ZIP Extractor Encryption (ZEE) Specification

ZIP Extractor by Visware LLC

Authors: Visware LLC Technical Staff

Version 1.0.1 FINAL - November 4, 2026. Suite ZEE-1 (SuiteID 1).

ZEE-1 uses ZIP compression method 91 for password-based encryption and authentication of file contents, with optional protection of file names and other metadata. It keeps the ordinary ZIP entry structure: each ZEE-1 file has its own encrypted payload, and the central directory is either public or encrypted in place. Ordinary ZIP entries may coexist with ZEE-1 files in public mode. The encrypted directory, ZECD, is part of this specification.

ZEE-1 builds on the entry-oriented design of WinZip AES. It uses established cryptographic primitives, shares password-derivation work across files, and keeps a CRC-32 inside each encrypted payload for checking decoded data. Encryption, authentication and CRC checking have distinct roles; Section 13 states what each protects and what remains visible.

Features

- **File protection:** AES-256-CTR encryption and a 128-bit HMAC-SHA-256 tag cover each compressed file and its encrypted CRC-32 trailer. The tag also authenticates the compression descriptor and directory mode.
- **Optional metadata protection:** ZECD encrypts and authenticates names, central-directory metadata and each file record's association with its payload. Public mode permits listing without a password.
- **Shared password derivation:** one 1,000,000-iteration PBKDF2-HMAC-SHA-256 derivation per credential/root-salt context, followed by separate keys for each object. Copied files may retain other contexts or passwords.
- **Ordinary ZIP integration:** method 91, explicit compression descriptors, ZIP64 and signed data descriptors support independent files and streamed writing without an archive-inside-an-archive wrapper.
- **Basic recovery:** a surviving local header and complete payload can support authenticated content salvage after directory loss. Recovery does not restore authenticated names or prove archive completeness.

Individual file sizes are not hidden. The archive comment is always public and unauthenticated. Neither a successful directory unlock nor password removal by itself establishes that all decoded file contents are valid.

PKWARE has assigned compression method 91 and extra-field Header ID `0x9101` to ZIP Extractor Encryption. The additional `0x455A` vendor-field registration and APPNOTE publication are pending. "Enhanced" is ZIP Extractor's product label and has no wire meaning.

Conventions and document map

This specification is for implementors familiar with ZIP records and basic cryptography.

MUST, MUST NOT, SHOULD, SHOULD NOT and MAY have the meanings of RFC 2119 and RFC 8174 [R34]. All multi-byte ZIP integers are unsigned little-endian unless explicitly stated otherwise (APPNOTE 4.3.3 [A1]). The AES counter is big-endian. ASCII strings in cryptographic formulas have no terminator. `||` denotes concatenation; `first n` means the initial n bytes.

`C` is an encrypted object's ciphertext length. For a file, `S` is the compressed-stream length, `C = S + 4` includes the encrypted CRC trailer, `P = 50 + C = 54 + S` is the complete stored payload length, and `U` is the uncompressed length. ZECD has no CRC trailer. "Compression descriptor" means the complete 12-byte `0x9101` extra field; "data descriptor" means the ZIP record following a streamed file payload. They are different structures. "Public mode" has an ordinary central directory; "protected mode" has a ZECD record. A directory entry is a filesystem directory, not the archive's central directory.

Sections 5-12 define the format and processing rules; Section 13 states their security scope. Section 14 supplies conformance cases and known answers. Section 16 gives a short implementation guide. The companion Implementer's Guide, Version 1.0.1 FINAL, provides copyable layouts and a complete worked archive. This document remains the normative reference; the document revision does not change SuiteID 1 or ZECD record version 1.

Rationales are non-normative. After publication, an incompatible entry contract requires another SuiteID; incompatible ZECD framing requires another record version. Clarifying errata may preserve the existing meaning. Unknown bytes are not blanket permission for future security semantics.

1. Purpose and authority

ZEE-1 encrypts and authenticates ZIP file contents using AES-256-CTR and HMAC-SHA-256-128. A password-derived root is expanded into independent object keys. Each file remains independently decryptable and verifiable from its payload, compression descriptor, directory mode and credential.

The format has two archive forms:

- **Public mode:** ordinary visible ZIP metadata, method-91 encrypted files, zero outer CRC fields for those files, and mandatory entry MACs and encrypted plaintext CRC checks.
- **Protected mode:** ZECD additionally encrypts and authenticates the central directory and the associations between records and file payloads. Files use the same encrypted CRC trailer and zero outer CRC fields.

A conforming ZEE-1 reader MUST support both forms and both specified data descriptor widths. A conforming writer MAY produce either or both forms and MAY omit descriptor writing. An entry-codec-only library may implement public mode and report ZECD as unsupported, but does not thereby implement the complete reader contract. Unsupported compression algorithms are reported as Section 10.3 specifies.

Rationale. Public mode has the integration shape of WinZip AES: an entry method and an extra field in ordinary ZIP. Protected mode adds metadata protection without putting the archive inside another archive member.

2. Scope and threat model

The adversary can read, copy and alter archive bytes without knowing the password. Security depends on password strength, correct randomness and an uncompromised implementation. Offline password guessing is expected. An externally observable password-testing service, endpoint compromise, traffic analysis and forensic reconstruction are outside this profile.

Individual file sizes are not protected. Payload lengths, compression methods, entry boundaries and physical order are visible; STORE reveals the original size. Public mode also exposes names, timestamps, attributes and other central metadata. Protected mode hides those fields, subject to length leakage described in Section 13.

The archive comment is public and unauthenticated in both modes. Basic recovery can authenticate surviving file contents and check their encrypted CRC trailers without restoring missing names or archive completeness.

3. Profile summary

3.1 Identity and parameters

Item	Value
Family and suite	ZIP Extractor Encryption (ZEE), ZEE-1, SuiteID 1
File method	91, bytes `5b 00`
Compression descriptor	`0x9101`, Data Size 8, signature `ZE`, SuiteID, ActualMethod, ActualFlags; 12 bytes total
Password bytes	Strict UTF-8 of the exact nonempty scalar sequence; Section 5
Root derivation	PBKDF2-HMAC-SHA-256, 1,000,000 iterations, 32 bytes
Root salt and object nonce	16 bytes each; Section 7.2
Object keys	HKDF-Expand-SHA-256(root, role label concatenated with nonce)
Cipher	AES-256-CTR, 128-bit big-endian counter starting at 1
File MAC	First 16 bytes of HMAC-SHA-256 over prefix, mode byte, descriptor and ciphertext
File payload	Section 7.1; ciphertext contains compressed stream and CRC32LE; `P = 54 + S`
Modes	Public `00`; protected `01`; mode is implicit MAC input, not a stored payload field
CRC-32	Encrypted four-byte file trailer is authoritative in both modes; every outer method-91 CRC field is zero
Empty regular file	Ordinary encrypted file; canonical STORE payload is 54 bytes
Directory entry	STORE, zero sizes and CRC, no file payload
Data descriptors	Signed 16-byte classic or 24-byte ZIP64 form; optional on write, supported on read
File ciphertext bound	`C <= 2^61 - 87`
ZECD	Header16, rootSalt16, nonce16, verifier2, ciphertext, tag16; `66 + C`, no CRC trailer; separate key labels and MAC prefix
Directory ciphertext bound	`C <= 2^61 - 94`; plaintext directory is uncompressed

3.2 Construction

A new single-password archive draws one root salt and stretches the password once. Each newly encrypted file or ZECD object draws its own nonce and expands the root into role-specific AES, MAC and verifier values. The file writer computes the plaintext CRC, compresses, appends CRC32LE, then encrypts and authenticates the result; the central directory is emitted publicly or protected by ZECD. Copied entries may retain other roots or passwords.

3.3 Authentication scope

Protected by each entry tag	Additional protection supplied by ZECD
Directory mode, complete compression descriptor, ciphertext including its CRC trailer, and ciphertext length	Names, sizes, timestamps, attributes, per-file comments, central extras, order and membership
Root salt and nonce select its authentication key	ZEAC records the expected entry tag; actual content still requires the entry MAC
The verifier is only an early-rejection hint	The public archive comment remains unauthenticated

Local headers and ordinary end-record fields are not authenticated. Public metadata remains unauthenticated even when the entry tag verifies. Mode binding prevents an unchanged protected-mode payload from validating as public mode; it does not authenticate a public filename or sender identity.

4. Relationship to WinZip AES and other formats

ZEE-1 retains WinZip AE-2's general entry shape: placeholder method, actual compression descriptor, password verifier, CTR encryption and a trailing MAC [W1]. It uses SHA-256, a stronger fixed password stretch, one root expanded into object keys, a 128-bit tag, explicit big-endian counters, strict UTF-8 credentials and authenticated interpretation metadata. Outer CRC treatment follows AE-2; an encrypted file trailer adds a separate plaintext CRC check in both directory modes. Method 99 and extra field `0x9901` never identify ZEE.

RAR5 and 7z cache password derivations for matching credential, salt and parameter contexts [R7] [R41]. ZEE-1 defines its own explicit hierarchy: a password-derived root and fresh public nonces select separate object keys. This does not imply that those formats use the same key schedule.

Like 7z encoded headers and RAR encrypted headers, ZECD protects the archive's metadata structure directly [R7] [R39]. Its record framing is defined in Section 8.11. It does not use APPNOTE's Archive Decryption Header, version-62 encryption signal or `0x0017` field [A1].

5. Password

5.1 Encoding rule

PBKDF2 receives strict UTF-8 of the exact Unicode scalar sequence supplied by the user [R33]. A conforming implementation MUST reject an empty string or any unpaired surrogate before encoding. It MUST NOT substitute U+FFFD, normalize, case-fold, trim, map spaces, truncate or fall back to another encoding. The format imposes no password-length limit. Readers MUST accept every well-formed nonempty scalar sequence as a credential.

HMAC hashes keys longer than 64 bytes before iteration and zero-pads shorter keys [R23]. Consequently, distinct UTF-8 strings can derive equal roots: for example, `password` and `password` followed by U+0000. Exact UTF-8 is injective as an encoding; derived keys are not claimed to be unique across password strings. Long password input does not multiply each iteration's cost.

5.2 Writer creation policy

When setting a NEW encryption password, a writer MUST also reject Unicode General_Category Cc: U+0000-U+001F and U+007F-U+009F, 65 code points. There is no additional noncharacter, invisible-format or normalization rule. This creation policy does not change a credential already accepted on read: editing under that credential uses it unchanged.

Rationale. Exact UTF-8 preserves credential semantics with a small, deterministic rule. Visually identical sequences may be different passwords. Filesystem filename-normalization rules do not define password input behavior. Legacy ZIP password encodings are separate contracts and do not supply ZEE fallbacks.

5.3 Conversion vectors

Input	Expected bytes or result
`password`	`70 61 73 73 77 6f 72 64`
U+00E9	`c3 a9`
U+0065 U+0301	`65 cc 81`, different from U+00E9
U+5BC6	`e5 af 86`
U+1F510	`f0 9f 94 90`
Lone U+D83D or U+DD10	Rejected before encoding
Empty string	Rejected
Leading/trailing spaces	Preserved
Embedded U+0000 on read	Preserved as `00`; refused for a new password
U+5BC6 U+1F510 U+0065 U+0301	`e5 af 86 f0 9f 94 90 65 cc 81`

Implementations MUST test these cases, including distinct byte encodings for canonically equivalent strings and equal roots for the documented HMAC zero-padding example. Historical low-byte encodings are not valid ZEE-1 credential preparation.

6. Key derivation

6.1 Password-derived root

```
R = PBKDF2-HMAC-SHA-256(passwordBytes, rootSalt, 1000000, 32)
```

PBKDF2 is RFC 8018 Section 5.2 [R32]. Its output is exactly the first 32-byte PBKDF2 block. The iteration count and output length are fixed by SuiteID 1, never stored, negotiated or adjusted for device speed.

