital space.

The insights from this research could guide future architectural and software design advancements, promoting innovation and adopting user-centric principles. As technology evolves, it's essential to reevaluate traditional data storage and application architectures to better protect user data and empower users in the digital age.

In summary, the significance of this study lies in its exploration of new architectural strategies. It provides a fresh outlook on data storage and has the potential to revolutionize application development. The research enhances user-centric principles, data protection, and user empowerment within the digital ecosystem, substantiated by a practical implementation and demonstrated through an application prototype.

CHAPTER 1

EXISTING PRODUCT ANALYSIS

1.1 Key representatives of the ecosystem

- **IPFS (InterPlanetary File System)** is an open-source distributed file system that uses peer-to-peer hypermedia protocol to store and share files. It provides a content-addressed system, meaning that files are addressed by their content hash rather than by their physical location, despite the focus on storing only files and not intended to store other structures or perform the role of a database, on top of IPFS surrender projects to manage personal records things like Filecoin and CopperDB. Filecoin is a peer-to-peer network that stores files online, with built-in economic incentives to ensure files are stored reliably over time. Filecoin and IPFS are complementary protocols for storing and sharing data on the decentralized web. Protocol Labs build both. The InterPlanetary File System (IPFS) is a peer-to-peer (p2p) storage network that seeks to connect all computing devices with the same system of files. Since Filecoin is part of IPFS separately will not be considered in this paper, as well as CopperDB, because it is a primitive project.
- **SOLID Project (Social Linked Data)** is an open-source initiative and specifications spearheaded by Sir Tim Berners-Lee, intending to establish a decentralized and user-centric web ecosystem. It introduces the concept of Personal Online Data Stores (PODs), which empower individuals to store and manage their data securely and privately. By adhering to the SOLID specifications, users gain greater control and ownership over their data, enabling them to decide how their information is shared and accessed by applications and services. The SOLID Project aims to reshape the web landscape, fostering a more transparent, user-driven, and privacy-aware online environment.
- **Decentralized Web Node** is a critical component of the decentralized web infrastructure that provides data storage and message relay capabilities. It is a distributed storage layer where individuals and organizations

can securely store and manage their data while maintaining control and ownership. The node facilitates access to public or private data associated with a specific Decentralized Identifier (DID). Additionally, it acts as a message relay mechanism, enabling secure peer-to-peer communication and data exchange within the decentralized web ecosystem. By leveraging decentralized technologies, the Decentralized Web Node promotes data sovereignty, privacy, and resilience, offering an alternative to centralized data storage systems.

1.2 IPFS (InterPlanetary File System)

1.2.1 Overview

IPFS (InterPlanetary File System) is a distributed protocol and network for storing and sharing files in a peer-to-peer manner. It was created in 2014 by Juan Benet and is open source software.[16]

IPFS aims to address some of the limitations of traditional HTTP-based file transfer protocols by creating a content-addressed system that is both decentralized and scalable. Rather than identifying files by their location on a specific server or network, IPFS identifies files by their content, using cryptographic hashes to generate unique identifiers.

When a file is added to IPFS, it is broken up into smaller pieces and distributed across the network in a peer-to-peer fashion. This means that there is no central point of control, and files can be accessed from any node on the network that has a copy of the file.

IPFS also includes a built-in version control system, which allows users to easily track changes to files and revert to previous versions if needed. This can be useful for collaborative projects or for ensuring the integrity of important files over time.

One of the key benefits of IPFS is its potential to reduce the reliance on centralized server infrastructure, making it more resilient to censorship, outages, and other forms of disruption. It is also designed to be more efficient than

traditional file transfer protocols, as it can cache files locally and retrieve them from nearby nodes, reducing the need for long-distance transfers.

IPFS is actively being developed and is supported by a growing community of developers and organizations. It is currently used for a wide range of applications, including decentralized web applications, scientific data sharing, and blockchain-based file storage.

1.2.2 System key components

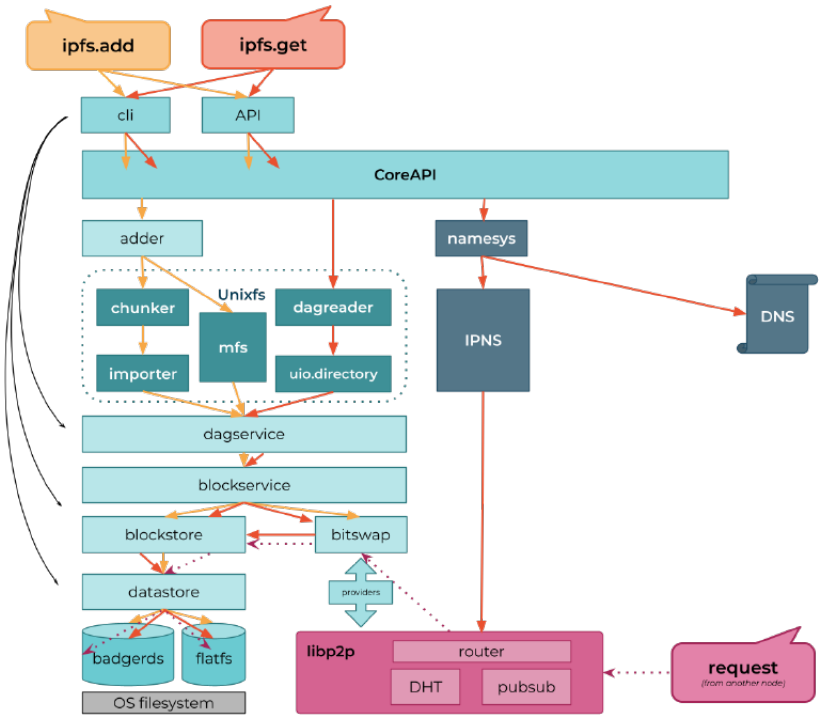

Figure 1.1. IPFS subsystems diagram[15]

IPFS is a distributed protocol and network designed for storing and sharing files in a peer-to-peer manner. It is implemented using a variety of technologies and approaches, including the following:

- **Content-addressing**: Rather than identifying files by their location on a specific server or network, IPFS identifies files by their content using cryptographic hashes to generate unique identifiers. This ensures that files can be easily located and retrieved from any node on the network that has a copy of the file.
- **Merkle DAG (Directed Acyclic Graph)**: IPFS uses a Merkle DAG to represent files as a directed acyclic graph of nodes, where each node

represents a unique piece of content. This allows IPFS to store and retrieve files in a more efficient manner, as nodes can be cached locally and retrieved from nearby nodes.

- **BitTorrent-based file transfer**: IPFS uses a BitTorrent-inspired approach to file transfer, where files are broken up into smaller pieces and distributed across the network in a peer-to-peer fashion. This ensures that there is no central point of control and that files can be accessed from any node on the network that has a copy of the file.
- **Git-inspired version control**: IPFS includes a built-in version control system, which allows users to easily track changes to files and revert to previous versions if needed. This is inspired by the Git version control system, and can be useful for collaborative projects or for ensuring the integrity of important files over time.
- **Libp2p networking stack**: IPFS uses the libp2p networking stack, which provides a modular, flexible framework for building decentralized peer-to-peer applications. Libp2p supports a wide range of networking protocols and transports, and can be used to build decentralized applications that run on a variety of platforms.

1.2.3 Network

The Content-Addressable Network (CAN) in IPFS is a distributed hash table (DHT) that allows nodes to store and retrieve content based on its unique content identifier, or CID. It works next way:

- Content is added to the IPFS network by creating a CID for the content. The CID is created using a hash function (usually SHA-2 or SHA-3) that generates a unique identifier for the content based on its contents.
- The content is then broken up into small pieces called blocks, which are typically 64 KB in size. Each block is given its own CID based on the content of the block.
- The blocks are then distributed across the IPFS network using a distributed hash table (DHT). Each node in the network maintains a portion of the DHT, which maps content identifiers to the nodes that store the content.
- When a node wants to retrieve content, it first looks up the content's CID

in the DHT to find the nodes that store the content.

- The node then retrieves the content by requesting the appropriate blocks from the nodes that store them. The blocks can be retrieved in parallel from multiple nodes to speed up the process.
- Once the node has retrieved all the blocks, it can reconstruct the original content using the CIDs of the blocks.

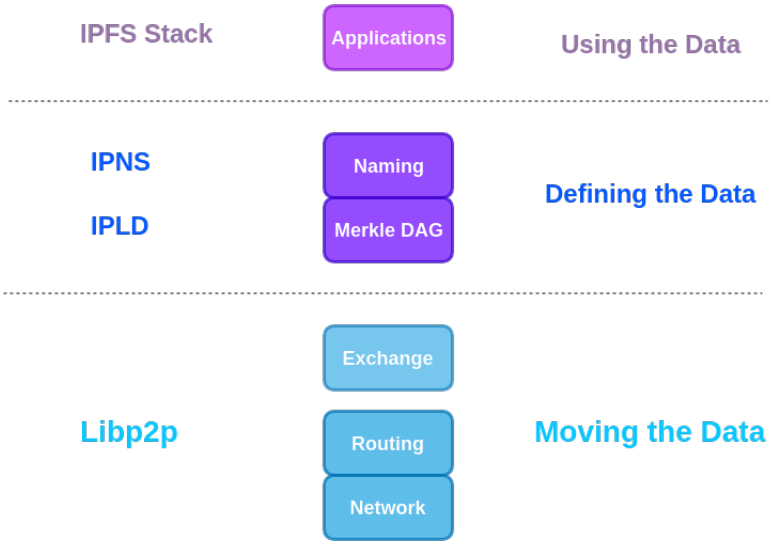

Figure 1.2. Libp2p moving data layer provides routing across all peers[15]

IPFS employs several key mechanisms to enhance the efficiency, resilience, and availability of content within its network.

Firstly, content-based routing is implemented, where requests for content are routed based on the content's CID (Content Identifier) rather than relying on the IP address of the node storing the content. This approach enables more efficient and robust routing of content requests.

Secondly, IPFS utilizes a peer-to-peer networking protocol known as libp2p (Figure 1.2), enabling direct communication between nodes. This peer-to-peer networking facilitates efficient and resilient content transfer across the network.

To ensure content availability, IPFS introduces the concept of pinning. Nodes can "pin" content, which means they preserve and make it accessible on the network even if the original node that added the content becomes offline. Pinning enhances the resilience and availability of content within the IPFS network.

Lastly, IPFS employs the BitSwap (Diagram 1.3) protocol to enable nodes

to exchange content blocks with each other. BitSwap ensures fair and efficient content trading among nodes without relying on a centralized server.

It's also worth considering IPFS DHT. IPFS uses a modified version of the Kademlia Distributed Hash Table (DHT) as its underlying routing protocol. The Kademlia DHT is a peer-to-peer network algorithm that allows nodes in the network to find and communicate with each other efficiently.

