Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 November 2026

V. Krivchikov
Independent

draft-sfc-container-format-01
5 May 2026

SFC: Self-Describing Resilient Container File Format

Abstract

This document specifies the Self-Describing Resilient Container (SFC) file format, a general-purpose binary container format designed for reliable transmission of arbitrary file data over unreliable, bandwidth-constrained, or asynchronous channels including physical media hand-carry, multi-session transfers, and heterogeneous delivery paths.

SFC encapsulates any file or directory tree into independently addressable, self-identifying chunks. Each chunk carries sufficient metadata to identify itself and verify its own integrity in isolation. Full reconstruction additionally requires a Global Header; this distinction is made explicit throughout.

This document defines a Core specification and five optional Profiles: Image (SFC/P1), Split Transport (SFC/P2), HTTP Hints (SFC/P3), Preprocessing (SFC/P4), and Directory (SFC/P5).

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 5 November 2026.

Copyright Notice

Table of Contents

Part I: Core Specification

---

1. Introduction

1.1. Motivation

File transfer over unreliable channels has long been addressed at the protocol layer: TCP retransmission, HTTP Range requests, BitTorrent peer-to-peer distribution. These solutions share a common dependency on live network connectivity and, in most cases, a coordinating server holding metadata about the transfer.

A distinct and underserved class of scenarios exists:

- Network connectivity is absent or intermittent (physical media hand-carry, sneakernet, disaster-area operations, deep-field scientific expeditions, spacecraft telemetry with high latency).
- Transfer is split across multiple sessions separated by days or weeks, with no persistent coordinator between sessions.
- Chunks arrive via heterogeneous carriers (USB drives, email attachments, postal storage media) in arbitrary order.

- The receiving party must derive maximum utility from whatever fraction of data has arrived before transfer is complete.

- The sender wishes to transfer an entire directory tree as a single resilient unit, with individual files extractable as their chunks arrive.

The Self-Describing Resilient Container (SFC) format addresses this gap. Each SFC chunk is self-identifying: it carries its own position identifier and integrity hash, enabling a receiver to recognize and verify individual chunks without any external resource. Full reconstruction additionally requires the Global Header; this is described in Section 2.

1.2. Design Goals

G1. Chunk self-identification and self-verification.
Every chunk MUST carry sufficient metadata to identify itself (File UUID, chunk index, chunk type) and independently verify its own byte integrity via BLAKE3 hash. A chunk does NOT carry sufficient information for full file reconstruction; that requires the Global Header (Sections 2.2 and 2.3).

G2. Loss tolerance.
The format MUST support recovery of the complete payload from a proper subset of transmitted chunks up to a configurable loss threshold, using erasure coding.

G3. Partial byte recovery.
A decoder MUST be able to reconstruct and present the contiguous byte prefix of the inner content from leading data chunks that have arrived, without requiring the full set. Whether this prefix is useful at the application level depends entirely on the inner format and is NOT guaranteed.

G4. Format agnosticism.
The format MUST accept any byte sequence as inner payload without requiring transformation of inner content.

G5. Transport agnosticism.
The format MUST NOT assume any specific transport mechanism.

G6. Implementation simplicity.
This specification defines two decoder implementation classes with distinct conformance status:

SFC-aware limited decoder: a decoder implementing only algorithm 0x00 (identity compression) can be built with no external dependencies beyond a BLAKE3 implementation and a Reed-Solomon GF(2^16) library. Such a decoder can process any SFC file whose compression algorithm is 0x00 and satisfies all Core MUST requirements for that file. It is NOT a conforming Core decoder and does not claim full Core conformance; it is a valid, useful subset implementation.

Conforming Core decoder: satisfies all MUST requirements in Part I of this specification, including mandatory support for algorithms 0x00 and 0x01 (zstd). Section 11 defines the complete conformance
checklist.

Throughout this document, "conforming decoder" means a conforming Core decoder unless explicitly qualified.

G7. Directory support.

The format MUST support encapsulation of an entire directory tree, with individual files extractable as their chunks arrive independently of other files in the same container.

1.3. Non-Goals

The following are explicitly outside scope:

- Encryption or confidentiality.
- Origin authentication or digital signatures.
- Streaming media optimization.
- Network protocol definition.
- Preservation of file metadata (permissions, timestamps).
- Symbolic links or empty directories.

1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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.

SFC File:
: A binary file conforming to this specification.

Inner Content:
: The byte sequence encapsulated in an SFC file. For standard files, the inner content is the file's bytes. For SFC Directory files (Inner Format ID 0x0050), the inner content is the concatenation of the Manifest bytes followed by all directory file bytes in Manifest order (Section 16.4).

Inner File Size:
: The total byte count of the inner content, excluding chunk padding. For standard files: the file's byte count. For SFC Directory: manifest_size + sum(file_sizes).

Data Chunk:
: A fixed-size fragment of the inner content, identified by a zero-based sequential index. Data chunk i carries inner content bytes [i*S .. (i+1)*S - 1]. For SFC Directory, the inner content begins with Manifest bytes; the Manifest spans chunks 0..K-1 where K = ceil(manifest_size / S). When manifest_size % S != 0, chunk K-1 is a mixed chunk: it contains the tail of the Manifest and the leading bytes of the first file's data. File and Manifest boundaries within a chunk are determined by byte offsets, not chunk boundaries. The chunk structure is otherwise identical to standard files.

Recovery Chunk:
: A chunk containing erasure-coded redundancy data derived from the set of data chunks.

Manifest:
: For SFC Directory (Inner Format ID 0x0050): a structured description of the directory tree, stored as the leading bytes of the inner content stream. K = ceil(manifest_size / S) chunks span the Manifest bytes. When manifest_size % S != 0, the last of these chunks (K-1) also contains leading bytes of the first file's data (mixed chunk). The Manifest is part of the inner content and subject to the same chunking, compression, erasure coding, and integrity rules as all other data.

File UUID:
: A 128-bit identifier uniquely identifying a particular SFC

encoding. All chunks and segments of the same encoding MUST share the same File UUID.

Chunk Self-Identification:
The property that a chunk alone can identify which encoding it belongs to (File UUID), its position (chunk index and type), and independently verify its own byte integrity via BLAKE3.

Chunk Self-Sufficiency (NOT a default property):
The hypothetical property that a chunk alone could participate in full dataset reconstruction. Standard SFC chunks do NOT have this property. Every chunk carries compression and erasure algorithm IDs in bytes 32-33, enabling decompression of that chunk's payload without the Global Header. This does NOT enable full reconstruction, which still requires N, M, S, Inner File Size, and the global hash from the Global Header.

Split Segment:
One physical file in an SFC/P2 Split Transport delivery, containing the Global Header and one or more chunks.

Terminal Segment:
The unique Split Segment that contains the Trailer.

Non-Terminal Segment:
Any Split Segment that is not the Terminal Segment.

Sibling Segments:
All Split Segments sharing the same File UUID.

Provisional Chunk:
A chunk that has passed phase-1 validation (D3a-D3e) but has not yet passed phase-2 validation (D4a-D4d). MUST NOT be attributed to any file or used in any output.

Benign Duplicate:
Two or more received chunks with identical chunk index and byte-for-byte identical content (same BLAKE3 hash value).

Contaminated Duplicate:
Two or more received chunks sharing the same chunk index and File UUID but having differing content (different BLAKE3 hashes).

Conforming Core Decoder:
A decoder implementation that satisfies all MUST-level requirements in Part I of this specification, including support for compression algorithms 0x00 and 0x01 (zstd). See Section 11. When this document says "conforming decoder" without further qualification, it means a Conforming Core Decoder.

SFC-Aware Limited Decoder:
A decoder implementation that supports a subset of Core (for example, only compression algorithm 0x00) and correctly handles all SFC files within that subset. NOT a Conforming Core Decoder and MUST NOT claim full Core conformance. Valid for constrained environments where zero-dependency operation is required.

2. Architecture Overview

2.1. Core and Profiles

This specification is divided into Core and Profiles.

Core defines the mandatory minimum: byte layout, chunk structure, integrity hashing, erasure coding, compression, reassembly, and

encoder/decoder behavior. All Conforming Core Decoders MUST satisfy the requirements in Part I. SFC-Aware Limited Decoders satisfy a subset; see Section 1.2 G6.

Profiles are optional extensions declared in the Global Header Flags field. A decoder encountering a Profile it does not implement MUST NOT fail; it MUST process the file using Core rules and MUST skip Profile-specific TLV extension fields.

Exception — SFC/P2 (Split Transport): SFC/P2 modifies the physical file structure by inserting a 16-byte Segment Header between the Global Header Region and the first chunk (Section 13.1). ALL decoders, including those that do not implement SFC/P2 semantics, MUST recognize the SPLIT_TRANSPORT flag (bit 0 of Flags) and skip the 16-byte Segment Header before parsing chunks. This is a Core-level requirement, not a Profile-level one, because the Segment Header affects byte alignment for all subsequent chunk parsing.

Profile declarations and recommended short identifiers:

Profile ID	Short ID	Name
0x01	SFC/P1	Image
0x02	SFC/P2	Split Transport
0x03	SFC/P3	HTTP Transport Hints
0x04	SFC/P4	Preprocessing
0x05	SFC/P5	Directory

2.2. What a Chunk Knows in Isolation

A standard SFC chunk, without the Global Header, provides:

- The File UUID it belongs to.
- Its own chunk index and type (data or recovery).
- Its compressed payload length.
- Independent hash-based integrity verification of its own bytes.

Additionally (present in version 0.1):
- Compression algorithm ID (chunk header byte 32).
- Erasure coding algorithm ID (chunk header byte 33).
- This enables decompression of that individual chunk’s payload without the Global Header. It does NOT enable reconstruction of the dataset, which still requires N, M, S, Inner File Size, and the global hash from the Global Header.

A chunk does NOT provide, in isolation:
- Total chunk count (N + M).
- Inner content size.
- Nominal chunk size (S).
- Global file hash.

2.3. What Requires the Global Header

Full reassembly requires the Global Header for:

- N and M (erasure coding parameters).
- Nominal chunk size S.
- Inner File Size (for trimming the last data chunk).
- Global file hash (for end-to-end integrity verification).
- Inner Filename (for output path).

Note: compression and erasure algorithm IDs are also available per-chunk in bytes 32-33 (Section 5.1), enabling per-chunk decompression without the Global Header. However, full reassembly still requires the Global Header for the parameters

listed above; per-chunk algorithm IDs alone are insufficient.

A decoder that has received chunks but not the Global Header MUST buffer received chunks as Provisional Chunks (applying D3a-D3e only) and defer reassembly.

2.4. Normative Requirements vs. Encoder Guidance

Type	Keyword	Sections
Interop normative (decoder behavior)	MUST/MUST NOT	3, 4, 5, 6.3, 6.4, 7.3, 8.1, 8.2, 9, 11
Interop normative (encoder behavior)	MUST/MUST NOT	4.8, 5.5, 9.5, 10.3
Encoder guidance (best practice)	SHOULD/ RECOMMENDED	6.2, 7.2, 10.1
Advisory (no interop impact)	MAY	10.2, 13.2
Profile normative (Profile implementors)	MUST/MUST NOT	12.3, 12.4, 13.1, 13.3, 13.4, 13.5, 13.6, 14, 15, 16.6, 16.7, 16.8, 16.9

3. File Structure

3.1. Magic Number and Version

Every SFC file or Split Segment MUST begin with:

Offset	Length	Value
0	4	0x53 0x46 0x43 0x00 (ASCII "SFC\0")
4	2	Major version (uint16, little-endian) = 0
6	2	Minor version (uint16, little-endian) = 1

This specification defines version 0.1. Decoders MUST reject files with a major version greater than they implement. Decoders SHOULD accept files with a higher minor version, treating unknown TLV fields per the TLV Contract (Section 3.2). For SFC/P2 segments sharing a File UUID, the stricter requirement of Section 13.5 applies: all segments MUST have identical major AND minor version numbers.

Minor version compatibility guarantee: within major version 0, any future minor version MUST NOT change the meaning of any existing known TLV tag or any existing assigned field in Core semantics. A future minor version MAY add new TLV tags and new Profile bit assignments in bits 9-15 of the Flags field via formal reassignment. It MAY also assign semantics to currently-reserved bytes in chunk headers (chunk header offsets 34-47) via formal reassignment.

Reserved bytes and forward compatibility: currently-reserved bytes in chunk headers (offsets 34-47 in version 0.1) MUST be zero when encoding for version 0.1. When a decoder encounters a file with a higher minor version than it implements and finds non-zero bytes in positions that were reserved in its implemented version, it MUST NOT treat this as a format error; instead it MUST treat those bytes as unknown extensions and ignore them. The hard format error for non-zero reserved bytes (Section 5.5) applies only when the file’s minor version is less than or equal to the decoder’s implemented minor version, where those bytes were defined to be zero.

Reserved bits 1-3 of the Flags field are reserved permanently within major version 0 and MUST remain zero in all minor versions.

3.2. Global Header and TLV Contract

The Global Header immediately follows the 8-byte preamble (Magic Number + Major Version + Minor Version; Section 3.1).

Offset	Length	Field
0	4	Header length H (uint32 LE; value excludes these 4 bytes; header region = H+4 bytes total)
4	16	File UUID
20	8	Inner File Size (uint64 LE; Section 4.5)
28	2	Inner Format ID (Section 4.2)
30	255	Inner filename (UTF-8, null-padded; Section 4.8)
285	32	Global file BLAKE3 hash (32 bytes; Section 8.2)
317	4	Data chunk count N (uint32 LE)
321	4	Recovery chunk count M (uint32 LE)
325	4	Nominal chunk size S in bytes (uint32 LE)
329	1	Erasure coding algorithm ID (Section 6.1)
330	1	Compression algorithm ID (Section 7.1)
331	2	Flags and Profile declarations (uint16 LE; Section 4.6)
333	2	Priority chunk count P (uint16 LE)
335	4*P	Priority chunk index list (uint32 LE each)
335+4P	var	TLV extension fields

Header length H MUST equal the total byte count of all fields from offset 4 through the end of all TLV extension fields. Decoders MUST validate H before allocating the header buffer.

Global Header Region: H+4 bytes starting at the Header length field (offset 0 of the Global Header). Magic bytes are NOT included. Used as input to the Trailer BLAKE3 hash.

TLV Contract:

Each TLV extension field:
- 2 bytes: type tag (uint16 LE)
- 4 bytes: value length L (uint32 LE)
- L bytes: value

TLV tag namespace:
- Tags 0x0001-0x7FFF: reserved for this specification.
- Tags 0x8000-0xFFFF: vendor-specific.

Known TLV tags:

- 0x0020 (uint64[] L=8*(N+M)): SFC/P3 chunk offset index. Requires SFC/P3 Profile bit (bit 6). Encoders MUST NOT include this tag unless the SFC/P3 Profile bit is set.
- 0x0030 (uint16 LE, L=2): SFC/P4 original inner format ID. Requires SFC/P4 Profile bit (bit 7). Encoders MUST NOT include this tag unless the SFC/P4 Profile bit is set.

User metadata tags (UTF-8 encoded strings, L MUST NOT exceed 4096):

- 0x0100 (UTF-8, L<=4096): Author. Name of the person or organization that created the content.
- 0x0101 (UTF-8, L<=4096): Description. Human-readable summary of the payload.
- 0x0102 (UTF-8, L<=4096): Location. Geographic location or place name associated with the

content (e.g., "77.8 S 166.7 E" or "Kabul field hospital").

0x0103 (UTF-8, L<=4096): Software. Name and version of the software that encoded the file (e.g., "sfc-cli 1.0").

0x0104 (UTF-8, L<=4096): Comment. Free-form note.

All metadata tags are OPTIONAL. Encoders MUST NOT emit a metadata tag with an empty value (L=0) or with L > 4096. Decoders MUST treat a known metadata tag with L=0 or L > 4096 as a format error per rule (f) and halt. All metadata values MUST be valid UTF-8; decoders MAY validate encoding but are not required to. Metadata tags are subject to the standard TLV rules (ascending order, at most once each).

Note: TLV tags 0x0010-0x0012 are permanently reserved and MUST NOT be used. Decoders encountering these tags MUST skip them silently without failing (as if they were unknown tags per rule c), because they may appear in files produced by older implementations.

Normative TLV rules:

a. TLV fields are contiguous; no padding between fields or between the priority list and the first TLV field.

b. Known TLV tags MUST appear at most once. Duplicate known tag: format error; report and halt.

c. Unknown TLV tags MUST be skipped without failing. Unknown tags MAY appear more than once; each occurrence ignored.

d. Encoders MUST serialize TLV fields in ascending type-tag order. This ordering is the canonical serialization of the TLV fields and is required for raw-byte identity of the Global Header Region across SFC/P2 segments (segment-specific metadata is carried in the Segment Header, not in TLV fields).

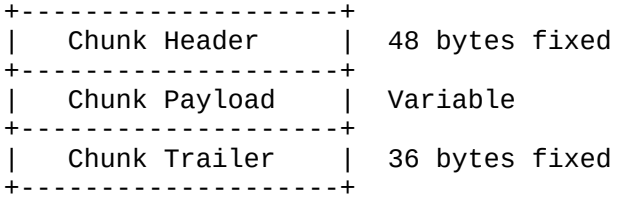
e. Each TLV value region MUST NOT extend beyond the Global Header boundary. Decoders MUST verify before reading TLV values.

f. L=0 is permitted for unknown tags. For known tags, unexpected length is a format error.

g. A known Profile-specific TLV tag without the corresponding Profile bit set is a format error. Report and halt.