6.2 Object keys

For each encrypted object, derive:

```
Kenc     = HKDF-Expand-SHA-256(R, ASCII(aesLabel)    || nonce16, 32)
Kmac     = HKDF-Expand-SHA-256(R, ASCII(macLabel)    || nonce16, 32)
verifier = HKDF-Expand-SHA-256(R, ASCII(verifyLabel) || nonce16,  2)
```

Role	aesLabel	macLabel	verifyLabel
File	`ZEE-1/aes`	`ZEE-1/mac`	`ZEE-1/verify`
Directory	`ZEE-1/dir/aes`	`ZEE-1/dir/mac`	`ZEE-1/dir/verify`

HKDF-Extract is not applied. R already serves as the pseudorandom key for RFC 5869's expansion step [R11]. Each result fits in one block:

```
Expand(R, info, L) = first L bytes of HMAC-SHA-256(R, info || 01)
```

The final `01` is the HKDF block counter, not the directory mode. Labels are exact ASCII with no terminator, separator or length prefix. Append the object's 16 raw nonce bytes, not their hex representation. Each output is a separate expansion with root R as PRK; Expand has no salt parameter. These fixed lengths make input boundaries unambiguous. File mode affects the MAC message, not the key schedule.

6.3 Sharing, copying and key lifetime

A writer creating a new single-password archive MUST use one root context for its newly encrypted files and directory. Each payload repeats the root salt and its own nonce so that it remains independently decryptable after copying or directory loss. Imported entries MAY retain their original roots and passwords. Readers derive once per distinct credential/root-salt context actually used, not necessarily once for every possible composite archive.

An implementation MAY cache roots and object keys within an authorized open/create session. Cache lookup MUST distinguish the credential as well as root salt; concurrent requests for the same context SHOULD share the in-flight derivation. References MUST be discarded when the credential is discarded or the session is locked/closed. Cancellation discards operation results; it need not erase a root still authorized for the open session. No physical-erasure guarantee is made for managed memory or platform keys.

New files normally use the target's active root context. For a public archive, a writer may reuse a matching context authenticated by a prior entry MAC, or create a fresh context. A verifier match alone MUST NOT be treated as authentication of an existing context. Directory rewrites can retain their root while drawing a new nonce.

Readers MUST NOT reject merely because root salts or nonces repeat: shared roots and unchanged copied payloads are legitimate. This does not relax the writer freshness rules in Section 7.2. A missing required primitive makes the operation unsupported. New encryption also requires a cryptographically secure random source. There is no weaker substitute.

6.4 Work factor and rationale

An offline password guess needs one 1,000,000-iteration derivation for the chosen root context, followed by an inexpensive verifier or MAC-key expansion. The attacker need not attack every file. Root sharing removes repeated honest work without reducing that per-guess stretch. Multiple verifiers can filter false positives cheaply after the root is derived; they do not bypass the stretch. Distinct hostile roots still impose distinct work, so implementations SHOULD derive on demand, bound concurrency and allow cancellation.

Deriving one PBKDF2 block and expanding it avoids the multi-block imbalance in WinZip AES-256's 66-byte PBKDF2-HMAC-SHA-1 output (four SHA-1 blocks). Related two-stage derivation has NIST and HKDF precedent [R10] [R11]. Compromise of a root exposes its children; compromise of one independently derived child key does not imply recovery of the root. Future changes to the fixed work factor require another suite after publication.

7. Payload

7.1 Layout

Every regular file has this payload in either directory mode:

```
rootSalt16 | nonce16 | verifier2 | encryptedStream[S] | encryptedCRC4 | tag16
C = S + 4
P = 50 + C = 54 + S
```

Offset	Length	Field
0	16	rootSalt
16	16	nonce
32	2	verifier
34	S	Encrypted compressed stream
34 + S	4	Encrypted CRC32LE of the uncompressed file
38 + S	16	tag

The root salt, nonce, verifier and tag are stored unencrypted. The two encrypted fields are one continuous CTR ciphertext. Append the four CRC bytes after compression and continue the same keystream without a reset, padding or alignment bytes. The MAC covers all C ciphertext bytes.

A file's ZIP compressed-size field counts all P bytes, excluding any following ZIP data descriptor. A file payload shorter than 54 bytes is malformed. Under STORE, an empty regular file has S=0, C=4, P=54 and CRC=0.

The decoder receives exactly S decrypted stream bytes. The last four plaintext bytes are the expected CRC, never compression input or file output. This boundary also applies when preserving compressed bytes during password removal; the CRC trailer MUST NOT be copied into an ordinary ZIP stream.

ZECD is different: after its 16-byte header it contains rootSalt16, nonce16, verifier2, C ciphertext bytes and tag16, with no CRC trailer. Its encrypted plaintext is exactly the central-record sequence and its full record remains 66+C bytes (Section 8.11), including C=0 for an empty directory.

7.2 Root salt and object nonce

A new root context MUST draw a fresh random 16-byte root salt. Every NEW encryption operation MUST draw a fresh random 16-byte nonce, including re-encryption and every directory serialization even if its plaintext is unchanged. Both draws MUST use a cryptographically secure random generator; generator failure aborts the operation. Neither value may be derived from a password, name, timestamp, content, offset or previous value. Deterministic inputs are permitted only in a test harness, not as normal writer settings.

Copying an unchanged payload or changing only its mode-bound tag is not a new encryption. Reusing the root for new encryption is intentional; reusing its nonce under the same role can repeat the AES keystream and is unsafe. A permanent history map is not required. Fresh 128-bit randomness provides negligible collision probability at the intended scale [R10].

7.3 AES-256-CTR

```
T_j  = integer j as an unsigned 128-bit big-endian block
KS_j = AES-256-Encrypt(Kenc, T_j), for j = 1, 2, ...
ciphertext = object plaintext XOR (KS_1 || KS_2 || ...)
```

File object plaintext is `compressedStream[S] || CRC32LE`; directory object plaintext is the central-record sequence without a CRC trailer. The keystream is truncated to the object plaintext length. The initial counter is `00000000000000000000000000000001`. There is no padding or separately stored CTR IV. The object nonce selects the key; it is not the counter block.

Chunking MUST produce identical ciphertext. Implementations MAY buffer partial blocks or use equivalent absolute-offset calls. At object byte offset o, the counter is `1 + floor(o/16)` and the first `o mod 16` bytes of that block's keystream are skipped. Every piece MUST preserve its position in the continuous keystream, including partial-block boundaries. Section 9's bound keeps every object below 2^57 counter blocks.

This is standard big-endian CTR [R15]. WebCrypto can use `AES-CTR`, that initial block and `length = 128`; `length = 64` is byte-equivalent throughout the permitted domain [R16]. WinZip's little-endian counter convention is not used. Native calls process many blocks at once: 16 bytes is the AES block size, not an API-call size. Independent chunk calls MUST NOT restart the object's keystream.

7.4 Authentication tag and verifier

For files:

```
mode = 00 for public mode, 01 for protected mode
tag  = first 16 bytes of HMAC-SHA-256(Kmac,
       ASCII("ZEE-1/tag") || mode || descriptor12 || ciphertext)
```

The mode is exactly one byte, inferred from the archive's resolved directory form during normal reading. It is not stored in the payload. Recovery uses the local form as an untrusted hypothesis (Section 8.7). The descriptor is the central copy by which the file was recognized. The MAC message is `22 + C` bytes; its prefix is the nine ASCII bytes `5a45452d312f746167`. Root salt and nonce select Kmac; they are not repeated in the MAC message. The two-byte verifier is not MAC input. Directory authentication uses Section 8.11's different message and role keys.

A reader MUST compare all 16 tag bytes using control flow independent of the first mismatch, such as XOR/OR accumulation. This is not a JavaScript constant-time guarantee. HMAC may be computed incrementally over arbitrary chunks or in one call; the entire defined message has the same result.

No extracted file contents may be presented as successful or cached as trusted until entry authentication, complete decoding, applicable size checks and the plaintext CRC check pass. Archive transformations have the specific verification scope in Sections 11.5 and 11.6.3. When a source can be reread stably, readers SHOULD verify ciphertext before decryption. Otherwise they may decrypt to a provisional sink, authenticate that pass and discard its output on failure. Checked and used bytes MUST come from the same source state; a two-pass check of one revision does not authorize decryption of another. Directory verification has the stronger verify-first requirement in Section 10.6.2.

The stored verifier is compared with Section 6.2's two-byte result before decrypting. A match permits continuation and proves nothing about integrity; roughly one in 65,536 wrong passwords passes it. A mismatch is a verifier failure, not proof of a wrong password: corruption of this unauthenticated field can reject the correct password. Its comparison need not follow the tag's data-independent rule. No verifier-repair procedure is required.

Rationale. Encrypt-then-MAC authenticates ciphertext before trusted use [R19] [R20]. The descriptor binds the decoder and mode binds the choice of metadata protection. An unchanged protected payload cannot be repackaged as a public-mode file with a valid tag. Legitimate mode conversion needs its MAC key and new tag, not new ciphertext. WebCrypto supports native PBKDF2, short key-expansion HMACs and chunked AES-CTR. Its HMAC interface is one-shot, however: bounded-memory processing of large files needs a separate incremental HMAC implementation. Independent chunk HMACs cannot be combined to obtain this tag. This implementation tradeoff does not change the format or permit unauthenticated output [R16].

7.5 Actual compression methods and flags

ActualMethod	Decoder	ActualFlags
0	STORE; the S decrypted stream bytes are output	0
8	Raw DEFLATE (RFC 1951), without zlib or gzip wrapping	Bits 1-2 accepted as authenticated no-op level hints; writer sets 0
9	Deflate64	As DEFLATE
12	BZIP2	0
14	ZIP LZMA, end-of-stream marker required	Bit 1 set; bit 2 clear
93	Zstandard	0

Readers also accept deprecated Zstandard method 20 as stored and decode it as 93. Newly created descriptors MUST use 93; unchanged pass-through may retain 20 under Section 11.5. Other ActualFlags bits MUST be zero. A method/flag combination outside this table is unsupported. A reader lacking a permitted decoder reports unsupported for that entry.

The compressed stream MUST be self-contained: no external dictionary or prior-entry history may be supplied. Every permitted decoder determines its end without relying on an unauthenticated uncompressed length [R26]. BZIP2 consists of one or more complete concatenated streams; Zstandard consists of one or more complete frames, including skippable frames. Decode them in order within the S-byte stream region. The final stream/frame MUST consume exactly those S bytes, excluding the four-byte CRC trailer. Truncation and trailing whole bytes fail; ordinary unused bits inside the final DEFLATE byte are not trailing bytes.

After authentication and complete decoding, compare output length to U. Reaching U is not successful stream termination and MUST NOT truncate the output into an apparent success. A reader MAY abort decoding once output exceeds U; extraction then fails, with Section 10.2's classification. Independent resource limits still apply, even to authentic data.

Writers may use any permitted method. A writer wrapping existing compressed content preserves its method only if its stream is self-contained and terminates exactly at the compressed boundary; ZIP LZMA needs its end marker. To wrap content that does not satisfy those conditions, first decompress and recompress with a permitted method. A writer may instead refuse the operation. This validation is distinct from copying an existing encrypted payload under Section 11.5. ActualFlags, not outer ZIP compression-hint bits, initialize the decoder.

8. ZIP records and archive layout

8.1 Method 91 and the 0x9101 compression descriptor

Each method-91 file header carries exactly one descriptor:

Offset	Size	Field	Value
0	2	Header ID	`0x9101` (`01 91`)
2	2	Data Size	8
4	2	Signature	`ZE` (`5a 45`)
6	2	SuiteID	1
8	2	ActualMethod	Section 7.5
10	2	ActualFlags	Section 7.5

Canonical STORE: `019108005a45010000000000`. Canonical DEFLATE: `019108005a45010008000000`.

The `ZE` signature identifies the format, not the writing application. All implementations use these same two bytes.

Parse bounds first. A truncated item, absent/duplicate descriptor or wrong signature is malformed. After checking the signature, an unknown SuiteID, including 0, is unsupported; its layout MUST NOT be parsed as suite 1. Suite 1 requires Data Size exactly 8, with no trailing descriptor bytes. A descriptor whose method or flags violate Section 7.5 is unsupported. An extra field `0x9101` on a non-91 entry does not identify that entry as ZEE.

Writers MUST place byte-identical descriptor copies in the local and central headers and use those same bytes in the MAC. Normal readers use only the central copy; the local copy exists for local reading and recovery. All 12 bytes, including the TLV wrapper, are authenticated.