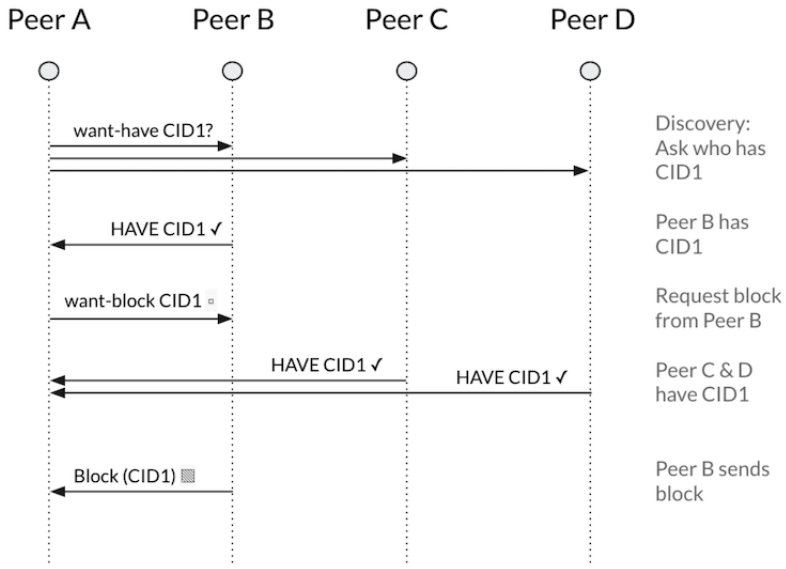

Figure 1.3. Sequence diagram of how Bitswap protocol works[17]

Kademlia Routing

In the IPFS DHT, nodes store and share information about the content that they are hosting, including the content's CID and the nodes that are currently hosting it. When a node wants to retrieve content from the network, it queries the DHT to find the nodes that are hosting the content. The IPFS DHT uses a Kademlia-like algorithm to route queries to the correct nodes in the network, based on the CID of the content being requested.

The use of a DHT allows IPFS to achieve efficient, decentralized content discovery and distribution, without relying on a centralized directory or naming system. The Kademlia-based routing algorithm also provides robustness and fault-tolerance, ensuring that the network can continue to function even if some nodes are offline or disconnected.

The modification to Kademlia that IPFS uses is called the "recursive

Kademlia DHT". The recursive Kademlia DHT is designed to address some of the limitations of the standard Kademlia algorithm, particularly with regards to content routing and peer discovery in large-scale distributed networks.

The recursive Kademlia DHT used in IPFS, on the other hand, allows nodes to store and share information about content as well as nodes. Each node in the network maintains a distributed hash table (DHT) that maps content identifiers (CIDs) to the nodes that are currently hosting the content. When a node wants to retrieve content from the network, it queries the DHT to find the nodes that are hosting the content.

The recursive Kademlia DHT also uses a more efficient algorithm for routing queries than the standard Kademlia algorithm. In the recursive Kademlia DHT, nodes forward queries to the "closest" nodes in the network, based on the distance between the CID of the queried content and the CIDs of the nodes in the network. This allows the network to efficiently route queries to the nodes that are most likely to have the requested content, while minimizing the number of hops required to retrieve the content.

1.2.4 Data Storage

Chuncking

IPFS uses chunking to break files into smaller data units, typically 64KB. When a file is added to IPFS, it undergoes a segmentation process into smaller chunks called "blocks." To ensure uniqueness and easy identification, these blocks are created using chunking. Chunking involves dividing the file into fixed-size chunks based on its content, aiming to assign a distinct Content Identifier (CID) to each block.

The chunking process utilizes Rabin fingerprinting, which breaks down the file into chunks by sliding a window of bytes. The window's size is determined using a next formula:

$$\text{Window Size} = \frac{\text{File Size}}{\text{Desired Chunk Size}} \times \text{Constant} \tag{1.1}$$

that considers the file's size and the desired chunk size. Rabin fingerprinting

generates a unique fingerprint for each chunk, which is the basis for creating the individual CIDs for the blocks.

Once the blocks have been generated and assigned unique CIDs, they are added to the IPFS network. This is accomplished by utilizing the Content-Addressable Network (CAN), which enables the distribution of the blocks across the network. In this way, IPFS ensures the efficient storage and retrieval of files by breaking them into smaller, identifiable chunks and distributing them through the network using unique CIDs.

Data Integrity

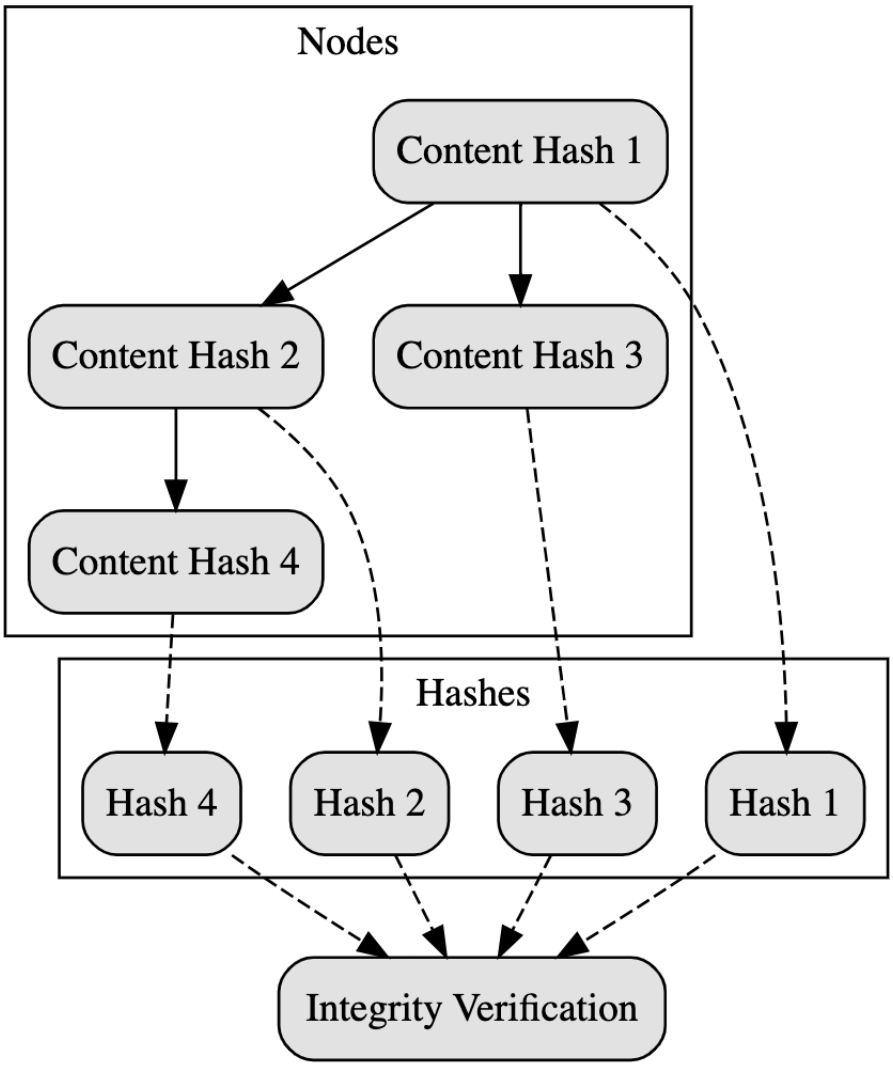

Figure 1.4. Simplified representation of the Merkle DAG in IPFS

The integrity of the data is achieved by using the previously mentioned cryptographic hashing, data chunking, and CID assignments mechanisms.

The Merkle DAG structure is formed by linking the nodes together (Illustrated on figure 1.4), each containing the hash of its content and the hashes of its child nodes. This creates a hierarchical structure where changes in any individual data chunk will propagate up the graph, affecting the hashes of the parent nodes.

IPFS can verify the integrity of the entire data set by traversing the Merkle DAG and comparing the hashes. Any alteration or tampering of the data will result in a different hash, which will be detected during verification.

The use of Merkle DAG in IPFS provides a robust and efficient mechanism for ensuring data integrity. It allows users to verify the authenticity and integrity of data without relying on a centralized authority. Additionally, it enables efficient content addressing and retrieval, as each piece of data can be uniquely identified by its CID, making it highly suitable for decentralized and distributed systems.

1.2.5 Data Transfer

IPFS utilizes a decentralized file transfer technique inspired by BitTorrent for data transfer across nodes and clients. This approach divides files into smaller pieces and distributes them across the network using a peer-to-peer architecture. Unlike traditional file transfer methods, IPFS eliminates the reliance on a central point of control, allowing files to be accessed from any node in the network that possesses a copy of the desired file.

When a user requests a file, IPFS employs a distributed hash table (DHT) to locate nodes that possess the corresponding blocks. This decentralized lookup mechanism enables efficient retrieval of file fragments from multiple sources simultaneously. As a result, IPFS optimizes file transfer speeds by leveraging the collective bandwidth and resources of the network's nodes.

IPFS employs a caching mechanism that facilitates data availability. Upon retrieving a file block, a node automatically caches it, enabling other nodes to access it if needed. This distributed caching system significantly improves the overall network performance and resilience by ensuring that frequently requested blocks are conveniently located near the requesting nodes.

1.3 SOLID Project

1.3.1 Overview

The SOLID Project, initiated by Sir Tim Berners-Lee, is an open-source endeavor aimed at creating a more decentralized and user-centric web. It introduces the concept of Personal Online Data Stores (PODs) that enable individuals to store and manage their data in a secure and controlled manner, granting users ownership and agency over their personal information. By adhering to SOLID specifications, users have the ability to determine how their data is shared and accessed by applications and services. The project aims to revolutionize the web landscape by promoting data privacy, user empowerment, and interoperability.

Regarding the use of Distributed Hash Tables (DHT) or other routing mechanisms, the SOLID project itself does not explicitly rely on DHT or specific routing protocols. The focus of SOLID is primarily on data storage, ownership, and access control. However, it is worth noting that the SOLID ecosystem can be complemented with additional technologies or protocols for decentralized communication and routing, depending on the specific implementation or integration.

Major technical components of the SOLID Project include:

- **PODs (Personal Online Data Stores)**: These are user-controlled storage spaces where individuals can securely store and manage their personal data. PODs can be hosted locally or in the cloud, and users have full control over access permissions.
- **Linked Data**: SOLID leverages Linked Data principles to enable decentralized and interconnected data on the web. It encourages the use of standardized RDF (Resource Description Framework) data models and URIs (Uniform Resource Identifiers) for representing and linking data.
- **WebID**: WebID is a core concept in SOLID that allows users to have a unique identifier (URI) associated with their online identity. WebIDs can be used for authentication, access control, and establishing trust in the decentralized web ecosystem.

- **Solid Server**: The Solid Server is an essential component that facilitates the storage, retrieval, and management of data in PODs. It acts as the interface between the user's data and applications, providing access control mechanisms and handling data synchronization.
- **Solid Protocol**: The Solid Protocol defines the communication and interaction standards for exchanging data between PODs and applications. It promotes the use of standard HTTP methods (GET, POST, PUT, DELETE) and RESTful principles for data manipulation.
- **Solid App Ecosystem**: SOLID encourages the development of a rich ecosystem of applications and services that can seamlessly interact with user-owned data stored in PODs. These applications can request access to specific data resources based on user permissions and can provide enhanced functionalities while respecting user privacy.

1.3.2 Data Storage

In SOLID, data is managed through Personal Online Data Stores (PODs). PODs are individual data repositories where users can store their personal data and have control over its access and management.

Data in PODs is typically stored in a structured format, such as Resource Description Framework (RDF) or JSON-LD, which allows for semantic representation and interoperability. Each piece of data in a POD is identified by a unique Uniform Resource Identifier (URI) that serves as its address within the SOLID ecosystem.