3.3. Chunk Layout

Each chunk consists of three sections in order:



In SFC/P2 segments, a 16-byte Segment Header (Section 13.1) is placed between the Global Header Region and the first chunk. In non-P2 files, chunks immediately follow the Global Header.

For SFC Directory (Inner Format ID 0x0050), data chunks 0..K-1 span the Manifest bytes as their payload content (Section 16.2).

When manifest_size % S != 0, chunk K-1 also contains leading bytes of the first file's data (mixed chunk; Section 16.4). There is no structural difference between Manifest chunks and file data chunks; both are standard data chunks processed identically by all Core rules.

3.4. Trailer

The Trailer is a 64-byte structure occupying the final 64 bytes of a complete single-file SFC, or the final 64 bytes of the Terminal Segment in SFC/P2.

Offset	Length	Field
0	4	0x54 0x52 0x4C 0x52 ("TRLR")
4	4	Reserved (MUST be zero)
8	32	BLAKE3 hash of the Global Header Region
40	8	Encoder timestamp (seconds since Unix epoch, uint64 LE)
48	16	Reserved (MUST be zero; Section 5.5)

The BLAKE3 hash at offset 8 covers the Global Header Region. Decoders MUST verify this hash (D2c).

The encoder timestamp at offset 40 is informational only. Decoders MUST NOT validate or reject based on its value; zero is a valid timestamp.

Note: bytes 4-7 are reserved. Decoders MUST treat non-zero values in bytes 4-7 as a format error for files with version <= 0.1.

In SFC/P2 Split Transport:

- Trailer MUST appear in the Terminal Segment only.
- For Non-Terminal Segments, D2c is DEFERRED until the Terminal Segment is located.
- If the Terminal Segment cannot be located, see Section 9.2 (Unverified Reconstruction) and Section 9.3 (Partial Reassembly) for the applicable recovery procedure.

3.5. Validation Dependencies

Dependency group 1 — Pre-allocation:

- D1a. Magic bytes verified before any further parsing.
- D1b. Major version verified before reading header fields.
- D1c. Header length H bounds-checked (Section 18.3) before allocating header buffer.

Dependency group 2 — Global Header parsing:

- D2a. All fixed header fields parsed before TLV fields.
- D2b. N, M, S validated against limits (Section 18.3) before any chunk memory is allocated.
- D2c. Trailer BLAKE3 hash verified before chunk processing. EXCEPTION: Deferred for SFC/P2 Non-Terminal Segments until Terminal Segment is located.

Dependency group 3 — Per-chunk phase 1 (Provisional):

- D3a. Chunk magic ("CHK\0") verified before reading further.
- D3b. File UUID read and stored; match against Global Header deferred to D4a.
- D3c. Compressed payload length bounds-checked before reading.
- D3d. BLAKE3 hash verified over (header bytes || payload bytes). Chunks failing verification MUST be discarded immediately and MUST NOT enter any buffer.
- D3e. Chunk end marker "/CHK" verified at expected position (Section 5.3). Missing or incorrect marker: discard chunk.

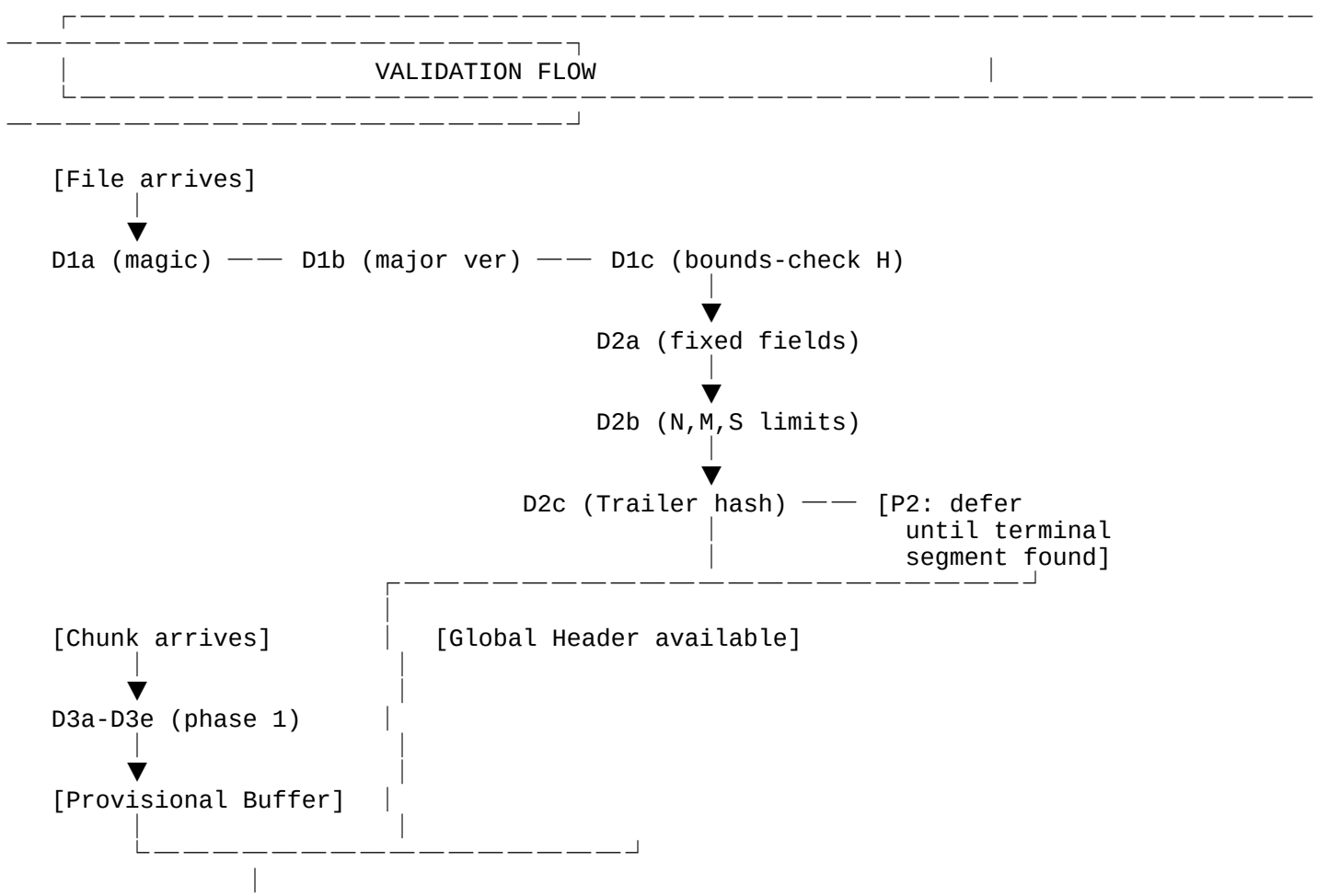
Provisional Chunks (passing D3a-D3e, not yet D4a-D4d) MUST
NOT be attributed to any file and MUST be stored in a
segregated provisional buffer (Section 18.5).

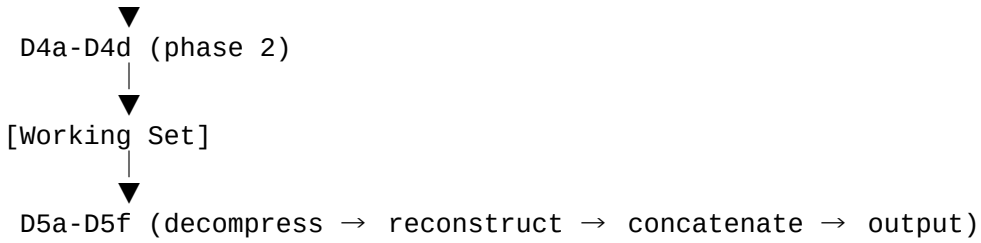
Optimization: decoders MAY deduplicate byte-identical chunks
within the provisional buffer before the Global Header arrives.

Dependency group 4 — Per-chunk phase 2 (requires Global Header):
D4a. File UUID match against Global Header UUID.
D4b. Chunk index range check: index in [0, N+M-1].
D4c. Reserved byte check (Section 5.5).
D4d. Algorithm ID check: bytes 32-33 match Global Header
algorithm fields (Section 5.1).
Chunks passing D4a-D4d are promoted to the working set.

Dependency group 5 — Reconstruction:
D5a. Duplicate handling (Section 9.5) before any decompression
or reconstruction.
D5b. Decompress each chunk payload independently (Section 7.3)
before erasure reconstruction. RS operates on uncompressed
S-byte blocks; decompression MUST precede RS reconstruction.
D5c. Erasure reconstruction on decompressed S-byte blocks
(Section 6.4), applied only to derive missing data chunks.
D5d. Concatenate decompressed blocks in ascending chunk index
order (indices 0..N-1).
D5e. Trim concatenated output to Inner File Size.
D5f. Global file hash verified after trimming, before output is
written or presented to user. D2c MUST also be verified
before D5f when the Terminal Segment is found.

Validation flow:





4. Global Header Fields

4.1. File Identifier

The File UUID is a 128-bit identifier generated by the encoder. Encoders SHOULD generate a version 4 (random) UUID [RFC9562]. All chunks and segments of the same SFC encoding MUST carry the same File UUID.

4.2. Inner Format Descriptor

ID	Format
0x0000	Unknown / unspecified
0x0001	Arbitrary binary data
0x0010	Plain text (UTF-8)
0x0011	Line-oriented text (CSV, NDJSON, log files)
0x0020	JPEG Baseline
0x0021	JPEG Progressive
0x0022	JPEG 2000
0x0023	JPEG XL
0x0024	PNG (non-interlaced)
0x0025	PNG (Adam7 interlaced)
0x0026	WebP (baseline)
0x0030	Fragmented MP4
0x0031	Matroska / WebM
0x0040	ZIP archive
0x0041	gzip
0x0042	zstd-compressed data
0x0043	tar + zstd (streamable archive)
0x0050	SFC Directory (see Section 16)
0x00FF	Nested SFC
0x0100	PDF
0x0101	ePub
0x8000+	Vendor-specific

Decoders MUST NOT fail on unknown Inner Format IDs; they MUST treat them as 0x0001 (arbitrary binary data).

4.3. Erasure Coding Parameters

N: count of data chunks encoding the complete inner content.
M: count of additional recovery chunks generated from N.
Any N of the (N+M) total chunks suffice for reconstruction.
M = 0: no redundancy; all N data chunks required.

4.4. Compression Parameters

The Compression Algorithm ID applies uniformly to ALL chunk payloads without exception. This includes Manifest chunks in SFC Directory files. Per-chunk algorithm overrides are not supported in version 0.1.

4.5. Chunk Count Fields and Inner File Size

Nominal Chunk Size S: target uncompressed size of each data chunk.

S MUST be even (required for Reed-Solomon GF(2^16) word alignment; Section 6.4). The last data chunk carries fewer than S bytes of meaningful inner content; it is zero-padded to exactly S bytes before compression (Section 5.2). Recovery chunks MUST be exactly S bytes in uncompressed form, zero-padded if necessary.

Inner File Size: total byte count of the inner content, excluding chunk padding.

```
For standard files:  Inner File Size = byte count of the file.
For SFC Directory:   Inner File Size = manifest_size
                                     + sum(file_sizes)
```

where manifest_size is defined in Section 16.2 and file_sizes are the actual sizes of each file listed in the Manifest.

Priority chunk list: The P field declares a count of priority chunk indices, followed by P uint32 LE values in the header. The following normative rules apply:

a. P MUST satisfy P <= N. Only data chunk indices (0..N-1) are valid priority indices; recovery chunk indices are not meaningful as priority hints. P > N is a format error.

b. Priority indices MUST be unique within the list. A duplicate index within the priority list is a format error.

c. Each priority index value MUST be in range [0, N-1]. An out-of-range value is a format error.

d. Sort order within the priority list is not required and carries no normative meaning.

e. The priority list is advisory. Outside of SFC/P1 (which defines specific semantics in Section 12.4), a decoder MAY use priority indices to prefetch or request those chunks first, but is NOT required to do so. The list has no effect on reconstruction correctness.

f. A non-empty priority list (P > 0) in a file that does not declare SFC/P1 (bit 4 not set) is permitted. Decoders MUST NOT treat this as a format error.

g. The priority list occupies 4P bytes inside the Global Header. Combined with all fixed fields (minimum 331 bytes) and TLV fields, the total MUST NOT cause H to exceed 65,536 bytes. Absent TLV fields, this limits P to at most 16,301 (floor((65,536 - 331) / 4) = 16,301). TLV fields reduce this bound further. Encoders MUST verify this constraint before serializing the header and report an error rather than produce a file whose H exceeds the maximum.

4.6. Flags and Profile Declarations

The 2-byte Flags field:

```
Bit 0 (LSB): SPLIT_TRANSPORT — this file is a Split Segment.
Bits 1-3:    Reserved permanently in major version 0.
             MUST be zero on encoding. Decoders MUST treat
             non-zero bits 1-3 as a format error and halt.
Bit 4:       SFC/P1 (Image) — encoder claims conformance.
Bit 5:       SFC/P2 (Split Transport) — encoder claims conformance.
Bit 6:       SFC/P3 (HTTP Hints) — encoder claims conformance.
Bit 7:       SFC/P4 (Preprocessing) — encoder claims conformance.
Bit 8:       SFC/P5 (Directory) — encoder claims conformance.
Bits 9-15:   Reserved for future profiles. MUST be zero in
             version 0.1. Decoders MUST ignore without failing.
```

A Profile bit indicates that the encoder claims conformance to that Profile's semantics, whether or not Profile-specific TLV fields are present. If a Profile bit is set, the encoder MUST conform to all MUST-level requirements of that Profile.

A decoder encountering a Profile bit it does not implement MUST NOT fail; it MUST apply Core rules and skip Profile-specific TLVs.

A decoder encountering a known Profile-specific TLV without the corresponding Profile bit set MUST treat this as a format error (TLV rule g).

A decoder detecting a Profile MUST-level violation SHOULD report the violation and MAY continue processing using Core rules only.

4.7. Protocol Contract: Header Authority

Rule 1 — Chunk self-authority:
A chunk's own magic, index, type, payload length, and BLAKE3 hash are authoritative within that chunk.

Rule 2 — File UUID and multi-encoding operation:
When a decoder processes multiple .sfc files simultaneously (e.g., decoder *.sfc), it MUST first group files by File UUID. Encountering files with different UUIDs is NOT an error; it is the expected multi-encoding case. UUID grouping precedes all other validation. Within a UUID group, every chunk's File UUID MUST match the group's Global Header UUID; mismatches within a group are reported per Section 9.4.

Rule 3 — Global Header authority:
All file-level parameters (N, M, S, compression algorithm, erasure algorithm, Inner File Size, global file hash) are sourced exclusively from the Global Header.

Rule 4 — Missing Global Header:
Buffer chunks as Provisional; report "Global Header not received". Do not reconstruct.

Rule 5 — Conflicting Global Headers (SFC/P2):
If two segments present Global Headers with the same UUID but differing raw bytes: report conflict with differing byte range and source files, halt, require explicit user resolution. Decoders MUST NOT silently resolve the conflict.

4.8. Inner Filename Safety

The Inner Filename field is 255 bytes, null-padded. The effective filename is the byte sequence before the first null byte. All bytes after the terminating null MUST be zero; non-zero bytes after the null are a format error.

The 255-byte field is part of the raw-byte identity of the Global Header Region.

Encoders MUST NOT include in the effective filename:
- Path separators: '/' (0x2F) and '\' (0x5C).
- The sequences ".." and "." as the entire effective filename.
- Any byte in range 0x00-0x1F (control characters).

Decoder filename processing:
a. Sanitization at byte level before UTF-8 interpretation. Each maximal contiguous sequence of forbidden bytes replaced with a single '_' (0x5F). After forbidden-byte replacement, validate the result as UTF-8. Replace each maximal invalid

UTF-8 byte subsequence (as defined by the W3C Encoding Standard "replacement of maximal subparts" algorithm) with a single '_' (0x5F). This two-pass process is mandatory and deterministic; MUST NOT use U+FFFD or any other substitute.

b. Empty effective filename: reject with "empty inner filename".
c. Apply sanitization regardless of encoder conformance.
d. Write output only within the designated output directory. Path traversal MUST NOT succeed even if sanitization fails.
e. If the sanitized effective filename equals "." or "..", reject with "inner filename is reserved path component". Neither "." nor ".." contains forbidden bytes, so rule (a) does not catch them; this explicit check is required.

For SFC Directory, Section 4.8 rules also apply to each path component in Manifest file entries (Section 16.3).

5. Chunk Structure

5.1. Chunk Header (48 bytes fixed)

Offset	Length	Field
0	4	"CHK\0" (0x43 0x48 0x4B 0x00)
4	16	File UUID
20	4	Chunk index (uint32 LE)
24	4	Chunk type (uint32 LE): 0x00000001 = Data chunk 0x00000002 = Recovery chunk All other values: unknown chunk type; discard this chunk (Section 9.4)
28	4	Compressed payload length (uint32 LE)
32	1	Compression algorithm ID (uint8)
33	1	Erasure coding algorithm ID (uint8)
34	14	Reserved (MUST be zero; Section 5.5)

In version 0.1, bytes 32-33 carry algorithm IDs in every chunk; the only valid compression IDs are 0x00-0x03 and 0x80+; the only valid erasure IDs are 0x00, 0x01, and 0x80+.
A decoder receiving a chunk without the Global Header MAY use bytes 32-33 to decompress that chunk's payload independently. This is the headerless decompression capability described in Section 2.2.

When the Global Header IS available, decoders MUST verify that bytes 32-33 match the Global Header's algorithm fields. A mismatch is a format error; the affected chunk MUST be discarded.

