Bedrock - Strategic Expertise Dossier

Dorian MOY - dorian.moy@epitech.eu
Florian LAUCH - florian.lauch@epitech.eu
Pablo LEVY - pablo1.levy@epitech.eu
Reza RAHEMTOLA - reza.rahemtola@epitech.eu

Contents

1. Expertise topic & context

A. Privacy-by-Design and Client-Side Encryption Architectures for Decentralized Storage Systems

This expertise topic focuses on how privacy-by-design principles are implemented in decentralized drive solutions, with a particular emphasis on client-side encryption, key ownership, and data confidentiality guarantees.

B. Project Context

Our product is a decentralized drive built on top of the Aleph network, leveraging IPFS for file storage. Files are encrypted on the client side using a private key that is never shared with the app, the network or any third party thanks to the use of a wallet provider, allowing connections with wallet solutions (Ledger, Metamask, Rabby) or an email solution. As a result, only the end user is able to access and decrypt their data.

Like in a classic drive solution, the user is able to upload its files, manage them like in a regular filesystem (duplicate, move, copy, rename, delete) with folder management. The user is also able to share files using a contact system where users can be identified by a username, as well as sharing them with a private LLM model.

If our project stops, the end user is still able to get his files back using the Aleph network, and decrypt them manually.

C. Stakes

Guaranteeing data confidentiality in a decentralized environment

- How can the user be certain that no one can access their files on a public network?

Preventing unauthorized access, including from infrastructure providers

- How can the user be certain that their data cannot be intercepted or accessed, through traffic or by platform administrators?

Balancing security, performance, and usability

- Is security at the cost of the regular & user-friendly features an online drive can offer?

Ensuring long-term trust in the system

- Is the user data dependent on the Bedrock platform? Can it be accessed again if the project is taken down?

2. Why this topic was chosen

Privacy-by-design is the core value proposition of Bedrock and a critical differentiator in the decentralized storage market.

From a technical perspective, incorrect architectural decisions in encryption or key management can lead to irreversible security vulnerabilities, or content loss as the user wouldn’t be able to get the unencrypted files back.

From a market perspective, zero-knowledge storage and user-owned data are becoming increasingly relevant due to growing concerns around surveillance, data leaks, regulatory pressure, companies' privacy policies and AI training. A survey led by Proton in June 2024 revealed that 80% of the votants don’t want their private data to be used to train AI models, and 60% don’t want their data to be shared with AI at all. One year earlier, a study from Proton and the UK governement highlighted the willingness of the population to be more cautious about their online privacy. Over half of people (52%) disagree that “free” online services monetizing their data is an ethical business model.

The primary objective of the project is therefore to provide a decentralized drive solution that guarantees strong privacy and data ownership to its users. Unlike traditional cloud storage systems, the project explicitly aims to ensure that no infrastructure component, including storage nodes or network operators would be able to access user data.

The topic of privacy-by-design and client-side encryption architectures directly supports this objective, as it defines how privacy guarantees are enforced at a technical level rather than through policy or trust assumptions.

Choosing this topic allows the project to move from a high-level promise (“your data is private”) to a verifiable and technically grounded architecture.

---

3. Research scope & methodology

A. Sources Considered

The research was conducted using multiple types of sources:

- Official documentation
  - Aleph Cloud
  - LibertAI
  - IPFS

- Technical articles and white papers
  - Article from a former Ledger developer
  - Aleph Cloud whitepaper
  - EIP-4337 whitepaper

- Developer discussions and feedback
  - Reddit discussions about a use case of aleph.im
  - Private conversations & meetings between the Bedrock project members and the developers from Aleph Cloud in order to get feedbacks

- Conference talks or videos
  - Introduction of aleph.im by its CEO in a talk session

B. Research Questions

- How do decentralized storage systems enforce privacy-by-design?
- What encryption models are used, and what are their trade-offs?
- How is key ownership and recovery handled?
- What metadata remains visible to the network?

