# ZIP Extractor Encryption (ZEE) Implementer's Guide

**ZIP Extractor by Visware LLC**

**Authors:** Visware LLC Technical Staff

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

This guide gives ZIP implementers the byte layouts, cryptographic inputs and reading procedure needed to decode ZEE-1 files, including archives with an encrypted central directory. It accompanies the **ZIP Extractor Encryption (ZEE) Specification, Version 1.0.1 FINAL**, which remains the normative reference: full specification. Section references labelled "Spec" refer to that document. This document revision does not change SuiteID 1 or any archive bytes.

Start here when adding ZEE to an existing ZIP reader. The worked archive in Section 9 supplies intermediate values so that a wrong key schedule can be distinguished from a wrong password. The full specification supplies the conformance cases and detailed rules for editing, recovery and less common ZIP forms. Documentation: https://zipextractor.app/docs/zee.html.

## 1. Format at a glance

ZEE retains ZIP's independent entries. Compression method **91** identifies an encrypted file, and extra field **0x9101** carries its actual compression method. Each payload has its own encrypted CRC-32 and authentication tag. There are two directory forms:

| Form | Directory at the resolved ZIP directory offset | File MAC mode byte |
|---|---|---|
| Public | Ordinary central-directory records; names and metadata are unverified | `00` |
| Protected | One `ZECD` record encrypting and authenticating those records | `01` |

Method 91 alone does not select a mode. Retain the resolved directory form while reading its files. Public mode can contain unrelated ordinary ZIP entries; protected writers emit only ZEE-1 regular files and directory entries. The archive comment is public and unauthenticated in both forms, including after unlock; identify it as unverified if displayed.

SuiteID 1 selects the complete key schedule and layout here. SuiteID 0 is invalid; 2-65535 are unassigned and unsupported. ZECD record version 1 is the only defined directory framing. Recognize the identifiers before parsing a known version's body; an unknown value is not permission to guess.

All multi-byte ZIP integers below are unsigned little-endian. The AES counter is big-endian. `||` concatenates raw bytes; `[a:b]` is a half-open byte range. Hexadecimal dumps represent bytes, not ASCII input to a cipher.

| Symbol | Meaning |
|---|---|
| U | Fully decoded file length |
| S | Compressed-stream length, excluding its CRC trailer |
| C | Object ciphertext length; for a file `C = S + 4`; ZECD has no CRC trailer |
| P | Complete file payload length: `P = C + 50 = S + 54` |
| compressionDescriptor12 | All 12 serialized bytes of extra field 0x9101, including Header ID and Data Size |
| header16 | All 16 serialized bytes of the ZECD record header |

A "compression descriptor" is not ZIP's bit-3 data descriptor. A filesystem directory entry is not the archive's central directory.

## 2. Exact cryptographic construction

### Password and root

Encode the exact, nonempty Unicode scalar sequence as strict UTF-8. Do not normalize, trim, replace invalid scalars, or prepend a byte-order mark. Supplied characters, including a literal U+FEFF, are not removed. Reject malformed UTF-8 and lone surrogates. New passwords additionally exclude Unicode General_Category Cc; reading preserves existing control characters. Legacy ZIP password encodings do not apply to ZEE. (Spec 5.)

```
root = PBKDF2-HMAC-SHA-256(passwordBytes, rootSalt16, 1000000, 32)
```

The output is 32 bytes. Cache it by **credential and root salt**, never salt alone. A new single-password archive shares one root context across its new files and directory; copied files may have other contexts or passwords. Derive those on demand. Discard session references on credential discard, lock or close. (Spec 6.1-6.3.)

### Object keys and verifier

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

| Object 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` |

Use the object's own nonce. Labels are exact, case-sensitive ASCII with no quotes, NUL, separator or length prefix. Append the 16 nonce bytes, not their hex representation. Each row output is a separate expansion. The PBKDF2 output is used directly as PRK: **there is no HKDF-Extract and no HKDF salt**. Each output fits in one block, so RFC 5869, Section 2.3 reduces to:

```
Expand(root, info, L) = HMAC-SHA-256(root, info || 01)[0:L]
```

Here `01` is one raw byte, the HKDF block counter. It is not the file's mode. The stored verifier is compared only after this derivation. A mismatch rejects this attempt; a match authorizes no content. About 1 in 65,536 wrong passwords passes it and must still face the complete MAC.