Encoders MUST write the correct algorithm IDs in bytes 32-33 of every chunk, matching the Global Header's compression and erasure algorithm fields.

Unknown chunk type: decoder MUST discard chunk and report "unknown chunk type" with chunk index and observed type value. Such chunks do not contribute to valid chunk count V (Section 6.3).

5.2. Chunk Payload

The chunk payload is the compressed (or raw, if algorithm 0x00) content.

Data chunk i carries inner content bytes [i*S .. (i+1)*S - 1]. This rule applies uniformly to all data chunks. For SFC Directory files, the Manifest is simply the leading portion of the inner content; Manifest chunk payloads obey this same rule.

The last data chunk (index N-1) MUST be zero-padded to exactly S bytes before compression and before erasure coding input.

Recovery chunks carry Reed-Solomon coded data per Section 6.4. Their uncompressed size is always exactly S bytes.

5.3. Chunk Trailer (36 bytes fixed)

Offsets relative to the start of the Chunk Trailer.

Trailer offset	Length	Field
0	32	BLAKE3(Chunk Header bytes || Chunk Payload bytes)
32	4	0x2F 0x43 0x48 0x4B ("/CHK")

Hash input: all 48 Chunk Header bytes concatenated with all Chunk Payload bytes. Chunks failing hash verification MUST be discarded immediately (D3d).

Decoders MUST verify that the 4-byte end marker "/CHK" is present at the expected position (immediately after the 32-byte BLAKE3 hash). A missing or incorrect end marker indicates truncation or corruption; the chunk MUST be discarded with error "chunk end marker invalid" (D3e).

The end marker enables fast chunk boundary detection by scanning for the "/CHK" byte sequence without computing BLAKE3 hashes, which is useful for recovery from partially corrupted streams. BLAKE3 verification (D3d) remains mandatory regardless.

5.4. Recovery Chunks

Recovery chunks follow the same physical structure as data chunks. Chunk indices: N to N+M-1. Payload wire format: Section 6.4.

5.5. Reserved Field Handling

Reserved bytes:

- Chunk Header offsets 34-47 (14 bytes).
  Note: offsets 32-33 are NOT reserved; they carry algorithm IDs in all modes (Section 5.1).
- File Trailer (Section 3.4) offsets 4-7 (4 bytes, reserved).
- File Trailer (Section 3.4) offsets 48-63 (16 bytes).

Encoders MUST write zero bytes in all reserved fields.

Decoders MUST treat non-zero reserved bytes as a hard format error for files with major.minor <= the decoder's implemented version:

- Report "non-zero reserved bytes" with location and byte offset.
- Discard the affected chunk, or halt if error is in Trailer.
- MUST NOT silently accept or ignore.

Note: bits 9-15 of the Flags field are explicitly exempt from this rule per Section 4.6; they MUST be ignored without failing, as they are reserved for future profile assignments.

6. Erasure Coding (Core)

6.1. Algorithm Identifiers

0x00	None (M MUST be 0; all N data chunks required)
0x01	Reed-Solomon GF(2^16), Cauchy matrix (REQUIRED support)
0x80+	Vendor-specific

Erasure algorithm 0x00 with M > 0 is a format error. Decoders MUST report "erasure algorithm 0x00 with M > 0" and halt before any chunk processing. Encoders MUST NOT produce such files.

Erasure algorithm ≠ 0x00 with M = 0 is also a format error: when M = 0 no erasure coding is active, so a non-None algorithm ID is inconsistent. Decoders MUST report "non-zero erasure algorithm with M=0" and halt. Encoders MUST set the field to 0x00 when M = 0 (Section 10.3).

A decoder that does not implement a declared erasure algorithm MUST report "unsupported erasure algorithm: 0xXX" as a distinct error and MUST NOT produce any output for that file. This is symmetric with the treatment of unsupported compression algorithms (Section 7.1). A file declaring a vendor-specific erasure algorithm is a valid SFC file; it is only processable by decoders that implement that algorithm.

6.2. Reed-Solomon Parameters

The REQUIRED algorithm is systematic Reed-Solomon over GF(2^16) with a Cauchy generator matrix. Any N of the (N+M) total chunks suffice to reconstruct all N data chunks. N+M MUST NOT exceed 65,535. This limit ensures the maximum column index M+N-1 fits within the field (elements 0 to 65,535), and also provides one unit of conservative margin.

M selection guidance (non-normative):

Transport	Suggested M/(N+M)
Reliable (local copy)	0.00
Email / messaging attachment	0.20
Physical media (multiple drives)	0.30
Highly unreliable channel	0.50

6.3. Reconstruction Algorithm

1. Group all received files by File UUID (Rule 2, Section 4.7).
2. For each UUID group:
   a. Apply D3a-D3e to each chunk. Discard invalids.
   b. Buffer Provisional Chunks; defer D4a-D4d.
   c. Verify Global Header availability (Rule 4); if absent, buffer all chunks as Provisional and defer reconstruction until the Global Header arrives.
   d. Apply D4a-D4d to Provisional Chunks; promote or discard.
   e. Apply duplicate handling (Section 9.5).
   f. Count valid chunks V (data + recovery in working set). Invalid, unknown-type, and duplicate-eliminated chunks MUST NOT contribute to V.
      If M=0 and any data chunk absent: report incomplete; halt.
      If M>0 and V<N: report incomplete; halt.
   g. If all N data chunks are available and valid: decompress each per Section 7.3; proceed to reassembly under Section 9.1 (if D2c passed) or Section 9.2 (if D2c deferred), step 2 in either case. Recovery chunks carry RS parity and MUST NOT contribute bytes to the assembled inner content stream.
   h. Otherwise: decompress all chunks in the working set per Section 7.3 to obtain S-byte uncompressed blocks. Select exactly N chunks for reconstruction: use all available data chunks first; if fewer than N data chunks are present, supplement with recovery chunks from the working set to reach exactly N, selecting them in ascending chunk index order. Any N-subset from a Cauchy construction will produce correct results. Apply Reed-Solomon reconstruction

per Section 6.4 on these N decompressed blocks to derive the missing data chunks.

i. Proceed to reassembly under Section 9.1 or Section 9.2, as applicable, with all N decompressed data blocks available.

6.4. Reed-Solomon Wire Format

Normative ordering rule: RS operates on uncompressed S-byte block payloads. Compression applied AFTER RS generation; decompression applied BEFORE RS reconstruction. Applying RS over compressed data is non-conforming.

S MUST be even. RS treats each S-byte block as a sequence of S/2 uint16 little-endian words.

Decoders MUST verify that decompressed chunk size equals exactly S bytes for every chunk (including the last data chunk, which was zero-padded to S before compression per Section 5.2).
A decompressed size other than S is a format error; the chunk MUST be discarded.

--- GF(2^16) Field ---

Field polynomial: x^16 + x^5 + x^3 + x^2 + 1 (0x1002D).
Implementations MUST use this polynomial.

Reference inverse values:
gf_inv(2) = 0x8016 [gf_mul(2, 0x8016) = 1, confirmed]
gf_inv(3) = 0xFFE4 [gf_mul(3, 0xFFE4) = 1, confirmed]

Implementations MUST validate against these values at conformance testing or at initialization.

N+M MUST NOT exceed 65,535. This limit derives from the GF(2^16) field size: row indices k ∈ {0..M-1} and column indices {M..M+N-1} must all fit within the field (0 to 65,535), and k XOR (M+j) must be non-zero for all valid (k,j).

--- Generator Matrix ---

Cauchy matrix C (M rows × N columns) over GF(2^16):
C[i][j] = gf_inv(i XOR (M + j))
for i in [0,M), j in [0,N)

Encoders MUST use this construction. Alternative Cauchy constructions and Vandermonde matrices are NOT conforming.

--- Encoding: Recovery Block Generation ---

Inputs: N blocks B[0..N-1], each exactly S bytes.
B[i] = decompressed payload of data chunk i.
B[N-1] MUST be zero-padded to S bytes (Section 5.2).

Each block is interpreted as S/2 uint16 little-endian words:
W[i][w] = uint16_le(B[i][2*w], B[i][2*w+1])
for w in [0, S/2).

Recovery block R[i] for i in [0, M):
R[i][w] = XOR over j in [0,N): gf_mul(C[i][j], W[j][w])
for each word position w in [0, S/2).

After computing R[i]: serialize words back to S bytes (uint16 LE), compress using the file's declared algorithm (independently per Section 7.3). Store as recovery chunk N+i.

```
--- Reconstruction ---

Given any N valid chunks, reconstruct ONLY missing data chunks.

Ordering: sort selected N chunks by ascending chunk index before
constructing the matrix.  Call this sorted list
available_chunks[0..N-1].

Semantic invariant: row p of A_inv yields coefficients for
recovering W[p] from the N available equations.

Procedure:
1.  Sort available_chunks by ascending chunk index.
2.  For each (idx, block) at position i in sorted list:
      Interpret block as S/2 uint16 LE words.
      If idx < N (data chunk):
        matrix_row[i] = identity row for position idx
                        (row[j] = 1 if j==idx, else 0)
      If idx >= N (recovery chunk, k = idx - N):
        matrix_row[i] = [gf_inv(k ^ (M+j)) for j in [0,N)]
3.  Form N×N matrix A where row i = matrix_row[i].
3a. Safety check: for each recovery chunk at position i (idx >= N,
    k = idx - N), verify that k XOR (M+j) != 0 for all j in
    [0,N).  By construction, k < M <= M+j, therefore k != M+j
    as integers, therefore k XOR (M+j) != 0 (XOR of distinct
    non-negative integers is always non-zero).
    A chunk index outside [N, N+M) would have been rejected by
    D4b.  Decoders MAY assert this invariant; if it fails the
    chunk set is malformed and MUST be discarded.
4.  Invert A over GF(2^16) to get A_inv.  If inversion fails
    (matrix is singular), the selected chunk set is incompatible
    with the Cauchy construction; this MUST NOT occur when chunks
    are produced by a conforming encoder.  If encountered, choose
    a different N-subset from the chunks already in the working
    set, excluding the recovery chunk that caused the singular
    matrix, and retry.  If no alternative N-subset exists,
    report "RS reconstruction failure" and halt.
5.  For each missing data chunk index p:
      recovered[p][w] = XOR over i in [0,N):
        gf_mul( A_inv[p][i], available_chunks[i].words[w] )
6.  Serialize recovered words to bytes (uint16 LE).
7.  Strip zero-padding from B[N-1] using Inner File Size.

--- Normative Pseudocode ---

# Encoding
# Interpret each block as uint16 LE words
W = []
for j in range(N):
    W.append([uint16_le(B[j][2*w], B[j][2*w+1])
              for w in range(S // 2)])

for i in range(M):
    R[i] = [0] * (S // 2)               # S/2 zero words
    for j in range(N):
        coeff = gf_inv(i ^ (M + j))
        for w in range(S // 2):
            R[i][w] ^= gf_mul(coeff, W[j][w])
    # Serialize R[i] back to S bytes (uint16 LE) before compression

# Reconstruction
# available_chunks: list of N (chunk_index, S-byte decompressed block
#                   pairs)
# missing_indices: data chunk indices not in available_chunks

available_chunks.sort(key=lambda x: x[0])  # MUST sort ascending
```

```
# Interpret blocks as uint16 LE words
words = []
for (idx, block) in available_chunks:
    words.append([uint16_le(block[2*w], block[2*w+1])
                  for w in range(S // 2)])

A = []
for (idx, block) in available_chunks:
    if idx < N:
        row = [1 if j == idx else 0 for j in range(N)]
    else:
        k = idx - N
        row = [gf_inv(k ^ (M + j)) for j in range(N)]
    A.append(row)
A_inv = gf_matrix_inverse(A)  # invert N× N matrix over GF(2^16)

# Row p of A_inv gives coefficients to recover W[p]
for p in missing_indices:
    recovered_words = [0] * (S // 2)
    for i in range(N):
        coeff = A_inv[p][i]
        for w in range(S // 2):
            recovered_words[w] ^= gf_mul(coeff, words[i][w])
    recovered[p] = serialize_uint16_le(recovered_words)

--- Worked Example ---

N=2, M=1, S=4 (2 words per block), field 0x1002D.
C[0][0] = gf_inv(0 XOR (1 + 0)) = gf_inv(1) = 1
C[0][1] = gf_inv(0 XOR (1 + 1)) = gf_inv(2) = 0x8016

B[0]=[0x01,0x00,0x02,0x00], B[1]=[0x03,0x00,0x04,0x00]
W[0]=[0x0001, 0x0002], W[1]=[0x0003, 0x0004]

Encoding — recovery block R[0]:
R[0][0] = gf_mul(1, 0x0001) XOR gf_mul(0x8016, 0x0003)
        = 0x0001 XOR 0x8017 = 0x8016
R[0][1] = gf_mul(1, 0x0002) XOR gf_mul(0x8016, 0x0004)
        = 0x0002 XOR 0x0002 = 0x0000
R[0] as words: [0x8016, 0x0000]
R[0] as bytes (uint16 LE): [0x16, 0x80, 0x00, 0x00]

Intermediate GF(2^16) values used above:
   gf_mul(0x8016, 0x0003) = gf_mul(0x8016, 2) XOR 0x8016
                          = 0x0001 XOR 0x8016 = 0x8017
   gf_mul(0x8016, 0x0004) = gf_mul(gf_mul(0x8016, 2), 2)
                          = gf_mul(0x0001, 2) = 0x0002

Reconstruction when B[1] is lost (available: [(0,B[0]), (2,R[0])]):

position i  chunk idx  type        matrix row
----------  ---------  ----------  ————————————————————————————————————
    0           0      data        [1, 0]        identity col 0
    1           2      recovery    [1, 0x8016]   Cauchy row C[0]

           col 0   col 1
           ———————  ———————
A[0]   [    1,      0     ]
A[1]   [    1,    0x8016  ]

det(A) = gf_mul(1, 0x8016) XOR gf_mul(0, 1) = 0x8016
gf_inv(0x8016) = 2  (since gf_mul(0x8016, 2) = 1)

A_inv = gf_inv(det) * [[0x8016, 0], [1, 1]]
```

```
            col 0   col 1
            ————————  ————————
A_inv[0]  [  1,      0  ]
A_inv[1]  [  2,      2  ]
```

Recover W[1] using row p=1 of A_inv:

```
recovered[1][0] = gf_mul(2, 0x0001) XOR gf_mul(2, 0x8016)
                = 0x0002 XOR 0x0001 = 0x0003
recovered[1][1] = gf_mul(2, 0x0002) XOR gf_mul(2, 0x0000)
                = 0x0004 XOR 0x0000 = 0x0004
```

recovered W[1] = [0x0003, 0x0004] matches original B[1].
Implementations MUST verify against these values.

7. Compression (Core)

7.1. Algorithm Identifiers

```
0x00  None (identity)                          (MUST support)
0x01  zstd [RFC8878]                            (MUST support)
0x02  brotli [RFC7932]                          (SHOULD support)
0x03  lz4 frame format [LZ4-Frame]              (SHOULD support)
      Wire format: LZ4 Frame Format as defined in the LZ4
      Frame Description specification.  Implementations MUST
      use frame format, not raw block format.
0x80+ Vendor-specific
```

Decoder support requirements:

A conforming Core decoder MUST support algorithms 0x00 and 0x01. A decoder that does not implement a declared algorithm (0x02, 0x03, or vendor-specific) MUST report "unsupported compression algorithm: 0xXX" as a distinct error and MUST NOT produce any output for that file. Such a decoder is still considered conforming: files using 0x02/0x03 are valid Core files, but they are only fully decodable by implementations that support the declared algorithm.

The set of decodable files depends on the decoder's supported algorithm set. Encoders choosing algorithms beyond 0x01 accept reduced decoder compatibility. Encoders SHOULD prefer 0x01 when maximum decoder compatibility is required.

7.2. Algorithm Selection Guidance

Non-normative encoder guidance.

Inner format class	Suggested algorithm
Already-compressed content	0x00 (none)
Text, JSON, XML, source code	0x01 (zstd)
Binary executables	0x01 (zstd)
Scientific / sensor data	0x01 (zstd, high level)
Latency-critical	0x03 (lz4)

Note: encoders MAY use any zstd compression level (including high levels such as 19) under algorithm ID 0x01; the compression level is an encoder-local parameter, not a wire-format distinction. All zstd levels produce valid RFC 8878 frames decodable by any zstd implementation.

Compressibility test: take first min(inner_size, 1,048,576) bytes of inner content; compress at default level; if result > 95% of input size, use algorithm 0x00 for the entire file.

For SFC Directory: apply the compressibility test to the inner content stream as a whole (Manifest + file data), since the declared algorithm applies uniformly to all chunks.

7.3. Per-Chunk Independence Requirement

Each chunk MUST be compressed independently. Stream compressors MUST reset state between chunks. Cross-chunk contexts PROHIBITED. A decoder MUST be able to decompress any single chunk payload without requiring any other chunk, provided it supports the declared compression algorithm. If the declared algorithm is not supported, the decoder MUST report the unsupported algorithm error per Section 7.1 rather than attempting to decompress.

8. Integrity Verification (Core)

8.1. Per-Chunk Integrity

Each chunk carries a 32-byte BLAKE3 hash of its header and payload bytes. BLAKE3 is the sole required hash algorithm in version 0.1.

8.2. Global File Integrity

The Global Header field at offset 285 carries the 32-byte BLAKE3 hash of the complete inner content, decompressed and trimmed to Inner File Size.

For standard files: BLAKE3 of the file’s bytes.

