Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 13 June 2026

M.J. Arcan
Arcan Consulting

December 2025

The MyClerk Virtual File System: Distributed Storage for Family Networks
draft-myclerk-vfs-00

Abstract

This document specifies the MyClerk Virtual File System (VFS), a distributed storage system designed for family networks. The VFS provides end-to-end encrypted, redundant storage across heterogeneous nodes using adaptive chunking and Reed-Solomon erasure coding [REED-SOLOMON]. Metadata synchronization uses Conflict-free Replicated Data Types (CRDTs) [CRDT] with a hybrid LWW-Register and OR-Set approach.

The system supports tiered storage, automatic health monitoring with self-healing capabilities, and federation between separate family installations. It is designed to be accessible to non-technical users while providing enterprise-grade data durability.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 June 2026.

Copyright Notice

Table of Contents

1. Introduction

Modern families have storage devices distributed across multiple locations: a primary home with NAS, a vacation house with a mini-PC, grandparents with a Raspberry Pi, and mobile devices requiring remote access. The MyClerk VFS unifies these resources into a coherent, redundant file system.

The VFS operates on top of the MyClerk Protocol (draft-myclerk-protocol) for all communication. This document specifies the storage layer, including data organization, redundancy mechanisms, and metadata synchronization.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Terminology

Chunk A fixed-size segment of file data, typically 4-16 MB, that forms the unit of storage, encryption, and distribution.

Fragment One piece of an erasure-coded chunk. A chunk with k=3, m=2 produces 5 fragments.

Core Node A storage node with high availability (>95% uptime over 30 days) that participates in guaranteed fragment storage.

Bonus Node A storage node with variable availability that provides additional redundancy when online.

CRDT Conflict-free Replicated Data Type. A data structure that can be replicated across multiple nodes and merged without conflicts.

LWW-Register Last-Writer-Wins Register. A CRDT where concurrent writes are resolved by timestamp.

OR-Set Observed-Remove Set. A CRDT that supports add and remove operations without conflicts.

2. Architecture Overview

The VFS consists of four layers:

1. *Virtual Namespace:* CRDT-based directory structure visible to users.
2. *Content-Addressed Chunk Store:* Encryption, chunking, and hashing layer.
3. *Distributed Fragment Storage:* Erasure coding and distribution across nodes.
4. *Federation Layer:* Sharing between separate VFS installations.

2.1. Design Principles

* *Zero Data Loss:* Erasure coding ensures data survives multiple node failures.
* *End-to-End Encryption:* Only authorized users can read data; storage nodes see only ciphertext.
* *Self-Healing:* Automatic fragment redistribution when nodes fail.
* *User-Friendly:* Complex internals hidden behind simple folder interface.

3. Adaptive Chunking

The VFS uses adaptive chunk sizes based on file size to optimize storage efficiency:

File Size	Chunk Size	Rationale
< 1 MB	No chunking	Avoid overhead
1-100 MB	4 MB	Good balance
> 100 MB	16 MB	Fewer fragments

Table 1: Adaptive Chunk Sizes

Files smaller than 1 MB are stored as a single unit. This avoids the overhead of chunk management for small configuration files and documents.

4. Erasure Coding

The VFS uses Reed-Solomon erasure coding to provide redundancy. Each chunk is encoded into k data fragments plus m parity fragments. Any k fragments are sufficient to reconstruct the original chunk.

4.1. Erasure Coding Profiles

Profile	k	m	Overhead	Tolerance	Use Case
Economy	4	1	25%	1 node	Non-critical data
Standard	3	2	67%	2 nodes	Recommended default
Critical	4	4	100%	4 nodes	Important documents
Paranoid	4	5+	>125%	5+ nodes	Maximum security

Table 2: Erasure Coding Profiles

The Standard profile with k=3, m=2 is RECOMMENDED for most use cases. It provides tolerance for two simultaneous node failures with reasonable storage overhead.

4.2. Fragment Distribution

Fragments MUST be distributed across different nodes to maximize fault tolerance. The distribution algorithm SHOULD:

* Place at least k fragments on Core Nodes
* Distribute fragments across different physical locations when possible
* Use Bonus Nodes for additional copies
* Avoid placing multiple fragments on the same node

5. Fragment Health Monitor

The VFS continuously monitors fragment availability and takes corrective action when necessary.