---

4. Compared approaches

The following approaches were analyzed and compared:

1. Aleph-based decentralized storage with client-side encryption
2. Centralized storage with client-side end-to-end encryption (Proton Drive)
3. Centralized storage with provider-managed encryption (Google Drive)

Each approach was evaluated based on security, scalability, developer experience, and privacy guarantees.

A. Approach 1 — Aleph-based Decentralized Storage (Client-Side Encryption)

Aleph provides a decentralized storage and compute layer that can be used to build privacy-preserving applications. In the context of this project, files are encrypted on the client side before being uploaded to the Aleph network. The Aleph network itself is only a cloud redistribution solution, but allows any application builder to add sufficient security levels to reach zero-knowledge storage.

Key characteristics

- Encryption implemented at the application layer
- Strong separation between encrypted data and infrastructure
- Designed to integrate storage, compute, and indexing

Privacy implications

- Zero-knowledge storage model at the application level
- Infrastructure providers cannot access file contents (if encrypted)
- Privacy guarantees depend on correct client-side implementation

B. Approach 2 — Proton Drive (Centralized, Client-Side E2EE)

Proton Drive is a centralized cloud storage solution designed with privacy as a primary goal. Unlike traditional cloud providers, Proton Drive implements **client-side end-to-end encryption**, ensuring that file contents are encrypted before leaving the user's device.

Encryption keys are derived from the user's credentials and are not accessible to Proton's servers, resulting in a zero-access model for file contents. However, the underlying storage infrastructure remains centralized and operated by a single provider: in case of server failure, or service outage in Proton services, the files of the user can be lost forever without any way to recover them.

Key characteristics

- Client-side encryption applied by default
- User-controlled encryption keys (derived from password)
- Centralized storage and metadata management
- Strong focus on privacy and regulatory compliance

Privacy implications

- File contents are not accessible to the provider
- Metadata (timestamps, file structure, access patterns) may still be visible
- Users must trust the provider for availability, integrity, and correct implementation

C. Approach 3 — Centralized Storage with provider-managed encryption (Google Drive)

Google Drive represents a centralized cloud storage system with strong infrastructure scalability and mature developer tooling. While encryption is used in transit and at rest, true end-to-end encryption is limited or optional, and key management is generally controlled by the provider.

Key characteristics

- Centralized infrastructure
- Provider-controlled encryption and key management
- High scalability and usability

Privacy implications

- Provider has potential access to data and metadata
- Trust is placed in the service operator
- Limited alignment with zero-knowledge or self-custodial models

5. Comparative benchmark

The benchmark was built using criteria that directly impact the feasibility and credibility of a privacy-by-design decentralized drive:

- **Security model and trust assumptions**
- **Key management and user ownership**
- **Metadata exposure**
- **Performance overhead**
- **Scalability and resilience**
- **Developer ergonomics**
- **Costs**

These criteria were selected to balance cryptographic guarantees, architectural trade-offs, and real-world usability.

A. Detailed Analysis by Criterion

Security Model & Trust Assumptions

- **Aleph**
  Security relies on decentralization given by Aleph & IPFS, ensuring no single entity has full control over data access or availability. Trust is minimized but shifted toward correct client wallet custody.

- **Proton Drive**
  Strong cryptographic guarantees are provided through client-side encryption. However, users must trust Proton as a centralized operator for availability, metadata handling, and correct enforcement of the zero-access model.

- **Google Drive**
  Security is primarily policy and provider-based. While encryption is used, the provider ultimately controls key management and access, requiring full trust in the operator.

Key Management & User Ownership

- **Aleph**
  The main keys are generated and stored through the wallet solution the user opted in. The user has exclusive ownership, but this also introduces the risk of irreversible data loss in case of key loss. No built-in recovery mechanism can be enforced, as it would counter the privacy-by-design approach.

- **Proton Drive**
  Keys are derived from user credentials. This allows password-based access and some recovery mechanisms but introduces dependence on authentication systems and password security.

- **Google Drive**
  Keys are managed by the provider, allowing easy recovery but eliminating true user ownership.

Metadata Exposure

- **Aleph**
  While file contents and metadata are encrypted, certain metadata (creation and edition timestamps) as well as the encrypted version of the file contents and metadata will remain visible at the network level.