For SFC Directory: BLAKE3 of the concatenation of all Manifest bytes followed by all directory file bytes in Manifest order. This is identical to the standard definition because the inner content is Manifest_bytes || file_0_bytes || ... || file_{F-1}_bytes and Inner File Size = manifest_size + sum(file_sizes).

After full reassembly, decoders MUST verify this hash (D5f) and MUST report an error on mismatch.

8.3. Partial File Integrity

When presenting output without global hash verification, decoders MUST:

- Label output as unverified at the container level.
- State that the global file hash has NOT been checked.
- Report which chunk index ranges are absent or unverified.

For SFC/P5 partial extraction: each extracted file that has passed per-file BLAKE3 verification MAY be labeled as individually verified even when the container remains unverified (Section 16.9).

8.4. Integrity vs. Authenticity

SFC’s BLAKE3 hashes detect accidental corruption and tampering under trusted metadata. They do NOT provide origin authentication. An adversary with write access can replace both payload and hash.

Implementations MUST NOT describe SFC integrity as "tamper-proof", "authenticated", or "secure against active adversaries".

9. Reassembly Procedure (Core)

9.1. Full Reassembly

Full reassembly applies when V >= N and the Trailer hash has been

verified (D2c passed).

1. Ensure all N decompressed data blocks are available.
   Section 6.3 handles decompression and RS reconstruction;
   on entry to step 2 all N blocks are decompressed S-byte
   sequences. (The last data chunk was zero-padded to S bytes
   before compression per Section 5.2; trimming to Inner File
   Size occurs in step 3.) This step is informational when §9.1
   is entered from §6.3 step g or step h, which already guarantee
   all N blocks are available; in that path, proceed to step 2.
2. For i = 0 to N-1: append decompressed block i to output.
3. Trim output to Inner File Size.
4. Verify output against global file hash (D5f).
5. For standard files:
   a. Sanitize Inner Filename per Section 4.8.
   b. Write to designated output directory.
6. For SFC Directory (0x0050): apply Section 16.7.

9.2. Unverified Reconstruction

Unverified reconstruction applies when V >= N but the Terminal Segment cannot be located (D2c remains unverified).

This is NOT partial reassembly. The decoder has sufficient chunks to reconstruct the complete inner content; what is absent is the Trailer hash providing end-to-end container metadata authentication.

Procedure:
1. Apply Section 6.3 to decompress and reconstruct all N data
   blocks (decompression before RS, per D5b/D5c).
2. Concatenate decompressed blocks; trim to Inner File Size.
3. D5f (global file hash): verify the reconstructed content
   against the Global Header's BLAKE3 hash. If this passes,
   the inner content's integrity is verified against the Global
   Header hash, even without Trailer verification.
   MUST communicate to the caller that reconstruction succeeded,
   inner content hash is verified, and container metadata is
   unverified (Trailer absent or unlocatable). The exact form
   of communication (error code, status flag, structured return
   value) is implementation-defined; the distinction from fully
   verified output MUST be observable through the API's return
   value or output metadata, not only through diagnostic logging.
4. For standard files: sanitize filename and write output.
5. For SFC Directory: apply Section 16.7.

The distinction from full reassembly: the Trailer's authentication of the Global Header Region (D2c) has not been verified. An adversary who could replace the Global Header bytes and also forge or omit the Trailer would be undetected. Under trusted channel assumptions, unverified reconstruction is acceptable.

Output MUST be labeled: "reconstructed; container metadata unverified (Trailer absent)".

9.3. Partial Reassembly

Partial reassembly applies when V < N: not enough chunks are available to reconstruct the complete inner content.

IMPORTANT: partial reassembly delivers a byte-level prefix of the inner content. Whether this is useful at the application level depends entirely on the inner format.

Procedure:
1. Identify the longest contiguous run of verified data chunks
   starting from index 0. If chunk index 0 is not in the

working set, this run has length zero and partial reassembly produces zero output bytes. Decoders SHOULD report "no contiguous prefix available" in this case.

2. Decompress each chunk in the run independently (Section 7.3).
3. Concatenate decompressed blocks.
4. If chunk N-1 is included: trim to Inner File Size.
   Otherwise: emit all bytes without trimming.
5. Label output as partial and unverified (Section 8.3).
6. Report missing chunk index ranges.

For SFC Directory: if V < N, Core partial reassembly delivers a byte prefix of the inner content. File-level extraction from partial data is defined in Section 16.8 and is a separate operation that may be possible if Manifest chunks are among the available chunks and some files' chunk ranges are complete.

9.4. Error Handling

Decoders MUST detect the following conditions and MUST make each distinguishable from the others in diagnostic output. The exact form of reporting (error codes, log messages, API returns) is implementation-defined:

- Invalid magic bytes.
- Unsupported major version.
- Header length H out of bounds.
- Duplicate known TLV tag.
- TLV value overruns header boundary.
- Known TLV with unexpected length for its tag.
- Known Profile TLV without corresponding Profile bit (rule g).
- Non-zero bytes after Inner Filename null terminator.
- Non-zero Flags bits 1-3 (Section 4.6).
- Non-zero bytes in Trailer bytes 4-7.
- Trailer BLAKE3 hash mismatch.
- File UUID mismatch within UUID group.
- Chunk BLAKE3 hash failure (chunk index).
- Chunk index out of range (index, N+M).
- Unknown chunk type (chunk index, observed value).
- Chunk header algorithm ID mismatch with Global Header (D4d).
- Non-zero reserved bytes (location, offset, values).
- Insufficient chunks (V available, N required, missing indices).
- RS reconstruction failure (singular matrix; no valid chunk subset found).
- Global file hash mismatch after full reconstruction.
- Truncated chunk (chunk index).
- Global Header conflict in SFC/P2.
- Terminal Segment conflict in SFC/P2 (Section 13.1).
- Missing or invalid Segment Header in SFC/P2 segment.
- Non-zero reserved bytes in Segment Header (offsets 13-15).
- Segment index >= total segment count (Section 13.1).
- Duplicate segment index across sibling segments (Section 13.1).
- Inconsistent total segment count across segments (Section 13.1).
- Invalid terminal flag value (not 0x00 or 0x01; Section 13.1).
- Chunk end marker "/CHK" invalid (D3e).
- Decompressed chunk size != S bytes.
- Compressed payload length exceeds 2*S (Section 18.3).
- Version mismatch across SFC/P2 segments (Section 13.5).
- Missing Global Header.
- Terminal Segment not found (warning; unverified reconstruction per Section 9.2 if V >= N, or partial reassembly if V < N).
- Dataset inconsistency: Contaminated Duplicate (Section 9.5).
- Empty inner filename.
- Profile MUST-level violation.
- SFC/P2 and SFC/P3 bits both set in the same file (Section 14).
- SPLIT_TRANSPORT bit (bit 0) set without SFC/P2 Profile bit (bit 5) set: decoder MUST treat as a format error and halt.

A file with bit 0 set but bit 5 absent is structurally ambiguous; the Segment Header MUST be skipped (bit 0 signals its presence) but P2 semantics cannot be applied.

- Manifest BLAKE3 hash failure (SFC/P5; Section 16.2).
- Case collision in Manifest paths (SFC/P5; Section 16.3).
- Manifest entry offset/size inconsistency (SFC/P5; Section 16.7).
- Erasure algorithm 0x00 with M > 0 (Section 6.1).
- Non-zero erasure algorithm with M = 0 (Section 6.1).
- Manifest entry path length L = 0 (Section 16.7).
- Nominal chunk size S is odd (Section 6.4).
- Unsupported erasure algorithm: 0xXX (Section 6.1).
- Unsupported compression algorithm: 0xXX (Section 7.1).
- Priority count P > N (Section 4.5 rule a).
- Duplicate index in priority list (Section 4.5 rule b).
- Priority index out of range [0, N-1] (Section 4.5 rule c).
- Field value below minimum (S = 0, or other violations of Section 18.3 lower bounds).
- Inner File Size = 0 with N != 1 (Section 18.3).

Error severity classification:

Chunk-level errors (invalid hash, reserved bytes, index out of range, algorithm mismatch, unknown type, end marker invalid, decompressed size mismatch, compressed length exceeds 2*S): decoder SHOULD discard the affected chunk and continue processing remaining chunks.

File-level errors (invalid magic, unsupported version, header length out of bounds, Global Header conflict, Trailer hash mismatch, non-zero Trailer reserved bytes, unsupported algorithm, erasure 0x00 with M>0, non-zero erasure algorithm with M=0, S is odd, Inner File Size = 0 with N != 1, version mismatch across segments, missing/invalid Segment Header, Segment Header reserved bytes, segment index/count violations, duplicate known TLV, TLV overrun, Profile TLV without bit, Manifest hash failure, Manifest offset/size inconsistency, Manifest entry path length L=0): decoder MUST halt.

9.5. Duplicate Chunk Handling

Case A — Benign Duplicate (same content, same hash):
Silently discard all but one copy.

Case B — Contaminated Duplicate (same index, different content):

B1: Exactly one copy passes BLAKE3 verification.
Discard failing copy; use passing copy; log event.

B2: Both copies pass BLAKE3 verification.
Both MUST be discarded. Report "dataset inconsistency" with chunk index. The chunk-level integrity model guarantees exactly one chunk per index per encoding; two valid differing chunks indicate data set contamination or malicious substitution. Note: this detects inconsistency within the chunk-level integrity model; it does not detect cryptographic forgery, which requires origin authentication. Attempt RS recovery if M > 0 and V >= N after discards.

B3: Neither copy passes BLAKE3 verification.
Discard both; report two corrupt chunks at same index.

10. Encoder Behavior (Core)

10.1. Chunking Strategy

N = ceil(Inner File Size / S)

Exception: when Inner File Size = 0, N MUST be 1 regardless of S. See Section 18.3 for the zero-length inner content encoding rule.

Suggested S values (non-normative):

Inner content size	Suggested S
< 1 MB	64 KB
1 MB -- 100 MB	1 MB
100 MB -- 1 GB	4 MB
> 1 GB	16 MB

10.2. Encoder Warnings (Non-Normative Guidance)

The following warnings are implementation guidance for encoder user interfaces. They are NOT wire-format requirements and are NOT testable against the encoded output. An encoder that omits these warnings still produces conforming SFC files.

Warning A — Already-compressed format:
Applies to: JPEG (any), MP4, ZIP, gzip, zstd data, WebP.
Recommended action: set compression algorithm to 0x00.
Suggested message: "Compression disabled: inner format already compressed."

Warning B — No partial utility:
Applies to: ZIP (0x0040), gzip (0x0041), PDF (0x0100), PNG non-interlaced (0x0024), WebP (0x0026).
Suggested message: "Under partial delivery, this format provides no application-level utility until delivery is complete. Consider converting (SFC/P4)."

Note: JPEG Baseline (0x0020) is NOT included in Warning B. Although its partial utility is limited to the top portion of the image (P1 Class S), it does provide non-zero utility under partial delivery. See Section 12.3.

10.3. Encoder Conformance Checklist

A conforming Core encoder MUST:

- Generate a fresh version 4 (random) UUID [RFC9562] per encoding; MUST NOT reuse the same UUID for distinct encodings (Section 4.1).
- Write the preamble with major version = 0, minor version = 1 (Section 3.1).
- Set Flags bits 1-3 to zero (reserved permanently; Section 4.6).
- Set Flags bits 9-15 to zero in version 0.1 (Section 4.6).
- Compute N = ceil(Inner File Size / S); when Inner File Size = 0, set N = 1 (Section 10.1).
- Ensure N + M does not exceed 65,535 (Section 6.2).
- Use a positive, even S (Section 6.4).
- Zero-pad data block N-1 to exactly S bytes before compression and RS input (Section 6.4).
- Apply RS encoding on uncompressed S-byte blocks; compress each recovery block independently afterwards (Section 6.4).
- When M = 0: set erasure algorithm ID to 0x00 (Section 6.1).
- When M > 0: MUST NOT use erasure algorithm 0x00; MUST use algorithm 0x01 (RS GF(2^16)) or a vendor-specific algorithm (0x80+) (Section 6.1).
- Set the same erasure algorithm ID in both the Global Header and each chunk header (Section 5.1).
- Set the same compression algorithm ID in both the Global Header and each chunk header; IDs MUST agree (Section 5.1).

- Compute the global BLAKE3 hash over the trimmed inner content (before zero-padding the last block) and store it in the Global Header (Section 8.2).
- Compute the Trailer BLAKE3 hash over the exact bytes of the Global Header Region (H+4 bytes) and store it in the Trailer (Section 3.4).
- Assign chunk indices 0..N-1 to data chunks and N..N+M-1 to recovery chunks (Section 5.2, Section 5.4).
- Write zero bytes in all reserved fields: chunk header offsets 34-47, Trailer offsets 4-7 and 48-63 (Section 5.5).
- Exclude from the Inner Filename: '/' (0x2F), '\' (0x5C), bytes in range 0x00-0x1F, and the sequences ".." and "." as the entire effective filename (Section 4.8).
- Null-pad the Inner Filename field to exactly 255 bytes; all bytes after the null terminator MUST be zero (Section 4.8).
- NOT produce files where erasure algorithm = 0x00 and M > 0 (Section 6.1).
- NOT produce files with an odd S value (Section 6.4).
- NOT produce a priority list with P > N (Section 4.5).
- NOT produce duplicate indices in the priority list (Section 4.5).
- NOT produce priority index values outside [0, N-1] (Section 4.5).
- NOT produce a header where 4P bytes plus fixed fields and any TLV bytes would cause H to exceed 65,536 bytes (Section 4.5 rule g).

A conforming Core encoder SHOULD:

- Generate UUIDv4 using a cryptographically secure random source (Section 4.1).
- Select S from the guidance table in Section 10.1.
- Finalize the Global Header Region before computing the Trailer hash, so the hash covers the complete header bytes including the global file hash (Section 3.4).
- Set the Trailer encoder timestamp to the current time in seconds since the Unix epoch (Section 3.4); writing zero is permitted but provides no diagnostic value.

11. Decoder Behavior (Core)

A conforming Core decoder MUST:

- Follow the validation dependencies of Section 3.5.
- Enforce TLV Contract rules (Section 3.2), including rule g.
- Group files by UUID before intra-group validation (Rule 2).
- Verify per-chunk 32-byte BLAKE3 hashes; discard failures.
- Enforce reserved-byte hard error per Section 5.5.
- Treat non-zero Trailer bytes 4-7 as format error.
- Discard chunks with unknown type values.
- When Global Header is available: verify chunk bytes 32-33 match Global Header algorithm fields (Section 5.1).
- Support erasure reconstruction (algorithm 0x01).
- Support decompression of algorithms 0x00 and 0x01 (zstd).
- Report "unsupported compression algorithm: 0xXX" for any algorithm it does not implement; produce no output for that file (Section 7.1).
- Verify chunk end marker "/CHK" (D3e).
- Perform full reassembly (Section 9.1) or unverified reconstruction (Section 9.2) and verify global hash (D5f).
- Sanitize Inner Filename per Section 4.8.
- Detect all error conditions in Section 9.4 and make each distinguishable in diagnostic output.
- Handle all duplicate chunk cases per Section 9.5.
- Skip unknown TLV tags without failing.
- Ignore non-zero bits 9-15 in Flags without failing.
- When SPLIT_TRANSPORT flag (bit 0) is set: skip the 16-byte

Segment Header before parsing chunks (Section 13.1).
- Support partial reassembly (Section 9.3) with correct labels (design goal G3).

A conforming Core decoder SHOULD:
- Support decompression of algorithms 0x02 (brotli) and 0x03 (lz4 frame format).

Part II: Profiles

12. Profile 1 — Image (SFC/P1)

12.1. Scope and Limitations

SFC/P1 applies when the inner file is an image. Setting the SFC/P1 bit declares conformance to the classification in Section 12.2 and priority chunk guidance in Section 12.4. SFC/P1 does NOT cause progressive rendering; that is a property of the inner format itself.

12.2. Inner Format Classification

Class P — Spatially progressive: partial data yields a spatially complete image at reduced quality.
- 0x0022 JPEG 2000
- 0x0023 JPEG XL

Class Q — Quality-layer progressive: partial data may yield a quality-reduced complete image. Behavior is encoding-dependent and viewer-dependent.
- 0x0021 JPEG Progressive

Class S — Sequential (top-down): partial data yields the top portion at full quality.
- 0x0020 JPEG Baseline

Class I — Interlaced: partial data yields a low-resolution full-image preview.
- 0x0025 PNG Adam7 interlaced

Class N — No application-level partial utility: full file required.
- 0x0024 PNG non-interlaced
- 0x0026 WebP (baseline)

12.3. Partial Utility by Class

Class P: Decoders SHOULD decode and display as reduced-quality complete image, updating as chunks arrive.

Class Q: Decoders MAY attempt display; MUST inform user result is encoding-dependent and may not display correctly in all viewers.

Class S: Decoders SHOULD display available top portion with clear boundary indicator.

Class I: Decoders SHOULD display available coarse preview.

Class N: Decoders MUST NOT attempt to display partial data as an image; SHOULD show a progress indicator.

12.4. Priority Chunks (SFC/P1)

For Class P formats: encoders MUST verify codestream structure to identify the chunk indices carrying the first quality layer or spatial resolution layer. Encoders MUST set P > 0 and list those chunk indices. Encoders MUST NOT assume a fixed byte layout; byte offsets MUST be derived from codestream inspection.

For Class S formats: encoders MUST set priority chunks to indices 0 through max(0, ceil(N*0.1) - 1), covering approximately the first 10% of chunks (the top rows of the image). The max(0,...) guard is unreachable given N 1 (Section 18.3) but is retained for defensive clarity.

