Hybrid Backup Service with Decentralized Sovereignty

Neil Roy, Kevin Lee, Edwin Medrano Villela, Awin Zhang, Suhrit Padakanti

Team Mysten

November 3rd, 2025 ~ PRD v1

Introduction

1.1 Problem Statement

Traditional cloud backup services often force users into a trade-off between usability, security, and long-term accessibility. Even when encrypted storage is offered, users remain dependent on a single centralized provider. This creates risks:

- **Lock-in:** Providers may shut down, increase prices, or alter policies (e.g., reduce encryption guarantees), leaving users with limited options.
- **Regulatory Pressure:** Centralized providers may be compelled to compromise security measures.
- **Data Loss:** If the central service fails, users may permanently lose access to backups.

Alternatively, decentralized storage using cryptographic protocols on a blockchain can be extremely difficult for a regular user with no background in computer science to understand.

1.2 Motivation and Innovation

Most existing backup solutions force users to choose between convenience and control. Traditional cloud services prioritize speed and usability, but often trap users in closed ecosystems. In contrast, decentralized storage platforms emphasize sovereignty at the expense of accessibility and ease of use. Our project eliminates this trade-off through a **hybrid architecture**, ensuring that users retain **full control** of their **encrypted data** while maintaining the simplicity and speed expected from modern backup services.

What makes our approach impactful:

- **Hybrid sovereignty model:** By layering Walrus as a decentralized backend, our system ensures that users retain full ownership and retrieval rights even if the centralized provider fails or changes its policies/pricing.
- **Decoupled control and infrastructure:** Encryption keys and user data remain separate. Users keep control of their encrypted backups, while infrastructure remains interchangeable. This separation enforces true data sovereignty rather than simply offering *encrypted storage*.
- **Operational simplicity for mainstream users:** Our caching and lifecycle-management layer hides blockchain complexity, handling WAL/SUI payments, renewal events, and bandwidth optimization. Allowing decentralized storage to become viable for everyday consumers, not just technically skilled users.
- **Sustainable, user-first ecosystem:** By standardizing how centralized services interact with decentralized backends, we enable future interoperability among backup providers, promoting a more open and competitive data-storage ecosystem.

In short, our **Hybrid Backup Service with Decentralized Sovereignty** delivers the best of both worlds: the seamless performance of centralized cloud backups and the autonomy of decentralized storage. It redefines what ‘ownership’ means in personal data management, offering **usability without compromise, resilience without complexity, and sovereignty without friction.**

1.3 Core Technical Advance

As stated before, the system implements a **hybrid storage architecture** that unites centralized performance with decentralized resilience. Its design ensures that users enjoy the speed and simplicity of a conventional cloud service while maintaining full control and recoverability of their encrypted data.

Centralized Layer

- **Fast-access caching:** A lightweight caching layer enables rapid file uploads and downloads, ensuring a seamless user experience. Frequently accessed data remains temporarily stored in this layer to minimize latency.
- **Efficient synchronization:** The centralized server manages all bandwidth-heavy communication with the Walrus network, insulating users from the complexity of blockchain transactions and high data amplification (up to 4–5x).
- **Payment and lifecycle automation:** The server abstracts Web3 interactions by automating WAL/SUI payments, renewal tracking, and blob expiration management. This allows users to interact with decentralized storage using familiar subscription or credit-card models.
- **Transparent encryption handling:** All files received by the server are already **AES-GCM-encrypted** on the client side. The centralized layer only handles ciphertext and metadata, ensuring **zero-knowledge operation**; the provider never has access to plaintext data or keys.

Decentralized Layer (Walrus)

- **Persistent, sovereign storage:** The Walrus network stores encrypted user data as **content-addressed blobs**, each verifiable and retrievable through a unique cryptographic identifier.
- **Independence and resilience:** Even if the centralized cache or payment service becomes unavailable, users can directly recover their encrypted data from Walrus using their local AES-GCM key.
- **Portability and trust minimization:** Because data persistence depends on the decentralized network rather than a single provider, users can migrate to other services or implement their own retrieval clients without re-uploading data.

Client-Side Encryption with AES-GCM

Every file is encrypted before transmission using **Advanced Encryption Standard (AES)** in **Galois/Counter Mode (GCM)**. This algorithm provides both **confidentiality** and **integrity** through authenticated encryption:

- **Confidentiality:** Only users who possess the encryption key can decrypt the file; the server and Walrus nodes see only ciphertext.
- **Integrity:** Each encrypted object includes an authentication tag that detects any modification to the ciphertext or metadata.
- **Zero-knowledge privacy:** Encryption and key management occur entirely on the client; no decryption capability exists on the server side.