- **Proton Drive**
  Metadata is managed centrally and visible to the provider. File contents remain private, but structure and access patterns may be exposed.

- **Google Drive**
  Both metadata and usage patterns are fully visible to the provider and may be used for optimization or analytics.

Performance Overhead

- **Aleph**
  Client-side encryption introduces CPU overhead on upload/download, and decentralized retrieval can increase latency compared to centralized services.

- **Proton Drive**
  Client-side encryption adds overhead, but centralized infrastructure mitigates latency through optimized data centers.

- **Google Drive**
  Performance is highly optimized, with minimal overhead from encryption and very low latency.

Scalability & Resilience

- **Aleph**
  Scalability depends on network participation and replication strategies. Resilience is achieved through decentralization but may vary depending on network conditions.

- **Proton Drive**
  High scalability backed by centralized infrastructure, but resilience is tied to a single provider.

- **Google Drive**
  Extremely high scalability and resilience due to global infrastructure and redundancy.

Developer Ergonomics

- **Aleph**
  Requires developers to manage encryption, key handling, and integration with Aleph APIs. Offers flexibility but increases complexity.

- **Proton Drive**
  Limited programmability and integration options. Primarily designed for end users

rather than custom application development.

- **Google Drive**
  Extensive APIs, SDKs, documentation, and tooling make integration straightforward.

Costs

- **Aleph**
  Costs depend on decentralized network pricing, storage usage, and potential token-based economics.

- **Proton Drive**
  Subscription-based pricing with predictable costs.

- **Google Drive**
  Tiered subscription pricing with economies of scale.

B. Benchmark Summary

Criterion	Aleph (Decentralized)	Proton Drive (Centralized E2EE)	Google Drive (Centralized)
Key ownership	User-only (wallet solution or locally-generated private key, never shared)	User (password-derived keys, provider has no content access)	Provider-managed
Metadata exposure	Network-visible partial metadata (encrypted file, timing)	Provider-visible metadata	Provider-visible metadata
Performance	Client-side encryption overhead + network latency	Client-side encryption overhead, optimized infra	Minimal overhead

Developer UX	Requires custom crypto & Aleph integration	Limited programmability	Mature APIs & SDKs
Trust assumptions	Minimal trust in infrastructure	Trust in single provider	Full trust in provider

C. Interim Conclusion

This benchmark shows that while centralized solutions (Google Drive, Proton Drive) offer strong usability and scalability, they rely on varying degrees of trust in a single provider. Proton Drive significantly improves privacy through client-side encryption but remains centralized.

The Aleph-based approach trades some usability and performance for stronger trust minimization and architectural privacy guarantees, aligning more closely with the project's privacy-by-design goals.

6. Experiments & observations

Several practical tests were conducted to evaluate the real-world impact of the selected architecture:

- Client-side encryption performance during upload
- Decryption latency during download
- Impact of file chunking on performance

A. Objectives of the Experiments

The goal of these experiments was not to benchmark raw performance exhaustively, but to **observe the concrete impact of privacy-by-design choices** on the system.

In particular, the experiments aimed to:

- Evaluate the overhead introduced by client-side encryption
- Observe performance differences between centralized and decentralized storage
- Identify practical trade-offs affecting user experience and developer decisions
- Validate assumptions made during the comparative benchmark

B. Experimental Setup

The experiments were conducted using a controlled test setup with the following characteristics:

- **Client environment:**
  Macbook Pro M1, running MacOS Sonoma, Wi-Fi 6 with 300 Mbps connection

- **Test data:**
  Files of varying sizes (e.g. small text files, medium documents, large binary files)
  Sizes tested: 10 Ko, 1 Mo, 100 Mo

- **Compared systems:**

    - Aleph-based storage with client-side encryption
    - Proton Drive (client-side E2EE)
    - Google Drive (baseline)

- **Metrics observed:**
    - Upload & download time
    - Encryption and decryption duration
    - CPU usage during encryption
    - Error or failure rates (if applicable)

C. Experiment 1 — Client-Side Encryption Overhead

Description
This experiment measured the time and CPU overhead introduced by encrypting files on the client before upload.

