AdaFS 1.0

Ada File System Technical Specification

Version 1.0
June 02, 2025

Clark Alesna
clark@saib.dev
SAIB Inc
https://github.com/SAIB-Inc/PaylKoyn

Contents

1 Abstract

AdaFS (Ada File System) is a novel protocol for storing arbitrary files directly on the Cardano blockchain using transaction metadata. This specification defines the encoding format, transaction structure, and retrieval mechanisms that enable decentralized file storage. AdaFS leverages Cardano's native metadata capabilities to create a permanent, censorship-resistant file storage system where files are reconstructed by following cryptographically-linked transaction chains.

2 Introduction

2.1 Motivation

Traditional file storage systems rely on centralized infrastructure that can be subject to censorship, single points of failure, and data loss. While blockchain-based storage solutions exist, most require specialized tokens or off-chain components. AdaFS provides a pure on-chain solution using only Cardano's existing transaction metadata capabilities.

2.2 Design Goals

The AdaFS protocol is designed with the following objectives:

- **Pure On-Chain Storage**: Files stored entirely within Cardano blockchain metadata
- **No Additional Tokens Required**: Uses only ADA for transaction fees
- **Permanent Storage**: Files persist as long as the Cardano blockchain exists
- **Cryptographic Linking**: Transaction chains provide integrity and ordering
- **Efficient Retrieval**: Optimized for reconstruction performance
- **Size Scalability**: Support for files larger than single transaction limits

2.3 Design Philosophy

This protocol represents a naive approach with a simple focus: store files on Cardano, nothing more, nothing less. We make no claims about being the next Filecoin or whether this is necessarily a useful thing to do. The goal is simply to demonstrate that arbitrary file storage is possible within Cardano's existing infrastructure, using only transaction metadata and standard blockchain mechanisms.

2.4 Scope

This specification covers AdaFS version 1.0, including:

- Metadata encoding format
- Transaction chaining mechanism
- File reconstruction algorithm
- Error handling procedures
- Implementation considerations

3 Protocol Overview

3.1 Core Concepts

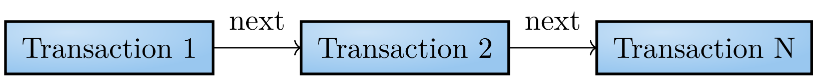
AdaFS operates on the principle of encoding file data as raw bytes within Cardano transaction metadata. Large files are automatically segmented across multiple transactions, with each transaction containing pointers to the next transaction in the chain.

3.2 Metadata Label

All AdaFS data uses metadata label `6673` (decimal). This number was chosen because it visually represents the hexadecimal `0x6673`, which corresponds to the ASCII characters "fs" (f=0x66, s=0x73) representing "file system".

3.3 Transaction Chain Structure

Files are stored as a linked list of transactions:



Each transaction contains:
- File data chunks encoded as raw bytes in MetadatumBytes
- Pointer to next transaction (if applicable)
- Metadata about the file (in final transaction)

4 Metadata Format Specification

4.1 Base Structure

All AdaFS metadata follows this JSON structure within label `6673`:

```cddl
metadata = {* transaction_metadatum_label => transaction_metadatum}

transaction_metadatum_label = uint .size 8

transaction_metadatum =
  {* transaction_metadatum => transaction_metadatum}
  / [* transaction_metadatum]
  / int
  / bytes .size (0 .. 64)
  / text .size (0 .. 64)

adafs_structure = {
  6673: {
    "payload": [* bytes],
    ? "next": bytes,
    ? "version": int,
    ? "metadata": {
      "filename": text,
      "contentType": text
    },
    ? "checksum": bytes
  }
}
```

4.2 Field Definitions

4.2.1 payload

An array of file data chunks.

- **Type**: `[* bytes]`
- **Format**: Each element contains raw binary data
- **Constraints**: Each bytes element limited to 64 bytes maximum (Cardano metadata constraint)
- **Structure**: Files are split into transactions, then each transaction's payload is split into 64-byte chunks
- **Presence**: Required in all transactions

4.2.2 next

Transaction hash pointing to the next transaction in the file chain.

- **Type**: `bytes`
- **Format**: 32-byte transaction hash (fits within 64-byte limit)
- **Constraints**: Must be valid Cardano transaction hash
- **Presence**: Present only in transactions that have a subsequent transaction

4.2.3 metadata

File metadata information included in the final transaction of a chain.

- **Type**: `{* transaction_metadatum => transaction_metadatum}`
- **Presence**: Present only in the final transaction
- **Fields**:
  - `contentType`: MIME type of the original file (`text`)
  - `filename`: Original filename (`text`)

4.2.4 version

Protocol version identifier.