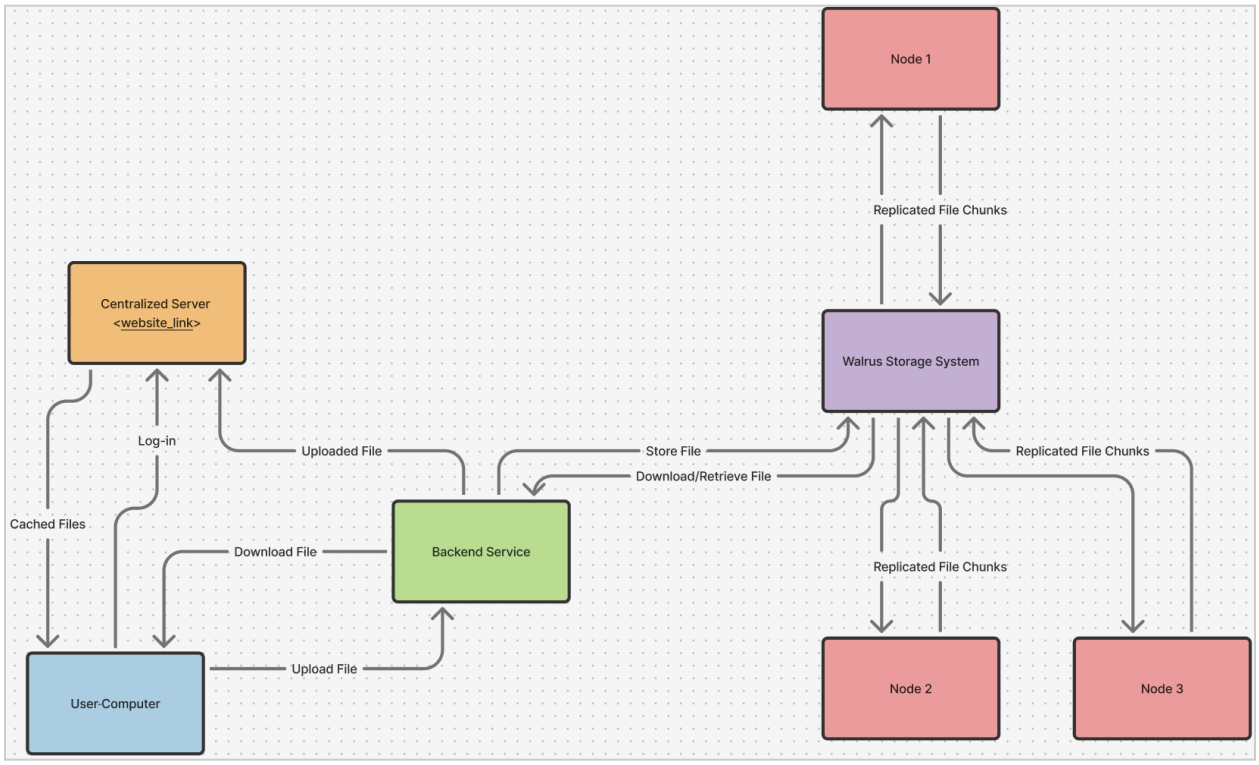
This cryptographic approach ensures that user sovereignty is preserved at a mathematical level—data remains secure even in untrusted environments.

System Advantages

- **Usability without compromise:** Users interact with a familiar web interface and instant caching layer while benefiting from end-to-end encryption and decentralized redundancy.
- **Resilience and direct recovery:** If the centralized cache fails, users can retrieve the same AES-GCM–encrypted blob directly from Walrus.
- **Security and compliance:** AES-GCM meets modern encryption standards (NIST SP 800-38D) and is widely used in secure communication protocols such as TLS 1.3, ensuring both strong protection and regulatory alignment.