Users have the ability to create, modify, and delete data within their PODs. They can define access control rules and permissions to determine who can access and modify their data. This enables users to maintain ownership and control over their personal information.

When a user interacts with an application or service that integrates with SOLID, the application makes HTTP requests to the user's POD to access and manipulate the data. The SOLID server acts as the intermediary between the application and the user's POD, handling authentication, authorization, and data retrieval/modification.

Data in a POD can be organized into different containers or folders, allowing users to structure and organize their data according to their needs. Additionally, data in PODs can be linked to other data resources within the user's POD or even across different PODs, enabling the creation of interconnected and semantic data networks.

SOLID emphasizes the concept of Linked Data, where data resources can be linked together using standardized vocabularies and ontologies. This enables the discovery and integration of data across different PODs and applications, fostering data interoperability and collaboration.

1.3.3 Data Access

PODs in the SOLID ecosystem use access control mechanisms to grant, revoke, and restrict access to data. These mechanisms allow users to have fine-grained control over who can access their data and what actions can be performed on it. Common components of the SOLID data access ecosystem are:

- **Access Control Lists (ACLs)**: ACLs are a common mechanism used in SOLID to define access permissions. An ACL is associated with each resource in a POD and contains a list of agents (users or groups) and their corresponding access permissions. This allows the owner of the resource to specify who can read, write, append, or delete the data.
- **Web Access Control (WAC)**: WAC is a standardized access control mechanism in SOLID that builds upon ACLs. It provides a more flexible and granular way of defining access permissions. WAC allows for complex access policies to be defined using rules and patterns, enabling users to express sophisticated access control requirements.
- **Authentication and Authorization**: PODs use authentication mechanisms to verify the identity of users and ensure they have the necessary permissions to access the data. This can involve authentication protocols like OAuth or OpenID Connect. Once the user is authenticated, authorization mechanisms are used to determine if the user has the required access rights based on the defined ACLs or WAC rules.
- **Access Delegation**: SOLID also supports access delegation, where users can grant access to their data to other individuals or applications. This

allows users to share specific resources or parts of their data with trusted entities without sharing their entire POD credentials.

- **Consent Management**: SOLID places importance on user consent for data access. Users have the ability to explicitly grant or revoke consent to applications or services requesting access to their data. This ensures that users have control over which entities can access their data and for what purposes.

Let's consider an example where a user, Alice, wants to share a specific file in her POD with two other users, Bob and Carol, and grant them read and write permissions. As shown in Figure 1.5, the figure illustrates next situation:

1) Alice creates a file named "document.txt" in her POD.
2) She sets up the WAC for the file by creating an ACL associated with it.
3) In the ACL, Alice adds two ACEs:
   - ACE 1: Subject = Bob's WebID, Modes = Read, Write
   - ACE 2: Subject = Carol's WebID, Modes = Read, Write

   This grants both Bob and Carol read and write access to the file.
4) Alice saves the ACL, which gets associated with the "document.txt" file.

Now, when Bob or Carol try to access the "document.txt" file in Alice's POD, the following happens:

- The Solid Server verifies their identities using their WebIDs.
- The server checks the ACL associated with the file to determine their access permissions.
- If the ACL allows read and write access for their WebIDs, they are granted access to the file.
- Otherwise, if the ACL denies their access or they don't have any matching ACE in the ACL, their access is denied.

1.3.4 Network

The SOLID network is fully based on HTTP. It utilizes standard HTTP methods (GET, POST, PUT, DELETE) for data retrieval, modification, and communication between nodes and services. Unlike IPFS, SOLID is not using

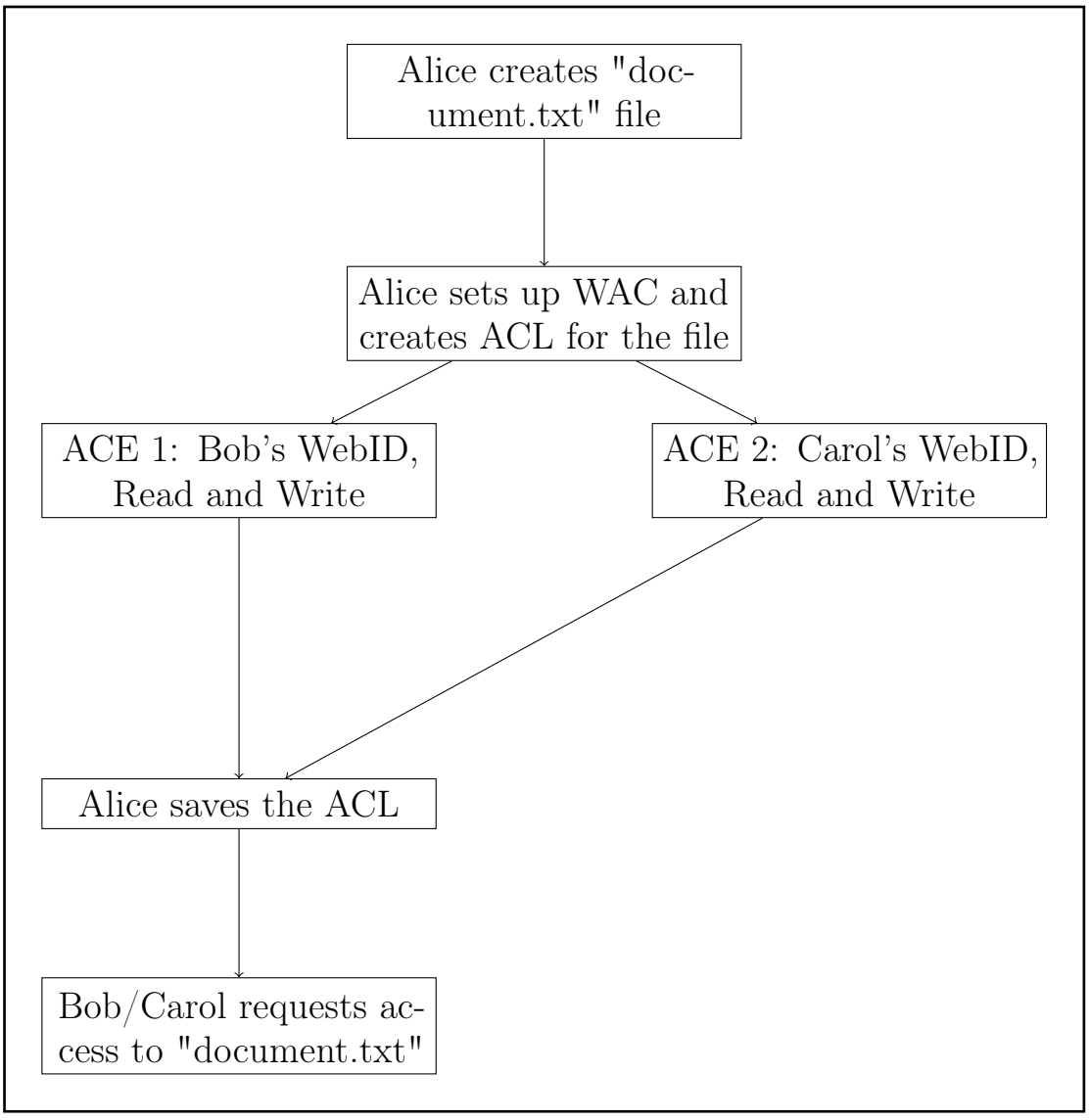

Figure 1.5. Example Usage of WAC and ACLs

DHT or other routing mechanism.

To do this, consider how clients and other PODs interact with the data in the original POD. Suppose a web application wants to retrieve a user's profile information from their POD. The application would send a GET request to the corresponding resource URI, such as "https://example.com/pod/profile". The Solid Server receives the request and authenticates the application using the user's WebID. The WebID is a unique identifier associated with a user and is used for authentication and authorization purposes. WebID is typically included as part of the HTTP headers. When a client, such as a web application, interacts with a POD, it includes the WebID in the HTTP headers to authenticate itself and gain access to the user's resources. The WebID is commonly included in the Authorization header using the "Bearer" authentication scheme.

1.4 Decentralized Web Node

1.5 Overview

A Decentralized Web Node is a fundamental building block of the decentralized web infrastructure purposed by Decentralized Identity Foundation (DIF), designed to provide secure and resilient data storage and message relay capabilities. It serves as a distributed storage layer that allows individuals and organizations to store and manage their data in a decentralized manner, promoting data sovereignty and ownership.

At its core, a Decentralized Web Node consists of several major technical components. First, it employs a distributed storage system, utilizing peer-to-peer networking protocols and distributed hash tables (DHTs), to store and replicate data across multiple nodes in a decentralized network. This ensures data availability and resilience, as there is no single point of failure.

Secondly, a Decentralized Web Node incorporates a decentralized identity system, such as the Decentralized Identifier (DID) standard, to uniquely identify entities and provide secure access control to data. DIDs enable individuals and organizations to have self-sovereign identities and control over their data,

allowing for fine-grained permissioning and data sharing.

Furthermore, the Decentralized Web Node serves as a message relay mechanism, facilitating secure peer-to-peer communication and data exchange within the decentralized web ecosystem. It enables entities to interact with each other, exchange messages, and request access to specific data related to a given DID.

The purpose of a Decentralized Web Node is to enable a decentralized and user-centric web experience. By leveraging distributed technologies, it aims to overcome the limitations of traditional centralized web architectures, such as data breaches, vendor lock-in, and lack of user control. It empowers individuals and organizations to store and manage their data securely, maintain ownership over their digital identities, and establish direct and secure communication channels within the decentralized web ecosystem.

It should be noted that DWN is much more theoretical framework for future systems than an application level program used by a large number of clients and receiving feedback on the functionality of the architecture.

1.5.1 Network

The specific distributed hash table (DHT) implementation used by the Decentralized Web Node (DWN) may vary depending on the underlying technology stack and design choices. As DWN is a conceptual framework, it does not specify a particular DHT implementation.

1.5.2 Decentralized Identifier (DID)

The Decentralized Identifier (DID) (Picture 1.6) is a unique identifier representing entities within the decentralized web ecosystem. DIDs serve as globally unique identifiers independent of any central authority or specific platform.

DIDs are designed to provide a decentralized mechanism for identifying and verifying entities within the decentralized web, such as individuals, organizations, or devices. They are based on the principles of self-sovereign identity,

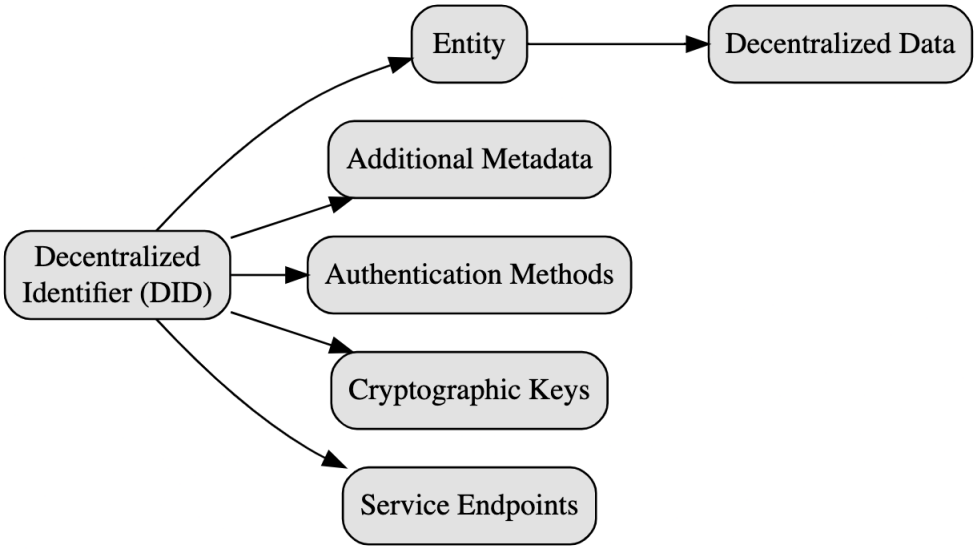

Figure 1.6. DID Structure

where individuals have control over their digital identities and can manage and authenticate their identity information without relying on centralized authorities.