- **Type**: `int`
- **Value**: `1` for AdaFS 1.0
- **Presence**: Present in final transaction

4.2.5 checksum

SHA-256 hash of the complete reconstructed file for integrity verification.

- **Type**: `bytes`
- **Format**: 32-byte SHA-256 hash (fits within 64-byte limit)
- **Presence**: Present in final transaction

5 Encoding Procedures

5.1 File Chunking Algorithm

Files are processed using the following algorithm:

1. **Calculate Transaction Capacity**: Determine how many 64-byte chunks fit per transaction
2. **Split File to Transactions**: Divide file across multiple transactions based on capacity
3. **Split Transaction Payload**: Within each transaction, split the payload into 64-byte chunks
4. **Store as Bytes Array**: Each transaction contains an array of bytes elements (max 64 bytes each)
5. **Optimize Distribution**: Maximize utilization within Cardano metadata limits

5.2 Encoding Pseudocode

The following pseudocode demonstrates the file encoding process:

```
function encode_file_to_transactions(file_data, max_tx_capacity):
    // Split file into transaction-sized chunks
    tx_chunks = split_file_into_transactions(file_data, max_tx_capacity)
    transactions = []

    for i in range(len(tx_chunks)):
        is_final = (i == len(tx_chunks) - 1)

        // Split transaction chunk into 64-byte pieces
        payload_chunks = split_into_64_byte_chunks(tx_chunks[i])

        // Create transaction metadata
        metadata = {
            6673: {
                "payload": payload_chunks  // Array of bytes (max 64 each)
            }
        }

        // Add next pointer (except for final transaction)
        if not is_final:
            // Note: next_tx_hash is pre-calculated due to deterministic transactions
            next_tx_hash = get_precalculated_hash(i+1)
            metadata[6673]["next"] = next_tx_hash

        // Add file metadata (only in final transaction)
        if is_final:
            metadata[6673]["version"] = 1
            metadata[6673]["metadata"] = {
                "filename": original_filename,
                "contentType": mime_type
            }
            metadata[6673]["checksum"] = sha256(file_data)

        transactions.append(create_transaction(metadata))

    return transactions

function split_into_64_byte_chunks(data):
    chunks = []
    for i in range(0, len(data), 64):
        chunk = data[i:i+64]
        chunks.append(chunk)
    return chunks
```

5.3 Transaction Capacity Calculation

Transaction capacity is determined by how many 64-byte chunks can fit within Cardano's metadata limits:

```
chunks_per_tx = (max_metadata_size - overhead) / (64 + cbor_overhead_per_chunk)
max_data_per_tx = chunks_per_tx * 64
```

Where:

- `max_metadata_size`: Cardano's per-transaction metadata limit
- `overhead`: Space for structure, next pointer, and metadata fields
- `cbor_overhead_per_chunk`: CBOR encoding overhead per bytes element

6 Transaction Chain Construction

6.1 Chain Construction Concept

Files are stored as linked lists of transactions using cryptographic linking:

1. **Transaction Generation**: Calculate all required transactions for the complete file
2. **Cryptographic Linking**: Each transaction references the next via transaction hash
3. **Deterministic Construction**: Since Cardano transactions are deterministic, all transaction hashes can be pre-calculated

4. **Metadata Distribution**: File chunks and metadata are distributed across the transaction chain
5. **Implementation Flexibility**: The protocol does not specify submission order or timing

6.2 Chain Integrity

The protocol ensures file integrity through:

1. **Hash Linking**: Each transaction explicitly references its successor
2. **Atomic Reconstruction**: Files can be reconstructed when all chain transactions exist on-chain
3. **Order Independence**: Transaction submission order does not affect reconstruction capability

7 File Reconstruction Algorithm

7.1 Retrieval Process

File reconstruction follows this algorithm:

```
function reconstruct_file(initial_tx_hash):
    file_chunks = []
    current_hash = initial_tx_hash
    metadata = null

    while current_hash is not null:
        transaction = get_transaction(current_hash)
        adafs_data = transaction.metadata["6673"]

        // Extract payload chunks (each chunk is bytes, max 64 bytes)
        for chunk in adafs_data.payload:
            file_chunks.append(chunk)  // chunk is already binary data

        // Store metadata if present
        if "metadata" in adafs_data:
            metadata = adafs_data.metadata

        // Follow chain
        current_hash = adafs_data.get("next", null)

    // Reconstruct file
    file_data = concatenate(file_chunks)

    // Verify integrity
    if metadata and metadata.checksum:
        verify_checksum(file_data, metadata.checksum)

    return file_data, metadata
```