Rationale. The assigned Header ID identifies the family, the signature follows APPNOTE's third-party collision guidance, and SuiteID selects the entire contract [A1]. One exact encoding keeps MAC input unambiguous.

8.2 Flags and ZIP data descriptors

Logical method-91 records MUST set bits 0 and 11 and clear bits 6 and 13. Bit 3 is set exactly when a data descriptor follows the payload. Writers clear compression-hint bits 1-2; readers ignore them and use ActualFlags. Other flag bits retain their ordinary ZIP meaning; they do not select ZEE cryptography. Public local headers use the same logical flags. Protected local flags follow Section 8.9. Directory entries clear bits 0 and 3.

Writers MAY use data descriptors for regular files. Readers MUST support the two forms below, each beginning with the required signature:

```
Classic, 16 bytes:
  50 4b 07 08 | CRC32:u32LE | P:u32LE | U:u32LE
ZIP64, 24 bytes:
  50 4b 07 08 | CRC32:u32LE | P:u64LE | U:u64LE
```

The record immediately follows the complete payload, including its tag, and is not counted in P. For public mode its values MUST be `0, P, U`. For protected mode all three values MUST be zero, following the encrypted directory convention of APPNOTE 4.3.9.6 [A1]. There is no unsigned descriptor form in ZEE-1 and no descriptor on a directory entry.

When bit 3 is set, classic local size fields are zero placeholders. A writer reserving ZIP64 sets both local size fields to `0xFFFFFFFF` and emits a ZIP64 size extra with both 64-bit placeholders zero. It MUST retain that ZIP64 size form in the final central record even if the values fit in 32 bits. Both central size fields then contain sentinels and the ZIP64 extra contains actual U followed by P. Reserving ZIP64 for small entries is permitted. A writer that might exceed a classic field MUST reserve ZIP64 before emitting a header it cannot later enlarge.

Descriptor width follows the central record's ZIP64 **size** form: if either size uses a ZIP64 expansion, the descriptor is 24 bytes; otherwise 16. An offset-only ZIP64 expansion does not widen the descriptor. Writers MUST use matching local/central size forms for streamed entries. Normal readers derive the width and P from the central record, locate the descriptor at the computed payload end, and validate its signature, complete length and values. They MUST NOT search ciphertext for its signature. Bit 3 belongs to ZIP framing and is not copied into ActualFlags or added to the entry MAC.

Without bit 3, a writer supplies final local sizes before publishing that header, using known-length input, staging or seekable output as appropriate. Permitting descriptors does not require any writer to implement streaming. Descriptors solve unknown sizes during writing; they do not provide per-chunk authentication or change Section 7.4's output-success boundary.

8.3 Version fields

A writer sets logical Version Needed to Extract to the maximum of 51 and the APPNOTE minimum for the entry's actual compression and other features. LZMA requires 63; ZIP64's 45 does not raise 51. Protected local headers use 51 as specified in Section 8.9. Version Made By is ordinary ZIP.

Readers MUST recognize method 91 before dispatch based on version or other encryption flags and MUST NOT use either version field as ZEE identity. A Version Needed below the writer formula is tolerated. Unknown suites and actual methods are handled by their own fields, not guessed from a version number. Version 62 is not a ZECD signal.

8.4 CRC-32

Every ZEE-1 regular file carries one authoritative CRC-32 of its complete uncompressed plaintext: the four little-endian bytes encrypted after its compressed stream in Section 7.1. Calculate ordinary ZIP CRC-32 [A1] while processing the original input, before compression. Extraction, content testing and recovery MUST compare the calculated CRC of fully decoded plaintext with this decrypted trailer after entry authentication, in either mode. A mismatch is `crc-mismatch`, not a password or MAC failure. Password removal may instead preserve the authenticated expected CRC without decoding, as Section 11.6.3 specifies; it makes no decoded-content validation claim.

All outer method-91 CRC fields MUST be zero: local header, central header and any ZIP data descriptor, in both modes. Readers MUST reject a nonzero value in any of these fields as malformed, including during local recovery. Normal reading checks the local CRC against zero; it does not compare other local metadata with the central record. An outer CRC, zero or nonzero, MUST NOT be used as an expected plaintext checksum. Other methods in a public ZIP retain their own CRC rules.

The trailer supplies an authenticated expected checksum only after the entry MAC verifies. Comparing it with fully decoded output checks processing errors beyond the ciphertext MAC's scope; it neither replaces MAC or ZEAC checks nor detects every fault. CRC is not a cryptographic authenticator, and a matching CRC MUST NOT authorize output, retagging or re-encryption after a failed MAC. Public CRCs can reveal short plaintext through guessing [W1]. Encrypting the sole expected CRC with its file avoids that disclosure even to a directory- password holder who lacks that file's different password.

8.5 Directories and empty regular files

A directory entry is identified by a name ending in `/` or directory attributes. Its writer form is method 0, zero sizes and CRC, bit 0 clear, and no payload, compression descriptor, association or data descriptor. A directory-shaped record with nonzero U is malformed. Protected-mode directories must use the complete writer form; other directory shapes outside that supported form are unsupported. Public ZIP import retains ordinary non-ZEE dispatch, and canonicalizes directories when creating protected output.

These size checks apply to resolved values. A directory record may carry a ZIP64 extra holding zero sizes, as Section 8.8.1 permits for small values; its 32-bit sentinel fields are not nonzero sizes. Implementation note: resolve the extra before checking the writer form, or a valid ZIP64-shaped directory is refused.

An empty regular file is an ordinary encrypted file. The canonical writer form is STORE, U=0, S=0, C=4, P=54 and actual CRC=0. A reader MUST also accept valid empty streams under other permitted methods. U=0 or any other field MUST NOT bypass entry authentication or protected-mode association. There is no empty-file KDF, verifier or authentication shortcut.

8.6 Registries and coexisting extra fields

SuiteID	Meaning
0	Unsupported; not a valid suite
1	ZEE-1, this complete contract
2-65535	Unassigned; unsupported

ZIP Extractor vendor extra field `0x455A` has a four-byte ASCII tag at the start of its data area. Each occurrence contains one tag and that tag's body, bounded by the enclosing Data Size. It is separate from the assigned `0x9101` descriptor. The registration request covers these tags:

Tag	Definition and permitted ZEE location
`ZEAC`	Section 8.10.2; a protected central file record
`ZEXT`	Optional producer marker, defined below; central records and public local headers

Extras are a bounded sequence of Header ID, Data Size and that many data bytes. Readers MUST scan by tag as well as Header ID: multiple `0x455A` items with different tags coexist. A last-wins map keyed only by Header ID cannot implement these rules. Unknown tags keep ordinary unknown-extra handling and confer no authentication. In their defined protected locations, ZEAC obeys its exact size and placement rules. ZIP64 extras coexist; masked locals admit only the fields in Section 8.9. Public writers omit ZEAC; public readers treat copies as ordinary unknown extras and MUST NOT use them as evidence of metadata authentication.

The optional ZEXT version-1 marker is fully described by:

```
HeaderID:u16LE = 0x455A, DataSize:u16LE = 8
data = ASCII("ZEXT") | version:u8 = 1 | major:u8 | minor:u8 | patch:u8
```

Thus its complete byte prefix is `5a4508005a45585401`, followed by three binary application-version components. A writer emitting it uses at most one per header and omits it rather than wrapping out-of-range components. Version 1 requires exactly eight data bytes. Unknown marker versions are opaque, not parsed as version 1. Readers prefer the central copy and may choose one if duplicates exist; this does not relax ZEAC's stricter rule. The marker is a producer self-claim and supplies no independent identity or authority, even when a password-authenticated directory carries it. No ZEE implementation is required to emit a ZIP Extractor producer marker.

ZECD itself is an archive-level record, not an extra field or another `0x455A` tag. Its signature consumes no PK-prefixed record signature. Leading tagged vendor data has APPNOTE precedent, including `0x0065` [A1].

8.7 Basic recovery without a usable directory

Recovery is explicit and best-effort, never automatic fallback after normal opening fails. An intact local compression descriptor and complete payload can permit content salvage. A candidate masked local header (bit 13 set) selects mode `01`; an ordinary unmasked candidate selects `00`. These are untrusted hypotheses until the entry MAC verifies.

Before offering recovered contents, require a file payload of at least 54 bytes and zero outer CRC fields wherever those records are used. A reader MUST authenticate the full payload with the candidate descriptor and mode, decode exactly its S-byte stream, check the encrypted CRC trailer and apply resource limits. Use generated names. Do not claim original names, order, membership, completeness or expected size were authenticated from local headers. A masked U=0 is not the expected plaintext length.

Known-size local entries retain straightforward boundaries. Streamed entries with unknown local sizes may not be recoverable after directory loss. This specification adds no ciphertext-signature search, verifier repair or forensic reconstruction procedure and makes no general recovery promise. An intact file carries its own expected CRC even when the directory is missing; that does not establish archive completeness.

8.8 Archive layout

A file record is a central file header describing a ZEE-1 regular file; a directory record describes a filesystem directory. A public ZIP may also contain other ordinary ZIP entry types. In protected mode the archive password encrypts ZECD; individual files may retain other passwords.

```
for each entry:
  local file header (masked in protected mode)
  file payload, if any
  signed data descriptor, if bit 3 is set
central directory:
  ordinary central records                public mode
  OR ZECD header16 + encrypted payload    protected mode
optional ZIP64 end record and locator
ordinary end of central directory record + public comment
end of file
```

This profile is single-file, not split across disks. Disk-number fields MUST be 0; a ZIP64 locator's total number of disks MUST be 1. Writers emit compact archives from offset zero, with entries contiguous and the current directory immediately following the last entry. No obsolete directory, deleted-entry remnant, preamble or unreferenced gap is emitted on rewrite.

Readers retain ordinary offset-based access and may accept earlier prefixes or gaps. Those bytes are not authenticated and gain no new semantics here. Entry headers, payloads and any descriptors MUST fit in the source before the resolved directory range. The directory and terminal records MUST be contiguous: directory, optional ZIP64 end record and locator, ordinary end record and its declared comment, then EOF. Extra terminal or trailing bytes are malformed. Writers emit the fixed version-1 ZIP64 end record (size 44 after its size field). Readers may accept an ordinary version-1 extensible sector when the entire declared record is bounded and correctly resolved; its bytes have no ZEE authentication or newly assigned semantics. Version-2 ZIP64 end records are unsupported and writers emit no digital signature record or Archive Extra Data Record.

An empty protected archive still contains ZECD: C=0 and the record is 66 bytes. With an ordinary end record, it is 88 bytes before any public comment or optional ZIP64 closing pair. An empty public ZIP has only ordinary ZIP structure and makes no password-protection claim.

8.8.1 End records, offsets and ZIP64

End-record directory size and offset describe the ordinary public directory or the entire ZECD record (`66 + C`). Counts are true central-record counts. Protected readers obtain the authoritative count by parsing the complete decrypted record sequence, never by stopping at the public count. Public end-record counts are not authenticated. The comment is public and unauthenticated in both forms (Section 8.12).

A reserved sentinel (`0xFFFF` for counts, `0xFFFFFFFF` for 32-bit sizes or offsets) requires the corresponding ZIP64 value. Values at or above those reserved boundaries need ZIP64. Writers MAY select ZIP64 even for small values. Readers MUST also accept a valid adjacent ZIP64 closing pair when ordinary end fields contain small actual values; any repeated actual values MUST agree. For this single-file profile, per-disk and total counts MUST agree in each end structure. Reserving ZIP64 entry sizes does not by itself require an archive-level end-record sentinel.

Resolve the end records by ordinary ZIP rules. The directory's end is the start of the ZIP64 end record if present, otherwise the ordinary end record. Subtract its resolved size to obtain its physical start; subtract the declared relative offset from that start to obtain the archive origin. The origin MUST be nonnegative and applies consistently to local-header offsets and the ZIP64 locator's offset. Use checked arithmetic and verify every range and terminal adjacency. An earlier prefix can shift the origin without changing relative offsets. Do not scan arbitrary other positions for ZECD during normal opening.

The selected ordinary end record and declared comment MUST end at EOF. Once selected, signatures inside the comment are just comment bytes. This profile retains ordinary end-record selection rather than defining a new ambiguity-search algorithm. Once ZECD is recognized at the resolved directory start, a later failure MUST NOT cause another interpretation.