Methodology

1. Encrypt a file locally using AES-256-GCM
2. Measure encryption time and CPU usage.
3. Upload encrypted file to storage backend.
4. Repeat for different file sizes.

Observations

- Encryption time scales approximately linearly with file size.
- For small files, encryption overhead is negligible compared to network latency.
- For large files, encryption becomes a noticeable part of total upload time.

Test Results Summary

File Size	Encryption Time (AES-256-GCM)	CPU Usage (Peak)	Total Upload Time (Encrypted)	% Time Spent Encrypting
10 KB	~0.4 ms	<1%	~0.4 s (network-dominated)	<1%
1 MB	~4 ms	~3%	~1 s	<1%

100 MB	~400 ms	15–30%	~6 s	~6.7%

Interpretation
Client-side encryption introduces predictable and manageable overhead, confirming that privacy-by-design is feasible without prohibitive performance costs for typical usage.

D. Experiment 2 — Upload & Download Latency Comparison

This experiment compared end-to-end upload and download times across the three storage approaches.

Methodology

- Upload encrypted files to each system
- Download and decrypt files
- Measure total elapsed time

Observations

- Google Drive consistently showed the lowest latency due to optimized centralized infrastructure.
- Proton Drive exhibited slightly higher latency due to encryption but remained stable.
- Aleph-based storage showed higher variance in latency, likely due to network distribution and node availability.

Test Results Summary (Average of 5 Runs)

Upload Time

File Size	Google Drive	Proton Drive	Aleph-based Storage
10 KB	0.10 s	0.14 s	0.30–0.60 s

1 MB	0.25 s	0.32 s	0.6–1.2 s
100 MB	2.9 s	3.4 s	4.5–9.0 s

Download + Decryption Time

File Size	Google Drive	Proton Drive	Aleph-based Storage
10 KB	0.09 s	0.12 s	0.25–0.50 s
1 MB	0.22 s	0.30 s	0.55–1.0 s
100 MB	2.7 s	3.2 s	4.2–8.5 s

Notes
Due to the way Google and Proton manage their servers, the uploads & downloads don't start immediately after taking action, so we only measured time when the action really started (using the Network tab as well as having a visual feedback or a confirmation the action was started)

Interpretation
Centralized infrastructure provides predictable performance, while decentralized storage trades some latency stability for reduced trust assumptions.

E. Experiment 3 — Metadata Visibility Analysis

This experiment aimed to identify which metadata elements remain visible to storage providers or the network despite encryption.

Methodology

- Inspect network requests and stored objects
- Identify visible attributes (file size, timestamps, identifiers)
- Compare across systems

Observations

- File contents remained encrypted in all client-side encryption setups.
- File size and upload timing were observable in all systems.
- Aleph-based storage exposed metadata at the network level rather than to a single provider.

Metadata Element	Google Drive	Proton Drive	Aleph-based Storage
File size	Visible to provider	Visible to provider	Encrypted file size visible to network nodes
Upload timestamp	Visible	Visible	Visible
File identifier	Provider-managed ID	Provider-managed ID	Encrypted version of the identifier visible to network nodes
User IP address	Visible to provider	Visible to provider	Visible to network nodes

Interpretation
Encryption alone does not eliminate metadata leakage. Decentralization changes *who* can observe metadata but does not fully remove exposure. Elements such as the upload timestamp are impossible to hide

F. Experiment 4 — Failure and Recovery Scenarios

This experiment explored how each system behaves under partial failure or key loss scenarios.

Observations

- Loss of encryption keys in the Aleph-based system resulted in irreversible data loss.
- Proton Drive allowed account-based recovery mechanisms, with implications for trust.
- Google Drive enabled full recovery via provider-controlled authentication.

Scenario	Aleph-based Storage	Proton Drive	Google Drive
Lost encryption key	Data permanently unrecoverable	Account recovery possible	Account recovery possible
Node unavailability	Temporary delays, eventual retrieval via replication	Not applicable	Not applicable
Account compromise	No central account to hijack	Account takeover possible	Account takeover possible

Interpretation
There is a direct trade-off between user sovereignty and recoverability. Strong privacy guarantees increase the importance of user education and key management strategies.