Within the DWN framework, DIDs play a crucial role in data storage and retrieval. They serve as references or pointers to specific data associated with an entity, such as personal information, credentials, or other relevant data. DIDs enable the linking and retrieval of decentralized data by providing a consistent and verifiable way to reference and access data within the decentralized web network.

DIDs typically consist of a unique identifier string and may be accompanied by additional metadata, such as cryptographic keys, authentication methods, or service endpoints. These additional components enhance the security and functionality of DIDs, allowing for secure authentication and interaction with the associated data.

Diagram 1.7 shows the topology of the communication scheme and DID application between Alice and Bob clients using DID Resolver.

1.6 Analysis Conclusion

Several critical implications may be taken from analyzing the explored methodologies in these three open-source data storages for personal data, influencing the direction of the proposed solution mentioned in this work. While

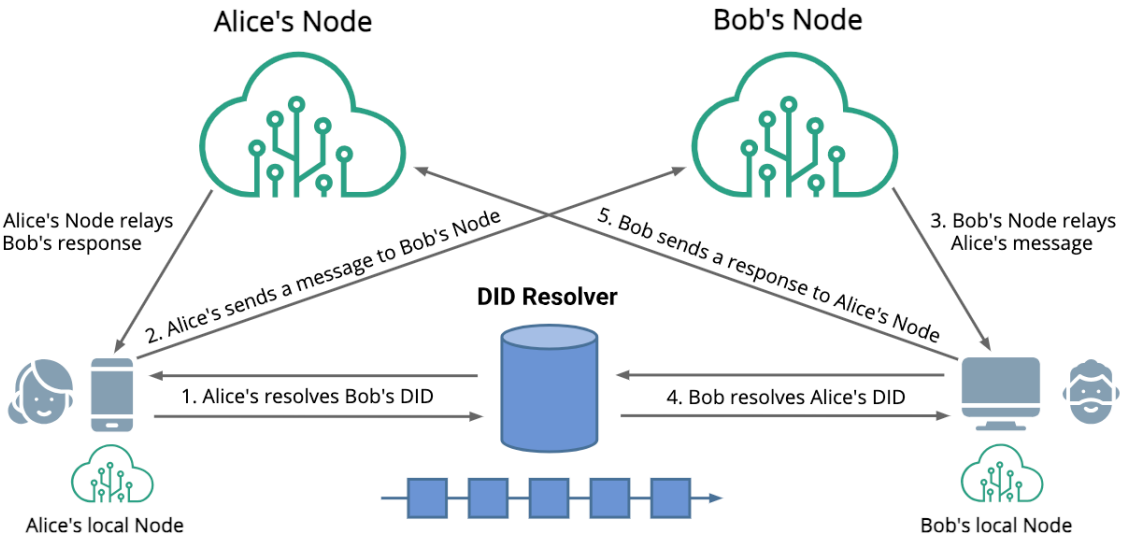

Figure 1.7. DID Topology [18]

these data storages appear diverse, they all share a common goal: to revolutionize data storage in a decentralized manner by detaching data from apps and enabling flexible administration of personal data in distributed networks.

One notable aspect is the SOLID Project, which emphasizes SOLID Data Access and Linked Data based on Access Control List and Access Delegation. This approach offers intriguing possibilities, granting users complete control over data storage and access. However, it lacks mechanisms to protect data on the server side, where services may receive the data. This limitation highlights the need for enhanced security measures to safeguard data throughout its lifecycle.

Data chunking mechanisms also warrant special attention, as they provide a means to securely store data on unfamiliar hosts without compromising its integrity. Algorithms like the Rabin fingerprint-based approach and the Merkel Directed Acyclic Graph (DAG) contribute to ensuring data integrity. However, there is room for improvement regarding client-side control over data utilization.

Another noteworthy concept is the Decentralized Web Node specification, which showcases the practical application of Decentralized Identity (DID) in our daily lives. This innovative approach envisions a future where physical documents and access cards are replaced by digital counterparts derived from an individual's identity. The potential of DID to revolutionize identity management is remarkable, opening avenues for increased convenience and security.

While these approaches differ in purpose, they share similar mechanisms

and engineering approaches, demanding special attention. Remarkably, implementing data routing and network protocols such as Distributed Hash Tables (DHT) offers an excellent opportunity to redefine data request routing, potentially replacing traditional DNS systems. The Kademlia protocol, known for its scalability, fault tolerance, and simplicity, is a good starting point for further exploration in this domain.

Regarding the physical storage of data, these approaches converge on the principle of chunking and distributing data to random nodes in the network. However, the final storage destination is expected to be the client's file system, ensuring data safety. In contrast, SOLID relies on server-level data storage in the file system, proposing a persistent server or provider akin to Network-Attached Storage (NAS).

Considering this comprehensive technical overview, it is possible to propose an architecture for a new system that addresses users' needs, resolves customer challenges, and tackles broader internet-related issues. The proposed solution lays the groundwork for a transformative data storage ecosystem by aligning users' desires with the internet's evolving landscape.

CHAPTER 2

DESIGN AND IMPLEMENTATION

2.1 Design Techniques Outline

The suggested solution and architecture have been developed using the information in this paper, and the arguments around the legitimacy of the storage and use of personal customer data by companies on enterprise cloud platforms that end users do not manage. As was previously stated, an audit is the only reliable approach to determine the security of any information system within an existing system. Companies, especially startups, need help to afford the expenditures of developing and maintaining such auditing and compliance with governmental regulators.

The goal of the solution is to provide an ethical PDS-style data warehouse for users and businesses, which will serve as a multifunctional distributed database and web server, allowing users and data creators to govern applications and data storage independently.

To address the challenges of data storage and modeling, it is valuable, to begin with a specific storage system that efficiently manages memory and stores data on disk. Rather than building a database from scratch, this project aims to identify the requirements for such a storage system and leverage existing solutions provided by the open-source community. Such a system should effortlessly combine document-oriented, graph-oriented, and relational approaches into a single framework. It should also provide a query language that is more scalable than conventional SQL. The system should run as a schema-less environment by default, changing to a schema-full environment as the data model develops. It should permit transactions between several tables and applications and offer real-time monitoring of data changes. In order to provide a high level of abstraction and separation, the database needs to be developed as a multi-tenant platform with a namespace implementation. This approach allows for creating multiple namespaces without constraints, ensuring each service remains independent. Below this layer lies the database layer, where any namespace can host numerous

databases.

Regarding network and routing, the solution employs cryptographic hashes to identify and locate files in a content-addressed data system. Object shards are lazily loaded, conserving disk space by storing only accessed blocks and caching the rest. The Kademlia routing protocol organizes nodes based on distance and maintains an efficient routing table.

Decentralized Identifiers (DIDs) are used to manage identities. DIDs offer distinctive identification for individuals, entities, and devices inside a decentralized and secure system. DIDs provide people control over their personal information, enhancing privacy and lowering the chance of identity theft. The approach lessens dependency on centralized identity providers like Google SSO by enabling interoperability between many identity systems. Removing the need for intermediaries to confirm identifying information also improves transaction efficiency and security.

The component in charge of automatic and self-executing licenses for data usage takes up a significant portion of the entire project and the database interface. Data access will be refused to systems or apps that do not adhere to the terms of the user license, protecting the privacy and authenticity of the data. Users still have control over how their data is used and shared, and they can modify, provide, or remove their consent at any time in compliance with the applicable data protection license issued by the user. This enables one to organize the system of data storage and use in a peer-to-peer network easily and offers high software data security by working with DID and a decentralized database.

2.1.1 Ethical Design Synopsis

Ethical Design is a concept previously mentioned in this paper, and it is now crucial to delve into its fundamental principles and approaches. This section aims to understand Ethical Design neutrally and scientifically comprehensively.

At its core, Ethical Design encompasses a set of principles and practices that prioritize ethical considerations in developing technological solutions. It seeks to address technology's ethical implications and societal impacts, ensuring

that these innovations align with moral values and respect the rights and well-being of individuals and communities.

Figure 2.1. Ethical Design Manifesto[19]

One of the key principles of Ethical Design is user empowerment. This involves giving users greater control and agency over their personal data, privacy, and digital experiences. It emphasizes the importance of informed consent, transparency, and user-friendly interfaces that enable individuals to make informed choices about their data and interactions with technology.

Another essential aspect of Ethical Design is privacy protection. It advocates for robust data protection measures, including secure storage, encryption, and privacy-enhancing technologies. By safeguarding personal information and limiting data collection and usage to legitimate purposes, Ethical Design seeks to mitigate the risks of data breaches, surveillance, and misuse.

Furthermore, Ethical Design promotes inclusivity and accessibility. It recognizes individuals' diverse needs and experiences and aims to ensure that technology is accessible and usable for all, regardless of their abilities or circumstances. This involves considering diverse user perspectives, designing for inclusivity, and avoiding discriminatory or exclusionary practices.

Ethical Design also emphasizes sustainability and environmental responsibility. It encourages the development of eco-friendly technologies that minimize

energy consumption, reduce waste, and mitigate the environmental impact of digital solutions. This includes considering the entire lifecycle of products, from sourcing materials to disposal and adopting sustainable practices throughout the design and production processes. Additionally, Ethical Design encourages accountability and transparency among technology creators and providers. It advocates for clear and comprehensible terms of service, responsible data governance, and mechanisms for recourse and redress in case of ethical violations. Ethical Design also promotes ethical business models prioritizing long-term societal benefit over short-term profit.

In summary, Ethical Design embodies principles and approaches that prioritize user empowerment, privacy protection, inclusivity, sustainability, and accountability in the Design and development of technology. By adhering to these ethical considerations, technology can better serve individuals and communities while upholding moral values and societal well-being.

2.2 Secure Data Protocol (Garlic-like encryption)

The diagram 2.2 introduces some more precise concepts and terminologies, while maintaining the garlic metaphor. We'll also consider the case where the garlic head, composed of multiple cloves, represents different data for various clients with distinct usage conditions.