P1 and P2 interaction: when SFC/P1 and SFC/P2 are combined, the priority list is advisory only. SFC/P2 receivers must download complete segment files; they cannot request individual chunks. Decoders operating in P2 mode MUST NOT treat the priority list as a mechanism for targeted chunk retrieval. Encoders SHOULD place priority chunks in early segments where possible, but this is a best-effort placement with no protocol enforcement.

13. Profile 2 — Split Transport (SFC/P2)

13.1. Segment Structure and Terminal Segment Discovery

Each segment is a self-contained SFC/P2 segment file containing:

- SFC Magic and version bytes (identical preamble as a standard SFC file).
- A complete copy of the Global Header — raw bytes of the Global Header Region MUST be byte-for-byte identical across all segments of the same File UUID (enforced by canonical TLV ordering, Section 3.2 rule d).
- A Segment Header (see below).
- One or more data and/or recovery chunks.
- Trailer: present in the Terminal Segment only; absent in Non-Terminal Segments (D2c deferred; Section 3.4).

A Non-Terminal Segment is structurally valid but does NOT constitute a complete SFC file: it lacks the Trailer required for full container authentication.

The SPLIT_TRANSPORT flag (bit 0) MUST be set in every segment. Profile bit 5 (SFC/P2) MUST also be set.

All other profile bits that apply to the encoded content MUST be preserved in every segment’s Global Header Flags field. The SPLIT_TRANSPORT and P2 bits are OR-added to the existing profile bits; they do NOT replace them. For example, a P2-split SFC/P5 directory container MUST have both bit 5 (SFC/P2) and bit 8 (SFC/P5) set in every segment.

--- Segment Header ---

The Segment Header immediately follows the Global Header Region (after the last byte of the Global Header, before the first chunk). It is NOT part of the Global Header Region and is NOT included in the Trailer BLAKE3 hash.

Offset	Length	Field
0	4	"SEG\0" (0x53 0x45 0x47 0x00) Note: the fourth byte is 0x00 (binary null), part of the magic value — not a C string terminator. Implementations MUST match all

		four bytes; string-comparison APIs that stop at 0x00 will mis-identify the magic.
4	4	Segment index (uint32 LE)
8	4	Total segment count (uint32 LE)
12	1	Terminal segment flag 0x00 = non-terminal 0x01 = terminal
13	3	Reserved (MUST be zero)

Total: 16 bytes.

Every SFC/P2 segment MUST include a valid Segment Header. Decoders MUST verify the "SEG\0" magic and reject segments without it. Decoders MUST treat non-zero reserved bytes (offsets 13-15) as a format error for segments with version <= the decoder's implemented version (same rule as Section 5.5).

Segment Header validation (decoders MUST enforce):

- Segment index MUST be less than total segment count.
- Total segment count MUST be >= 1.
- Across all segments of the same UUID, total segment count values MUST be identical; a mismatch is a format error.
- Across all segments of the same UUID, segment indices MUST be unique; a duplicate index is a format error.
- Terminal flag MUST be 0x00 or 0x01; other values are a format error.

--- Terminal Segment Detection ---

Terminal Segment detection requires all of:

- Segment Header terminal flag = 0x01.
- Final 64 bytes form a complete Trailer block.
- First 4 bytes of that block are "TRLR" magic.
- Trailer BLAKE3 hash at offset 8 passes verification.

Terminal Segment Conflict Table:

Scenario	Action
One segment has terminal flag 0x01 AND passes Trailer detection.	Terminal found; proceed.
One segment has terminal flag 0x01 but fails Trailer detection.	Hard error: "Terminal flag present; Trailer invalid." Halt.
No terminal flag set; one segment passes Trailer detection.	Hard error: "Terminal by Trailer only; missing Segment Header flag." Halt.
No terminal flag set; two or more pass Trailer detection.	Hard error: "Multiple candidate Terminals." Halt.
Two or more segments have terminal flag 0x01.	Hard error: "Multiple Terminal flags." Halt.
No terminal flag and none pass Trailer detection.	Warning: "Terminal Segment not found." If V>=N: unverified reconstruction (Section 9.2). If V<N: partial reassembly (Section 9.3).

13.2. Segment Naming Convention (Advisory)

Segments SHOULD be named:
<base>.<uuid8>.<segment_index_4d>.sfc

Example: report.f47ac10b.0000.sfc, report.f47ac10b.0001.sfc

Decoders MUST NOT require this convention.

13.3. Sibling Discovery and Multi-File Operation

Single-encoding discovery (for one UUID group):

Step 1: Name-based — enumerate <base>.<uuid8>.*.sfc in directory.
Tools that use a different segment naming convention (e.g.,
<base>-NN.sfc) MUST NOT rely on Step 1 and MUST proceed
directly to Step 2.
Step 2: UUID scan — read first 28 bytes of each .sfc file
(Preamble 8 + H field 4 + UUID 16); accept UUID matches.
MUST NOT read full file contents during scan.
The seed file (the fragment provided by the user) MAY be
read in full to determine whether it is a P2 segment and
to extract its UUID; the MUST NOT applies to the directory
scan of candidate siblings.
Step 3: User-directed — if V < N, report shortfall and prompt
user for additional files or directories.
Step 4: Watch mode (OPTIONAL) — monitor directory for new .sfc
files; retry on new match.

Multi-encoding operation (e.g., decoder *.sfc):

When presented with multiple .sfc files simultaneously:
1. Read first 28 bytes of each file; extract UUID.
2. Group files by UUID. Different UUIDs are separate encodings;
this is NOT an error. Grouping precedes all validation.
3. For each UUID group, apply single-encoding discovery and
reconstruction independently.
4. Report results per UUID group.

13.4. Conflict Resolution

All segments of the same UUID MUST have byte-for-byte identical
Global Header Regions. (The Segment Header is NOT part of the
Global Header Region and is expected to differ between segments.)

Case 1 — Conflicting Global Headers: report conflict with
differing byte range and source files; halt; require
user resolution. MUST NOT silently resolve.
Case 2 — Contaminated Duplicates: apply Section 9.5 Case B.
Case 3 — Benign Duplicates: apply Section 9.5 Case A.

13.5. Version Skew Handling

All segments sharing the same File UUID MUST have identical major
AND minor version numbers. Decoders MUST reject a segment set
where any two segments differ in either major or minor version;
report "version mismatch across segments" and halt.

Rationale: future minor versions may assign semantics to bytes
that are reserved in version 0.1 (Section 3.1). Combining
segments with different minor versions would create ambiguity
about which byte semantics apply. Requiring an exact version
match eliminates this class of interoperability issues entirely.

13.6. Incomplete Segment Handling

Segment truncated mid-chunk ("/CHK" trailer absent):
- Complete, verified chunks within segment MAY be used.

- Incomplete terminal chunk MUST be discarded.
- Report by filename and discarded chunk index.

14. Profile 3 — HTTP Transport Hints (SFC/P3)

Applies to complete single-file SFC objects only.
NOT applicable to SFC/P2 segments.

Mutual exclusion with SFC/P2: Encoders MUST NOT set the SFC/P3 Profile bit (bit 6) or include TLV 0x0020 in any file where the SPLIT_TRANSPORT flag (bit 0) or SFC/P2 Profile bit (bit 5) is set. A decoder encountering both SFC/P2 and SFC/P3 bits set in the same file MUST report "SFC/P2 and SFC/P3 bits both set" (Section 9.4), MUST ignore P3 semantics, and MUST proceed using SFC/P2 rules.

Servers SHOULD include Content-Type: application/sfc and Accept-Ranges: bytes in HTTP responses serving SFC files.

Optional TLV 0x0020: array of (N+M) uint64 LE byte offsets of each chunk's "CHK\0" from file start. Enables targeted HTTP Range requests without linear scan of chunk length fields.

Applicability constraint: TLV 0x0020 has L = 8*(N+M) bytes. Given the TLV value length limit of 16,384 bytes (Section 18.3), this TLV is only encodable when N+M <= 2048. Encoders MUST NOT include TLV 0x0020 when N+M > 2048. For larger files, clients MUST locate chunk boundaries by scanning chunk payload length fields sequentially from the start of the file.

15. Profile 4 — Preprocessing (SFC/P4)

Encoders MAY offer format-aware conversion before encoding.

From	To	P1 class improvement
JPEG Baseline	JPEG 2000/XL	S → P
PNG non-interlaced	PNG Adam7	N → I
MP4 (moov@EOF)	Fragmented MP4	None → streamable
ZIP	tar + zstd	None → sequential

Requirements: SHOULD inform the user that conversion is available; MUST NOT modify the source file on disk; MUST encode the converted content as the inner payload; MUST update Inner Format ID to the converted format; MUST store the original (pre-conversion) format ID in TLV 0x0030 (uint16 LE); SHOULD report the conversion performed.

16. Profile 5 — Directory (SFC/P5)

16.1. Overview

SFC/P5 enables encapsulation of an entire directory tree into a single SFC encoding. The directory contents are encoded as a single inner content stream: the Manifest occupies the leading portion, followed by the concatenated bytes of all files in the directory.

The Manifest is NOT a separate out-of-band structure. It is part of the inner content and subject to the same chunking, compression, Reed-Solomon erasure coding, and per-chunk integrity rules as all other data. Specifically:

- The Manifest spans data chunks 0..K-1 where K = ceil(manifest_size / S). When manifest_size % S != 0, chunk K-1 is a mixed chunk containing the tail of the Manifest and leading bytes of the first file; boundaries within a chunk are determined by byte offsets, not chunk structure.
- The declared compression algorithm applies to Manifest chunks identically to file data chunks.
- The Reed-Solomon code protects Manifest chunks identically to file data chunks. Missing Manifest chunks are recovered via RS within the M-loss budget.
- The global BLAKE3 hash covers the complete inner content.

The Inner Format ID MUST be 0x0050 when SFC/P5 is active. The SFC/P5 Profile bit (bit 8 of Flags) MUST be set.

SFC/P5 requires S 8 bytes so that the Manifest Magic (bytes 0-3) and body-length B (bytes 4-7) are both fully contained within chunk 0, as required by § 16.2 and § 16.8 step 2a. Encoders MUST reject S < 8 for SFC/P5 files. (In practice, S 64 is the minimum for non-identity compression per § 18.3.)

16.2. Manifest Structure

The Manifest is a variable-length structure stored as the leading bytes of the inner content stream.

Offset	Length	Field
0	4	Magic: 0x4D 0x46 0x53 0x54 ("MFST")
4	4	Body length B (uint32 LE). B = 4 + (total byte count of all F file entries). Equivalently, B is the byte count from offset 8 through the last byte of the last file entry.
8	4	File count F (uint32 LE).
12	var	F file entries (Section 16.3). Total byte count of entries = B - 4.
8+B	32	BLAKE3 hash of bytes 0..8+B-1 inclusive (covering Magic, B, F, and all file entries).

manifest_size = 8 + B + 32

Derivation check: if F=5 and total entry bytes = 346, then B = 4 + 346 = 350, manifest_size = 8 + 350 + 32 = 390. K = ceil(390 / S).

Decoder requirements:
- Decompress chunk 0 to read Magic and B (at minimum).
- Compute K = ceil(manifest_size / S) = ceil((8+B+32) / S).
- Collect and decompress all K leading data chunks (0..K-1) to obtain the complete Manifest bytes.
- Verify the Manifest BLAKE3 hash at offset 8+B before using any file entries. A failing hash is a format error; report "Manifest BLAKE3 hash failure" and halt extraction.

The Manifest BLAKE3 provides self-contained integrity for the Manifest structure, independently of the Trailer. It allows safe use of Manifest data for partial extraction (Section 16.8) before the global hash can be verified.

16.3. File Entry Format

Each file entry within the Manifest:

Offset	Length	Field

0	2	Path length L in bytes (uint16 LE)
2	L	Relative file path (UTF-8)
2+L	8	Byte offset of this file's content in the inner content stream (uint64 LE). This is the position of the first byte of this file's data relative to the start of the inner content (i.e., relative to the start of the Manifest).
10+L	8	File size in bytes (uint64 LE; actual byte count, not including chunk padding).
18+L	32	BLAKE3 hash of this file's content bytes.
50+L	2	Inner Format ID of this file (uint16 LE).

Total entry size: 52 + L bytes.

Byte offset derivation: for the first file,
   byte_offset_0 = manifest_size
For each subsequent file i > 0:
   byte_offset_i = byte_offset_{i-1} + file_size_{i-1}
Files are concatenated without inter-file padding.

Chunk index derivation (for use in partial extraction):
   If file_size = 0: no chunk range is associated with this file.
   The file is written as empty; no chunk data is needed.
   If file_size > 0:
     first_chunk = byte_offset / S   (integer division)
     last_chunk = (byte_offset + file_size - 1) / S

Note: files are not required to start at chunk boundaries.
A single chunk may contain bytes from multiple files or from
both the Manifest tail and a file's beginning.

Path format:
- Relative to the encoded directory root.
- Components separated by '/' (0x2F) regardless of host OS.
- MUST NOT begin with '/'.
- MUST NOT contain the component ".." (path traversal).
- MUST NOT contain the component "." (current-directory alias).
- MUST NOT contain '\' (0x5C).
- MUST NOT contain any byte in range 0x00-0x1F.

Section 4.8 sanitization applies to each path component
(split on '/') individually. The forbidden-component rules
above (including "." and "..") mirror the Section 4.8 rules
applied at the component level. Decoders MUST reconstruct the
directory hierarchy by creating intermediate directories.

Case collision rule: two paths in the same Manifest are in
collision if they are identical after Unicode Simple Case Folding
as defined in Unicode 15.1.0 [Unicode-CaseFolding], applied code
point by code point using the "C" (common) and "S" (simple)
mappings. Implementations using a later Unicode version MUST
apply the same folding behavior for code points defined in Unicode
15.1.0. Encoders MUST reject encoding if any two paths are in
collision (Section 16.6). Decoders MUST report a collision as an
error if detected during extraction and MUST NOT overwrite an
already-written file with a colliding path (Section 16.7).

16.4. Inner Content Layout

The inner content for an SFC Directory file is:

```
inner_content = Manifest_bytes || File_0_bytes || File_1_bytes
             || ... || File_{F-1}_bytes
```

This concatenation is chunked, compressed, and erasure-coded
using standard Core rules, identically to any other SFC file.

The standard chunk assignment applies: data chunk i carries inner content bytes [i*S .. (i+1)*S - 1].

The Manifest begins at byte 0 of chunk 0. K = ceil(manifest_size / S) chunks are required to hold the complete Manifest. However, when manifest_size % S != 0, chunk K-1 is a mixed chunk: its decompressed content contains the tail of the Manifest bytes followed by the leading bytes of the first file's data. There is no structural separation between Manifest content and file data within a chunk; boundaries are determined purely by byte offsets.

File data begins at byte offset manifest_size within the inner content stream. This offset falls within chunk K-1 when manifest_size is not a multiple of S, or at the start of chunk K when manifest_size is exactly a multiple of S.

Inner File Size:
   manifest_size + file_0_size + file_1_size + ... + file_{F-1}_size

Global BLAKE3 hash: BLAKE3 of the decompressed, trimmed inner content (Manifest bytes concatenated with all file bytes).

16.5. Chunk Allocation

Chunk index ranges (all inclusive):

- Data chunks 0..K-1: span Manifest bytes.
  K = ceil(manifest_size / S).
  Chunk K-1 may also contain leading bytes of the first file when manifest_size % S != 0 (mixed chunk).
- Data chunks covering file data: determined per-file using the byte_offset and file_size fields from the Manifest per Section 16.3. Multiple files may share a chunk boundary.
  N = ceil(Inner File Size / S).
- Recovery chunks N..N+M-1: Reed-Solomon parity over all N data chunks (Manifest + file data combined).

The last data chunk (N-1) is zero-padded to S bytes before compression and RS input.

For SFC/P2: encoders MUST place chunks 0..K-1 in the first segment. Without the Manifest (chunks 0..K-1), receivers cannot determine file boundaries, byte offsets, or per-file hashes; file-level partial extraction is impossible until the Manifest arrives. Note that when chunk K-1 is a mixed chunk, it contains the start of the first file's data; including it in the first segment enables early extraction of that file if its remaining chunks are also available.

16.6. Encoder Behavior (SFC/P5)

Encoders MUST:

- Recursively traverse the source directory.
- Skip symbolic links silently.
- Skip empty directories silently.
- Collect all regular files.
- If no regular files remain after traversal (e.g., directory contains only empty subdirectories and/or symbolic links), MUST report "no encodable regular files" and refuse to encode. F >= 1 is a hard requirement (Section 18.3).
- Check for case collisions: compute the case-folded form of each file's relative path. If any two paths have the same case-folded form, MUST report an error and refuse to encode. This prevents extraction failures on case-insensitive filesystems.
- Sort files in ascending lexicographic order by relative path using UTF-8 byte ordering. This ordering is MANDATORY and

MUST NOT be altered for alignment purposes or any other reason. The Manifest order is normative and determines byte_offset values, chunk allocation, and the inner content byte sequence.

- Compute the BLAKE3 hash of each file's content.
- Compute byte_offset for each file per Section 16.3.
- Compute manifest_size = 8 + B + 32 per Section 16.2.
- Set Inner Format ID to 0x0050 in the Global Header.
- Set the SFC/P5 Profile bit (bit 8) in Flags.
- Set Inner File Size per Section 4.5.
- Set N = ceil(Inner File Size / S).
- Apply the declared compression algorithm uniformly to ALL chunks including Manifest chunks.