## AES-256-CTR and MAC

```
initialCounter = 00000000000000000000000000000001
counter[j]     = unsigned 128-bit big-endian integer j, for j = 1, 2, ...

fileTag = HMAC-SHA-256(Kmac,
    ASCII("ZEE-1/tag") || modeByte || compressionDescriptor12 || ciphertext)[0:16]

directoryTag = HMAC-SHA-256(Kmac,
    ASCII("ZEE-1/dir/tag") || header16 || ciphertext)[0:16]
```

The nonce selects keys; it is not the counter or an IV. CTR has no padding. Encrypt a file's compressed stream and four-byte CRC trailer in **one continuous keystream**, including a trailer beginning midway through a block. At byte offset o, use counter `1 + floor(o/16)` and skip `o mod 16` keystream bytes. Never restart at counter 1 for each chunk. (Spec 7.3.)

The MAC inputs above are exhaustive. Do not append salt, nonce, verifier, ZIP data descriptor or archive comment. The file mode is one raw byte, not stored in its payload. The directory MAC has no mode byte. Salt and nonce already select the keys. Compare all 16 tag bytes without early mismatch termination. The two-byte verifier is not an authenticator. (Spec 7.4, 8.11.)

Writers draw a fresh random 16-byte root salt for each new root context and a fresh random 16-byte nonce for every new encryption, including directory rewrites. Use a cryptographic random generator and abort on its failure. Unchanged payload copying is not new encryption. Do not use the example's fixed values for ordinary writing. (Spec 7.2.)

# 3. Byte layouts

## Compression descriptor: extra field 0x9101

| Offset | Bytes | Value |
|---|---|---|
| 0 | 2 | Header ID `0x9101`: `01 91` |
| 2 | 2 | Data Size 8: `08 00` |
| 4 | 2 | ASCII `ZE`: `5a 45` |
| 6 | 2 | SuiteID 1: `01 00` |
| 8 | 2 | ActualMethod |
| 10 | 2 | ActualFlags |

For DEFLATE with zero flags: `019108005a45010008000000`. For STORE: `019108005a45010000000000`. Bounds, signature and suite identify the layout; suite 1 then requires exactly eight data bytes. Missing, duplicate, truncated or wrong-signature descriptors are malformed. Unknown suites or method/flag combinations are unsupported. Extra field 0x9101 on another outer method does not select ZEE.

Writers emit exactly one byte-identical copy in the local and central headers and authenticate those bytes. **Normal readers use the central copy**, not a reconstructed structure or the local copy. The local copy is available for explicit recovery. (Spec 8.1.)

## File payload

| Offset from payload start | Bytes | Field |
|---|---|---|
| 0 | 16 | rootSalt |
| 16 | 16 | nonce |
| 32 | 2 | verifier |
| 34 | S | Encrypted compressed stream |
| 34 + S | 4 | Encrypted CRC32LE of the complete uncompressed file |
| 38 + S | 16 | File tag |

```
payloadStart = physicalLocalHeaderOffset + 30 + nameLength + extraLength
payloadEnd   = payloadStart + P
ciphertext   = payload[34 : P − 16]
C = P − 50
S = P − 54
```

Check lengths, overflow and archive bounds before slicing. P is the resolved ZIP compressed size and includes all encryption overhead. P must be at least 54. A bit-3 data descriptor starts at payloadEnd, after the tag, outside P. Decrypt C bytes, separate the final four plaintext bytes as little-endian CRC, and give **only S bytes** to the compression decoder. Use ordinary ZIP CRC-32: reflected polynomial 0xEDB88320, initial value 0xFFFFFFFF and final XOR 0xFFFFFFFF, over the uncompressed bytes. (Spec 7.1.)