5.1. Health Levels

Status	Condition	Wait Time	Action
GREEN	>= k+2 online	-	No action
YELLOW	k+1 online	6 hours	Warning, then redistribute
ORANGE	k online	30 minutes	Immediate redistribution
RED	< k online	0	Emergency recovery

Table 3: Fragment Health Status

5.2. Node Failure Escalation

When a node becomes unreachable, the following escalation timeline applies:

0-30 minutes Ping retry with exponential backoff (10s, 30s, 90s). No status change or alerts.

30 minutes - 1 hour Alert sent to family administrator: "Node XY unreachable".

1-6 hours YELLOW status. System evaluates redistribution options and checks capacity on other nodes.

6-24 hours Redistribution begins. Fragments are copied to available nodes. Priority: Critical > Standard > Economy. Bandwidth limit: 50% of available capacity.

>72 hours Node downgrade: Core Node becomes Bonus Node. Fragments are no longer primarily placed on this node.

6. CRDT-Based Metadata

The VFS uses Conflict-free Replicated Data Types (CRDTs) for metadata synchronization across nodes. This ensures eventual consistency without coordination.

6.1. Hybrid CRDT Approach

The VFS employs a hybrid CRDT strategy:

* *LWW-Register:* Used for file content references, file attributes (size, mtime), and single-value fields.

* *OR-Set:* Used for directory entries, tag lists, and permission sets.

This combination provides optimal semantics for file system operations: LWW-Register ensures the latest file version wins, while OR-Set correctly handles concurrent file creation and deletion in directories.

6.2. LWW-Register Specification

The following CDDL [RFC8610] schema defines the LWW-Register structure. Values are encoded using CBOR [RFC8949].

```
lww-register<T> = {
    value: T,
    timestamp: uint,          ; Unix microseconds
    node-id: bstr .size 16,   ; Writer node ID
}

; Merge rule: Higher timestamp wins
; Tie-breaker: Lexicographically higher node-id wins
```

6.3. OR-Set Specification

```
or-set<T> = {
    elements: [* or-set-element<T>],
}

or-set-element<T> = {
    value: T,
    add-id: bstr .size 16,    ; Unique ID for this add operation
    removed: bool,            ; Tombstone flag
}

; Add: Create new element with unique add-id
; Remove: Set removed=true for all elements with matching value
; Merge: Union of all elements, removed flags propagate
```

6.4. File Metadata Structure

```
file-metadata = {
    id: bstr .size 16,                    ; Unique file ID
    name: lww-register<tstr>,             ; File name
    parent: lww-register<bstr .size 16>,  ; Parent directory ID
    content-hash: lww-register<bstr .size 32>, ; BLAKE3 hash
    size: lww-register<uint>,             ; File size in bytes
    mtime: lww-register<uint>,            ; Modification time
    chunks: lww-register<[* chunk-ref]>,  ; Chunk references
    tags: or-set<tstr>,                   ; User tags
    permissions: or-set<permission>,      ; Access permissions
}

chunk-ref = {
    hash: bstr .size 32,      ; BLAKE3 hash of chunk
    offset: uint,             ; Offset in file
    size: uint,               ; Chunk size
    ec-profile: tstr,         ; Erasure coding profile
}

permission = {
    principal: bstr .size 16, ; User or group ID
    access: uint,             ; Access flags (read, write, etc.)
}
```

6.5. Directory Metadata Structure

```
directory-metadata = {
    id: bstr .size 16,                    ; Unique directory ID
    name: lww-register<tstr>,             ; Directory name
    parent: lww-register<bstr .size 16>,  ; Parent directory ID
    children: or-set<bstr .size 16>,      ; Child file/directory IDs
    mtime: lww-register<uint>,            ; Modification time
    permissions: or-set<permission>,      ; Access permissions
}
```

7. Encryption

All data is encrypted end-to-end using ChaCha20-Poly1305 [RFC8439]. Storage nodes never see plaintext.