System Architecture

2.1 High-Level System Diagram



- **User Computer:** Logs in, uploads encrypted files, and downloads cached or recovered files.
- **Centralized Server:** Takes care of user authentication, manages cached data for fast access.
- **Backend Service:** Processes file operations and communicates directly with Walrus for storage and retrieval.
- **Walrus Storage System (SUI Blockchain):** Stores encrypted file blobs across multiple decentralized nodes for redundancy.
- **Nodes (1–3):** Hold replicated file chunks to ensure fault tolerance and persistent availability. (In practice, there will be many more nodes)

Requirements Specification

3.1 Functional Requirements

- The system should encrypt files and decrypt them after download using a symmetric encryption scheme (e.g. AES-GCM).
- The system should handle lazy upload so the user perceives the upload as happening much faster than it actually is.
- The system must ensure that user files and wallet identifiers remain private and inaccessible to developers or third parties.
- The user should be able to upload their data as a blob and retrieve it with a linked decryption key.
- The user should be able to access their data even if the server fails as a property of decentralized storage systems.

3.2 Non-Functional Requirements

- The user should experience a smooth and responsive file upload experience, ensuring that users perceive upload times as comparable to standard cloud storage services (ie.y Google Cloud).
- Users should only be able to access data blobs if they have a linked decryption key to maintain privacy.
- Users should be able to upload large file sizes up to 5GB without experiencing large lag times or errors.
- Encrypted blobs should maintain 100% bit integrity(won't be corrupted).
- There shouldn't be any logging of user data or metadata beyond operational necessity.

3.3 User Stories

- As a user, I want to download my files securely even if the centralized system fails, so that my data remains accessible under all circumstances.
- As a user, I want to view my SUI and WAL balances so that I can confirm I have enough tokens to store new files.
- As a user, I want to view, manage, and delete my uploaded files from a single interface so that I can easily organize and control my stored data.
- As a user, I want my files to persist on Walrus so that I can still access them if the centralized cache or provider shuts down.
- As a user, I want to be able to upload multiple files at once, so I don't have to waste time uploading files one at a time
- As a user, I want to be able to keep my encryption keys in a safe and accessible location so that they don't get lost and I can easily use them to access data.
- As a user, I want to verify that my stored data has not been corrupted, so that I can trust the integrity and safety of my files.

- As a user, I want to access my data at any time, regardless of the state of the service provider, so that I retain full control over my information.
- As a user, I want to see clear, upfront costs for uploading and storing data, so that I can make informed decisions about how I use storage resources.
- As a user, I want to pay for storage using traditional payment methods (e.g., credit card) so that I don't need to manage cryptocurrency wallets or additional blockchain tools.

Appendix

4.1 Technologies

- Programming languages, frameworks, libraries
  - **Languages**: TypeScript, JavaScript, Python
  - **Frameworks**: React, Next.js, Vite
  - **Libraries**: Tailwind CSS, Axios, Crypto libraries
- APIs, SDKs, databases, and cloud services
  - **APIs:** Walrus API
  - **SDK:** Vercel SDK
- Version control, deployment, and CI/CD tools
  - **Version Control:** Git
  - **Deployment:** Vercel, Netlify
  - **CI/CD:** GitHub Actions
- Testing tools and methodologies
  - **Testing tools:** Jest

Glossary

- **AES-GCM:** Advanced Encryption Standard with Galois/Counter Mode. A symmetric encryption algorithm provides both confidentiality and data integrity.
- **Backend:** The server-side components responsible for logic, data management, and communication with decentralized networks.
- **Blob ID:** A unique identifier or hash assigned to each uploaded file to verify its authenticity and integrity.
- **Caching Layer:** A temporary storage component that stores frequently accessed data or recently uploaded/downloaded files to reduce latency and improve system responsiveness. Examples include in-memory caches such as Redis or local browser caches.
- **CLI:** Abbreviation for Command Line Interface
- **Decentralized Storage:** A storage model in which files are distributed across multiple nodes, eliminating single points of failure and enhancing data sovereignty.
- **Encryption:** The process of converting data into a coded format that can only be read or decrypted by authorized parties.

- **Frontend:** What our end-users interact with, and houses all the client-side operations.
- **Lazy Upload:** A technique where files are uploaded asynchronously, allowing users to continue interacting with the system before the upload fully completes.
- **Sui:** A Layer-1 blockchain designed for fast, secure, and scalable transactions, used in this system for managing wallet balances and transaction validation.
- **Symmetric Encryption:** A method of encryption that uses the same key for both encryption and decryption operations.
- **Wallet:** A digital account or cryptographic key pair used to manage tokens or digital assets on the Sui blockchain.
- **Walrus:** A decentralized data storage framework built by Mysten Labs that provides secure and distributed file persistence.