- **Garlic Peel (End-to-End Encryption)**: The garlic cloves (data) are enveloped by a protective layer (encryption). This is analogous to employing end-to-end encryption which provides confidentiality, a crucial attribute in Information Theory. Using cryptographic techniques such as RSA, this security measure mathematically transforms plaintext data into ciphertext, rendering it incomprehensible to unauthorized parties. The security of such encryption hinges on the computational difficulty of factorizing large prime numbers, a problem considered intractable with current computational resources. It possible to note the encyption process

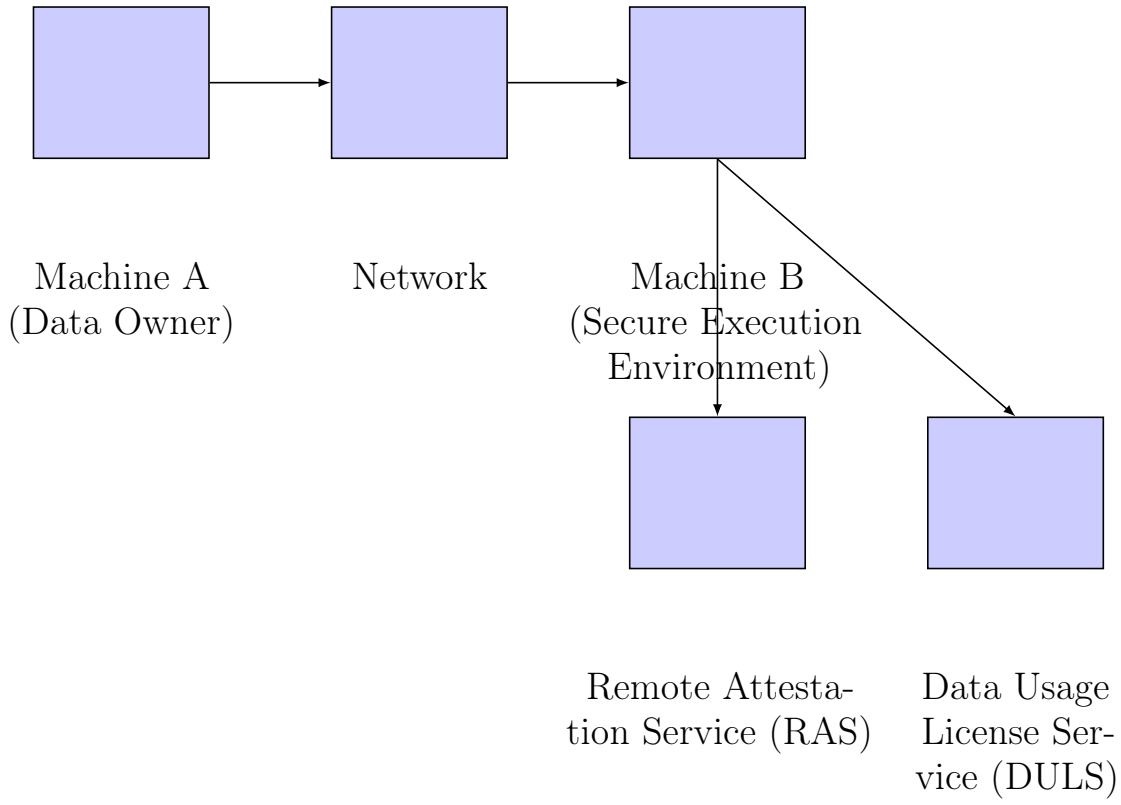

Figure 2.2. Diagram of data transmitted over network

as follows:

$$E : D \to D'$$

, where D is the original data and D' is the encrypted data. For RSA encryption, E might involve raising the data to the power of the public key and taking the modulus with respect to a large prime number. The decryption process is the inverse of this:

$$D : D' \to D$$

- **Peeling Process (Remote Attestation)**: Upon receipt of the garlic clove (encrypted data), the recipient evaluates the quality of the peel (verifies the encryption) prior to accessing the garlic (data). This aligns with the concept of remote attestation in cybersecurity, where the integrity of a system's software stack is verified before data transmission. This process mitigates the risk of data compromise, providing assurance that the data will be processed in a secure and trusted environment. It possible to model remote attestation as a binary function that verifies the integrity

and authenticity of the software stack:

$$R : (S, K) \rightarrow 0, 1$$

, where S is the software stack of the receiving machine, and K is a known good configuration. R returns 1 if the software stack matches the known good configuration and 0 otherwise.

- **Garlic Press (Secure Execution Environment)**: The approved garlic (data) is then processed within the confines of a garlic press (secure execution environment). Analogous to this, a secure execution environment in cybersecurity ensures that data remains secure during processing, a notion central to the concept of data integrity. Such environments offer protection from potential threats by segregating data processing from other system activities, a concept rooted in the principle of least privilege in access control models. It possible to model the secure execution environment as a function that processes the data:

$$S : (D, L) \rightarrow O$$

, where D is the decrypted data, L is the data usage license, and O is the output of the processing.

- **Recipe (Data Usage License)**: Finally, the garlic (data) is used following specific instructions from a recipe (data usage license). This corresponds to pre-defined data access and usage policies in data governance. These policies dictate permissible data interactions, thereby ensuring adherence to the data owner's terms and conditions. It possible to model the data usage license as a function that restricts the actions that can be performed on the data:

$$L : (D, A) \rightarrow 0, 1$$

, where D is the data and A is an action. L returns 1 if the action is allowed on the data and 0 otherwise.

The broader structure of a garlic head containing multiple cloves allows the metaphor to be extended to consider data sharing with multiple clients.

Each garlic clove represents a distinct data set, perhaps with different usage conditions or destined for different end users. Just as each clove is enclosed in its own peel, each data set is encrypted separately, ensuring the privacy and integrity of each dataset. The distinct 'peel' also signifies the uniqueness of the usage rules (recipes) for each clove (data set). This approach underpins the principle of data minimization and segmented access control, which are fundamental to secure data management with respect Ethical Design in section 2.1.1.

The secure data transfer algorithm proposed herein offers a comprehensive solution to the challenge of securely transferring data from one machine to another over an unsecured network, while also controlling the use of that data on the client machine. It incorporates a robust layer of security through the use of RSA encryption, remote attestation, and a secure execution environment, ensuring that the data can only be accessed and used under the conditions defined by the data owner.

The algorithm starts on the data owner's machine (Machine A) by defining the data to be transmitted and encrypting it using RSA encryption. This encrypted data is then sent over the network to the client machine (Machine B), along with the public key for decryption.

Upon receipt of the data, the client machine performs a remote attestation to verify the integrity and security of its software environment. If the attestation is successful, the data is decrypted using the private key and then processed within a secure execution environment. The usage of the data is controlled by a license function that defines the permissible actions on the data. Any action that violates the license leads to the termination of that action and a violation is logged.

If the remote attestation is unsuccessful, the data is rejected and the connection is terminated. This multilayered approach ensures that the data is not only transferred securely but also used within the constraints defined by the data owner.

The algorithm (figure 2.2) is described in detail as follows:

Algorithm 1 Secure Data Transfer from Machine A to Machine B

Machine A (Data Owner):
Generate RSA public and private keys: (n, e), (n, d).
Define data D to be transmitted.
Encrypt D using RSA public key: $D' = D^e \mod n$.
Send encrypted data D' and public key (n, e) over the network.
Machine B (Client):
Receive encrypted data D' and public key (n, e).
Perform remote attestation R to check software environment.
if R is successful **then**
    Decrypt data using RSA private key: $D = D'^d \mod n$.
    Define usage license function L.
    Process data D in secure execution environment S with license L.
    if any action violates license L **then**
        Terminate action and log violation.
    end if
else
    Reject data and terminate connection.
end if

2.3 Data Usage License Service (DULS)

In the realm of secure data transmission and handling, the imposition of data usage licenses has emerged as an essential paradigm, providing the legal and procedural framework that governs the permissible use, storage, and data transmission between entities.

A data usage license can be conceptualized as a function $L : D \times E \to R$, where D denotes the data being transmitted, E represents the entities (e.g., users, applications) that interact with the data, and R refers to the set of permissible actions or restrictions applicable to that entity for the data in question. This function encapsulates the core tenets of the data usage license, translating the raw data and the entity's characteristics into a distinct set of actions permissible under the given license.

The data usage license integrates seamlessly within the overall architecture of the secure data transmission protocol. The protocol serves as a vehicle for data and its corresponding usage license, ensuring secure delivery to the recipient. This integration can mathematically represent a composite function $T(L(D, E))$

where T represents the secure transmission function.

This approach offers multiple benefits:

- **Contextual access control**: The license function $L(D, E)$ allows for granular and context-specific access control, enabling unique restrictions based on the combination of data and entity attributes. This is essential in environments where data sensitivity varies, and the access rights of entities are not uniform.
- **Auditability**: By defining the permissible actions through a mathematical function, it is possible to audit the actions performed by entities on data retrospectively, thus ensuring compliance.
- **Legal Compliance and Enforcement**: The usage license serves as a legal contract between the data provider and the recipient, ensuring adherence to the agreed-upon terms of data use and providing a basis for legal recourse in case of violations.
- **Enhanced Security**: Incorporating usage licenses within the secure data transmission process reduces the risk of data misuse, as each data item is associated with clear usage terms that must be adhered to.

In order to delve more into the complexities of secure data transmission and data usage licenses, it becomes evident that these components are inextricably linked. Adequate data security and privacy preservation demand robust, secure transmission technologies and extended data usage licenses, all working together to preserve data in the digital landscape.

2.3.1 License implementation

In the pursuit of secure data exchange and usage protocols, it is fundamental to establish a comprehensive and enforceable data usage license (DUL). The role of this license is to provide an explicit contractual agreement regarding the terms and conditions for data storage, usage, copying, and other pertinent factors. The model proposed earlier encompasses a broad range of factors, including permissions, timeframes, storage restrictions, and legal considerations.

Let us consider an instance where Alice (data producer, Machine A) desires to share data with Bob (data consumer, Machine B). Before the data

exchange, Alice must provide Bob with a DUL detailing how the data should be handled (Figure 2.3). Data transfer can proceed Only after Bob's acceptance, either manually or automatically.

During this process, ensuring that the terms stipulated in the DUL are enforced is paramount, which can be significantly challenging, especially in low-trust or untrusted environments. The technical enforcement of these conditions can be achieved through several measures. Secure Execution Environments (SEEs), such as Intel SGX, ARM TrustZone, or AMD SEV, offer a hardware-protected memory area allowing isolated data processing. This isolation ensures that data usage within the SEE adheres to the DUL.

In parallel, Remote Attestation can offer a way for Alice to verify that Bob is operating within a secure environment and adhering to the DUL. This process may involve a challenge-response protocol, where Alice sends a challenge that Bob must respond to in a way that proves his secure state.

Policy Enforcement Points (PEPs) can enforce the DUL at various points in the data life cycle. For example, a PEP could block data from being written to disk if the DUL forbids this action.

Automated Policy Checking and Enforcement may also be necessary, depending on the complexity of the DUL. This could involve techniques derived from formal methods or artificial intelligence to ensure the correct and comprehensive enforcement of the DUL.

Moreover, Auditing and Monitoring mechanisms are essential for ensuring continuous adherence to the DUL. These mechanisms could involve automated logging and monitoring systems and periodic audits by security professionals.

Lastly, Legal and Contractual Measures can serve to enforce the DUL beyond the scope of technical measures. This could involve contracts, Service Level Agreements (SLAs), and legal sanctions for violations of the DUL.