Encoders SHOULD:

- Place Manifest chunks (0..K-1) in the first SFC/P2 segment.
- Warn the user when any individual file would trigger Warning B (Section 10.2) if encoded standalone.

16.7. Full Extraction

Full extraction is the normative SFC/P5 procedure following Core full reassembly (Section 9.1) or unverified reconstruction (Section 9.2), where all N data chunks are available.

1. The inner content has been fully reconstructed (all N data chunks decompressed, concatenated, trimmed to Inner File Size).
2. Parse the Manifest from inner content bytes 0..manifest_size-1:
   a. Verify Magic bytes 0-3 are "MFST".
   b. Read B from bytes 4-7.
   c. manifest_size = 8 + B + 32.
   d. Verify Manifest BLAKE3 at bytes 8+B..8+B+31.
      If fails: report "Manifest BLAKE3 hash failure"; halt.
   e. Parse F file entries from bytes 12..8+B-1.
   f. Validate file entry consistency (MUST enforce):
      - Each file entry's path length L MUST be at least 1. An entry with L=0 is a format error; report "Manifest entry path length is zero" and halt extraction.
      - byte_offset of first file MUST equal manifest_size.
      - For each subsequent file i > 0: byte_offset_i MUST equal byte_offset_{i-1} + file_size_{i-1}.
      - For each file: byte_offset + file_size MUST NOT exceed Inner File Size.
      - Violations indicate a corrupt or malicious Manifest; report "Manifest entry offset/size inconsistency" and halt extraction.
3. Determine the output directory name from the Inner Filename field of the Global Header, sanitized per Section 4.8.
4. For each file entry in Manifest order:
   a. Sanitize each path component per Section 4.8.
   b. Check for case collision with already-written files.
      If collision: the first-listed file entry in Manifest order is written; subsequent colliding entries are skipped and reported. Since § 16.6 mandates lexicographic Manifest ordering by UTF-8 byte ordering, the path that sorts earlier under UTF-8 byte ordering always survives a collision.
   c. Reconstruct directory structure; create intermediate directories as needed.
   d. If file_size = 0:
      Write an empty file to output_directory / sanitized_path.
      Verify per-file BLAKE3 hash against BLAKE3 of the empty byte string (zero input bytes).
      Label file: complete and verified.
   e. If file_size > 0:
      Extract bytes byte_offset..byte_offset+file_size-1 from the reconstructed inner content.

Verify against per-file BLAKE3 hash. If fails: report
"per-file BLAKE3 hash mismatch" with path; do not write.
Write file to: output_directory / sanitized_path.
MUST confine to output_directory.

5. After extracting all files, the global BLAKE3 hash verification (D5f from Section 9.1 step 4) provides end-to-end integrity.

16.8. Partial Extraction

Partial extraction allows individual files to be extracted before all chunks have arrived. This is the primary SFC/P5 use case for unreliable transport.

Partial extraction is distinct from Core partial reassembly (Section 9.3): it delivers complete individual files rather than a byte prefix of the inner content.

--- Step 1: Determine output directory and chunk availability ---

Derive the output directory name from the Inner Filename field of the Global Header, sanitized per Section 4.8. This is the same derivation as Section 16.7 step 3. All file writes in both Case A and Case B below MUST be confined to this output_directory.

Compute V (valid chunks in working set per Section 6.3 step f).

--- Step 2: Obtain Manifest bytes ---

First, try to read chunk 0 to learn B (body length) and compute manifest_size and K:

a. If chunk 0 is directly in the working set: decompress it per Section 7.3. Verify bytes 0-3 are "MFST" magic; if not, report "invalid Manifest magic" and halt. Read B from decompressed bytes 4-7. Verify B against hard limits (57 <= B <= 64 MB per Section 18.3); if out of range, report "Manifest body length out of bounds" and halt.
   Compute manifest_size = 8 + B + 32.
   Compute K = ceil(manifest_size / S).

b. If chunk 0 is NOT in the working set:
   - If V >= N: apply Case A below; RS will recover chunk 0.
   - If V < N: report "Manifest unavailable; file-level extraction impossible" and halt partial extraction. Core partial reassembly (Section 9.3) may deliver a byte prefix if any leading chunks are present.

Once manifest_size and K are known, check whether all K Manifest chunks (0..K-1) are in the working set:

c. If all K chunks are directly present: decompress them; concatenate to obtain Manifest bytes; proceed to Step 3.

d. If some Manifest chunks are absent:
   - If V >= N: apply Case A below; RS recovers missing chunks.
   - If V < N: report "Manifest unavailable; file-level extraction impossible" and halt.

--- Step 3: Verify Manifest ---

Verify the Manifest BLAKE3 hash at offset 8+B of the Manifest bytes. If verification fails: report "Manifest BLAKE3 hash failure" and halt extraction. Parse F file entries.

--- Case A: V >= N (full RS reconstruction possible) ---

```
When V >= N (regardless of which specific chunks are missing):
1.  Apply Section 6.3 in full: decompress all chunks in the
    working set; perform RS reconstruction to derive all N
    decompressed data blocks (Manifest and file data).
2.  Concatenate all N decompressed data blocks; trim to Inner
    File Size.  Verify the global BLAKE3 hash (D5f) against the
    Global Header's hash value.  If hash verification passes,
    the inner content integrity is confirmed.  If the Trailer
    has also been verified (D2c), the container is fully
    authenticated; otherwise label "inner content hash verified;
    container metadata unverified (Trailer absent or unverified)."
3.  All N data blocks are now available.  Extract all files using
    the procedure in Section 16.7 steps 2-5 (step 2d, Manifest
    BLAKE3 hash verification, was already performed in § 16.8
    Step 3 and need not be repeated; all other sub-steps apply).
    Each file extracted and passing its per-file BLAKE3 hash is
    labeled individually verified.

--- Case B: V < N (insufficient chunks for full RS) ---

When V < N, RS reconstruction is not possible.  Only files whose
complete chunk range is directly present in the working set can
be extracted.

Before processing any file entry, apply all checks from Section
16.7 step 2f:
   - Each file entry's path length L MUST be at least 1.
     An entry with L=0 is a format error; report "Manifest
     entry path length is zero" and halt.
   - byte_offset[0] MUST equal manifest_size.
   - For each i > 0: byte_offset[i] MUST equal
     byte_offset[i-1] + file_size[i-1].
   - For each file: byte_offset + file_size MUST NOT exceed
     Inner File Size.
If any byte_offset check fails: report "Manifest entry
offset/size inconsistency" and halt.  These checks MUST precede
any chunk access; a crafted Manifest with invalid byte_offset
values could otherwise cause out-of-bounds reads on the chunk
working set.

1.  For each file entry in Manifest:
    a.  If file_size = 0:
        i.   Sanitize path per Section 4.8.
        ii.  Check case collision with already-written files;
             if collision: report error and skip this entry.
        iii. Verify per-file BLAKE3 against BLAKE3 of the empty
             byte string; if fails: report and skip.
        iv.  Write empty file to output_directory / sanitized_path.
             MUST confine to output_directory.
        v.   Label: complete and individually verified.
    b.  If file_size > 0:
        first = byte_offset / S  (integer division)
        last  = (byte_offset + file_size - 1) / S
        Check whether all chunks [first..last] are directly
        present in the working set (not requiring RS).
    c.  If all chunks [first..last] are present:
        i.   Decompress chunks first..last (Section 7.3).
        ii.  Extract: start at (byte_offset - first*S) within
             decompressed chunk first; take file_size bytes.
        iii. Verify per-file BLAKE3; if fails: report and skip.
        iv.  Sanitize path; check case collision; write file.
        v.   Label: complete and individually verified.
    d.  If any chunk in [first..last] absent: report file as
        pending with missing chunk indices.
2.  Report: N_extracted files complete, N_pending pending.
```

3. Label container: "partially extracted; container hash unverified."

16.9. Trust Model

SFC/P5 provides four integrity levels:

Level 1 — Per-chunk: each chunk's BLAKE3 covers header and payload. Verified during D3d. Detects corrupt chunks before RS.

Level 2 — Manifest: the Manifest BLAKE3 covers all Manifest bytes (Magic through last file entry). Verified after collecting Manifest chunks. Self-contained; independent of Trailer or global hash. Allows safe use of Manifest data (paths, offsets, per-file hashes) for partial extraction before global hash is available.

Level 3 — Per-file: each file entry carries a BLAKE3 over that file's content bytes. Verified after extracting each file. Provides file-level integrity for partial extraction when the global hash cannot yet be checked.

Level 4 — Global: the Global Header carries a BLAKE3 over the complete inner content. Verified after full reconstruction (D5f). In SFC/P2, requires Trailer verification (D2c); if Trailer is unavailable, global hash may still be verified against the reconstructed inner content (unverified reconstruction, Section 9.2).

Trust hierarchy: a decoder MAY use Manifest data (Level 2) and per-file hashes (Level 3) to safely extract individual files even when the global hash (Level 4) has not yet been verified, provided Level 1 has passed for all processed chunks. Output produced without Level 4 verification MUST be labeled as unverified at the container level per Section 8.3, but individual files passing Level 3 verification MAY be labeled as individually verified.

Part III: Security, IANA, References

17. Implementation Status

(Note: This section will be removed before publication per RFC 7942 Section 7.)

Name: sfc (reference implementation)

Description: Open-source C++23 library (libsfc_lib) and command-line interface implementing the core SFC format and all five profiles (SFC/P1 through SFC/P5).

Level of maturity: prototype / pre-standard.

Coverage: Core encoding and decoding (§3-§11); SFC/P1 priority chunk classification (§12); SFC/P2 split transport with multi-segment encode and decode (§13); SFC/P3 TLV 0x0020 (§14); SFC/P4 TLV 0x0030 (§15); SFC/P5 directory encode, full extraction, and partial extraction (§16). GF(2^16) RS over Cauchy matrix verified against reference inverse values (§6.4).

Correctness: 303 automated tests covering wire-format round-trips, error injection, RS reconstruction across partial chunk sets, directory extraction, and CLI integration. Tests pass under AddressSanitizer, UndefinedBehaviorSanitizer, and ThreadSanitizer.

Contact: V. Krivchikov <sfcietf@gmail.com>

URL: https://github.com/tnxbutno/sfc

Last updated: 2026-05-06

18. Security Considerations

18.1. Integrity Without Confidentiality

SFC provides integrity detection but no confidentiality.

18.2. Tampering Detection Under Trusted Metadata

SFC's BLAKE3 hashes detect accidental corruption and tampering under trusted metadata. They do not authenticate origin. Implementations MUST NOT represent SFC integrity as equivalent to authenticated content.

18.3. Denial of Service

The following are hard protocol limits. Encoders MUST NOT produce files with values outside these ranges. Decoders MUST treat a declared value outside the permitted range as a format error (report and halt) before allocating any memory.

Field	Minimum	Maximum
Inner File Size	0 bytes	1 TB
Data chunk count N	1	65,534
Recovery chunk count M	0	65,534
N+M combined	1	65,535
Nominal chunk size S	2 bytes	256 MB
Compressed payload len	0	2*S bytes
Header length H	331	65,536 bytes
TLV value length	—	16,384 bytes
Manifest body B (P5)	57 bytes	64 MB
File count F (P5)	1	1,048,576
File path length L (P5)	1 byte	4,096 bytes

Note on H: H is the field value (uint32 LE), not the total header region size. The actual header region is H+4 bytes (Section 3.2). The limits above apply to the field value H. Therefore the maximum allocatable header buffer is 65,536 + 4 = 65,540 bytes. Minimum H = 331 (all fixed fields through priority count P with P=0 and no TLV fields): 4+16+8+2+255+32+4+4+4+1+1+2+2 = 335, so H >= 331 (= 335 - 4, because the 4-byte H field itself is excluded from H's value, being located before it).

Note on P: the priority list (4P bytes) occupies header space. Combined with the 331-byte fixed-field minimum and any TLV bytes, the total MUST NOT exceed H_max = 65,536 bytes. Absent TLV fields, P 16,301 (= floor((65,536 - 331) / 4)). See Section 4.5 rule g.

Note on S: S MUST be even (required for GF(2^16) word alignment in Reed-Solomon; Section 6.4). Minimum S = 2 bytes. When compression algorithm is not 0x00 (identity), encoders MUST choose S large enough that the compressed output of an S-byte

block fits within 2*S bytes. In practice, all standardized compression algorithms (zstd, brotli, lz4) have frame overhead exceeding 4 bytes, so S >= 64 bytes is the practical minimum for non-identity compression. A decoder receiving a chunk whose compressed payload exceeds 2*S rejects it regardless of S value.

Note on N+M: the combined limit of 65,535 derives from the GF(2^16) field size. Row indices k ∈ {0..M-1} and column indices {M..M+N-1} must fit within the field (0 to 65,535), and k XOR (M+j) must be non-zero for all valid (k,j) — which holds because k < M ≤ M+j implies k ≠ M+j. The limit of 65,535 (rather than 65,536) is a one-unit conservative margin.

Note on compressed payload length: the maximum of 2*S prevents allocation of unbounded memory from a single chunk payload length field. In practice, compressed output rarely exceeds the uncompressed size; this limit is a safety bound.

Note on minimum B and F: F >= 1 is required because an SFC/P5 container with zero files is not meaningful (empty directories are a non-goal per Section 1.3). The minimum B = 57 derives from one file entry with a 1-byte path: B = 4 + (52 + 1) = 57.

Note on F and B interaction: the F and B limits are not independent. With typical 20-character paths (entry ≈ 72 bytes), the effective maximum F is approximately 64 MB / 72 ≈ 909,000 files — well below the stated 1,048,576 limit. The F limit of 1,048,576 is only achievable with paths of 10 characters or fewer. Encoders MUST report an error when computing B would exceed 64 MB (suggested message: "Manifest body length would exceed 64 MB") regardless of whether F is within its stated limit.

Note on Trailer integrity: the Trailer BLAKE3 hash (offsets 8-39) covers the Global Header Region, not the Trailer itself. Trailer fields outside this hash — reserved bytes 4-7, the encoder timestamp (offsets 40-47), and reserved bytes 48-63 — are not authenticated by any hash. An adversary with write access to the file can corrupt these fields without invalidating D2c. In particular, writing any non-zero byte to Trailer offsets 4-7 causes a format error and halt (Section 5.5) while leaving the BLAKE3 hash intact — a denial-of-service requiring only file write access, not cryptographic capability. Deployments requiring Trailer authenticity MUST establish integrity over the full file through an external mechanism (e.g., code signing, transport-layer authentication).

Zero-length inner content (Inner File Size = 0): permitted, but requires a defined encoding because N = ceil(0 / S) = 0 would produce no data chunks. When Inner File Size = 0, encoders MUST set N = 1, producing exactly one data chunk whose decompressed payload is S bytes of zero-padding; this chunk is trimmed to 0 bytes on reassembly. M MAY be 0 or greater as normal. Decoders encountering Inner File Size = 0 with N = 1 MUST apply the standard reassembly procedure, trim to 0 bytes, verify the global BLAKE3 hash (which covers the empty byte string), and produce a zero-length output file. N != 1 when Inner File Size = 0 is a format error.

Compressed payload length: decoders MUST reject a chunk whose declared compressed payload length exceeds 2*S bytes. This prevents unbounded allocation from a single malformed length field. Encoders MUST NOT produce chunks exceeding this limit (in practice, compressed output very rarely exceeds S).

Decompressed output: decoders MUST verify that decompressed output per chunk equals exactly S bytes. If a decompressor

produces more or fewer than S bytes, the decoder MUST discard the chunk (format error per Section 6.4). Truncation to S bytes is NOT permitted; the mismatch indicates a corrupt or malicious payload. This defends against compression bombs and ensures RS block alignment.

Implementations MAY enforce lower local upper limits or higher local lower limits appropriate to their deployment environment; such tighter limits do not affect wire-level conformance provided the encoder produces values within the hard limits above.

18.4. Path Traversal and Platform Safety

Two path traversal attack vectors exist:

Vector 1 — Inner Filename (single-file SFC): addressed by 4.8.

Vector 2 — Manifest file paths (SFC Directory): addressed by Section 16.3 and 16.7. Each path component sanitized per 4.8.

For both: output directory confinement MUST be enforced independently of sanitization correctness (Section 4.8 rule d).

Platform-specific reserved names: some operating systems and filesystems reserve certain filenames regardless of extension. Decoders SHOULD sanitize path components that match platform-reserved names on the extraction target. Known reserved names include (non-exhaustive):

- Windows: CON, PRN, AUX, NUL, COM1-COM9, LPT1-LPT9 (with or without extensions).
- Characters that are legal in SFC paths but forbidden on some platforms: ':', '*', '?', '"', '<', '>', '|'.

Decoders SHOULD replace reserved names by prepending '_' to the component (e.g., "CON" → "_CON"). Decoders SHOULD replace platform-forbidden characters with '_' (0x5F) on the target platform. These sanitizations are platform-dependent and are NOT required for wire-level conformance, but they are essential for safe extraction on affected platforms.

18.5. Provisional Chunk Buffering

Before the Global Header is received, N and S are unknown; buffer limits cannot use N*S. Decoders MUST enforce an unconditional hard cap before the Global Header arrives. Recommended hard cap: 10 GB total provisional buffer. This value derives from N_max * S_max = 65,534 * 268,435,456 bytes (~16 PB total), which is impractical as a hard cap; 10 GB is chosen as a ceiling for unconstrained desktop-class deployments. Constrained environments (embedded, IoT) SHOULD set a lower cap appropriate to available memory; the 10 GB figure is a ceiling, not a floor. Implementations SHOULD document their actual cap.

After the Global Header, N*S*2 MAY be used as soft guidance for working-set size but MUST NOT replace the pre-header hard cap.