8.9 Local file headers

In public mode, writers use ordinary local headers with real UTF-8 names, ordinary metadata and the file descriptor. CRC is zero for method 91; size fields contain actual sizes or Section 8.2's streamed placeholders.

In protected mode, every entry has the following masked local form:

Field	Value
Signature	`50 4b 03 04`
Version Needed	51
Flags	Bit 13, plus bit 3 for a streamed file; all other bits clear
Method	91 for a file, 0 for a directory entry
Time, date, CRC	0
Classic sizes, no bit 3	P and 0
Classic sizes, bit 3	0 and 0
ZIP64 size form	Both 32-bit fields `0xFFFFFFFF`; extra below
Name	1-based physical ordinal, lowercase hexadecimal without leading zeros
File extras	Optional ZIP64 size extra first, then exactly one descriptor12
Directory extras and payload	None

The masked ZIP64 size extra is `01 00 10 00`, followed by U=0 and P, both uint64 little-endian. With bit 3 set, both values are zero placeholders. A writer uses this form when P needs ZIP64 and may reserve it for smaller or unknown sizes; the corresponding central size form follows Section 8.2. The local name and extra lengths count their actual byte lengths. No ZEXT, timestamp extra or other confidential metadata is emitted in masked locals. Their visible method, payload size and ordinal naming follow APPNOTE's masking precedent; ZEE's cryptographic dispatch remains its own [A1].

For one ordinal byte, a masked file header is 43 bytes, or 63 with ZIP64; a directory header is 31 bytes. The payload begins immediately after the fixed 30-byte header, filename and extras.

Normal readers use the local signature and the two length fields to locate that payload and MUST check that its local CRC is zero. They MUST NOT compare or repair other local fields against the central record or use them as alternate decoding metadata. In particular, normal mode selection comes from the directory form, not local bit 13. The precise local form is a writer obligation; Section 8.7 is the separate path that uses local metadata as a recovery hypothesis.

8.10 Central directory records

8.10.1 Records and text

Public mode uses ordinary central headers. Its method-91 records follow Sections 8.1-8.5 and 10.1; other ZIP formats retain ordinary dispatch. A public directory supplies no ZEE authentication of names, lengths, attributes, order or membership. A successful entry MAC does not elevate it.

Protected plaintext consists only of ordinary central file headers. Writers emit ZEE-1 file records and Section 8.5 directory records only. Supported file records require a ZEAC association and zero outer CRC. A different regular-file kind is unsupported, not an invitation to extract it through a generic decoder; it may be listed as unsupported without preventing other entries from opening. A method-91 record with malformed framing remains malformed, not an unsupported escape.

Every central record written under the ZEE profile MUST set bit 11 and encode names and per-file comments as strict UTF-8 without normalization. Protected readers reject missing bit 11 or malformed text. Public method-91 readers apply the same rule to their ZEE records, without imposing it on unrelated legacy entries. Import transcodes resolved text; merely setting bit 11 on legacy bytes is insufficient. No new duplicate-name, path or filesystem policy is defined. Binary extra fields keep their own encodings.

Protected central headers clear bits 6 and 13; file bit 3 follows the descriptor rule and directory entries clear it. Central offsets are relative to the resolved archive origin. Ordinary ZIP fields and ZIP64 extras retain their meanings, with P counting the complete encrypted file payload.

The header rules every protected record shares (bit 11 set, bits 6 and 13 clear, disk number 0 under Section 8.8, strict UTF-8 text) apply before its kind is considered. A record that breaks one is malformed, even when its kind alone would be unsupported; only a correctly formed record of another kind is unsupported.

8.10.2 ZEAC association

Each protected ZEE-1 file record carries exactly one association:

```
HeaderID:u16LE = 0x455A, DataSize:u16LE = 20
data = ASCII("ZEAC") || expectedTag16
```

The complete prefix is `5a4514005a454143`, followed by the expected entry tag. Missing, duplicate, wrong-size or directory-entry ZEAC is malformed. ZEXT and other tagged vendor records can coexist; scan by tag.

Before deriving or using the file keys, a reader MUST compare the payload's stored tag with this authenticated expected tag. A mismatch is an entry authentication failure, never a password error. A match does not verify the payload's interior. Successful extraction additionally requires its actual entry MAC, decoding, size and CRC checks.

Same-mode pass-through MUST carry forward the authenticated source's expected tag, not replace the expectation by rereading a possibly substituted payload suffix. The suffix comparison is required before copying. It can preserve pre-existing interior damage, which actual entry verification detects later; it MUST NOT be called full-payload verification.

Rationale. Offsets, equal lengths and even matching plaintext CRCs do not identify contents. Another genuine encrypted payload can share both; its own valid MAC does not identify it as the intended record. ZEAC supplies that association against substitution of a different tag. It is not a claim of collision-resistant commitment against an entry-key holder deliberately constructing colliding 128-bit tags.

8.10.3 Exact consumption and count

Protected plaintext MUST consist entirely of complete central headers beginning `50 4b 01 02`. A field or extra running past its bounds, any archive-extra record or leftover byte is malformed. No inner end record is stored. The count is the number of parsed central headers. An empty sequence is valid, irrespective of any public archive comment.

Precedence. This framing is checked over the whole sequence before any record is interpreted, and a failure in it makes the whole listing malformed: nothing is listed, and it is not a password failure. Within a correctly framed sequence, the record rules of Sections 8.1-8.5, 8.10.1 and 8.10.2 apply entry by entry: a record that breaks one is listed as malformed or unsupported, and the other entries still open. Rationale: framing decides where every later record starts, so nothing after a framing failure is known to be a record; a record rule concerns only its own record. The same extra-item overrun is therefore a whole-listing failure here and one malformed entry in a public directory.

8.11 ZECD record and directory cryptography

Offset	Size	Field
0	4	`ZECD` (`5a 45 43 44`)
4	2	Record version 1
6	2	SuiteID 1
8	8	Ciphertext length C, uint64 little-endian
16	16	rootSalt
32	16	nonce
48	2	verifier
50	C	ciphertext
50 + C	16	tag

Total length is `66 + C`. The first 16 bytes are `header16`. Plaintext is exactly Section 8.10's record sequence, uncompressed and without a CRC trailer. Use Section 6's root derivation and directory role labels, and Section 7.3's CTR convention:

```
tag = first 16 bytes of HMAC-SHA-256(Kmac,
      ASCII("ZEE-1/dir/tag") || header16 || ciphertext)
```

The prefix has 13 ASCII bytes and no terminator. There is no mode byte in this MAC: the distinct role, prefix and ZECD header identify its meaning. Public comment bytes are not part of this message, and no comment hash or comment-authentication record is stored.

Root salt may be shared with files and retained across edits. Every new directory encryption MUST draw a fresh nonce. There is no directory/file salt inequality or permanent nonce registry.

At the resolved directory start, recognize the four-byte signature first. Read the complete version field first; an unknown version is unsupported without parsing its body. For version 1, read the complete SuiteID; an unknown suite is unsupported without parsing its body. A missing or partial required identifier is malformed. Known version 1/suite 1 requires the full 16-byte header and at least 66 record bytes, a valid C within Section 9.5's bound and resolved declared directory size exactly `66 + C`. Verify the tag before decrypting or parsing the directory. C is only an untrusted candidate boundary until authentication, never permission for an unchecked allocation or read. Normal opening examines no alternate ZECD position.

8.12 Public archive comment

The archive comment is the final field of the ordinary end record. It is public and unauthenticated in both modes. If displayed, it MUST be identified as unverified, including after directory unlock. It is not an input to any ZEE MAC and has no protected copy, hash or authentication record. Per-file comments inside ZECD remain encrypted and authenticated central metadata.

Newly supplied archive-comment text uses strict UTF-8 without normalization; unrelated pass-through preserves existing bytes. Comment contents confer no authentication and do not control access to files. Ordinary end-record length, bounds and EOF rules still apply; structural damage can prevent normal opening.

9. Sizes, ZIP64 and resources

9.1 File cryptographic bound

HMAC's inner SHA-256 input has 64 key-pad bytes, 9 prefix bytes, one mode byte, the 12-byte descriptor and C ciphertext bytes: `86 + C`. SHA-256 accepts at most `2^61 − 1` whole message bytes [R29], giving:

```
C <= 2^61 − 87
S = C − 4 <= 2^61 − 91
```

The outer hash and counter space impose no tighter limit; at most `2^57 − 5` counter blocks are needed. Implementations MUST use checked, exact integer arithmetic for lengths and offsets, not lossy double or 32-bit arithmetic, and enforce this bound before and during processing. It is a construction limit, not an advertised product capacity.

9.2 ZIP64

File sizes are P=50+C=54+S and U. ZIP64 expands size and offset fields at their reserved sentinel values by the ordinary rules, with Section 8.2's explicit streamed-size form. Payload framing and cryptography do not otherwise change. The 64-bit ZIP uncompressed-size representation and implementation resource limits also apply. File-size expansion and archive-end expansion are distinct.

9.3 Resource envelope

Implementations MUST document supported sizes and hard resource limits, check known limits before starting and enforce them as data flows. Runtime storage failures remain possible after preflight. A tested size target is not a new format cap. Output, decompression-ratio and time limits apply to authenticated data too. Resource refusal or cancellation must not be shown as successful extraction or as a conclusive wrong-password diagnosis.

9.4 Practical validation

Large compressed plaintext can exercise ZIP64 while producing little ciphertext. Validation therefore needs both container-size tests and an independent long, bounded-memory MAC/cipher stream. Section 14.7 defines that evidence. Product capacity claims need measurements of the complete implementation and its storage path.

9.5 Directory bound

Directory HMAC inner input is `64 + 13 + 16 + C = 93 + C`, so:

```
C <= 2^61 − 94
```

The record's full size is 66+C. Compare that against the resolved ZIP64-aware declared directory size, never the raw sentinel field. Apply implementation resource limits before buffering, decrypting or parsing the directory.

10. Reader

10.1 Entry recognition

Normal reading obtains a central record from the resolved public directory or from a verified and parsed ZECD. Recognize a method-91 entry before any generic encryption or version dispatch:

1. Locate exactly one bounded `0x9101` descriptor; check signature and SuiteID as Section 8.1 specifies. An unknown suite is unsupported.
2. For suite 1, require Data Size 8, logical bits 0 and 11 set and bits 6 and 13 clear. Apply the directory-kind and UTF-8 rules of Section 8.
3. Require supported ActualMethod/ActualFlags and P at least 54; derive C=P-50 and S=C-4 with checked arithmetic and enforce the ciphertext bound.
4. Require central CRC zero in either mode. Protected mode additionally requires the file's ZEAC under Section 8.10.
5. Use bit 3 and central ZIP64 size form for any following data descriptor.

The central record supplies method, flags, descriptor, sizes and offset. The local signature and length fields locate the payload; its CRC MUST be zero. Other local metadata is neither preferred nor compared. The directory form selects MAC mode. A recognized entry never falls back to WinZip AES, ZipCrypto, another encryption format or suite, or a different password encoding.

10.2 Entry authentication and decoding

1. Resolve and bound the local header, payload and any data descriptor; require zero outer CRC fields under Section 8.4. With ZECD, compare the stored payload tag to the authenticated ZEAC expectation before any object- key derivation or use. Abort on mismatch.
2. Read rootSalt, nonce and verifier. Obtain R from the authorized cache or derive it, then expand the file keys and verifier. A verifier miss ends this attempt with the non-conclusive result in Section 10.3.
3. Authenticate the exact prefix, implicit mode, central descriptor and C ciphertext bytes, including the encrypted CRC trailer. Prefer a stable verify-first pass; otherwise decrypt and decode provisionally while computing HMAC over the bytes actually used. Supply only the first S decrypted bytes to the codec; retain the last four as the expected CRC32LE.
4. Compare all 16 tag bytes. On failure discard provisional output and report authentication failure. If decoding fails during a combined pass, discard its output but finish the bounded ciphertext MAC pass to classify the failure: a bad tag is authentication failure; a valid tag permits reporting the decoding, size or CRC error. Resource refusal, cancellation or I/O failure may end processing without an authentication verdict.
5. Require complete decoding of the S-byte stream and output length U. In either mode, compute the decoded plaintext CRC and compare it with the decrypted four-byte trailer. A matching CRC never excuses a bad tag.
6. Validate any data descriptor at the computed endpoint against Section 8.2, if not already done. Only after all applicable checks pass may the output be committed or reported as successfully extracted.