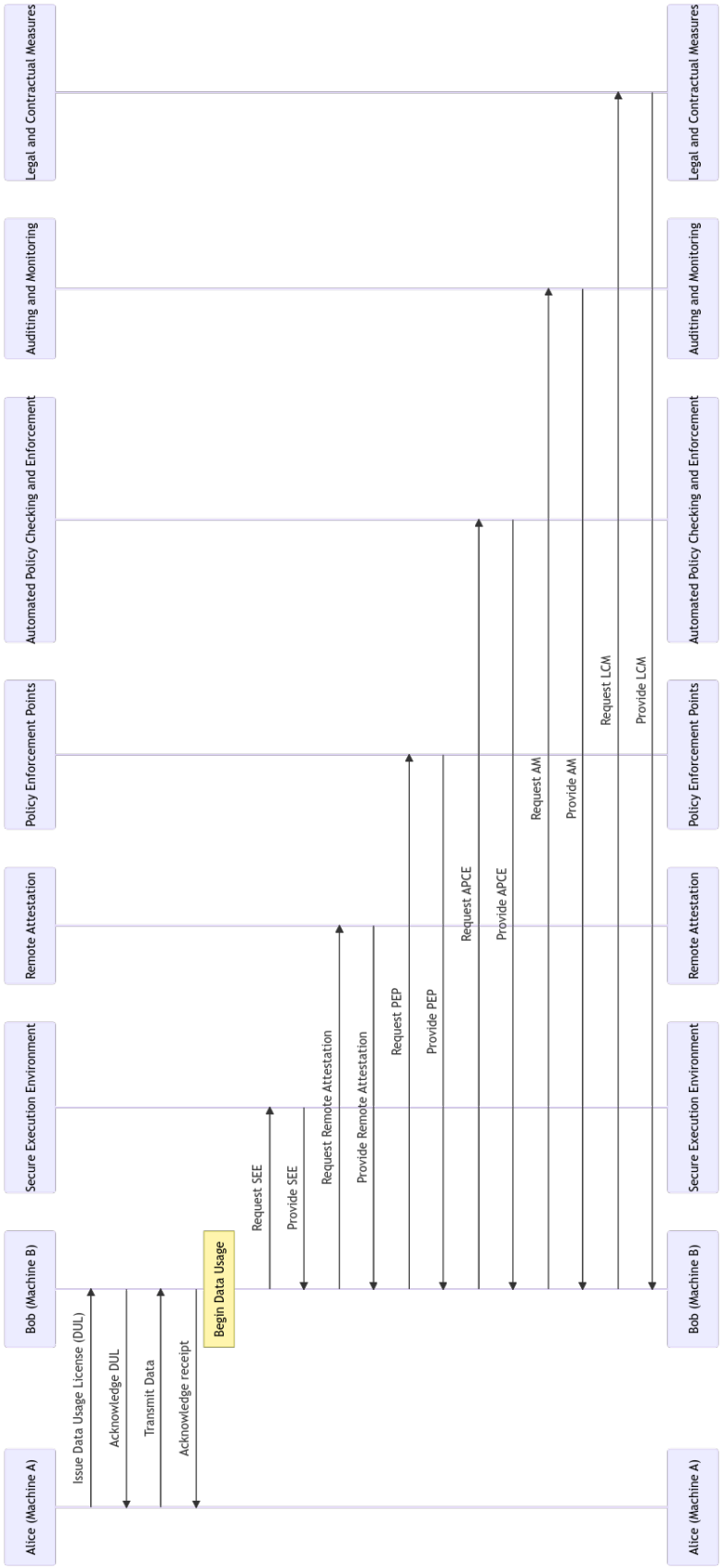

Figure 2.3. Sequence Diagram of using Data Usage License

2.3.2 Designing a Machine-Readable Data Usage License (DUL)

In the context of secure data transmission and usage regulation, a critical component is the Data Usage License (DUL). This comprehensive document encapsulates all relevant permissions, constraints, and guidelines governing the use of the data. Given the computational context of this operation, this license must be created in a machine-readable format.

Machine-readable formats are standardized data structures that can be efficiently parsed and processed by software. JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are two primary formats that could serve this purpose effectively. Both these formats offer structured, hierarchical data representation, enabling a detailed layout of the DUL terms and conditions.

Each rule, restriction, or term contained within the DUL - such as permissions, usage restrictions, timeframes, storage restrictions, etc. - would be represented as a key-value pair within the JSON or XML document. This approach provides a streamlined way for the software to fetch and understand each license component.

Each DUL should be assigned a unique identifier to ensure uniqueness and ease of tracking. Universally Unique Identifiers (UUID) serve this purpose well. These identifiers are designed to be globally unique, ensuring each DUL can be individually referenced and managed.

The following is the diagram 2.4 and description of the Data Usage License model structure.

- **License**: Represents a data usage license agreement.
  - *LicenseID*: A unique identifier for each license agreement. This allows each license to be individually tracked and managed.
  - *reference*: A reference to the specific data this license pertains to, represented by an instance of *DataKeyReference*.
  - *permissions*: Details of what actions the license is permitted to do with the data, represented by an instance of *Permissions*.
  - *restrictions*: Additional rules about how the data can be used,

represented by an instance of *Restrictions*.
  - *metadataAccess*: Permissions related to metadata about the data, represented by an instance of *MetadataAccess*.
  - *revocation*: Rules about when and how the license can be revoked, represented by an instance of *Revocation*.
  - *attestation*: Requirements for the license to prove their adherence to the license, represented by an instance of *Attestation*.
  - *legalJurisdiction*: Legal jurisdiction and governing law of the license, represented by an instance of *LegalJurisdiction*.
- **DataKeyReference**: A reference to the specific data this license pertains to.
  - *identifier*: An identifier for the data. This could be any string that uniquely identifies the data.
  - *hashValue*: A cryptographic hash of the data. This allows the data's integrity to be verified.
- **Permissions**: What actions the license is permitted to do with the data.
  - *read*: Can the license read the data?
  - *write*: Can the license modify the data?
  - *delete*: Can the license delete the data?
  - *share*: Can the license share the data with others?
- **Restrictions**: Additional rules about how the data can be used.
  - *temporal*: When can the data be accessed or used? Represented by an instance of *TemporalRestrictions*.
  - *spatial*: Where can the data be used? Represented by an instance of *SpatialRestrictions*.
  - *derivativeWorks*: Are derivative works permitted?
- **MetadataAccess**: Permissions related to metadata about the data.
  - *read*: Can the license read the metadata?
  - *write*: Can the license modify the metadata?
  - *share*: Can the license share the metadata?
- **Revocation**: Rules about when and how the license can be revoked.
  - *expirationDate*: When does the license expire?

  - *conditions*: Under what conditions can the license be revoked? Represented by an instance of *RevocationConditions*.
  - *method*: How is the license revoked? Represented by an instance of *RevocationMethod*.

- **Attestation**: Requirements for the license to prove their adherence to the license.
  - *remoteAttestation*: Is remote attestation required?
  - *logging*: Are logging and monitoring required?
  - *auditReports*: Are regular audit reports required?

- **LegalJurisdiction**: Legal jurisdiction and governing law of the license.
  - *country*: What country's law applies to the license?
  - *state*: What state's law applies, if applicable?
  - *applicableLaws*: Any specific laws or regulations that apply to the license.

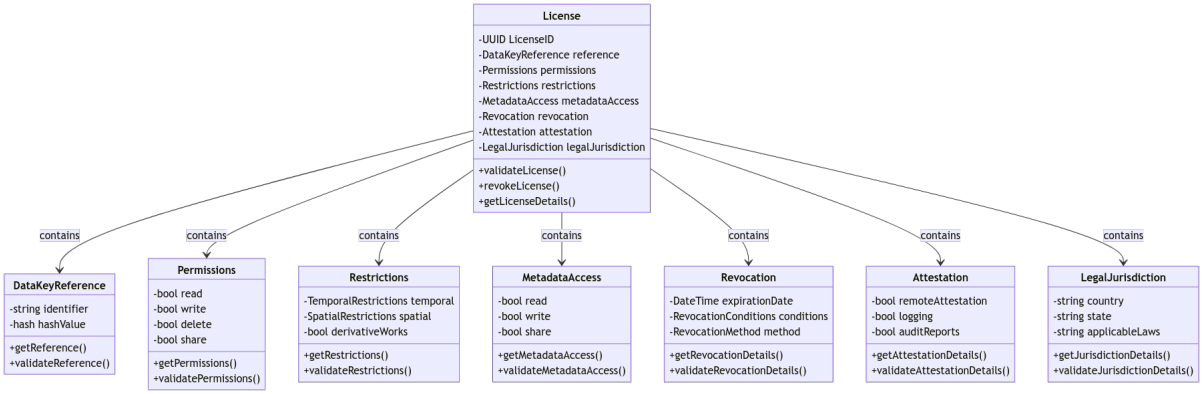

Figure 2.4. Data Usage License Structure

In conclusion, the methodologies and techniques discussed in Section 3.3.3, focusing on Data Access and the utilization of Access Control Lists (ACLs), Web Access Control (WAC), and Access, have played a significant role in shaping the concept of Data Usage Licenses (DUL). The application of these approaches, as seen in the SOLID Project, has facilitated the development of a robust system architecture capable of safeguarding user data residing on third-party servers or other client systems. By leveraging these methodologies, the proposed solution can effectively address data security concerns and establish a framework for responsible data handling in distributed environments.

2.4 Data Organisation

As previously mentioned, the main focus of this project is to demonstrate the practicality of combining decentralized techniques and a suitable multi-model storage approach to create a personal web server capable of storing personal data and making it accessible to web applications. In this project, particular attention is given to the open-source multi-model Surreal DB.

Surreal DB serves as a foundation for document storage, where each record is stored using a key-value storage engine that accommodates various types of data, including arrays and objects. However, Surreal DB offers more than just document storage capabilities. It can effectively handle time-series ordered data and highly-connected graph data by leveraging its unique approach to Record IDs and retrieving individual records from the underlying key-value storage engine. Additionally, Surreal DB introduces SurrealQL, an SQL-like query language, facilitating easy data manipulation across the database.

The versatility of Surreal DB enables it to run in different environments and deployment modes. It can be utilized as an embedded database, a vertically-scalable single-node server, or a horizontally-scalable distributed cluster. While Surreal DB shares similarities with traditional relational and document databases, it possesses distinct features that set it apart.

Surreal DB is designed as a multi-tenant database platform, offering a high-level namespace layer to separate applications, source of information. The number of namespaces on Surreal DB is unrestricted. Within each namespace, the database layer operates similarly to other database management systems, allowing an unlimited number of databases. Data can be stored within table definitions, referred to as collections in other systems. Each row or document in Surreal DB is known as a record. The platform also supports multiple authentication scopes, enabling custom authentication for tables, records, and fields.

2.4.1 Database Implementation

The foundation for working with the SurrealDB database will be a class DataStorage, which is an additional layer of abstraction over the SurrealDB implementation. The DataStorage class is technically implementing an Adapter pattern.

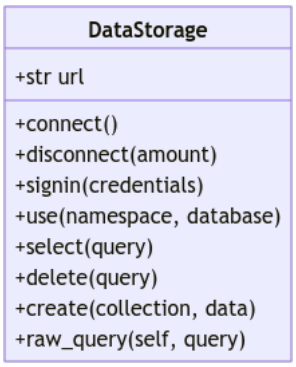

Figure 2.5. Implemented DataStorage Class

The implementation of SurrealDB offers a promising solution for achieving the following use case:

By leveraging the Namespace feature, it becomes possible to allocate a distinct namespace specifically for the application that the user intends to engage with. For instance, when utilizing a text recognition service, the service no longer retains data on its own infrastructure. Instead, the client side creates a Namespace with the prefix "APP_" appended to its name, ensuring clear separation and organization of data.

Within this Namespace, the service that the user interacts with can share its data models, facilitating the creation of databases tailored to the specific application's requirements. This enables efficient and streamlined data management within the designated Namespace, ensuring a coherent and well-structured data environment.