Provisional Chunks MUST NOT be exposed as belonging to any file or used in any output.

19. IANA Considerations

This document requests registration of one media type.

19.1. Media Type Registration: application/sfc

This document registers the media type "application/sfc" per RFC 6838 [RFC6838].

Type name:	application
Subtype name:	sfc
Required parameters:	none
Optional parameters:	version (e.g., "0.1"). If present and inconsistent with the in-file version (Section 3.1), the in-file version takes precedence.
Encoding considerations:	binary
Security considerations:	See Section 18
Interoperability considerations:	The format is self-describing. Decoders MUST accept any conformant encoder output. Partial files (missing chunks) are valid inputs to partial reassembly. See Section 9.
Applications that use this media type:	File transfer and archival tools operating over unreliable channels; command-line and library encoders and decoders implementing this specification.
Additional information:	
Deprecated aliases:	none
Magic number(s):	53 46 43 00 (bytes 0-3)
File extension(s):	.sfc
Macintosh file type code(s):	none
Person & email to contact for further information:	V. Krivchikov <sfcietf@gmail.com>
Intended usage:	COMMON
Restrictions on usage:	none
Author:	V. Krivchikov
Change controller:	IETF

The "x-" prefix convention is deprecated per RFC 6648 [RFC6648] and MUST NOT be used for new deployments.

20. References

20.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[RFC9562] Davis, B. et al., "Universally Unique IDentifiers (UUIDs)", RFC 9562, May 2024. (Obsoletes RFC 4122.)

[RFC8878] Collet, Y. and Kucherawy, M., "Zstandard Compression", RFC 8878, February 2021.

[RFC7932] Alakuijala, J. and Szabadka, Z., "Brotli Compressed Data Format", RFC 7932, July 2016.

[BLAKE3] O'Connor, J., Aumasson, J.-P., Neves, S., and Wilcox-O'Hearn, Z., "BLAKE3: One Function, Fast Everywhere", USENIX Security Symposium, 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/oconnor

The normative specification is the BLAKE3 team's
published paper and associated test vectors. The
reference implementation repository is at
https://github.com/BLAKE3-team/BLAKE3 (commit 2966085,
2023-08-08 used for implementation verification).
Should BLAKE3 be published as an RFC or NIST standard,
that publication MUST supersede this reference.

[LZ4-Frame]
Collet, Y., "LZ4 Frame Format Description", v1.6.2,
2022. https://github.com/lz4/lz4/blob/dev/doc/
lz4_Frame_format.md
Note: LZ4 has no current IETF RFC. Algorithm 0x03
(LZ4 Frame Format) is at risk for removal or demotion
to informative if a stable archival specification is
not established before this document advances. The
IESG is expected to flag this reference.
SFC algorithm 0x03 MUST use the LZ4 Frame Format
(not the raw LZ4 Block Format).

[Unicode-CaseFolding]
Unicode Consortium, "CaseFolding.txt, Unicode 15.1.0",
https://www.unicode.org/Public/15.1.0/ucd/CaseFolding.txt

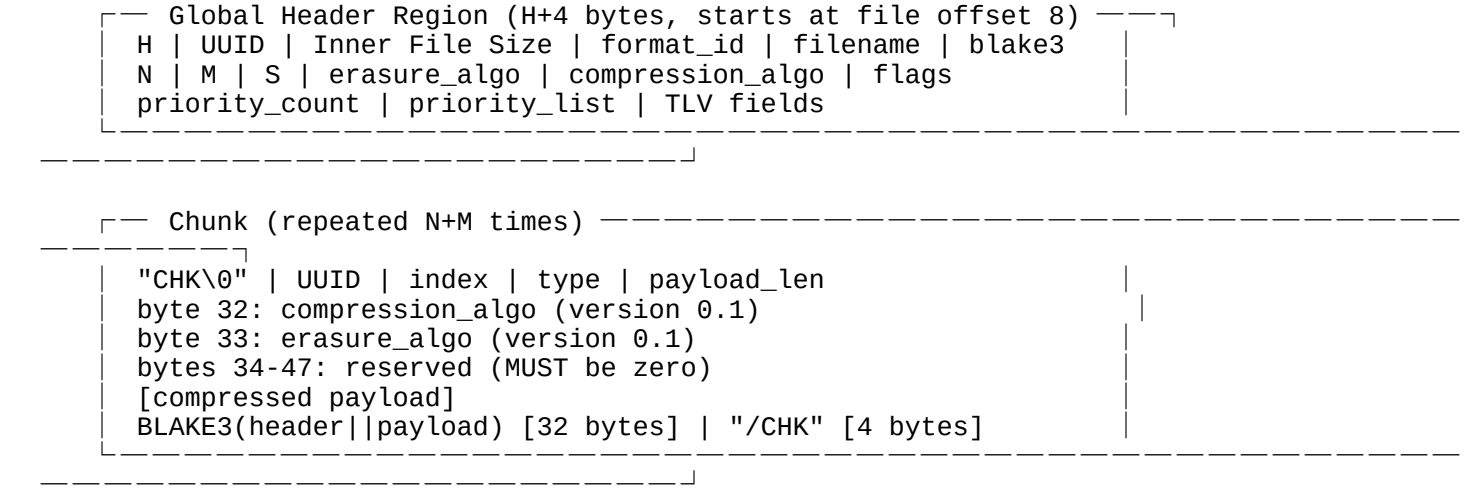
20.2. Informative References

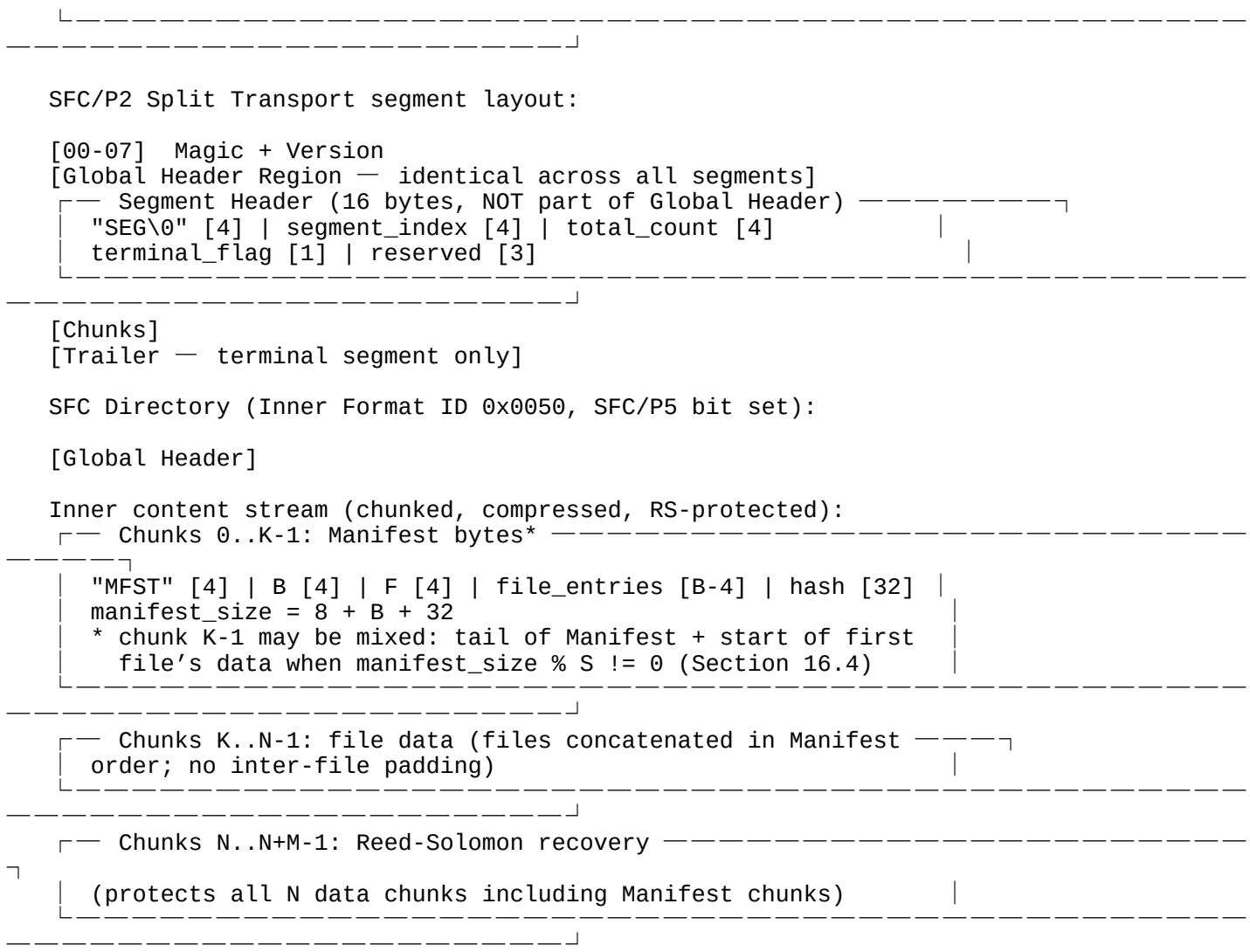
[RFC6838] Freed, N. et al., "Media Type Specifications and
Registration Procedures", RFC 6838, January 2013.
[RFC6648] Saint-Andre, P. et al., "Deprecating the 'X-' Prefix
and Similar Constructs in Application Protocols",
RFC 6648, June 2012.
[Matroska] https://www.matroska.org/technical/specs/index.html
[RFC1951] Deutsch, P., "DEFLATE", RFC 1951, May 1996.
[RFC5854] Bryan et al., "Metalink", RFC 5854, June 2010.
[BEP3] Cohen, B., "BitTorrent Protocol",
https://www.bittorrent.org/beps/bep_0003.html

Appendix A. Format Summary Diagram

Standard single-file SFC (version 0.1):

[00-07] Magic + Version: 53 46 43 00 00 00 01 00





Appendix B. Inner Format Compatibility Table

Note: IDs 0x0000 (Unknown/unspecified) and 0x0001 (Arbitrary binary data) are catch-all values and are not listed; all other assigned IDs are enumerated below.

Format	ID	Byte-str?	App utility?	P1 class	Notes
JPEG Baseline	0x0020	Yes	Partial	S	Top rows only
JPEG Progr.	0x0021	Yes	Partial†	Q	Viewer-dependent
JPEG 2000	0x0022	Yes	Good	P	Full spatial
JPEG XL	0x0023	Yes	Good	P	Recommended
PNG standard	0x0024	Yes	None	N	Full file needed
PNG interlaced	0x0025	Yes	Partial	I	Coarse preview
WebP (baseline)	0x0026	Yes	None	N	Full file needed
Plain text	0x0010	Yes	Good	-	Line by line
CSV / NDJSON	0x0011	Yes	Good	-	Record by record
MP4 fragmented	0x0030	Yes	Good	-	Plays prefix
Matroska/WebM	0x0031	Yes	Good	-	Streamable; plays prefix

ZIP	0x0040	Yes	None	-	Directory at EOF
gzip	0x0041	Yes	None	-	Full stream needed
zstd data	0x0042	Yes	None	-	Full stream needed
tar + zstd	0x0043	Yes	Good	-	Sequential
SFC Directory	0x0050	Yes	Good‡	P5	Per-file extraction
PDF	0x0100	Yes	Poor	-	Structure varies
ePub	0x0101	No	Poor	-	Spine at end; full needed
Nested SFC	0x00FF	Yes	Good§	-	Inner SFC rules apply

† JPEG Progressive: utility is encoding and viewer dependent.
‡ SFC Directory: file-level extraction once Manifest chunks (0..K-1) available; each file extractable when its chunk range is complete.
§ Nested SFC: partial utility depends on the inner SFC's own encoding parameters; a decoder MUST parse the inner SFC format to determine utility class.

Byte-str? column: "Yes" means the format can be parsed as a forward-only byte stream without backtracking or seeking, regardless of partial utility. "No" means the format requires seeking or end-of-stream structures to be usable.

Appendix C. Example Encoding Sessions

C.1. Single File

Input: quarterly_report.pdf (10,485,760 bytes), S=1MB
N=10, M=3, algorithm 0x01, compression 0x00
UUID: f47ac10b-58cc-4372-a567-0e02b2c3d479

Output (SFC/P2, one chunk per segment):
report.f47ac10b.0000.sfc ... report.f47ac10b.0012.sfc

Recovery scenario: data chunk 9 lost. V=12 >= N=10.
RS reconstruction recovers chunk 9. Trim last block.
Global BLAKE3 verified.

C.2. Directory Encoding

Input: ./mission_data/

images/photo_001.jpg	2,100,000 bytes
images/photo_002.jpg	1,800,000 bytes
report.pdf	5,200,000 bytes
sensors/pressure.csv	400,000 bytes
sensors/temp.csv	300,000 bytes

Parameters: S = 1,048,576 bytes, compression = 0x00.

Step 1: Lexicographic sort (UTF-8 byte ordering):
images/photo_001.jpg
images/photo_002.jpg
report.pdf
sensors/pressure.csv
sensors/temp.csv

Step 2: Manifest construction.

File entries (each 52 + path_length_in_bytes):

"images/photo_001.jpg"	20 bytes →	entry size 72
"images/photo_002.jpg"	20 bytes →	entry size 72
"report.pdf"	10 bytes →	entry size 62
"sensors/pressure.csv"	20 bytes →	entry size 72
"sensors/temp.csv"	16 bytes →	entry size 68

Total entries: 346 bytes

```
   B = 4 (F field) + 346 (entries) = 350
   manifest_size = 8 + 350 + 32 = 390 bytes
   K = ceil(390 / 1,048,576) = 1  (Manifest fits in chunk 0)

   Step 3: Byte offsets and chunk ranges.

   File                    byte_offset    file_size    chunks
   ————————————————————————————————————  ————————————————  ————————
————————  ————————
   (Manifest)              0              390          0..0
   images/photo_001.jpg    390            2,100,000    0..2
   images/photo_002.jpg    2,100,390      1,800,000    2..3
   report.pdf              3,900,390      5,200,000    3..8
   sensors/pressure.csv    9,100,390        400,000    8..9
   sensors/temp.csv        9,500,390        300,000    9..9

   Note: chunk 0 contains Manifest (bytes 0-389) and the start of
   photo_001.jpg (bytes 390-1,048,575).  Several chunks contain
   bytes from two adjacent files; decoders extract the correct byte
   range using byte_offset and file_size from the Manifest.

   Step 4: Inner File Size and RS parameters.

   Inner File Size = 390 + 2,100,000 + 1,800,000 + 5,200,000
                   + 400,000 + 300,000 = 9,800,390 bytes
   N = ceil(9,800,390 / 1,048,576) = 10
   M = 3 (30% overhead)
   Total segments: N + M = 13

   Chunk allocation:
      Chunks 0-9:   data (Manifest + file data)
      Chunks 10-12: recovery

   Output (SFC/P2, one chunk per segment, UUID a1b2c3d4):
      mission_data.a1b2c3d4.0000.sfc  ← chunk 0 (Manifest + start
                                        of photo_001.jpg)
      mission_data.a1b2c3d4.0001.sfc  ← chunk 1
      mission_data.a1b2c3d4.0002.sfc  ← chunk 2
      mission_data.a1b2c3d4.0003.sfc  ← chunk 3
      mission_data.a1b2c3d4.0004.sfc  ← chunk 4
      mission_data.a1b2c3d4.0005.sfc  ← chunk 5
      mission_data.a1b2c3d4.0006.sfc  ← chunk 6
      mission_data.a1b2c3d4.0007.sfc  ← chunk 7
      mission_data.a1b2c3d4.0008.sfc  ← chunk 8
      mission_data.a1b2c3d4.0009.sfc  ← chunk 9
      mission_data.a1b2c3d4.0010.sfc  ← recovery 0
      mission_data.a1b2c3d4.0011.sfc  ← recovery 1
      mission_data.a1b2c3d4.0012.sfc  ← recovery 2 (terminal)

   Partial extraction scenario (chunks 0, 1, 2, 3 received):

      Chunk 0: Manifest recovered and verified → directory listing
               known.

      images/photo_001.jpg (chunks 0..2, all received):
         chunk 0: bytes 0..1,048,575; file bytes start at offset 390
         chunk 1: bytes 1,048,576..2,097,151; all file bytes
         chunk 2: bytes 2,097,152..2,100,389; remaining file bytes
         → Extract: (chunk 0 bytes from 390) || (chunk 1 all) ||
                    (chunk 2 bytes to offset 2,100,389)
         → Verify per-file BLAKE3 → write

      images/photo_002.jpg (chunks 2..3, all received):
         Shares chunk 2 with photo_001, chunk 3 with report.pdf.
         file bytes span: offset 2,100,390 to 3,900,389
         → Extract correct byte range from chunks 2-3
```

```
         → Verify per-file BLAKE3 → write

      report.pdf (chunks 3..8; only chunk 3 received): pending.
      sensors/pressure.csv (chunks 8..9; missing): pending.
      sensors/temp.csv (chunk 9; missing): pending.

      Status: 2 files extracted and verified; 3 files pending.
      Container: partially extracted; container hash unverified.

   decoder *.sfc operation:
      Groups by UUID:
        a1b2c3d4: 13 files → mission_data encoding
        f47ac10b: 13 files → quarterly_report encoding
      Each group processed independently.
```

Appendix D. Conformance Test Cases

A conforming decoder MUST produce the specified outcome for each case.

D.1. Minimal Valid File

N=1, M=0, algorithm 0x00, inner = single byte 0x42.
Expected: reassembly succeeds; output=[0x42]; global hash verified.