An implementation may order independent structural checks earlier without treating unauthenticated metadata as trusted. It MUST NOT turn a decoder error, early stream termination or partial output into success.

10.3 Failure classes

Class	Meaning
Not a ZEE entry	Method is not 91; public ZIP uses ordinary dispatch; a different regular-file kind inside ZECD is unsupported
Malformed	A recognized layout, length, flag, text or record rule fails
Unsupported	Unknown suite/version, unsupported actual method or flags, missing decoder or required primitive
Did not verify	The two-byte verifier did not match; this is not proof of a wrong password
Authentication failure	Entry/directory MAC failure, or protected entry association mismatch
Decoding or size failure	The tag verified but the compressed stream or decoded length is invalid
crc-mismatch	Authenticated and decoded content disagrees with its encrypted CRC trailer
Resource/cancel/I/O failure	Operation stopped without claiming successful output or an authentication verdict it did not establish

Readers MUST distinguish these results internally. These are failure categories, not prescribed user-interface strings. Applications MAY use their common incorrect-password prompt for verifier and MAC failures while retaining the detailed result for diagnostics. Neither failure proves whether the password is wrong or the archive is damaged: wrong passwords can reach the MAC and damaged verifier bytes can reject correct passwords. Resource, unsupported-format and authenticated-content failures are not password verdicts and MUST NOT be classified as such.

10.4 No reinterpretation

After recognizing ZECD or method 91, failure MUST NOT trigger another normal interpretation, local-header fallback, lower work factor, alternate hash, cipher, actual method or password transformation. Retrying with another user-supplied credential is permitted. Explicit recovery is Section 8.7. This rule does not claim protection against replacing the whole archive with unrelated non-ZEE material.

10.5 Successful output and resources

An authenticated directory is a verified listing and set of expectations, not proof that every referenced file's contents have been verified. A keyless copy is likewise not a contents test. Successful extraction needs every applicable check in Section 10.2; archive transformations use the checks specified in Sections 11.5 and 11.6.3. Provisional storage and remote uploads must not be committed or shown as complete before the operation's required checks pass. Cancellation stops scheduling new work and discards cancelled results; an already-running platform cryptographic call need not be abortable.

Public names, offsets and sizes remain attacker-controlled metadata even after successful entry authentication. Protected names are password- authenticated, not intrinsically safe filesystem paths. Ordinary extraction path safety and resource policies remain the application's responsibility; ZEE does not redefine ZIP filename uniqueness or filesystem semantics.

10.6 Archive recognition, unlock and extraction

10.6.1 Recognition

Resolve the terminal chain, directory range and archive origin under Section 8.8.1. Before ordinary central-header parsing or version-62 decryption dispatch, inspect the resolved directory start for `ZECD`. A match selects protected mode and Section 8.11's bounds-first parser. No other location is examined during normal recognition. A nonmatch leaves ordinary ZIP processing, including public-mode method-91 recognition, in effect.

10.6.2 Unlock

For ZECD, obtain the directory root and expand its role keys. Compare the verifier, then MUST verify the complete directory MAC before decrypting or parsing any directory plaintext. This can use retained ciphertext or a pinned stable source; if stability cannot be established, retain the record before verification. Do not weaken directory verify-first to unverified parsing. A verifier miss and tag failure are distinct, non-conclusive results; either may be retried with another credential.

After authentication, decrypt and parse the complete directory grammar. Invalid authenticated grammar is malformed, not a password failure. Directory root authentication authorizes reuse of that credential/root context, not acceptance of every file key or the assumption that all entries use the directory password.

10.6.3 Listing

Public metadata can be listed without password derivation and MUST NOT be presented as authenticated. Protected metadata is listed only after the directory MAC and grammar succeed. Unsupported protected record kinds may be listed as unsupported; masked local names never supply logical names. The archive comment remains unverified after unlock.

10.6.4 Extraction

Apply Sections 10.1 and 10.2 to each selected ZEE-1 record. Protected mode uses authenticated metadata and association; public metadata remains unauthenticated. Both forms require the encrypted plaintext CRC check. A file needing another credential may be retried separately. One entry's unsupported algorithm, verifier failure, association failure, bad MAC or bad CRC does not invalidate other independently processable files.

10.6.5 Directory failures

A recognized ZECD with an unknown record version or suite is unsupported. A truncated header, invalid range, `66+C` size mismatch or invalid protected record grammar is malformed. A verifier miss is a verifier failure; a tag failure is an authentication failure. None permits another directory interpretation or an automatic recovery listing.

11. Writer

11.1 Creation

Select public or protected directory mode explicitly. Apply Section 5's new-password rule, draw a root salt and derive the root. A normal new single-password archive uses that context for all newly encrypted objects. The writer's supported compression methods and output strategy may be limited without adding alternate cryptography. An unsupported operation is refused rather than encoded with different suite parameters.

11.2 File encryption

For each regular file being encrypted:

1. Draw a fresh nonce; expand file encryption, MAC and verifier values.
2. Build the compression descriptor once from the chosen permitted method and actual flags. Compress or validate a self-contained source stream. Track U and the complete plaintext CRC in either mode. For existing compressed source input, establish these from decoding or a trustworthy source for the same unchanged content, never an unauthenticated ZIP CRC.
3. Enforce known and streaming size/resource limits; append CRC32LE to the S-byte compressed stream, encrypt both continuously with the counter appropriate to each offset, and compute the mode-bound MAC over all C bytes.
4. Emit the local header and complete payload with the required copies of the descriptor and zero local CRC. If using bit 3, append its signed descriptor. Store actual P and U in the final central record.
5. Store zero central CRC in either mode. Store the emitted tag as ZEAC for protected mode; omit ZEAC for public mode.

Empty regular files follow exactly this procedure. Directory entries use Section 8.5's no-payload form.

11.3 Header emission

The descriptor MUST occur once in each file header and be byte-identical to the MAC input and its other copy. Local form, flags, ZIP64 reservation and descriptor values follow Section 8. These are writer obligations even though normal readers do not compare local metadata against central data. The writer's conformance tests MUST verify this identity and the exact canonical headers for both modes and descriptor widths.

11.4 No downgrade

A writer MUST NOT silently change the selected directory protection, cipher, suite, work factor or password preparation in response to a failure. Unavailable primitives, randomness failure, resource refusal, cancellation or failed required source verification abort the operation. An explicit request to change directory mode is Section 11.6.3, not a failure fallback.

11.5 Same-mode pass-through

An unchanged ZEE-1 payload MAY be copied without decrypting or retagging when its directory mode stays the same. Preserve its complete payload, compression descriptor and cryptographic context. This preserves an existing descriptor using method 20 or nonzero permitted hint flags; it is not newly encoding compressed data.

For protected sources, authenticate the source directory first, compare the payload suffix with its expected ZEAC, and carry forward that expected tag and associated metadata from the authenticated source. Do not replace trusted expectations with a reread payload suffix. Public-mode pass-through preserves bytes without claiming authentication of their source names, metadata or contents. A public zero CRC is not a computed plaintext checksum.

Every check and subsequent use MUST refer to the same source state. Retained bytes or a pinned source revision are ways to ensure that. Rebuild local headers from the selected central metadata and current mode; do not copy stale masked ordinals or source offsets. If a copy changes the framing strategy, regenerate its data descriptor and ZIP64 fields consistently. Changing bit 3 alone does not change the file MAC message.

Independent payload copying supports different roots and passwords. It does not verify interior ciphertext without the file key, even if ZEAC matches. An operation advertised as testing or verifying file contents must actually perform Section 10.2's checks.

11.6 Archive creation, editing and conversion

11.6.1 Completing an archive

After entries, write the central records in public mode, or build the protected plaintext (central headers only), draw a fresh directory nonce and emit ZECD. Finish with the ordinary closing records. In protected mode all new regular files are ZEE-1 files, and an empty archive still has ZECD. A public ZIP may mix ordinary entry formats; ZEE rules apply to its method-91 entries without redefining the others.

11.6.2 Editing

Same-mode edits such as rename, reordering or removal need not retag unchanged files. Preserve their cryptographic bytes and protected expectations according to Section 11.5. Protected output re-encrypts the current directory with a fresh nonce, normally under its existing root. Archive comments remain unauthenticated on rewrite. Compact the output as Section 8.8 requires rather than carrying old directories or deleted-entry remnants.

11.6.3 Conversion, password removal and password change

Changing directory mode requires authenticating the source file MACs and calculating new tags with the target mode byte. A protected source also requires directory authentication and ZEAC checks. The old and new MACs can be calculated in one ciphertext pass, but a new tag MUST NOT be accepted unless the old one verifies. Otherwise conversion could authenticate previously damaged ciphertext. Preserve ciphertext, salt, nonce and descriptor when nothing else changes; update payload tags and protected ZEAC values. This operation needs every affected file's credential/root, not just the directory password. Reusing ciphertext is not a new encryption operation.

Protected-to-public conversion publishes the chosen logical metadata, preserves zero outer CRC fields and omits ZEAC. It can reuse the authenticated source sizes and encrypted CRC trailer without decoding, subject to the source-MAC check above. Public-to-protected conversion MUST establish actual U and decoded correctness through Section 10.2's authentication, complete decoding, size and CRC checks, unless a trustworthy source for the same unchanged content already establishes those facts. An authenticated expected CRC alone does not establish U; public size fields are not such a source. A forged public U fails the ordinary conversion size check, rather than becoming an authenticated size claim. Public names become chosen input metadata, not retroactively authenticated historical assertions; the archive comment stays public and unverified. A public archive's other regular-file formats must be converted to supported ZEE-1 files for protected output. WinZip AES ciphertext cannot be copied as ZEE-1 ciphertext.

Removing file encryption MUST verify every affected file's original MAC and, for a protected source, its directory and ZEAC association. It MAY preserve the unchanged decrypted compressed stream without decoding it: emit exactly S stream bytes, strip the encryption framing and four-byte trailer, and use the authenticated trailer value as the expected CRC in the ordinary ZIP records. Restore ActualMethod as the ordinary method and ActualFlags as its general purpose bits 1-2, and rebuild sizes, offsets and closing records for the ordinary output. The trailer MUST NOT become ordinary file data or a compressed-stream suffix. Source metadata retains its existing trust scope: a public U remains an unverified claim, not a measured decoded size.

This form of password removal authenticates the source bytes but is not a decoded-contents test. It MUST NOT silently replace an incorrect authenticated trailer CRC with a newly calculated value. An operation that also validates contents applies Section 10.2 and fails on a CRC mismatch. Normal extraction, content testing, recovery and public-to-protected conversion retain their full decoding, applicable size and CRC checks. Protected mode permits only encrypted regular files, so removing their encryption produces an ordinary public ZIP.

Changing the password for the whole archive re-encrypts all affected files under a new root salt and fresh object nonces, and re-encrypts ZECD if selected. For a ZEE source, password changes and other re-encryption MUST verify the original file MAC before accepting new output; protected sources also require directory authentication and ZEAC checks. Unchanged compressed content and its CRC trailer may be re-encrypted without decoding; this is not a decoded-contents test. Operations that decode contents apply Section 10.2 before accepting the resulting output. A matching CRC cannot replace original-MAC verification: CTR permits changes to ciphertext and CRC together without knowledge of the key. Re-encrypting only the directory does not change file passwords. No password-rewrapping scheme is defined.

11.6.4 Atomicity

Required verification failure, cancellation or another operation failure MUST abort the write without modifying the source or presenting partial output as a completed archive. Temporary output may be staged, but committed output is gated on the operation's required checks. Report the failing entry and class where possible; mixed-password archives may be retried with additional credentials. Same-mode pass-through has the explicitly limited verification scope of Section 11.5, not an implied full contents test.

12. Application integration

12.1 Credentials

Password/session UI is implementation-specific. Passwords and secret keys MUST NOT be serialized into archive records or diagnostic logs. A reader may retain a supplied credential for an authorized session; exact password bytes and the creation/read distinction remain Section 5's requirements.