## ZECD archive-level record

| Offset from ZECD start | Bytes | Field |
|---|---|---|
| 0 | 4 | ASCII `ZECD`: `5a 45 43 44` |
| 4 | 2 | Record version 1 |
| 6 | 2 | SuiteID 1 |
| 8 | 8 | Ciphertext length C |
| 16 | 16 | rootSalt |
| 32 | 16 | nonce |
| 48 | 2 | verifier |
| 50 | C | Encrypted central-directory records |
| 50 + C | 16 | Directory tag |

The complete record is **66 + C bytes**; header16 is bytes `[0:16]`. The resolved ZIP directory size must equal 66+C exactly. The plaintext is the ordinary central-record sequence, uncompressed, with no inner end record, CRC trailer or comment-authentication record. Authenticate before decrypting or parsing it. (Spec 8.10-8.11.)

## Vendor extra field 0x455A

Walk extras as a sequence of `HeaderID:u16LE, DataSize:u16LE, data[DataSize]`. Each 0x455A occurrence holds **one** four-byte ASCII tag followed by its body. Several occurrences with different tags can coexist. A map keyed only by Header ID can overwrite a required association.

| Tag | Complete field prefix | Body and rule |
|---|---|---|
| ZEAC | `5a4514005a454143` | 16-byte expected file tag; exactly one per protected ZEE regular-file central record |
| ZEXT | `5a4508005a455854` | Version byte 1, then major, minor, patch as one byte each; optional producer marker |

ZEAC is 24 bytes including its ZIP wrapper, with Data Size 20. Missing, duplicate (even identical), wrong-size or directory-entry ZEAC is malformed. Compare its authenticated expected tag to the file's stored trailing tag **before file-key work**. This binds the named central record to that tag; the file MAC still has to validate the ciphertext. Do not substitute a reread payload suffix for an authenticated expected ZEAC during copying.

ZEXT version 1 requires Data Size 8. Writers emit at most one per header; readers may choose among duplicate markers, preferring the central copy. Unknown marker versions remain opaque. Unknown vendor tags use ordinary unknown-extra handling, confer no authentication, and must still fit their declared bounds. Public writers omit ZEAC; public readers ignore copies without attributing authentication to them. Masked local headers admit neither ZEXT nor other metadata extras. (Spec 8.6, 8.9, 8.10.2.)

# 4. Integrating with ZIP records

## Locate the directory before choosing its parser

Read the ordinary ZIP closing structures, with ZIP64 expansion where sentinels require it. The selected end record and its declared public comment must end at EOF. Immediately before it is either the directory or the adjacent ZIP64 end record and locator. The directory ends where that terminal chain begins. Use checked, exact integer arithmetic:

```
directoryStart = startOfTerminalChain − resolvedDirectorySize
archiveOrigin  = directoryStart − resolvedDirectoryRelativeOffset
physicalLocalHeaderOffset = archiveOrigin + centralLocalHeaderOffset
```

The origin must be nonnegative and apply consistently, including to the ZIP64 locator offset. Earlier prefixes or gaps are not authenticated. Writers emit compact archives from origin zero. Entries must fit before the directory; the directory and closing structures are contiguous. This profile is single-file: disk fields are zero and the locator's disk count is one.

At **that one resolved directory start**, inspect for `ZECD` before ordinary central-header parsing or version-62 dispatch. A recognized ZECD failure must never cause a different interpretation or a signature scan. Otherwise use ordinary public ZIP central-directory parsing. Method 91 is likewise recognized before generic encryption/version dispatch. (Spec 8.8.1, 10.6.1.)

Writers store true central-record counts. Protected readers parse the entire authenticated sequence and take its count from that parse; public counts are not a stopping rule or proof of membership. Within each closing structure per-disk and total counts agree. Repeated actual values across ordinary and ZIP64 end structures agree. Reserved values `0xffff` / `0xffffffff` require their ZIP64 values; a valid adjacent ZIP64 pair is also accepted when ordinary fields contain small actual values. Entry-size reservation alone does not require ZIP64 closing records. Spec 8.8.1 defines the complete resolution rules.