7.1. Key Hierarchy

Keys are derived hierarchically using HKDF [RFC5869]:

```
Family Master Key (FMK)
    |
    +-- Folder Key: /family/photos/
    |       |
    |       +-- File Key: photo.jpg
    |               |
    |               +-- Chunk Key: a1b2c3...
    |
    +-- Folder Key: /family/documents/
    |
    +-- Session Keys (for Federation)
```

7.2. Key Derivation

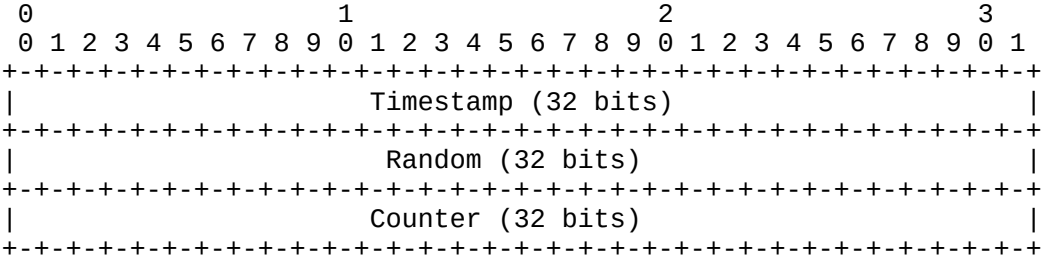
```
folder_key = HKDF-SHA256(
    IKM  = family_master_key,
    salt = folder_id,
    info = "myclerk-vfs-folder-v0",
    L    = 32
)

file_key = HKDF-SHA256(
    IKM  = folder_key,
    salt = file_id,
    info = "myclerk-vfs-file-v0",
    L    = 32
)

chunk_key = HKDF-SHA256(
    IKM  = file_key,
    salt = chunk_index (4 bytes, big-endian),
    info = "myclerk-vfs-chunk-v0",
    L    = 32
)
```

7.3. Nonce Construction

Each encryption operation requires a unique 96-bit nonce:



This construction provides collision probability less than 2^-80 per year at 1 million operations per second.

7.4. Content Hashing

All content is hashed using BLAKE3 [BLAKE3] for content addressing:

* Chunks: BLAKE3 hash of plaintext content

* Files: Merkle tree of chunk hashes

8. Node Classification

Nodes are classified based on their reliability:

Type	Criteria	Role
Core Node	>95% uptime (30 days), <100ms latency	Guaranteed fragment storage (k fragments)
Bonus Node	Variable availability	Additional redundancy when online

Table 4: Node Classification

Classification is automatic based on observed metrics. A node that has been offline for more than 72 hours is automatically downgraded from Core to Bonus.

9. Tiered Storage

Data is automatically migrated between storage tiers based on access patterns:

Tier	Storage Type	Latency	Content
Hot	NVMe/SSD	<100ms	Last 7 days
Warm	SSD/HDD	<500ms	7-30 days
Cold	HDD/Archive	>1s	>30 days

Table 5: Storage Tiers

10. Caching

Each client maintains a three-level cache:

L1 (RAM) Hot metadata and small files. Configurable size (default 512 MB). Latency <10ms.

L2 (SSD) Frequently accessed chunks. Configurable size (default 50 GB). Latency <100ms. LRU eviction.

L3 (Remote) On-demand retrieval from storage nodes. Multi-source streaming for large files.

11. VFS Operations

VFS operations use operation codes in the range 0x0500-0x05CF of the MyClerk Protocol.

11.1. Basic Operations (0x0500-0x050F)

Code	Name	Description
0x0500	VFS_MOUNT	Mount VFS
0x0501	VFS_UNMOUNT	Unmount VFS
0x0502	VFS_STAT	Get file/directory info
0x0503	VFS_LIST	List directory contents
0x0504	VFS_READ	Read file data
0x0505	VFS_WRITE	Write file data
0x0506	VFS_CREATE	Create file/directory
0x0507	VFS_DELETE	Delete file/directory
0x0508	VFS_RENAME	Rename/move
0x0509	VFS_SYNC	Force sync

Table 6: Basic VFS Operations

11.2. Chunk Operations (0x0510-0x051F)

Code	Name	Description
0x0510	CHUNK_STORE	Store chunk
0x0511	CHUNK_RETRIEVE	Retrieve chunk
0x0512	CHUNK_VERIFY	Verify integrity
0x0513	CHUNK_DELETE	Delete chunk
0x0514	CHUNK_LOCATE	Find chunk locations
0x0515	CHUNK_REPLICATE	Replicate chunk