12.2 Session keys

Section 6.3 governs root/key caching and disposal. Cache scope must not cross independent authorization sessions merely because public root salts match. A cached root does not authenticate newly encountered payloads; their verifier, MAC and other checks still apply.

12.3 Mixed roots and formats

Multiple credentials or root salts are permitted in either directory mode. Shared salt values do not prove password equality. Public archives may contain other ZIP formats; protected writers emit only supported ZEE-1 regular files and ordinary directory records. Unsupported protected entry kinds are handled as Section 8.10.1 specifies.

12.4 Reporting results

Applications should distinguish a public listing, an authenticated listing, and verified extracted contents. Directory unlock verifies metadata and expected file tags, not every referenced payload. Password removal can preserve authenticated compressed bytes without testing decoded contents. Report success for the operation actually performed; do not present either operation as a completed content-integrity test.

Worker boundaries, progress reporting and storage destinations are implementation choices. They do not change the wire or authorize weaker verification. For each operation, apply its required checks regardless of whether the output is local, downloaded or sent to a remote destination.

13. Security scope

Before unlock, protected archives expose object boundaries, payload lengths, compression descriptors, physical order, ordinal local names, directory/file distinction, root salts, nonces, verifiers, tags, directory length and the public comment. Repeated root salts link objects to a claimed derivation context but do not prove a common password. STORE reveals U exactly. Uncompressed directory length leaks aggregate variable-field lengths; with one entry, isolating filename length still depends on knowing other comments, extras and variable contributions. There is no size-hiding or padding claim. Public mode also exposes the logical central metadata.

Under the threat model and correct implementation:

- Fresh object keys protect file contents using AES-256-CTR; the mode, descriptor and ciphertext are authenticated by a 128-bit truncated HMAC. The ideal per-attempt forgery scale is 2^-128, subject to the strength of the primitives and password. This is not a collision-resistance claim.
- Fresh random nonces under a given root/role make accidental key and keystream reuse negligible at ordinary scale; the birthday collision approximation for N draws is `N(N−1)/2^129`.
- One full PBKDF2 stretch is needed per offline password guess against a chosen root context. More files do not multiply that cost. A leaked root exposes its children; a leaked password exposes matching archive contexts.
- For a fixed authentic payload, the tag binds its decoder and directory mode. Protected-to-public stripping cannot preserve a valid file tag.
- ZECD authenticates its logical metadata and expected file tags. Actual file MACs still verify contents, and encrypted CRC trailers separately check decoded output in both modes. A matching CRC never substitutes for either MAC or ZEAC.

The profile does not claim:

- Sender identity, freshness, replay prevention or external archive identity. Copying an authentic archive does not require its password; replacing the whole file with unrelated material is not detected as a ZEE modification.
- Public-mode metadata integrity or archive-comment authentication. Its content MAC does not authenticate public filenames.
- Availability after damage, guaranteed recovery, trusted local metadata, authenticated preambles/gaps/end fields, or reconstruction of lost records.
- Keyless verification of every payload byte. ZEAC is not a collision- resistant commitment against a key holder constructing colliding tags.
- Formal key commitment for this exact composition; its 128-bit tag is not advertised as a 128-bit collision-resistant commitment [R30].
- Security after releasing unverified plaintext, endpoint compromise, password strength, or resistance to an exposed password-partition oracle [R25] [R30]. A cached root can make repeated trials cheap; PBKDF2 is not a universal per-request rate limiter.
- Constant-time execution in managed runtimes, mathematical impossibility of random collisions, detection of every processing fault by CRC, or immunity to decompression/storage/KDF denial of service.

These limits do not weaken the mandatory output checks. They separate file authentication, protected metadata, CRC checking and recovery rather than making any one of them stand in for another.

14. Conformance and known answers

The algorithms and layouts above are the authority. Reference programs and known answers must reproduce them, not define hidden implementation rules. An implementation may use any library that produces the specified results; compatibility does not depend on ZIP Extractor source code. Fixed salts and nonces are test inputs only.

The known-answer corpus contains complete archive bytes and intermediate values. The entry and archive reference programs and independent WebCrypto checker use this corpus. The SHA-256 pins below identify the exact artifacts for this specification.

14.1 Primitive and entry known answers

The entry oracle reproduces the primitive and entry answers using Python standard-library PBKDF2/HMAC and the `cryptography` AES primitive. Its recorded output includes RFC 7914 [R36], RFC 4231 [R37], RFC 5869 [R11] and the 2001 FIPS 197 AES-256 example [R35]. Hexadecimal fields denote byte sequences unless labeled `crc32`: those values are unsigned hexadecimal numbers, serialized as four little-endian bytes. For example, `crc32 = 414fa339` becomes trailer bytes `39 a3 4f 41`. A file's CRC trailer continues the same cipher stream; it does not select another key or counter sequence.

All entry examples use rootSalt `000102030405060708090a0b0c0d0e0f`, nonce `101112131415161718191a1b1c1d1e1f`, and 1,000,000 iterations. The repeated test inputs are fixtures, not writer randomness policy. HKDF info is the specified label immediately followed by the nonce.

KDF-1. Password bytes `70617373776f7264`.

```
root = d9de601903a32772e93c2c6bdd99733fda92d1a7cdfdd85246500182c27bb641
kenc = 67ec324cd7080c641f4e6ede2df4363128f2d06cfe9aff061653241086093a59
kmac = dc34a149f47197dc51cc6b2742d05b63dca2befe86f3e6fbe27b5e292b313e9a
verifier = da37
```

KDF-2. Password bytes `e5af86f09f949065cc81`.

```
root = 2fdffa57df7cf1a5bdd9ba676a750d794b7a263bcaf57d46b28fef0793868431
kenc = 871457715866e528052ff0f71f9cf8366e7039028c5676160b8a78d6de1a345a
kmac = 7df27060b0cc232197a20250367aeffe834f3374ee9fac30178d4e06d76e4b6e
verifier = e5c8
```

KS-1. AES counter blocks under KDF-1, starting at BE128(j).

```
j=1  83719ecf15b702d3d0a4c6930f6f8560
j=2  569b4efe1b8ba02faead75ad584ee6b7
j=255  e36d44df60e3f7a95b90f630bfb39747
j=256  869df8a7b212dab25a5cf0adc1276756
j=257  44e836f3f7ec67c3e605ee81e83e68ec
j=65535  fea481df456075fb3ef47fe0a452d00f
j=65536  076013d9e5f4b7384f35d17e1a06c6ed
j=4294967295  f53c3bf55f340897b997f9cbfbf192ee
j=4294967296  c2aee47dc41eea27cb47da8bf65c3ec6
j=4294967297  6a90fc983b4501ee7e2dd05da06e676f
```

The carry at 2^32 is checked by seeking directly to the counter; no 64 GB (2^36-byte) fixture is needed. A separate check compares blockwise AES and the CTR API and checks chunked encryption.

Empty and nonempty STORE entries. Use descriptor `019108005a45010000000000`. The empty file encrypts CRC32LE zero: S=0, C=4, P=54. The nonempty plaintext is ASCII `The quick brown fox jumps over the lazy dog` (43 bytes): S=43, C=47, P=97. Both modes encrypt the same stream and CRC; the mode byte changes the tag. Every outer CRC field is zero.

EMPTY-PUBLIC.

```
crc32 = 00000000
objectPlaintext = 00000000
ciphertext = 83719ecf
tag = 0f0d13b0747f5373a7693724a321f888
payload =
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1fda3783719ecf0f0d13b0747f5373a7693724a3
21f888
```

EMPTY-PROTECTED.

```
crc32 = 00000000
objectPlaintext = 00000000
ciphertext = 83719ecf
tag = bef6912702cd1742f92e4161136a908a
payload =
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1fda3783719ecfbef6912702cd1742f92e416113
6a908a
```

ENTRY-PUBLIC.

```
crc32 = 414fa339
objectPlaintext =
54686520717569636b2062726f776e20666f78206a756d7073206f7665722074686520c617a7920646f6739a34f41
ciphertext =
d719fbef64c26bb0bb84a4e16018eb4030f436de71fecd5fdd8d1adb3d3cc6c32afac737d1179e14b93da50471b7e5
tag = d2ab0a080a5e56bca80dd2e2ff4390f3
payload =
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1fda37d719fbef64c26bb0bb84a4e16018eb4030
f436de71fecd5fdd8d1adb3d3cc6c32afac737d1179e14b93da50471b7e5d2ab0a080a5e56bca80dd2e2ff4390f3
```

ENTRY-PROTECTED.

```
crc32 = 414fa339
objectPlaintext =
54686520717569636b2062726f776e20666f78206a756d7073206f7665722074686520c617a7920646f6739a34f41
ciphertext =
d719fbef64c26bb0bb84a4e16018eb4030f436de71fecd5fdd8d1adb3d3cc6c32afac737d1179e14b93da50471b7e5
tag = 9c9ee709b85e8e18be661e9b64489e93
payload =
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1fda37d719fbef64c26bb0bb84a4e16018eb4030
f436de71fecd5fdd8d1adb3d3cc6c32afac737d1179e14b93da50471b7e59c9ee709b85e8e18be661e9b64489e93
```

The corpus also contains CRC-CROSS-13, CRC-CROSS-14 and CRC-CROSS-15 in both modes. Their STORE input is the byte sequence 00 through S-1. Each four-byte CRC crosses an AES block boundary, exercising a continuing counter rather than a new encryption for the trailer.

Authenticated compression flags. These primitive MAC vectors use KDF-1, protected mode, and ciphertext `00 01 .. ff`; that byte string is not presented as a decodable compressed stream.

```
TAG-DEFLATE-0
descriptor = 019108005a45010008000000
tag = 218159ce8363fd95bfa1dc21a8d58d03
```

```
TAG-DEFLATE-2
descriptor = 019108005a45010008000200
tag = a2f51a55f89cadcb379a6efaf6cc0f13
```

Reference SHA-256 pins:

- `docs/ze1/zee1-kat-verify.py`: `19490efb2550aa3efc0412b89dbe8f4a022b680beeedab5d256a6a8344a55f89`.
- `docs/ze1/zee1-kat-verify-output.txt`: `ffc665df196c735b2c97c2097496bc182fbe81bedc30876c68365b5b126da02d`.
- `docs/ze1/zee1-v014-vectors.json`: `495e47b5d2f9d9e78668f59d78dcec287722aa62ea398e82ef48fb3d6818a80f`.

14.2 Required positive and negative cases

A conformance harness MUST cover password encoding, exact descriptors, empty files, both modes, same-root and imported-root contexts, different object nonces/roles, counter carry, and both signed descriptor widths. It MUST also cover protected empty archives, public ordinary empty ZIPs, ZIP64 sizes versus offset-only ZIP64, and ZIP64 tails both with ordinary sentinels and with consistent small actual values.

Required failures include:

- Missing, duplicate, truncated or wrong-signature descriptors; suite 1 Data Size 7 or 9: malformed. Wrong signature includes `45 5a`.
- SuiteID 0, 2 or `0x4445` (data begins `5a 45 45 44`): unsupported, not a malformed signature. Forbidden actual methods/flags are unsupported.
- P below 54, explicitly including lengths 50-53; range overflow; missing, wrong-width or bad-value data descriptors; nonzero local, central or validated descriptor CRC in either mode; and incomplete terminal framing.
- Mode stripping, changed authentic descriptor, changed ciphertext/tag, wrong credential with a verifier collision, and protected association swaps.
- Same-length plaintexts with equal CRC but different valid payload tags: substitution MUST fail ZEAC. A concrete pair is hexadecimal `474f4f4444415441` and `4556494c843e2ba0`, both length 8 and CRC `1d6696f6`.
- Ciphertext changed while retaining a valid source directory and tag suffix: keyless copy may retain the damage, but extraction and mode conversion MUST fail original entry authentication; conversion or re-encryption must not accept new output derived from that unauthenticated source.
- On extraction, content testing or a transformation that decodes, a valid tag over an incorrect encrypted CRC trailer: `crc-mismatch`. Non-decoding password removal preserves that authenticated expected CRC; it MUST NOT silently repair it. A failed tag with a matching plaintext CRC: authentication failure, never successful extraction, mode conversion, password removal or password change.
- Authenticated bad sizes, malformed compressed streams, trailing bytes, external-dictionary requirements and resource refusals. Counterpart positive cases include complete BZIP2 concatenations and Zstandard frame sequences.