To further regulate data access and storage, users can employ Data Usage Licenses (DULs). Through the issuance of DULs, users can grant the service explicit permission to utilize specific data and store relevant user-related data on their servers. This mechanism provides a means to establish clear boundaries and control over data usage and storage, enhancing transparency and user autonomy.

By implementing SurrealDB, this use case demonstrates the potential to create a secure and efficient environment where applications and services operate within dedicated namespaces. This approach allows for effective data separation and management, ensuring that data associated with a particular application remains distinct and properly licensed. Such a framework promotes privacy, data governance, and user-centric control, contributing to an improved and responsible data ecosystem.

This usercase can be implemented programmatically on SurrealDB side as follows:

```
DEFINE NAMESPACE APP_EXAMPLE;
USE NS APP_EXAMPLE;
DEFINE DATABASE APP_EXAMPLE_INTERMEDIATE_DATA;
```

and it is possible to define accessible token for application inside SurrealDB and expose to service:

```
USE NS APP_EXAMPLE;

DEFINE TOKEN APP_EXAMPLE_ACCESS_TOKEN
  ON NAMESPACE
  TYPE HS512 # HS512 used only for example
  VALUE "APP_EXAMPLE_PUBLIC_ACCESS_KEY";
```

Also, SurrealDB has functionality to define scope with limited session duration and add this scope to the access based on token for previous example:

```
DEFINE SCOPE APP_EXAMPLE_TEMP_SESSION SESSION 24h
        SIGNUP ( CREATE user SET email = "")
        SIGNIN ( SELECT * FROM user WHERE email = "");
```

By harnessing the power of SurrealDB and Data Usage Licenses (DULs), it becomes feasible to establish seamless access to user data irrespective of the specific node where it is stored. The flexibility provided by SurrealDB allows for the implementation of additional functionalities that can enhance the system's capabilities. For instance, it is conceivable to develop features that enable the sharding of Namespaces and Databases while ensuring the preservation of critical authentication properties. Moreover, the system can facilitate the

localization or distribution of data across network members, thereby ensuring rapid and efficient access to the associated services. This collaborative approach of leveraging SurrealDB and extending its functionality empowers the creation of a robust and scalable data storage system with enhanced accessibility and performance.

2.5 Network

As previously stated, the Kademlia protocol was chosen to enable routing between nodes in this project. This protocol's implementation in IPFS was also considered previously. The focus will now be on implementing this protocol in this solution. First, investigate the approaches used in the Kademlia protocol. Efficiency in Kademlia is gained by the way it manages the "network distance" between nodes. This distance is not about the physical distance or latency, but a function of the XOR operation on node IDs.

The defining characteristic of Kademlia is that it uses a XOR metric to define the "distance" between two points in the system, which has some valuable properties:

- The distance between A and B is the same as the distance between B and A. (It is symmetric)
- The distance between A and A is zero.
- The distance between A and B is positive if A $\neq$ B.
- The triangle inequality (The distance between A and C is at most the distance between A and B plus the distance between B and C)

Let's define the node identifiers in the system as binary strings. The distance is then defined as the XOR of these two strings, interpreted as an integer.

In mathematical terms, let's consider two nodes, A and B, with binary identifiers a and b. The distance 'd' between A and B is defined as:

$$d(A, B) = a \oplus b$$

where $\oplus$ denotes the bitwise XOR operation.

This XOR metric presents a unique quality, the unidirectionality. If A is closer to B than A is to C, then B is closer to A than C is to A. This property is exploited by the Kademlia routing algorithm.

In terms of efficiency, the Kademlia protocol specifies that any message can be sent to any node in the network with $O(\log(n))$ messages where n is the total number of nodes in the network. This logarithmic efficiency is achieved by maintaining certain invariants about the "routing table", a data structure each node keeps about the state of the network.

Each node's routing table is divided into 'k-buckets'. Each k-bucket corresponds to an interval of the XOR distance metric space, containing nodes at distance $[2^i, 2^{(i+1)})$, where i is the index of the k-bucket. Nodes in the network try to keep their k-buckets filled with live nodes, thus having more information about nodes that are closer (lower i) to them, as they have more k-buckets for closer nodes.

When a Kademlia node is looking for the node with a specific ID, it sends a FIND_NODE request to nodes in its own buckets that are in the direction of the target ID. This guarantees that each iterative step gets us closer (based on the XOR metric) to the target, and due to the logarithmic distribution of nodes in buckets, this lookup process takes $O(\log(n))$ time complexity, which is highly efficient.

2.5.1 Network implementation

In the context of a distributed system, the integration of a secure data transfer protocol, a Data Usage License (DUL), and the Kademlia Distributed Hash Table (DHT) creates a cohesive solution for secure data distribution, access control, and fault tolerance.

The secure data transfer protocol provides a mechanism for the reliable transmission of data across the network. This protocol ensures confidentiality and integrity by encrypting the data before transmission and verifying it upon reception. Additionally, it embeds the DUL within the transmitted data, protecting the usage constraints with the same level of security as the data

itself.

The DUL is a digital agreement that dictates how the received data can be used. It's encoded within the data payload and is extracted and verified by the client upon data reception. The DUL includes information on usage permissions, access time, existence duration on the client machine, and other metadata that specify the data handling procedures. Therefore, it serves as an automated way to enforce data usage policies and user agreements.

Kademlia DHT adds a level of fault tolerance and network efficiency to this system. It provides a way to distribute and locate data within the network based on a unique identifier, which is mapped to the data. This mapping allows efficient data retrieval even in a highly dynamic network.

Suppose the originating node that initially stored data is now offline. In such a scenario, thanks to Kademlia's data redundancy feature, the data (and its DUL) would have been replicated on several other nodes in the network. A client wishing to retrieve this data initiates a query within the Kademlia network using the data's unique identifier. The Kademlia protocol efficiently routes this request to the nearest available node storing the requested data. This data, still encrypted and bound by the DUL, is then securely transmitted to the client. Upon reception, the client applies the secure data transfer protocol's verification mechanism to ensure the data's integrity and decrypts it for use, adhering to the embedded DUL's terms.

This integrated system represents a robust solution for secure, controlled data distribution in a distributed environment. It leverages the strengths of secure data transfer protocols, usage licensing, and efficient, fault-tolerant data management via Kademlia DHT.

The basic Node based on Kademlia can be implemented as follow:

```
class InitialNode:
    def __init__(self, port, bootstrap_address, bootstrap_port):
        Thread.__init__(self)
        self.port = port
        self.bootstrap_port = bootstrap_port
        self.bootstrap_address = bootstrap_address

```

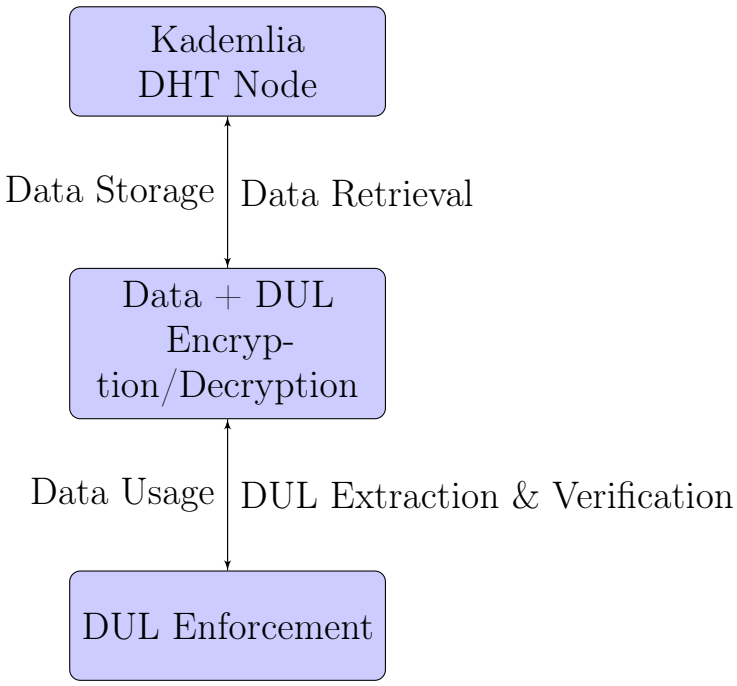

Figure 2.6. Implementing Kademlia DHT into the entire system

```
8     def start_node(self):
9         loop = asyncio.new_event_loop()
10        asyncio.set_event_loop(loop)
11        loop.set_debug(True)
12
13        kademlia_server = Server()
14        loop.run_until_complete(kademlia_server.listen(self.port))
15
16        if self.bootstrap_port is not None:
17            bootstrap_node = (self.bootstrap_address,
              ↪  self.bootstrap_port)
18            loop.run_until_complete(
19            kademlia_server.bootstrap([bootstrap_node]))
20
21        try:
22            loop.run_forever()
23        except KeyboardInterrupt:
24            pass
25        finally:
26            kademlia_server.stop()
27        loop.close()
```

2.6 Identity

In the context of our distributed system architecture, the concept of Decentralized Identifiers (DIDs) might not occupy the most conspicuous position, but its significance is central and paramount. DIDs serve as the core foundational element, a cornerstone that supports every component of the system. They not only define the digital persona of an entity but also dictate the access, transfer, and utilization of data within the system.

- **Identity Establishment:** Each user or application within the system fabricates a DID Document encapsulating the particulars of their digital identity, including public keys, authentication protocols, and service endpoints. These documents, stored in the Kademlia Distributed Hash Table (DHT) network, can be conveniently retrieved using the corresponding DID as a key.
- **Authentication Mechanism:** When a data access request is initiated by an entity, it undergoes an authentication process, where a message signed using the entity's private key is verified using the public key retrieved from the DID Document. This robust authentication process ensures the legitimacy of the entity demanding access to data.
- **License-Driven Authorization:** Post-authentication, the system checks for data access permission concerning the entity. The permissions are embedded within the Data Usage License associated with the data. These licenses comprise a set of verifiable credentials, each associated with an entity's DID, thereby implementing a secure and reliable authorization mechanism.
- **Secure Communication Pathways:** The DID Document also holds information about service endpoints, which assist in the establishment of secure communication channels between entities.
- **Data Integrity Validation:** The validation of data integrity can be achieved using the public key provided in the data owner's DID Document. The data, signed by the owner's private key, can be verified for its integrity.
- **Key Revocation and Recovery:** DIDs also allow for key revocation and recovery, ensuring that entities maintain control in cases of key theft or loss.

- **Interoperability and Portability:** The global uniqueness and high availability of DIDs support interoperability and portability, contributing to a seamless data sharing experience across different systems.
- **Privacy and Anonymity:** DIDs support pairwise pseudonymous identifiers, enabling entities to use distinct DIDs for different interactions, thereby preventing cross-interaction activity correlation.

In essence, DIDs serve as the bedrock of our proposed system. They provide a decentralized identity mechanism that enables secure and controlled data transfer, storage, and utilization. The effectiveness of our architecture's components is deeply reliant on the robustness of the DID-based identity mechanism, underlining the pivotal role of DIDs in the entire ecosystem.

2.6.1 Identity Implementation

The in Appendix A section 2.6.1 presents an implementation of a Decentralized Identifier (DID), a modern system for managing digital identities without the need for centralized authority. It provides an identity management scheme that is both highly robust and secure.