G. Summary of Findings

Across all experiments, the following conclusions were observed:

- Client-side encryption is technically viable and introduces acceptable overhead.
- Centralized solutions offer superior performance consistency.
- Decentralized storage aligns better with strict privacy-by-design goals.
- Key management remains a critical usability and security challenge.
- Metadata exposure persists across all models and must be explicitly addressed.

---

7. Technical Recommendations

Based on the research and benchmark:

- Client-side encryption is recommended to achieve a complete privacy-by-design solution.
- Key ownership must remain exclusively on the client. We can provide third-party solutions allowing registration using an email, but the selection of the provider must remain a user decision.
- Metadata exposure must be minimized, both by encrypting it as much as possible, but also by making changes unpredictable.

A. Client-Side Encryption

Client-side encryption is strongly recommended as a core design principle. Encrypting data before it leaves the client ensures that:

- Storage providers and network nodes never access plaintext data
  Confidentiality does not rely on infrastructure trust
- Privacy guarantees remain valid even in case of node compromise or malicious operators

Experimental results show that the performance overhead introduced by client-side encryption is predictable and acceptable for typical file sizes, making it a viable default approach.

B. Key Ownership and Control

Encryption keys must remain exclusively under client control. Exclusive client-side key ownership ensures:

- True data sovereignty for end users
- No implicit trust in service operators or identity providers
- Alignment with strict privacy-by-design and zero-knowledge principles

While this approach removes recovery options provided by centralized services, it enforces a clear security boundary: loss of access is a user responsibility rather than a provider-controlled process.

C. Metadata Exposure Mitigation

Metadata exposure should be minimized wherever technically feasible.

Proposed Strategies

- Encrypt file names and directory structures
- Use content-addressed identifiers instead of user-defined identifiers
- Batch uploads or add timing obfuscation where possible
- Avoid storing user-identifying metadata at the protocol level

Although some metadata (e.g. access timing) remains observable in most storage systems, reducing unnecessary metadata leakage significantly improves the overall privacy posture.

D. Rejected or Deprioritized Approaches

Centralized End-to-End Encrypted Storage
While solutions such as Google Drive or Proton Drive provide strong usability and recovery mechanisms, they were deprioritized due to:

- Centralized trust assumptions
- Provider-controlled authentication and account recovery
- Potential exposure to legal or operational access requests

Protocol-Level Encryption Only
Relying solely on protocol-level encryption without client-side guarantees was rejected because:

- It does not protect against malicious or compromised storage operators
- It shifts trust away from users toward infrastructure providers

Hybrid Key Escrow Models
Key escrow or partial recovery mechanisms were excluded due to:

- Increased attack surface
- Ambiguous trust boundaries
- Incompatibility with strict user-sovereignty goals

E. Final Architectural Direction

The chosen architecture prioritizes:

- User-controlled encryption
- Decentralized storage infrastructure
- Minimal trust assumptions
- Explicit trade-offs between usability and privacy

This direction aligns with the core objectives of the project and differentiates it from mainstream cloud storage solutions by making privacy a foundational constraint rather than an optional feature.

---

8. Application to the Project

The following changes were applied to the project as a direct result of this expertise work:

1. Usage of the Aleph Cloud network to decentralize storage and guarantee user sovereignty
2. Implementation of a double encryption system for secure file sharing
3. Addition of a dedicated metadata encryption layer to limit metadata visibility

These decisions were justified based on the comparative research and observed results.

8.1 Decentralization & User Sovereignty

To align the system with strong privacy-by-design principles, we migrated from a centralized backend prototype to a decentralized storage infrastructure built on Aleph Cloud. Files are encrypted client-side and then stored on the decentralized network, ensuring that no single infrastructure provider can access user data in plaintext form.

Before the implementation of the Aleph Network, the project relied on a centralized backend architecture. This model provided predictable stability, low latency, and full control over infrastructure behavior. However, it implicitly required users to trust the project's servers for data availability and integrity. If the platform were to shut down, users would lose practical access to their stored files.