D.2. Invalid Magic Bytes

First 4 bytes: 0x53 0x46 0x43 0x01.
Expected: reject "invalid magic bytes"; no further parsing.

D.3. Unsupported Major Version

Major version = 255.
Expected: reject "unsupported major version: 255".

D.4. Chunk BLAKE3 Hash Failure

N=1, M=0; payload one bit flipped; hash not updated.
Expected: discard chunk 0; report hash failure; no output.

D.5. File UUID Mismatch in Chunk

N=2, M=0; chunk 1 UUID differs from Global Header.
Expected: discard chunk 1; report mismatch with both UUIDs.

D.6. Benign Duplicate Chunk

N=2, M=0; chunk 0 received twice, byte-identical.
Expected: accept one copy silently; reassembly succeeds.

D.7. Contaminated Duplicate — One Valid (Case B1)

N=2, M=0; chunk 0 twice, different content; one fails hash.
Expected: use passing copy; log event; reassembly succeeds.

D.8. Contaminated Duplicate — Both Valid (Case B2)

N=2, M=1; chunk 0 twice, different content; both pass hash.
Expected: discard both; report dataset inconsistency; attempt RS recovery; if V>=N after discards, reassembly succeeds.

D.9. Non-Zero Reserved Bytes

N=1, M=0; chunk header reserved bytes non-zero; hash covers them.
Expected: report "non-zero reserved bytes"; discard chunk; no output.

D.10. Global Header Conflict in SFC/P2

Two segments, same UUID, Inner File Size differs by 1 byte.
Expected: report conflict; halt; no output.

D.11. Truncated Final Chunk

Last chunk payload truncated; chunk trailer absent.
Expected: discard incomplete chunk; report truncation.

D.12. Erasure Reconstruction

N=3, M=2 (total 5 chunks). For every withheld set of size <= M=2, reconstruction MUST succeed. C(5,3)=10 + C(5,4)=5 + C(5,5)=1 = 16 valid subsets; all MUST yield identical output with global hash verified.

D.13. Header Length Field Parsing

D.13a: H correct → header parsed; reassembly succeeds.

D.13b: H = correct+4 (encoder bug) → format error.

Common encoder defect: encoder includes the 4-byte H field itself in the value of H, producing H = (actual body length) + 4 instead of H = (actual body length). The decoder reads H, allocates H+4 bytes for the Global Header Region, then parses the fixed fields and finds that the declared H does not match the actual byte count of fixed fields plus priority list plus TLV — specifically, the H field value exceeds the maximum of 65,536 (Section 18.3) or causes the header region to overrun the file.

Expected: validate H against bounds [331, 65,536]; if H exceeds 65,536 or the resulting H+4 region overruns the file boundary, report "Header length H out of bounds" and halt.

D.14. Duplicate Known TLV Tag

TLV tag 0x0030 (SFC/P4 original format ID) appears twice. SFC/P4 bit is set.

Expected: reject "duplicate known TLV tag"; halt.

D.15. Unknown TLV Tag Filling Remaining Header Space

Unknown TLV with L set to consume all remaining header space.

Expected: tag skipped; reassembly proceeds normally.

D.16. Multiple Terminal Candidates — Trailer Present on Both

Two segments; both have Segment Header terminal flag = 0x01; both also carry a structurally valid Trailer (magic + correct hash). This tests that a passing Trailer does not cause the decoder to accept one segment as the authoritative terminal instead of counting flags.

Expected: report "Multiple Terminal flags"; halt.

D.17. Multiple Terminal Candidates — No Trailer

Two segments; both have Segment Header terminal flag = 0x01; neither carries a valid Trailer.

Expected: report "Multiple Terminal flags"; halt.

D.18. Chunk Index Out of Range

Chunk index = N+M (one past valid).

Expected: discard chunk; report "chunk index out of range".

D.19. Erasure Recovery After Case B2 Discard

N=2, M=1; chunk 0 is Contaminated Duplicate (both pass hash). V after discards = 2. V >= N=2; RS recovers chunk 0.

Expected: reassembly succeeds; global hash verified.

D.20. Malformed Priority List

P set such that priority list exceeds header boundary.

Expected: report header length inconsistency; halt.

D.21. TLV Value Overruns Header Boundary

TLV L causes value to extend beyond header boundary.

Expected: report "TLV value overruns header boundary"; halt.

D.22. Unknown Chunk Type

Chunk type = 0x00000003.

Expected: discard chunk; report with chunk index and type.

D.23. RS Reconstruction with Non-Trivial Matrix After B2 Discard

N=3, M=2; chunk 1 is Contaminated Duplicate (both pass hash).
Working set: chunks 0, 2, 3, 4. Select any 3; RS inverts a 3x3 matrix with at least one Cauchy row.
Expected: RS succeeds for each valid 3-chunk subset; all yield identical output; global hash verified.

D.24. Zero-Length Unknown TLV
Unknown TLV tag with L=0.
Expected: accepted and skipped; reassembly proceeds.

D.25. Known TLV with Wrong Declared Length
TLV 0x0030 (expected length 2) with declared L=8. SFC/P4 bit set.
Expected: reject "known TLV with unexpected length"; halt.

D.26. Non-Zero Bytes After Inner Filename Null Terminator
Inner Filename has valid name, null terminator, then non-zero bytes before end of 255-byte field.
Expected: reject "non-zero bytes after null terminator"; halt.

D.27. Profile TLV Present Without Profile Bit
TLV 0x0020 present but SFC/P3 bit (bit 6) not set.
Expected: reject per TLV rule g; halt.

D.28. SFC/P1 Profile Violation — Empty Priority List for Class P
SFC/P1 bit set; Inner Format ID 0x0022 (JPEG 2000, Class P).
Priority list is empty (P=0). Section 12.4 requires encoders to verify codestream structure and list priority chunks for Class P. P=0 violates this Profile MUST.
Expected: decoder implementing SFC/P1 SHOULD report Profile MUST-level violation; MAY continue with Core rules. Decoder not implementing SFC/P1 MUST NOT fail on Profile bit alone; P=0 is valid under Core rules (Section 4.5 does not require P > 0).

D.29. Trailer Bytes 4-7 Non-Zero
Trailer bytes 4-7 contain non-zero values.
Expected: report "non-zero reserved bytes in Trailer"; halt.

D.30. Terminal Segment Not Found, V >= N — Unverified Reconstruction
SFC/P2 with 3 segments; none has terminal flag set and none passes Trailer detection. All 10 data chunks available (V >= N).
Expected: report "Terminal Segment not found"; perform unverified reconstruction (Section 9.2); verify inner content global hash; label output "reconstructed; container metadata unverified". MUST NOT treat missing Terminal as a hard error when V >= N.

D.31. Terminal Segment Not Found, V < N — Partial Reassembly
SFC/P2 with 3 segments; none has terminal flag set and none passes Trailer detection. Only 7 of 10 data chunks available.
Expected: report "Terminal Segment not found"; perform partial reassembly (Section 9.3); label output partial and unverified.

D.32. Per-Chunk Decompression Without Global Header
Chunk header bytes 32-33 carry compression 0x01 (zstd) and erasure 0x01 (RS). Global Header has not yet arrived.
Expected: decoder decompresses chunk payload using byte 32's algorithm ID; chunk passes D3a-D3e and enters provisional buffer.

D.33. Chunk Algorithm ID Mismatch — Real
Chunk byte 32 = 0x01 (zstd) but Global Header declares compression 0x02 (brotli).
Expected: report algorithm mismatch; discard chunk.

D.34. SFC Directory — Manifest Hash Failure
Inner Format ID 0x0050; Manifest BLAKE3 field incorrect.
Expected: report "Manifest BLAKE3 hash failure"; halt extraction.

D.35. SFC Directory — Correct Manifest Arithmetic
Inner Format ID 0x0050; F=2 files; paths "a.txt" (5 bytes) and "b.txt" (5 bytes); each entry size = 52+5 = 57 bytes; total entries = 114 bytes.
B = 4 + 114 = 118.
manifest_size = 8 + 118 + 32 = 158.
BLAKE3 hash at byte offset 126 (= 8+118) covering bytes 0..125.
Expected: decoder computes manifest_size = 158; K = ceil(158/S); file byte offsets begin at 158.

D.36. SFC Directory — Partial Extraction
Inner Format ID 0x0050, F=3 files; Manifest in chunk 0. All Manifest chunks present. File B's complete chunk range present; files A and C missing chunks.
Expected: extract file B; verify per-file hash; report files A and C as pending; label container partially extracted, unverified.

D.37. SFC Directory — Manifest RS Recovery
Inner Format ID 0x0050; Manifest in chunk 0; M=1; chunk 0 missing; recovery chunk N present; V=N valid chunks available.
Expected: RS reconstructs chunk 0 per Section 6.3; Manifest verified; extraction proceeds.

D.38. SFC Directory — Path Traversal in Manifest
Manifest path "../../../etc/passwd".
Expected: path sanitized or rejected; no file written outside output directory.

D.39. Multi-UUID Operation
Two .sfc files in directory with different File UUIDs.
Command: decoder *.sfc.
Expected: two UUID groups; each processed independently; encountering different UUIDs is NOT an error.

D.40. SFC Directory — File Byte Range Spanning Chunk Boundary
Inner Format ID 0x0050; two files; second file starts at byte_offset not a multiple of S (shares a chunk with file 1 tail). All relevant chunks present.
Expected: decoder extracts second file using correct byte range within decompressed chunk content; per-file BLAKE3 verified.

D.41. SFC Directory — Manifest Unavailable After RS Failure
Inner Format ID 0x0050; Manifest chunks 0..K-1 missing; V < N even with recovery chunks.
Expected: report "Manifest unavailable; file-level extraction impossible"; halt P5 partial extraction.

D.42. SFC Directory — Case Collision in Manifest
Manifest contains paths "Data/File.txt" and "data/file.txt". (Same under Unicode case-folding.)
Expected: encoder MUST have rejected this at encoding time. Decoder detecting collision during extraction: report "case collision in Manifest paths"; do not write second file.

D.43. SFC Directory — Case Collision Under Simple Case Folding
Manifest contains "README.md" and "readme.md". Under Unicode Simple Case Folding both map to "readme.md": this IS a collision.
Expected: encoder MUST have rejected this at encoding time. Decoder detecting collision during extraction: report "case collision in Manifest paths"; do not write second file.

D.44. P3 TLV 0x0020 Size Limit Enforcement
SFC/P3 file with N+M = 3000; encoder includes TLV 0x0020 (would require 24,000 bytes, exceeding the 16,384 TLV limit).
Expected: encoder MUST NOT produce such a file. A decoder encountering TLV 0x0020 with L > 16,384 MUST treat it as a

format error (TLV value length exceeds hard limit).

D.45. P3 TLV 0x0020 Absent for Large File
SFC/P3 file with N+M = 2049 and no TLV 0x0020.
Expected: valid P3 file; decoder locates chunks by sequential scan; no error due to absent offset index.

D.46. SFC Directory — Zero-Length File
Manifest contains one file entry with file_size = 0.
Expected: decoder writes an empty file; per-file BLAKE3 hash verified (BLAKE3 of zero bytes has a known fixed value); no chunk-range formula applied; labeled complete.

D.47. Protocol Limit Exceeded — N Too Large
Global Header declares N = 100,000 (exceeds hard limit of 65,534).
Expected: decoder reports format error; halts before allocating any memory for chunk structures.

D.48. RS Pipeline Order — Decompress Before Reconstruct
N=3, M=1; chunk 1 is missing; V=3 >= N=3. Received chunks 0, 2 (data) and chunk 3 (recovery) are in compressed form.
Expected: decoder decompresses chunks 0, 2, 3 independently before applying RS; RS operates on uncompressed S-byte blocks; recovered chunk 1 matches the original byte-for-byte.

D.49. SFC Directory — Partial Extraction Case B (V < N)
Inner Format ID 0x0050; N=10, M=3; V=7 (insufficient for RS). Manifest chunks (0..K-1) present. One file’s complete chunk range [3..4] fully in working set; another file’s range [6..8] has chunk 7 missing.
Expected: Case B applies; file with range [3..4] extracted and verified; file with range [6..8] reported as pending (chunk 7 absent); no RS reconstruction attempted.

D.50. SFC Directory — Partial Extraction Case A (V >= N)
Same as D.49 but M=3 recovery chunks are present so V=10 >= N=10.
Expected: Case A applies; full RS reconstruction performed; all N data blocks available; all files extracted and verified; container labeled unverified (Trailer may be absent).

D.51. Unsupported Compression Algorithm
Valid Core file with compression algorithm 0x02 (brotli). Decoder does not implement brotli.
Expected: report "unsupported compression algorithm: 0x02"; produce no output; decoder is still conforming.

D.52. SFC Directory — Zero-Length File in Full Extraction
Manifest contains file entry with file_size = 0. All N data chunks present; full reassembly complete.
Expected: decoder writes empty file; verifies per-file BLAKE3 against BLAKE3 of the empty byte string; labels complete. Behavior is identical to partial extraction D.46.

D.53. Unsupported Erasure Algorithm
Valid SFC file with erasure algorithm 0x82 (vendor-specific). Decoder does not implement this algorithm.
Expected: report "unsupported erasure algorithm: 0x82"; produce no output; decoder is still conforming.

D.54. Priority List — P > N
Global Header declares N=5, P=6, with 6 priority indices.
Expected: report "priority count P > N"; halt (format error per Section 4.5 rule a).

D.55. Priority List — Duplicate Index

Global Header declares P=3; priority list contains [1, 2, 1].
Expected: report "duplicate index in priority list"; halt (format error per Section 4.5 rule b).

D.56. Priority List — Out-of-Range Index
Global Header declares N=5; priority list contains index value 5 (valid range is [0, 4]).
Expected: report "priority index out of range [0, N-1]"; halt (format error per Section 4.5 rule c).

D.57. Priority List Outside SFC/P1 — Advisory Only
Global Header has P=2, priority list [0, 1]; SFC/P1 bit (bit 4) is NOT set. All indices in range [0, N-1], no duplicates.
Expected: no format error; priority list treated as advisory; reassembly proceeds normally (Section 4.5 rule f).

D.58. Bytes 32-33 Always Carry Algorithm IDs
Chunk header bytes 32-33 are 0x01 and 0x01 (zstd, RS GF(2^16)).
Global Header also declares 0x01 and 0x01.
Expected: bytes 32-33 are valid algorithm IDs; chunk accepted. Decoder MAY use bytes 32-33 for headerless decompression (Section 5.1).

D.59. P2 and P3 Bits Both Set
Global Header has both SFC/P2 bit (bit 5) and SFC/P3 bit (bit 6) set, and includes TLV 0x0020.
Expected: decoder MUST report "SFC/P2 and SFC/P3 bits both set"; MUST ignore P3 semantics; MUST proceed using SFC/P2 rules.

D.60. Inner File Size Zero — Valid Encoding
Global Header declares Inner File Size = 0, N = 1, S = 65536.
One data chunk with 65536 bytes of zero-padding.
Expected: decoder performs standard reassembly; trims to 0 bytes; verifies global BLAKE3 hash (of empty byte string); produces a zero-length output file.

D.61. Inner File Size Zero — Invalid N
Global Header declares Inner File Size = 0, N = 3.
Expected: decoder reports format error "Inner File Size = 0 with N != 1" (Section 18.3); halts.

D.62. Nominal Chunk Size S = 0
Global Header declares S = 0.
Expected: decoder reports format error (below minimum per Section 18.3); halts.

D.63. SFC Directory — Mixed Chunk K-1
Inner Format ID 0x0050; S = 1,048,576 bytes; manifest_size = 600 bytes (manifest_size % S != 0). K = 1 (one Manifest chunk).
Chunk 0 contains Manifest bytes 0..599 followed by file data bytes 600..1,048,575.
Expected: decoder correctly identifies the Manifest boundary at byte 600 within chunk 0; extracts Manifest bytes 0..599 and treats the remainder of chunk 0 as file data; per-file extraction and per-file BLAKE3 hash verification succeed.

D.64. SFC/P2 Segment Header Validation
SFC/P2 segment with SPLIT_TRANSPORT flag set but no "SEG\0" magic at expected Segment Header position.
Expected: report "missing or invalid Segment Header"; halt.

D.65. SFC/P2 Segment Header Reserved Bytes Non-Zero
SFC/P2 segment with valid "SEG\0" magic; bytes 13-15 contain non-zero values; segment version <= decoder’s version.
Expected: report "non-zero reserved bytes in Segment Header"; halt.

D.66. SFC/P2 Version Mismatch Across Segments
Two segments, same UUID; one has minor version 1, the other has minor version 2.
Expected: report "version mismatch across segments"; halt.

D.67. Chunk End Marker "/CHK" Missing
N=1, M=0; chunk payload present, BLAKE3 hash correct, but final 4 bytes are not "/CHK".
Expected: discard chunk; report "chunk end marker invalid".

D.68. Compressed Payload Length Exceeds 2*S
N=1, M=0, S=65536; chunk declares compressed payload length = 200,000 (> 2*65536 = 131,072).
Expected: reject chunk before reading payload; report "compressed payload length exceeds 2*S".

D.69. Odd Nominal Chunk Size S
Global Header declares S = 65537 (odd number).
Expected: report format error (S must be even per Section 6.4); halt.

D.70. Erasure Algorithm 0x00 with M > 0
Global Header declares erasure algorithm 0x00, M = 2.
Expected: report "erasure algorithm 0x00 with M > 0"; halt before any chunk processing.

Authors' Addresses

Valerii Krivchikov
Independent

Email: sfcietf@gmail.com