The code represents a Python class, 'IdentityIssuer', which utilises the 'indy' library, an open-source project that provides a toolkit for verifiable self-sovereign identity. This class is designed to operate on entities referred to as 'wallets', which are secure, private data structures where DIDs, associated metadata, and other relevant data are stored.

The IdentityIssuer class is initialised with a wallet_config and wallet_credentials. wallet_config is a dictionary containing configuration parameters for the wallet (e.g., id, storage type), while wallet_credentials holds credentials to open the wallet (e.g., key, key derivation method).

The class contains four methods: issue_user_did, issue_webapp_did, revoke_did, and resolve_did. The issue_user_did and issue_webapp_did methods are responsible for generating DIDs for users and web applications, respectively. They open the wallet, create a new DID, optionally store some metadata, then close the wallet, returning the DID and verkey. In the case of 'issue_user_did', it also issues a credential to the new DID, with the definition

stored in the wallet.

The revoke_did method allows for the revocation of a specific DID. This involves opening the wallet, revoking the credential on the ledger associated with the DID, and then closing the wallet.

Finally, the resolve_did method retrieves metadata associated with a particular DID from the wallet. This can be useful for validating the identity associated with a DID or checking the status of a DID.

In sum, this code illustrates a foundational component of decentralized identity systems. It provides a concrete mechanism for issuing, revoking, and resolving DIDs, laying the groundwork for more sophisticated operations in a self-sovereign identity system.

CONCLUSIONS

The purpose of this study was to explore a decentralized paradigm for data storage and processing, focusing on an architecture that prioritizes the user. By emphasizing the user's control and flexibility over their data, this study offers a path to a digital ecosystem where data is viewed not just as a commodity but as a personal asset that deserves respect and responsible management.

This study aims to design a decentralized data storage system where users control their data rather than apps or services. Such a design is ethical, separating control of the data from the application infrastructure and transferring the tools of control to the user.

Software architectures are a potential foundation for a new application generation that functions independently of data control. These applications access user data under strict conditions defined by the user through data use licenses (DULs). These licenses protect the user's data by providing a clear framework for using the data as the user desires.

Kademlia's distributed hash table (DHT) facilitates efficient data storage and retrieval in this decentralized system. Kademlia's DHT ensures reliable data distribution over an extensive network of nodes, increasing system resilience and ensuring uninterrupted operation during node failures or network outages.

Decentralized identifiers (DIDs) solve the digital identity verification problem. They are trusted, verifiable identifiers facilitating secure interactions, contributing to a more robust and convenient digital ecosystem.

Key findings of this study include:

- **Decentralization as a practical solution:**Decentralized systems provide privacy, security, and data control. Kademlia's use of DHTs and DIDs has proven the potential of decentralized systems to create reliable, high-performance solutions for data storage and verification.
- **User empowerment and data sovereignty:** The proposed DUL system architecture technically extends user empowerment and data sovereignty. This approach demonstrates the ability of technology to empower users digitally.

- **Ethical design paradigms:**The project demonstrated the application of ethical design principles in building application architecture, emphasizing that technology can be ethical, responsible, and user-centered.

In the future, this research could be applied to creating software where web-centric applications are not built around data models developed by developers but those semantic data models that the user has at their disposal to enhance data reuse when the third data output of a service is easily reused in another service based on the fact that this data potentially exists in the customer's data warehouse. So, a user's medical data can be effectively reused in another domain, regardless of its regional location or availability of API between services, but based only on the semantics of the data being available at all times in a distributed system.

Overall, this study looked at alternative approaches to replace traditional data management systems and highlighted the potential of user-centered ethical design principles in the digital ecosystem.

BIBLIOGRAPHY

1. Белоченко О.Є, Lightweight distributed data storage for web-oriented data centric apps, «Стан, Досягнення та Перспективи Інформаційних Систем і Технологій», Одеса, 2023, с. 84-85
2. “The social economy: Unlocking value and productivity through social technologies,” McKinsey & Company, Jul. 01, 2012. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-social-economy
3. “BAE Systems| Emerald Insight,” Oct. 01, 2003. https://www.emerald.com/insight/content/doi/10.1108/aeat.2003.12775eaf.006/full/html
4. L. Irwin, “Human Error is Responsible for 82% of Data Breaches - GRC eLearning Blog,” GRC eLearning Blog, Jul. 01, 2022. https://www.grcelearning.com/blog/human-error-is-responsible-for-85-of-data-breaches
5. S. Atske, “Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information,” Pew Research Center: Internet, Science & Tech, Nov. 15, 2019. https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/
6. J. Dunn, "Digital Clutter Can Be Overwhelming. Here’s How to Clean Up," New York Times, Feb. 24, 2023. https://www.nytimes.com/2023/02/24/well/live/digital-clutter.html
7. R. Browne, “Italy became the first Western country to ban ChatGPT. Here’s what other countries are doing,” CNBC, Apr. 04, 2023. https://www.cnbc.com/2023/04/04/italy-has-banned-chatgpt-heres-what-other-countries-are-doing.html
8. “ArtStation is hiding images that protest against AI art,” ArtStation is removing anti-AI protest artwork from its homepage - The Verge, Dec. 23, 2022. https://www.theverge.com/2022/12/23/23523864/artstation-removing-anti-ai-protest-artwork-censorship

9. "Artists Are Revolting Against AI Art on ArtStation," Artists Are Revolting Against AI Art on ArtStation. https://www.vice.com/en/article/ake9me/artists-are-revolt-against-ai-art-on-artstation
10. "GitHub Users Want to Sue Microsoft For Training an AI Tool With Their Code," GitHub Users Want to Sue Microsoft For Training an AI Tool With Their Code. https://www.vice.com/en/article/g5vmgw/github-users-want-to-sue-microsoft-for-training-an-ai-tool-with-their-code
11. "The lawsuit that could rewrite the rules of AI copyright," The lawsuit against Microsoft, GitHub and OpenAI that could change the rules of AI copyright - The Verge, Nov. 08, 2022. https://www.theverge.com/2022/11/8/23446821/microsoft-openai-github-copilot-class-action-lawsuit-ai-copyright-violation-training-data
12. T. Davis and S. Rajamanickam, "Ethical Concerns of Code Generation Through Artificial Intelligence," SIAM News. https://sinews.siam.org/Details-Page/ethical-concerns-of-code-generation-through-artificial-intelligence
13. "GitHub Copilot investigation Joseph Saveri Law Firm& Matthew Butterick," GitHub Copilot investigation Joseph Saveri Law Firm& Matthew Butterick. https://githubcopilotinvestigation.com/
14. "Understanding Personal Data Stores (PDS) - Mydex," Mydex. https://web.archive.org/web/20151001065833/https://mydex.org/understand-pds/
15. "IPFS Architecture," Launchpad. /curriculum/ipfs/subsystems-architecture/
16. "InterPlanetary File System - Wikipedia," InterPlanetary File System - Wikipedia, Apr. 01, 2022. https://en.wikipedia.org/wiki/InterPlanetary_File_System
17. "Bitswap | IPFS Docs," Bitswap | IPFS Docs. https://docs.ipfs.tech/concepts/bitswap/
18. "DIF Decentralized Web Node," DIF Decentralized Web Node. https://identity.foundation/decentralized-web-node/spec/
19. "Ind.ie — Ethical Design Manifesto," Ind.ie — Ethical Design Manifesto.

https://ind.ie/ethical-design/

20. “SurrealDB Documentation,” SurrealDB. https://surrealdb.com/docs/introduction/concepts
21. “DBOS: A Database-Oriented Operating System,” DBOS: A Database-Oriented Operating System. https://dbos-project.github.io/blog/intro-blog.html
22. “Splitting Data with Content-Defined Chunking,” Splitting Data with Content-Defined Chunking | Gopher Academy Blog, Dec. 01, 2018. https://blog.gopheracademy.com/advent-2018/split-data-with-cdc/
23. “DIF - Decentralized Identity Foundation,” DIF - Decentralized Identity Foundation. https://identity.foundation/
24. “ACM Digital Library,” ACM Digital Library. https://dl.acm.org/doi/fullHtml/10.1145/3427228.3427262
25. Y. Andrian, H. Kim, and H. Ju, “A Distributed File-Based Storage System for Improving High Availability of Space Weather Data,” MDPI, Nov. 21, 2019. https://www.mdpi.com/2076-3417/9/23/5024
26. “Distributed Hash Tables with Kademlia mdash; Stanford Code the Change Guides documentation,” Distributed Hash Tables with Kademlia; Stanford Code the Change Guides documentation. https://codethechange.stanford.edu/guides/guide_kademlia.html

APPENDICES

Appendix

```python
import json
from typing import Dict, Any, Optional, Tuple

from indy.anoncreds import issuer_create_and_store_credential_def, issuer_revoke_credential
import indy.wallet


class IdentityIssuer:
    def __init__(self, wallet_config: Dict[str, Any], wallet_credentials: Dict[str, Any]):
        self.wallet_config = wallet_config
        self.wallet_credentials = wallet_credentials

    async def issue_user_did(self, seed: Optional[str] = None) -> tuple[Any, Any, Any]:
        # Open the wallet
        wallet = await indy.wallet.open_wallet(**self.wallet_config, **self.wallet_credentials)

        # Create a new DID
        did_info = {'seed': seed} if seed else {}
        (did, verkey) = await indy.did.create_and_store_my_did(wallet, json.dumps(did_info))

        # Issue a credential to the DID
        schema = {
            'id': 'schema_id',
            'name': 'User Credential',
            'version': '1.0',
            'attributes': ['name', 'email'],
        }
        cred_def = await indy.anoncreds.issuer_create_and_store_credential_def(wallet, did, json.dumps(schema), 'CL', False)
```

```
29
30          # Close the wallet
31          await indy.wallet.close_wallet(wallet)
32
33          return did, verkey, cred_def
34
35      async def issue_webapp_did(self, webapp_name: str) -> tuple[Any,
        ↪  Any]:
36          # Open the wallet
37          wallet = await indy.wallet.open_wallet(**self.wallet_config,
            ↪  **self.wallet_credentials)
38
39          # Create a new DID
40          (did, verkey) = await indy.did.create_and_store_my_did(wallet,
            ↪  '{}')
41
42          # Write metadata to the ledger
43          metadata = {'webapp_name': webapp_name}
44          await indy.did.set_did_metadata(wallet, did,
            ↪  json.dumps(metadata))
45
46          # Close the wallet
47          await indy.wallet.close_wallet(wallet)
48
49          return did, verkey
50
51      async def revoke_did(self, did: str) -> None:
52          # Open the wallet
53          wallet = await indy.wallet.open_wallet(**self.wallet_config,
            ↪  **self.wallet_credentials)
54
55          # Revoke the DID on the ledger
56          await issuer_revoke_credential(wallet, did)
57
58          # Close the wallet
59          await indy.wallet.close_wallet(wallet)
60
61      async def resolve_did(self, did: str) -> Optional[Dict[str, Any]]:
62          # Open the wallet
63          wallet = await indy.wallet.open_wallet(**self.wallet_config,
            ↪  **self.wallet_credentials)
```

```
64
65          # Resolve the DID metadata from the ledger
66          metadata = await indy.did.get_did_metadata(wallet, did)
67
68          # Close the wallet
69          await indy.wallet.close_wallet(wallet)
70
71          if metadata:
72              return json.loads(metadata)
73          else:
74              return None
75
```