## Regular-file header forms

These are the writer forms; normal reader checks are described below.

| Field | Public local header | Central record, either mode | Protected local header |
|---|---|---|---|
| Method | 91 | 91 | 91 |
| Flags | Logical flags | Logical flags | Exactly `0x2000`, or `0x2008` with bit 3 |
| Version needed | max(51, required ZIP method/features) | Same formula | 51 |
| Name | Real UTF-8 | Real UTF-8 | 1-based physical ordinal, lowercase hex, no leading zeros |
| Time/date | Ordinary metadata | Ordinary metadata | Zero |
| ZIP CRC | Zero | Zero | Zero |
| Known sizes | P, U | P, U | P, 0 |
| Extras | Compression descriptor; ordinary permitted extras | Compression descriptor; ZEAC in protected mode; permitted extras | Optional ZIP64 size extra first, then compression descriptor only |

Logical flags set bits 0 and 11, clear bits 6 and 13, and set bit 3 exactly when a data descriptor follows. Writers clear bits 1-2; readers ignore them and use ActualFlags instead. Other bits retain ordinary ZIP meaning. The common writer values are 0x0801 and 0x0809. Protected locals have **bit 0 clear**, despite holding encrypted data. Do not use that bit to bypass method-91 decoding. LZMA raises logical version needed to 63; version 62 is not a ZECD identifier. Names and per-file comments use strict UTF-8. Newly supplied archive-comment text also uses UTF-8; unchanged comment bytes may be preserved. (Spec 8.2-8.4, 8.9.)

In normal reading the governing central record supplies the method, compression descriptor, mode context, P and U. Use the local signature and its name/extra lengths to locate the payload, and require the local CRC to be zero. **Do not compare or repair other local metadata against the central record.** This includes the local descriptor. Byte-identical copies are a writer requirement, not an additional normal-reader check. Masked local names, U=0 and cleared bit 0 are expected. (Spec 8.9, 10.1.)

## Streamed sizes and data descriptors

With bit 3, classic local sizes are both zero. ZIP64 reservation instead puts `0xffffffff` in both local size fields and supplies extra field 0x0001 with Data Size 16 and two zero uint64 placeholders. Retain that size form in the final central record even when the sizes fit in 32 bits; its ZIP64 extra supplies actual U then P. Without bit 3, masked ZIP64 locals store U=0 then P. Ordinary ZIP64 offset expansion retains its standard position.

| Descriptor form | Bytes after the complete payload |
|---|---|
| Classic, 16 bytes | `504b0708 \|\| CRC:u32LE \|\| P:u32LE \|\| U:u32LE` |
| ZIP64, 24 bytes | `504b0708 \|\| CRC:u32LE \|\| P:u64LE \|\| U:u64LE` |

Public values are **0, P, U**; protected values are **0, 0, 0**. Require the signature. Width follows the central record's ZIP64 **size** form, not an offset-only expansion. Compute the endpoint from P; never search ciphertext for the signature. Validate width, values and bounds. These bytes are neither part of P nor MAC input. All outer method-91 CRC fields must be zero. Writers may omit data descriptors; full readers support both widths. (Spec 8.2.)

## Directories and empty files

Directory entries are identified by names ending `/` or directory attributes. Protected central directories use STORE, zero resolved sizes/CRC, bit 11 set, bits 0/3/6/13 clear, and no payload, compression descriptor or ZEAC. A ZIP64 extra holding zero sizes is allowed. Their masked local headers use method 0, version 51, bit 13 only, ordinal names, zero metadata/sizes, and no extras or payload. Public unrelated directory forms retain ordinary ZIP handling. A directory-shaped record with nonzero U is malformed. (Spec 8.5.)

Empty regular files remain encrypted files. Canonical STORE has U=S=0, C=4, P=54 and trailer CRC zero; other valid empty streams are accepted. Never bypass authentication because U is zero. An empty protected archive still has a 66-byte ZECD object; an empty public ZIP has ordinary framing and makes no password-protection claim.