- Changed source between verification and use, cancellation, and incomplete output publication. Nothing aborted is reported as complete.
- Forged public U on public-to-protected conversion: the ordinary size check fails; a new authenticated directory must not endorse that false value.
- Changed or removed public comment with valid terminal framing: comment remains unverified in both modes, and intact files remain accessible.

Positive cases MUST include CRC trailers crossing a 16-byte AES block boundary and an implementation chunk boundary. Check that only S bytes reach the decoder or an ordinary ZIP stream during password removal. After removing the directory, known-boundary local recovery must still authenticate the file and check its trailer CRC, without claiming recovered names or completeness.

Writer tests MUST check identical compression descriptors in both headers, fresh random inputs on new encryptions, nonce changes on directory rewrites, preserved contexts on unchanged copies and correct mode-conversion tags. Normal reading of a valid central descriptor does not depend on equality with its local copy; explicit local recovery must authenticate the local candidate instead. Both paths reject nonzero local CRCs. Test these paths separately.

14.3 Independent reproduction

An independent implementation made from this text without the reference-oracle source MUST reproduce the published known answers. Conformance harnesses MUST reproduce the entry answers in Section 14.1 and the archive outcomes in Section 14.9. A writer supplied the fixture's fixed inputs must reproduce the canonical bytes for the methods it implements. This does not require production APIs to export secret keys.

The independent WebCrypto checker implements this specification without consulting either Python oracle's source. It uses the JSON only for fixture inputs and expected values, native WebCrypto for cryptography, and its own CRC implementation. Results are reference-format evidence, not production-path validation. Its SHA-256 is `9990bee071130e4de2cf6acaa29a0f216eb3712529a4268906a0034dd45d4926`.

14.4 Round trips

Exercise new content, empty content, permitted compression methods, Unicode credentials and metadata, shared roots, imported roots and both archive forms. Validate plaintext, sizes, trailer CRCs, MACs and the consistently unverified archive-comment status. Compression decoders not implemented must report unsupported, never silently reinterpret bytes. Exercise the implementation's actual read and write paths, not only isolated cryptographic helpers.

14.5 Implementation evidence

Reference-vector success is not verification of production workers, streaming HMAC, parser limits, UI success boundaries or storage behavior. Those paths require implementation tests and review. A format writer need not implement data descriptors, but a conforming reader must consume the descriptor fixtures through its real parsing and extraction paths.

14.6 Regression scope

The method-91 dispatch must leave ordinary ZIP, WinZip AES and ZipCrypto contracts distinct. Test recognizable unsupported suites and failures that must not trigger another decoder or password transformation. Root caching must not cross credentials or sessions or bypass later entry checks.

14.7 Long-stream evidence

Before claiming large-file support, test ZIP64 using a logical file above `2^32` bytes and separately process more than `2^32` ciphertext bytes through the streaming MAC/CTR path with bounded memory. Compressible input alone is not evidence of long ciphertext processing. Test counter carry around block `2^32` by seeking to that counter, without allocating the preceding 64 GB (`2^36` bytes). One-shot and chunked encryption must agree, including partial final blocks. Check independent length arithmetic at the suite bounds.

14.8 Editing and conversion

Test rename/reorder/remove, preserved protected expectations, mixed-password same-mode copying, public comment edits, both mode conversions and full password removal/change. Every conversion requiring authentication must abort atomically on failure. Public-to-protected import must establish actual U and decoded correctness; reading an authenticated CRC trailer alone is not enough. Password removal must exclude the trailer from ordinary output and retain its authenticated expected CRC in the resulting ordinary ZIP records. Where implemented, test non-decoding removal separately from a contents test: a valid-MAC bad trailer is preserved as an expected checksum, not silently repaired; a failed source MAC or protected association aborts removal. Copying public U does not establish its correctness.

14.9 Complete archive known answers

The archive oracle reproduces the complete archives, intermediate values and required negative outcomes below. Its SHA-256 is `2dac7b2de89c4351f13b908a23dd19478507b4777275a82ba682bf8cc9b58ced` . The JSON corpus supplies complete bytes; this table identifies each fixture.

Fixture	Bytes	Archive SHA-256
`public-empty`	22	`8739c76e681f900923b900c9df0ef75cf421d39cabb54650c4b9ad19b6a76d85`
`public-empty-file`	178	`28b33cab05a4e1944a129460ea336c52ce3bf012eb5a4491486f95f237aee36e`
`public-store`	205	`9bda7561e8a5415e639af8d24c5e8bb09735930c50309cfcaf65ade4faa8d415`
`public-dd32`	221	`36a9fd8384a1515531ef7de4265e357231f57656c73b78133a64c7e4f30418d7`
`public-dd64`	345	`d68a57c33fdb695f1308b78f9b853fa30ac76a0f4ed8e9980a0dd2d6245b525a`
`public-trailer-chunk-boundary`	255	`7e93264e4a73520ad011d70422a7352b106c272a9c787c558e5f9c9460045a95`
`protected-empty`	88	`386d3412c01e05883c69ef275b6dbfc06c4ffc9662ae09ad9f7161724f067e9f`
`protected-empty-file`	268	`719dfe0e82c55e26986c92cd9531a10ad6ba341473274aaba40253c8058153b7`
`protected-store`	286	`1081b40f5052760e560af2fb95ccf0931114740cbd4e78349b6eaef71c076de6`
`protected-dd32`	302	`bd3390ae95c3174dec927032259d63917ab2268cf7c4a584632e2ee2dadcb86c`
`protected-dd64`	426	`63791fb7bbf96db81ab4d92e4a83b91795db62823a56b94a0cf55e9d2f7751a0`
`protected-trailer-chunk-boundary`	338	`17bdb018adad097e025d658035d2e4063b7c4cbdcfa197cf3f9eb0c5b20dae79`
`protected-comment`	99	`281a693e1cd65a45e3e1cdc6fcace02b5b2e6aed2ed0470b794bd78952d9ddef`
`public-offset64-dd32`	233	`664b0ccc9ef2952eb427044f8f07d249ea83e0a78bb0514716171e92390409ac`
`protected-dd64-classic-values`	426	`273b3abd8abd3208ac2b4965170a9533f555087b19838592b5279bbb111f90bf`

Fixture	Bytes	Archive SHA-256
`public-dd64-no-tail`	269	`1515e8c7ab37be4fa052934c182bdbb030094096895f23d16683df8c47881de0`
`protected-deflate-unicode`	312	`a3721fb4ce003a13d101e3dcea7c800a4dd13d18b76061b291ead3a0b2e30acb`
`protected-shared-root`	538	`4f980fc48429b27bd1d747ade428ff5d5130f858447be6edb3b82e01a7c68266`
`protected-mixed-contexts`	639	`433539733db60bb90c0c249e70f25470688799ac1f3e9503e0cb0b6a5b6d5524`
`protected-collision-original`	276	`ab6ecff89a4670e0f58d3d160697e19f9705819d9d76b033df8db6dc9958161d`
`protected-collision-donor`	276	`c3d6bbc808fa4c70511ad3fd36150f2a0ca06e794a6165c1ea6b966efe7e01ba`

Unless a fixture supplies another credential or salt, use ASCII `password`, rootSalt `00 01 .. 0f`, first entry nonce `10 11 .. 1f`, and directory nonce `30 31 .. 3f`. Every fixture must specify the exact credential, root salt and nonce for each object. Directory keys retain their separate HKDF labels.

The fixtures cover public and protected empty archives, empty regular files, STORE and DEFLATE, Unicode, signed classic and ZIP64 data descriptors, offset-only ZIP64, reserved sizes without ZIP64 tails, and ZIP64 end records with small actual values. The trailer-chunk-boundary fixtures have S=63, so the CRC spans an AES block and a 64-byte implementation chunk boundary.

The STORE fixture plaintext remains ASCII `123456789`, whose CRC-32 is `cbf43926`; its trailer plaintext is `26 39 f4 cb` in either mode. Every outer method-91 CRC field is zero. Empty regular files have a 54-byte payload. A public empty archive remains an ordinary 22-byte ZIP. A protected empty archive has a 66-byte ZECD record followed by EOCD, with no CRC trailer in that directory object.

The descriptor cases retain public values zero CRC, P and U, and protected values all zero. The shared-root case includes regular files and a directory under one derivation. The mixed-context case needs three derivations: two root salts under one credential and a different credential under the original salt. A cache indexed only by rootSalt must fail this distinction.

Protected empty archive.

```
5a4543440100010000000000000000000000010203040506070809 0a0b0c0d0e0f303132333435363738393a3b3c3d3e3faa9328
698c5b5118d0d5a5a471a12728b638504b0506000000000000000042000000000000000000
```

Directory root: `d9de601903a32772e93c2c6bdd99733fda92d1a7cdfdd85246500182c27bb641`.

Directory kenc: `c7fb797bd2ea86b1de404fc43840a6d363dee4bdab4177f2615b27a26bf90899`.

Directory kmac: `c111bbb02170d6c10da188e62c4b58e89fdd511cc01c6ab1a460f6bdafd49c74`.

Directory verifier: `aa93`.

Directory tag: `28698c5b5118d0d5a5a471a12728b638`.

The public-comment case keeps ASCII `Public note` in EOCD and contains no comment record inside ZECD. Replacing or removing those public bytes with valid end framing leaves file and directory authentication intact. The comment remains unverified after unlock.

The CRC-collision pair is `474f4f4444415441` and `4556494c843e2ba0`, each eight bytes with CRC-32 `1d6696f6`. Both payloads must authenticate separately. Substituting the donor beneath the unchanged original protected directory must fail ZEAC; restoring the original tag on the donor must fail its actual entry MAC. Matching CRCs do not excuse either failure.

Mode conversion must first verify the source MAC, preserve ciphertext and check the required size/decoding facts before accepting the target tag. Include the bad-source retagging and re-encryption cases, the forged public-U case and local recovery after directory loss. Both reference implementations exercise trailer, conversion, association, recovery and comment cases; these small fixtures do not establish production-path conformance.

14.10 Verification scope

The corpus checks the cryptographic and framing contract with small deterministic fixtures. Reference decoding covers STORE and raw DEFLATE; other codec implementations need their own conformance evidence. It is not an exhaustive ZIP parser, a claimed performance result, a substitute for production streaming tests or proof that unaware tools handle the format gracefully. Validate the same cases through the implementation's parser, workers and output-commit paths.

14.11 Interoperability and registration

Unaware readers may report unsupported methods, reject ZECD, expose masked ordinal names through recovery, emit placeholders or incorrectly report an empty archive. Measure exact fixture hashes, tool versions, invocation mode, exit status, listings and produced files. File-path and piped-input behavior can differ. Use the actual writer output for compatibility claims; a modified diagnostic fixture does not establish the same behavior.

The compatibility record must distinguish unsupported errors, silent omission, false success, zero-byte placeholders, nonempty zero-filled output and actual plaintext disclosure. These outcomes require product dispositions; they are not automatically waived because no plaintext was revealed. They also do not authorize changing the cryptographic contract for an unaware parser.

ZEE uses method 91, the `0x9101` descriptor, tagged `0x455A` extras and bit 13 for masked local headers. ZECD is recognized at the resolved directory start. It uses no PK-prefixed signature, version-62 encryption signal, `0x0017` field or Archive Decryption Header. Bit 13 marks the masked local headers; the directory format selects the decryption algorithm. Method 91 alone does not imply an encrypted directory.

15. Versioning and conformance artifacts

The document version identifies this specification; SuiteID 1 selects the ZEE-1 cryptographic contract, and ZECD record version 1 selects its framing. These identifiers have different roles. After publication, incompatible entry semantics require another SuiteID, and incompatible directory framing requires another record version. Unknown suites and versions remain unsupported, not permission to guess a compatible interpretation.

A conformance package identifies the specification, known-answer corpus and reference programs by version and hash. Reference results establish reproducibility of those artifacts; implementation correctness, supported capacity and interoperability need their own evidence under Section 14.

16. Implementation guide

This guide summarizes existing rules; it adds no requirements. The detailed rules, including their failure classes, remain authoritative.