8 Implementation Considerations

8.1 Performance Optimization

8.1.1 Caching Strategy

Implement caching at multiple levels:

1. **Transaction Cache**: Cache blockchain transaction lookups
2. **File Cache**: Cache complete reconstructed files
3. **Metadata Cache**: Cache file metadata separately

8.1.2 Parallel Processing

For large files with multiple transactions:

1. Fetch transactions in parallel where possible
2. Maintain ordering during reconstruction
3. Optimize blockchain API request patterns

8.2 Security Considerations

8.2.1 Data Integrity

1. Always verify SHA-256 checksums when available
2. Implement transaction hash validation
3. Detect and handle chain manipulation attempts

8.2.2 Privacy Implications

1. File data is permanently public on blockchain
2. Metadata includes original filenames
3. Consider encryption for sensitive data

8.3 Scalability Factors

8.3.1 Transaction Limits

Current Cardano limitations:

1. Maximum metadata size per transaction: 16KB
2. Practical payload size: 8KB per transaction after encoding overhead
3. No limit on transaction chain length

8.3.2 Cost Considerations

1. Each transaction requires minimum ADA fee
2. Large files require multiple transactions
3. Fee calculation: `base_fee * transaction_count`

9 Reference Implementation

9.1 Core Functions

Essential functions for AdaFS implementation:

```typescript
interface AdaFSMetadata {
  contentType: string;
  fileName: string;
  checksum: string;
}

interface AdaFSTransaction {
  payload: string[];
  next?: string;
  metadata?: AdaFSMetadata;
}

function encodeFile(fileData: Uint8Array): AdaFSTransaction[] {
  // Implementation details
}

function reconstructFile(startTxHash: string): {
  data: Uint8Array;
  metadata: AdaFSMetadata;
} {
  // Implementation details
}
```

9.2 Test Vectors

Live test vectors and implementation examples can be found in the PaylKoyn repository:

Repository: https://github.com/SAIB-Inc/PaylKoyn

The repository contains:
- Working AdaFS implementation
- Real transaction examples on Cardano testnet
- Test files with various sizes and types
- Complete metadata structures
- Validation tools for implementation testing

10 Implementation Notes

10.1 Current PaylKoyn Implementation

The PaylKoyn project implements AdaFS 1.0 with the following specifics:

10.1.1 Transaction Chain Implementation

The PaylKoyn reference implementation demonstrates one approach to transaction chaining:

- Pre-calculates all transaction hashes using Cardano's deterministic transaction system
- Constructs the complete linked list before any submission
- Submits transactions independently (order does not affect protocol compliance)

10.1.2 Metadata Requirements

For proper metadata preservation, file uploads must include:

- **Name**: Original filename (required for proper Content-Disposition)
- **ContentType**: MIME type (required for proper Content-Type headers)
- **File data**: Binary content

10.1.3 Retrieval Compatibility

The retrieval implementation handles both storage formats:

- **MetadatumBytes**: Raw binary data (current implementation)
- **MetadataText**: Hex-encoded strings (legacy compatibility)

This dual compatibility ensures the system can read files stored with different encoding approaches.

11 Conclusion

AdaFS 1.0 provides a robust foundation for decentralized file storage on the Cardano blockchain. The protocol's design prioritizes simplicity, reliability, and compatibility with existing Cardano infrastructure while enabling permanent, censorship-resistant file storage.

Key benefits of the AdaFS approach:

1. **No Infrastructure Dependencies**: Files stored entirely on-chain
2. **Cryptographic Integrity**: Transaction hashes provide tamper evidence
3. **Scalable Design**: Supports files of arbitrary size through chaining
4. **Standard Compliance**: Uses established Cardano metadata mechanisms
5. **Binary Efficiency**: Direct binary storage for optimal space utilization

The PaylKoyn implementation demonstrates successful operation with files up to 100MB, automatic transaction chaining, and complete metadata preservation from blockchain to cache to HTTP response.

12 References

1. Cardano Documentation: Transaction Metadata
   https://developers.cardano.org/docs/transaction-metadata/

2. CBOR RFC 8949: Concise Binary Object Representation
   https://datatracker.ietf.org/doc/html/rfc8949

3. RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
   https://datatracker.ietf.org/doc/html/rfc3986

4. NIST FIPS 180-4: Secure Hash Standard (SHS)
   https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

—

Specification Version: 1.0
Last Updated: June 02, 2025
Author: Clark Alesna (clark@saib.dev)
Organization: SAIB Inc
Contact: https://github.com/SAIB-Inc/PaylKoyn