## 5. Select and finish the actual codec

| ActualMethod | Stream and ActualFlags |
|---|---|
| 0 | STORE: output exactly S bytes; flags 0 |
| 8 | Raw DEFLATE (RFC 1951), not zlib or gzip wrapping; writers use flags 0; readers accept bits 1-2 as no-op level hints |
| 9 | ZIP Deflate64; same flag rule as DEFLATE |
| 12 | Complete concatenated BZIP2 streams; flags 0 |
| 14 | ZIP LZMA representation, including its properties header and end marker; bit 1 set ( `0x0002` ), bit 2 clear |
| 93 | Complete Zstandard frames, including skippable frames; flags 0 |
| 20 | Deprecated Zstandard number, accepted on read and unchanged pass-through; new content uses 93 |

Other ActualFlags bits are zero. Unsupported combinations or unavailable decoders are reported as unsupported, never guessed. Every compressed stream is self-contained: no external dictionary or prior-entry history. Follow Spec 7.5 and APPNOTE's corresponding method representation, not a generic wrapper chosen by a compression library.

Consume exactly the S-byte region. Decode all BZIP2 streams or Zstandard frames in sequence. Reject truncation, missing required end markers and trailing whole bytes. DEFLATE's ordinary unused bits in its final byte are allowed. Do not stop merely because U output bytes have appeared. Complete decoding must produce exactly U bytes and the encrypted trailer's expected ZIP CRC-32. Apply independent storage/output limits even to authenticated content; U alone is not a decompression-bomb defense.

## 6. Ordered reader procedure

For an initial implementation, use a seekable, stable source and authenticate before decrypting. A source revision must not change between checking and use.

1. Resolve the bounded ZIP directory range and archive origin as in Section 4. Recognize ZECD at that position, or retain public mode.
2. For ZECD, check identifiers, lengths and limits. Derive its directory keys, compare its verifier, verify its MAC, then decrypt. Parse the **entire** plaintext as central headers with bounded fields and extras. A framing overrun or leftover byte makes the whole listing malformed; record-specific failures within sound framing affect the relevant entry. Do not use a public count to truncate this parse. Public mode instead parses ordinary central records without assigning them metadata authentication.
3. Select a supported regular file. Validate its governing central descriptor, logical flags, strict UTF-8, zero outer CRC and resolved sizes. A different regular-file kind inside ZECD is unsupported, not a generic-decoder escape. Apply the protected common flag, UTF-8 and single-disk checks before classifying that other kind as unsupported. (Spec 8.10.1.) Find its payload using the local header, check local CRC zero, and require the complete header/payload/data-descriptor range before the directory.
4. In protected mode, compare ZEAC's expected tag with the stored file tag before deriving or using file keys. A mismatch is association failure.
5. Obtain the root for this file's credential and salt, expand its file keys, and compare the verifier. If it matches, verify the full file MAC with the directory's mode and central compression descriptor. Do not accept content on the verifier or ZEAC comparison alone.

6. Decrypt C bytes continuously, take the final four as CRC32LE, and decode the preceding S bytes using ActualMethod and ActualFlags. Require complete decoding, output length U and matching CRC. Validate any data descriptor at the computed endpoint if it was not checked earlier.
7. Only now expose the contents as successfully extracted or cache them as trusted. On failure discard provisional output. Preserve internal failure categories; a common password-retry UI may handle verifier/MAC rejection.

One-pass file processing may decrypt/decode into a provisional sink while computing the MAC; no successful or trusted output is committed before all checks pass. A remote upload is suitable only if its partial result remains provisional and can be discarded. Directory processing always verifies its MAC before decryption. If provisional decoding fails, finish the bounded MAC pass to distinguish a bad tag from an authenticated content failure; resource, cancellation or I/O failure may stop without that verdict. (Spec 7.4, 10.2.)

A file may need a different password from its directory. A verifier miss and a MAC failure can also result from corruption; neither proves the password is wrong. Retrying another user-supplied credential is allowed. Never change the suite, iteration count, encoding or algorithm to make a recognized object pass. (Spec 10.3-10.4.)

## 7. Writing, copying and password removal

For a new file: compute the original plaintext CRC, compress, append CRC32LE, encrypt continuously, and append the tag for the selected mode. Emit zero outer CRCs and the matching compression descriptors. New protected output adds ZEAC and encrypts the central records under a fresh directory nonce. Finish ordinary ZIP closing records; the directory size includes all ZECD overhead. (Spec 11.1-11.4.)