1. **Resolve the container.** Establish the terminal chain, archive origin and bounded directory range (8.8.1). Recognize ZECD at that one position before generic directory or version dispatch (10.6.1).
2. **Establish metadata authority.** Public central records are unverified input. For ZECD, verify its MAC before decryption, frame the entire plaintext, then interpret individual records (8.10.3, 10.6.2).
3. **Prepare one file.** Use its governing central record, implicit directory mode and descriptor. Locate the payload from the local header lengths; check the local CRC but do not substitute other local metadata. In protected mode, compare ZEAC before file-key work (10.1-10.2).
4. **Perform the selected operation.** Extraction and content testing need the file MAC, complete decoding, size and trailer CRC checks. Copying, conversion and password changes or removal have the specific scopes in 11.5-11.6.3. Do not borrow a weaker operation's success rule.
5. **Commit or discard.** Use the same source state that was checked, apply resource limits throughout, and publish output only after the operation's required checks pass. A failed recognized ZEE object is never another encryption format (10.4-10.5, 11.6.4).

For writing, start with 11.1-11.3 and the matching header tables in Section 8. Compression precedes the CRC trailer; both share one continuous cipher stream. Known sizes, staging, seekable output and signed data descriptors are output strategies, not different cryptographic profiles. Section 7.3 explains native AES-CTR chunking; Section 7.4 explains the incremental HMAC needed for bounded- memory browser processing.

Run the known answers and failure cases in Section 14 through the actual implementation paths. They exercise details that are easy to miss: exact UTF-8 credentials, credential-aware root caching, role labels, implicit MAC mode, CTR position across partial blocks, S-byte decoder input, tagged extras, ZIP64 size forms and the distinction between copying and verifying contents.

16.1 Worked protected archive

This explanatory example accompanies the Implementer's Guide. It does not replace the conformance cases in Section 14 or change their requirements.

This complete 353-byte archive contains one raw-DEFLATE file. The password is `test`, encoded as `74 65 73 74`. The filename is `test for grok to decode.txt`; the 36 content bytes are `Hello Grok! Nice job decoding this!\n`, with one final LF byte. All offsets below are decimal and zero-based; ranges are half-open. The public password and keys are test data.

The EOCD at offset 331 declares directory offset 135 and directory size 196. At 135, `ZECD` selects protected mode. Its header declares C=130, so the full record is 66+130=196 bytes and ends immediately before the EOCD. The directory has no CRC trailer and decrypts to one ordinary central header.

Range	Meaning
[0,43)	Masked local header, ordinal name `1`, compressionDescriptor12
[43,59), [59,75), [75,77)	File root salt, nonce, verifier
[77,119), [119,135)	File ciphertext42, tag16
[135,151)	ZECD header16
[151,167), [167,183), [183,185)	Directory root salt, nonce, verifier
[185,315), [315,331)	Directory ciphertext130, tag16
[331,353)	Ordinary EOCD; one entry, no comment

Both objects use this root salt and PBKDF2 result:

```
rootSalt = b1bc492db9d101f01b3137b51c5d8b40
root     = eb3333deb33556417e06da8219090403547451d2e1cdc27b90fb9a74a32ea506
```

Independent HKDF-Expand operations produce these values:

```
file nonce    = 0e3fd7fa9a77957b88d84698e268ccbd
file Kenc     = 6501b5f2d488c13312f73288761fe66aead861e2db2f68a134e6176cc15827db
file Kmac     = b14cad14ec61f951f7e5d339f3b47cabad9d0cb500908e68aa86aff0685568d4
file verifier = abd7
file tag      = 2eea46231902d4900c0ca4517b9acaeb

dir nonce     = 0150a561821e5eafc537156fdb2e2332
dir Kenc      = 8b6060f31420112ee908a6c499052a4949eec8bf9217406659da190cf7ec055c
dir Kmac      = cfb2b97e6c8a041a6d583c4358cd4654a78b540ced57f0eb3f61a6c069d5b994
dir verifier  = ece6
dir tag       = 6bd7238a62dc1a3882b09c0e811a2f33
```

Directory authentication uses `ASCII("ZEE-1/dir/tag") || header16 || ciphertext130`. After it succeeds, decrypt with counter BE128(1). The 130-byte plaintext is:

```
504b01023f00330001085b002d654a5d000000005c000000240000001b003900
0000000000000000000000000000007465737420666f722067726f6b20746f2064
65636f64652e747874019108005a450100080000005a4514005a4541432eea46
231902d4900c0ca4517b9acaeb5554050001e794ca6a5a4508005a4558540105
1400
```

The central record has flags 0x0801, method 91, P=92, U=36, a 27-byte name, and 57 extra bytes. In order, the extras are compressionDescriptor12, ZEAC24, extended timestamp9, and ZEXT12. The two separate `0x455A` occurrences are valid: ZEAC contains the expected file tag, while `ZEXT 01 05 14 00` identifies marker version 1 and producer version 5.20.0. Scan every extra by tag; do not overwrite one with the other. The masked local flags are 0x2000, its U=0, and its name is `1`; these deliberately differ from the central metadata.

The descriptor is `019108005a45010008000000`: actual method 8, actual flags 0. Compare ZEAC's expected tag with the file suffix, then verify `ASCII("ZEE-1/tag") || 01 || compressionDescriptor12 || ciphertext42` under the file MAC key. After authentication, file decryption yields:

```
compressed stream (38 bytes):
f348cdc9c957702fcacf5654f0cb4c4e55c8ca4f5248494dce4fc9cc4b5728c9
c82c56e40200
CRC32LE (4 bytes): a50283d0
```

The CRC begins at object byte 38, using counter 3 starting at keystream byte 6. There is no cipher reset at that boundary. Counter blocks 1, 2 and 3 give these file keystream blocks:

```
d966d1a7ac31839df33204bbcf1fdd84
696014119a134e49d76ea2f8c22b06f9
11e486e9797813d35016cd3369e24653
```

Raw DEFLATE consumes exactly the first 38 bytes, terminates, and produces 36 bytes. Their CRC-32 is `D08302A5`, matching trailer bytes `A5 02 83 D0`. The entire encrypted payload is 54+38=92 bytes, although the compression stream alone has only 38 bytes.

Complete archive, hex (32 archive bytes per line):

```
504b0304330000205b0000000000000000005c0000000000000001000c003101
9108005a45010008000000b1bc492db9d101f01b3137b51c5d8b400e3fd7fa9a
77957b88d84698e268ccbdabd72a2e1c6e6566f3b239fd52ef3fd491ca3ca8de
5ec85b070419216b34897c2e30d9c8d00d7b78b6d1d3c62eea46231902d4900c
0ca4517b9acaeb5a45434401000100820000000000000b1bc492db9d101f01b
3137b51c5d8b400150a561821e5eafc537156fdb2e2332ece6ce32293ebc3f24
92dea4f7c9d8544cdee86fe8ed714b077159127662e3d80579cf4e64b8cd0e3f
eb2166231847e5e99ce030b4f628f709c81afa8e2b90d0b740310ba02804c288
c16b62a7a2657ad3f3f534bc956154d81351d938dcd0d47d96980d414b97cb76
19df8ea462d7ef7be16181b761828466750f88d9f4db4c8732efb46bd7238a62
dc1a3882b09c0e811a2f33504b0506000000000100010c40000008700000000
00
```

Archive SHA-256:

```
dafe57b20e929516d7f08a3b71bd2988cc57eafb41e7360cf19aa77aa7795bcd
```

The repository includes the example archive. Its machine-readable companion includes the complete MAC inputs, ciphertexts, plaintexts, keys, offsets and decoded field values. The standalone verifier checks this example only; the full conformance corpus covers other ZIP forms.

References

- [A1] PKWARE, APPNOTE.TXT .ZIP File Format Specification, version 6.3.10: sections 4.3.7, 4.3.9-12, 4.3.16, 4.4.3.2, 4.4.4, 4.4.5, 4.5.1-3, 4.6.1, 4.6.3, 7.2.2, 7.3 and 7.5. https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.10.TXT
- [W1] WinZip, AES Encryption Information: Encryption Specification AE-1 and AE-2, version 1.04 (2009-01-30), as mirrored verbatim in the minizip 2.7.1 source tree. https://chromium.googlesource.com/external/github.com/nmoinvaz/minizip/+/2.7.1/doc/winzip_aes.md

- [R7] RARLAB, RAR 5.0 archive format (technote); UnRAR 7.3.1, `crypt5.cpp`, `CryptData::SetKey50` (derivation cache). https://www.rarlab.com/technote.htm , https://www.rarlab.com/rar/unrarsrc-7.3.1.tar.gz
- [R10] NIST SP 800-132, Recommendation for Password-Based Key Derivation, sections 5.1, 5.2 and 5.4. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
- [R11] RFC 5869, HMAC-based Extract-and-Expand Key Derivation Function, sections 2.3, 3.3 and appendix A. https://www.rfc-editor.org/rfc/rfc5869
- [R15] NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation, section 6.5 and appendix B. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
- [R16] W3C, Web Cryptography API (AES-CTR: the rightmost `length` bits of the counter block increment as a big-endian integer). https://w3c.github.io/webcrypto/
- [R19] Bellare and Namprempre, Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm (2000), ePrint 2000/025. https://eprint.iacr.org/2000/025
- [R20] RFC 5116, An Interface and Algorithms for Authenticated Encryption, section 2.1. https://www.rfc-editor.org/rfc/rfc5116
- [R23] RFC 4868, section 2.3 (HMAC-SHA-256-128); RFC 2104, sections 2 and 5. https://www.rfc-editor.org/rfc/rfc4868 , https://www.rfc-editor.org/rfc/rfc2104
- [R25] Andreeva, Bogdanov, Luykx, Mennink, Mouha, Yasuda, How to Securely Release Unverified Plaintext in Authenticated Encryption (2014), ePrint 2014/144. https://eprint.iacr.org/2014/144
- [R26] RFC 1951 (DEFLATE), section 3.2.3; RFC 8878 (Zstandard; section 5 defines dictionaries supplied outside the frame); the bzip2 stream format (end-of-stream magic and stream CRC); Igor Pavlov, LZMA Specification (Draft), 2015-06-14 (raw LZMA end-of-stream marker; APPNOTE [A1] defines ZIP method-14 framing and flags). https://www.rfc-editor.org/rfc/rfc1951 , https://www.rfc-editor.org/rfc/rfc8878 , https://sourceware.org/pub/bzip2/docs/manual_3.html , https://www.7-zip.org/a/lzma-specification.7z
- [R29] FIPS PUB 180-4, Secure Hash Standard (SHA-256 message length below 2^64 bits). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
- [R30] Len, Grubbs, Ristenpart, Partitioning Oracle Attacks, USENIX Security 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/len
- [R32] RFC 8018 (PKCS #5 v2.1), section 5.2 (PBKDF2). https://www.rfc-editor.org/rfc/rfc8018
- [R33] RFC 3629 (UTF-8). https://www.rfc-editor.org/rfc/rfc3629
- [R34] RFC 2119 and RFC 8174 (requirement keywords). https://www.rfc-editor.org/rfc/rfc2119 , https://www.rfc-editor.org/rfc/rfc8174
- [R35] NIST FIPS PUB 197 (2001), Appendix C.3, AES-256 example. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
- [R36] RFC 7914, section 11 (PBKDF2-HMAC-SHA-256 test vectors). https://www.rfc-editor.org/rfc/rfc7914
- [R37] RFC 4231 (HMAC-SHA-2 test vectors), test case 5. https://www.rfc-editor.org/rfc/rfc4231
- [R39] 7-Zip, 7z format description, SignatureHeader and kEncodedHeader. https://github.com/ip7z/7zip/blob/main/DOC/7zFormat.txt
- [R41] 7-Zip, 7z AES derivation and cache implementation. https://github.com/ip7z/7zip/blob/main/CPP/7zip/Crypto/7zAes.cpp

Copyright and trademarks

Copyright (c) 2026 Visware LLC.

ZIP Extractor is developed and published by Visware LLC. This specification is authored by Visware LLC Technical Staff.

ZIP Extractor and Visware are registered trademarks of Visware LLC. Other product names and trademarks belong to their respective owners.

Describing an implementation as compatible with this format does not imply endorsement or certification by Visware LLC.