After, storage is handled by a decentralized network. This introduces slightly higher latency and variability compared to centralized cloud systems, as availability depends on distributed nodes rather than a single optimized provider. However, it guarantees user sovereignty: encrypted files remain accessible independently of Bedrock's operational status. Even if the project ceases to exist, users can still retrieve their encrypted data directly from the network and decrypt it with their own keys.

In summary, we lose some infrastructure stability, but we guarantee user sovereignty and significantly reduce trust assumptions.

8.2 Secure File Sharing

To allow secure sharing while preserving zero-knowledge guarantees, a double encryption architecture was introduced. Files are encrypted using a symmetric content key, and this content key is itself encrypted using the owner’s public key. When a file is shared, the content key is additionally encrypted with the recipient’s public key, without modifying the encrypted file itself.

Before, encryption was strictly single-user oriented. A file encrypted by one user could only be decrypted by that same user. Sharing would have required revealing private key material or re-encrypting the entire file separately for each recipient, which would be inefficient and insecure.

After, files can be securely shared without exposing the original encryption key and without duplicating encrypted payloads. The data stored on the decentralized network remains unreadable to the public, and only explicitly authorized recipients can decrypt the file. The blockchain or storage layer may expose encrypted objects, but these objects are cryptographically meaningless without the correct private key.

This double encryption mechanism therefore enables secure file sharing while ensuring that publicly exposed encrypted data remains accessible only to intended recipients.

8.3 Encrypted Metadata Layer

Research demonstrated that encrypting file contents alone is insufficient to guarantee strong privacy. Metadata such as file names, folder structures, and timestamps can reveal sensitive information about user behavior and activity patterns.

To address this issue, we introduced an additional encryption layer specifically dedicated to metadata. File names, directory structures, and internal references are encrypted before being stored on the decentralized network.

Before, metadata elements could potentially be interpreted or analyzed, allowing indirect inference of user activity. Even if file contents were encrypted, observers might still deduce usage patterns, relationships between files, or organizational structures.

After, metadata stored on the network is unreadable without the appropriate decryption keys. Observers may still see that data exists and that transactions occur, but they cannot meaningfully interpret file names, folder hierarchies, or structural relationships. This significantly reduces the risk of activity logging or behavioral profiling based solely on metadata analysis.

By encrypting metadata, we deny meaningful visibility into user activity and reinforce the overall privacy guarantees of the system.

9. Limits & Future Work

Identified Limitations

Despite the depth of the conducted research and experimentation, several aspects remain outside the scope of this study.

Key Recovery Mechanisms
This work does not fully address secure key recovery solutions. While exclusive client-side key ownership provides strong privacy guarantees, it also introduces the risk of irreversible data loss. Designing recovery mechanisms that do not weaken the trust model remains an open challenge.

User Error Scenarios
The study does not extensively evaluate real-world user error cases such as lost devices, forgotten credentials, or accidental key deletion. These scenarios are critical for usability but difficult to reconcile with strict privacy-by-design constraints.

Long-Term Cryptographic Agility
The research assumes the continued security of currently used cryptographic primitives. The impact of future cryptographic breaks, algorithm deprecation, or post-quantum requirements is not fully explored.

Operational and Legal Considerations
Regulatory, legal, and jurisdictional implications of decentralized storage systems were not deeply analyzed, despite their potential impact on deployment and adoption.

Future Work and Research Directions

Future work could extend this study in several directions:

- Exploration of privacy-preserving key recovery approaches (e.g. social recovery, threshold cryptography, or user-managed backups)
- Usability studies focusing on key management and user education
- Evaluation of post-quantum cryptographic algorithms for client-side encryption
- Long-term performance evaluation under real user load
- Formal threat modeling and security audits of the implemented architecture

---

Closing Perspective

The limitations identified in this study highlight an essential reality: strong privacy guarantees require explicit trade-offs. Rather than eliminating these trade-offs, future work should aim to make them clearer, safer, and more manageable for end users, without compromising the core principles of decentralization and user sovereignty.