The checks depend on what an operation claims. This table concerns existing ZEE source files; it does not authorize bypassing format or bounds checks.

| Operation | Required checks and meaning |
|---|---|
| Extract or test contents | Directory MAC and ZEAC if protected; file MAC; complete decode, U and CRC |
| Same-mode payload copy/edit | If protected, authenticate directory, compare ZEAC, and preserve that authenticated expectation; file-key work is not required. This does not test interior ciphertext or decoded content |
| Change protected to public | Directory MAC, ZEAC, original file MAC; retag with mode 00; authenticated source sizes may be reused without decoding |
| Change public to protected | Original file MAC, complete decode, U and CRC unless a trustworthy source already establishes the same unchanged contents and size; retag with mode 01 and create ZEAC |
| Remove password | Original file MAC and protected directory/ZEAC when applicable; may preserve exactly S decrypted compressed bytes and transfer the authenticated trailer CRC into ordinary ZIP records without decoding |
| Change password/re-encrypt | Verify original file MAC and any protected directory/ZEAC; use fresh nonces. A whole-archive password change also creates a new root salt. Unchanged compressed bytes and CRC may be reused without decoding |
| Explicit local recovery | Authenticate candidate mode/descriptor and complete payload, decode and check CRC; use generated names and make no original metadata/completeness claim |

Password removal does not silently repair an authenticated but incorrect CRC, and it does not establish a public U claim. A public-to-protected conversion must not promote a forged size to authenticated metadata. Protected files cannot be stripped into public mode without verifying and replacing their mode-bound tags. Same-mode copying preserves existing roots, nonces and payload bytes. Failed transformations leave the source unchanged. See Spec 8.7, 11.5-11.6 for the complete contracts.

## 8. Implementation checks and practical limits

WebCrypto supports PBKDF2, the short HMAC expansions and AES-CTR with the counter above ( `length: 128` ). Process useful chunks, not one API call per AES block. WebCrypto HMAC is one-shot: bounded-memory large-file processing needs an incremental HMAC implementation. Separate chunk MACs cannot be combined to obtain ZEE's whole-object tag.

The file ciphertext bound is `C <= 2^61 − 87`; the directory bound is `C <= 2^61 − 94`. These come from SHA-256's maximum whole-byte message length minus HMAC's 64-byte inner pad and the fixed 22-byte/file or 29-byte/directory MAC prefixes. They are cryptographic construction limits, not browser or product capacity promises. Use exact checked lengths and enforce practical limits before allocations and as data flows. (Spec 9.)

Protected mode still exposes entry count and boundaries, physical order and offsets, compression methods, payload lengths where framed, salt-sharing relationships and the public comment. STORE reveals original size; no individual-size privacy is promised. Salt sharing alone does not prove password equality or which application imported a file. (Spec 13.)

For implementations starting from WinZip AES:

| Detail | WinZip AE-1/AE-2 | ZEE-1 |
|---|---|---|
| Outer method / extra | 99 / 0x9901, Data Size 7 | 91 / 0x9101, Data Size 8; `ZE` precedes SuiteID |
| Password derivation | PBKDF2-HMAC-SHA-1, 1,000 iterations per entry salt | PBKDF2-HMAC-SHA-256, 1,000,000 iterations per root context, then labeled expansion |
| AES counter | Little-endian | 128-bit big-endian, first block 1 |
| Authentication | 10-byte HMAC-SHA-1 tag | 16-byte HMAC-SHA-256 tag binding mode, descriptor and ciphertext |
| Expected plaintext CRC | Public in AE-1; omitted in AE-2 | Encrypted four-byte trailer in both directory forms; outer CRCs zero |

Test the examples here, then Spec 14's negative cases: wrong-password verifier collisions, bad tags, valid tags with invalid CRC/codec/size, swapped payloads, duplicate descriptors or ZEAC, repeated vendor fields, ZIP64 reservations, data descriptors, mixed roots, and CTR chunk/carry boundaries. A successful decode of one archive is a useful integration check, not complete conformance evidence.

## 9. Worked protected archive

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
00000000000000000000000000007465737420666f722067726f6b20746f2064
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.

## Copyright and trademarks

Copyright (c) 2026 Visware LLC.

ZIP Extractor is developed and published by Visware LLC. This guide 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. Compatibility does not imply endorsement or certification by Visware LLC.