Table 7: Chunk Operations

11.3. Fragment Operations (0x0520-0x052F)

Code	Name	Description
0x0520	FRAGMENT_STORE	Store fragment
0x0521	FRAGMENT_RETRIEVE	Retrieve fragment
0x0522	FRAGMENT_STATUS	Get fragment status
0x0523	FRAGMENT_REDISTRIBUTE	Redistribute
0x0524	FRAGMENT_HEALTH_REPORT	Health report

Table 8: Fragment Operations

11.4. Metadata Operations (0x0530-0x053F)

Code	Name	Description
0x0530	META_SYNC_REQUEST	Request metadata sync
0x0531	META_SYNC_DIFF	Send diff
0x0532	META_CONFLICT_RESOLVE	Resolve conflict

0x0533	META_SNAPSHOT	Create snapshot
0x0534	META_RESTORE	Restore from snapshot

Table 9: Metadata Operations

11.5. Sharing Operations (0x0550-0x055F)

Code	Name	Description
0x0550	SHARE_CREATE	Create share
0x0551	SHARE_ACCEPT	Accept share
0x0552	SHARE_REVOKE	Revoke share
0x0553	SHARE_LIST	List shares
0x0554	SHARE_UPDATE	Update permissions
0x0555	COMMON_FOLDER_CREATE	Create common folder
0x0556	COMMON_FOLDER_JOIN	Join common folder
0x0557	COMMON_FOLDER_LEAVE	Leave common folder
0x0558	COMMON_FOLDER_SYNC	Force sync

Table 10: Sharing Operations

11.6. Emergency Operations (0x05C0-0x05CF)

Code	Name	Description
0x05C0	EMERGENCY_REVOKE	Immediate revocation
0x05C1	EMERGENCY_KEY_INVALIDATE	Invalidate all keys
0x05C2	EMERGENCY_CACHE_WIPE	Request cache wipe
0x05C3	EMERGENCY_CONFIRM	Client confirmation
0x05C4	EMERGENCY_STATUS	Query cut-off status

0x05C5	EMERGENCY_RESTORE	Restore access

Table 11: Emergency Operations

12. Federation

The VFS supports sharing between separate installations (families). Two sharing modes exist:

Simple Share Owner retains data on their Core Nodes. Recipient gets cache access only. Connection loss = no access.

Common Folder Both parties have full copies. Both can upload. Connection loss does not affect access.

12.1. Cache Security

Shared access uses time-limited tokens:

* Validity check interval: 10 minutes
* Offline tolerance: 1 hour
* Key rotation interval: 24 hours

Emergency Cut-Off immediately invalidates all tokens and keys.

13. Security Considerations

13.1. Encryption

All data is encrypted with ChaCha20-Poly1305 before leaving the client. Storage nodes never see plaintext. Key compromise affects only the specific folder/file/chunk level in the hierarchy.

13.2. Nonce Handling

Nonce reuse with the same key completely compromises security. The timestamp+random+counter construction ensures collision probability below 2^-80 per year.

13.3. CRDT Security

CRDT operations are authenticated using the MyClerk Protocol's session keys. Unauthorized nodes cannot inject or modify metadata.

13.4. Federation Security

Federation uses separate session keys. Emergency Cut-Off can immediately revoke all access. Cached data becomes cryptographically inaccessible when keys are rotated.

14. IANA Considerations

This document has no IANA actions.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

15.2. Informative References

[BLAKE3] O’Connor, J., "BLAKE3: One function, fast everywhere", 2020, <https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf>.

[CRDT] Shapiro, M., "A comprehensive study of Convergent and Commutative Replicated Data Types", 2011, <https://hal.inria.fr/inria-00555588>.

[REED-SOLOMON] Reed, I.S. and G. Solomon, "Polynomial Codes Over Certain Finite Fields", 1960, <https://ieeexplore.ieee.org/document/1057464>.

Acknowledgements

This specification is part of the MyClerk project, a privacy-first family orchestration platform currently in development. For more information, visit https://myclerk.eu.

Author’s Address

Michael J. Arcan
Arcan Consulting
Email: rfc@arcan-consulting.de
URI: https://myclerk.eu