FoldLock + WhistleLock

Full papers and source listings

Aziel / GodLock.AZ
2 September 2026 · Apache-2.0
No repository or DOI claimed.

WhistleLock does not implement rotating sender mail or IP masking. Those are an untraceable-origin send path and are refused.

FoldLock

Algorithmic tether suppression

Paper ID: FL-WP-0.3 Product: FoldLock v0.3 Supersedes: FL-WP-0.2, FL-WP-0.1 Aziel / GodLock.AZ 2 September 2026 License: Apache-2.0

> *Pull the tethers. Restore the bytes. Do not ship a zip and call it a fold.*

Abstract

FoldLock is a text compressor whose method is algorithmic suppression of tether words. A tether is a high-frequency function word that holds sentences together without carrying the payload: as, is, has, to, and, or, etc, and the function-word net around them.

Fold walks UTF-8 text, pulls those tethers out of the byte stream, and writes a `.fld` whose body is residual bytes plus compact tether opcodes. Unfold puts the tethers back and will not emit a file unless the restored bytes match the original size and SHA-256.

This is not a zip wrapper. v0.2 used zlib. That draft is retired. FoldLock's claim in this mesh is the suppression method — a linguistic fold with exact restore — not a DEFLATE clone and not a fabricated bake-off against general binary compressors.

Runtime: `foldlock.py`. Magic: `FLD3`. Lexicon: `TETH-1`. No repository or DOI is claimed as of this paper.

---

1. Why v0.2 was the wrong machine

A zlib header plus a compressed body is a zip-class bundle. It reduces bytes by a general entropy coder. That work already has a name.

FoldLock is named for the fold: take the tethers out of the language surface, keep the residual, restore on the way back. If the product cannot name the words it removed and put them back to the original byte length, it is not FoldLock.

Operator correction, 2 September 2026:

- folding compresses by reducing bytes via algorithmic suppression;
- suppression removes tether words (as, is, has, to, and, or, etc);
- unfold restores those tethers to the proper byte-size file;
- a zip bundle is not the product.

---

2. Purpose

1. Suppress, don't delete. A deleted word cannot be restored to the original bytes. Each tether becomes an opcode that still knows which word, which case, and which surrounding spaces were pulled. 2. Restore to the same file. Unfold is lossless. `orig_size` and `orig_sha256` are the gate. 3. Refuse binary. FoldLock v0.3 is a text fold. A PNG is not a sentence. Do not silently wrap it in zlib to look useful. 4. Name the lexicon. `TETH-1` is a table, not a mood. Reordering IDs orphans containers.

---

3. What this is / is not

This is

- reversible tether-word suppression on UTF-8 text;
- a 3-byte opcode for each suppressed tether (escape + marker + id);
- exact restore of letters and of single ASCII spaces bound to those

tethers;

- Path A (refold to a new file) / Path B (do not rewrite a `.fld`).

This is not

- zip, zlib, gzip, DEFLATE, zstd, or lzma;
- a registration store (FL-WP-0.1, retired);
- a claim that every file class shrinks — short strings and identifier-

heavy source can grow by the header and by 3-byte opcodes on one- letter tethers such as `a` / `i`;

- UL, EmployeeLock, TemporalLock, EmbryoLock, ARK, or GodLock;
- a published industry bake-off. No corpus ranking is asserted here.

Product stance in this mesh: FoldLock is the tether-suppression compressor. Leadership is the method class. Ratios track tether density. They are receipts, not trophies.

---

4. Tether lexicon TETH-1

Operator core, first seven IDs, never reordered:

```
as is has to and or etc
```

Then the function-word net that makes suppression a fold instead of a seven-word toy: articles, auxiliaries, pronouns, demonstratives, common prepositions and conjunctions. The live ordered table is `TETHERS` in `foldlock.py`. Capacity cap: 255 IDs. v0.3 ships 112.

A token becomes a tether only if:

- it is a letter-run `[A-Za-z]+`;
- its lowercase form is in `TETH-1`;
- its case is all-lower, Title, or ALL-CAPS.

Mixed case (`Mc` style, or `tHe`) stays literal. That keeps restore exact without a fourth case page.

`etcetera` is in the table. `etc.` with the period: the letters `etc` suppress; the period stays residual.

---

5. Algorithmic suppression

5.1 Tokenise

Split the decoded UTF-8 string with

```
[A-Za-z]+|[^A-Za-z]+
```

Letter runs and the leftover (spaces, digits, punctuation, newlines) alternate. Nothing is dropped on the floor.

5.2 Space shapes

A tether often sits in a single ASCII space on one or both sides. Those spaces are part of the fold, not leftover dust.

Shape	Bytes pulled with the word
bare	`word`
lead	`word`
trail	`word`
both	`word`

Only U+0020 counts. Tabs, newlines, and double spaces stay residual so restore cannot invent layout.

Adjacent tethers: if the previous opcode already ate the trailing space, the next opcode does not eat it again.

5.3 Stream codec

Not an entropy coder.

Sequence	Meaning
`0xFF 0xFF`	one literal byte `0xFF`
`0xFF marker id`	tether. `marker = (case << 2) \
any other byte	literal residual byte

Case codes: 0 lower, 1 Title, 2 UPPER. Code 3 is illegal in a container.

A tether that was `and ` (5 bytes) becomes 3 bytes. A tether that was `a` (1 byte) becomes 3 bytes and is a net loss. That is honest and visible in the fold receipt (`tether_bytes_saved` can be negative on that hit).

5.4 Header

Little-endian `<4sBBQ32sQ`:

Field	Size	Rule
magic	4	`FLD3`
lexicon_id	1	`1` = TETH-1
reserved	1	`0`
orig_size	8	original UTF-8 length
orig_sha256	32	raw digest of original bytes
body_len	8	length of the suppression stream
body	body_len	stream from 5.3

`FLD2` files raise: zlib wrapper is retired; refold with v0.3.

---

6. Unfold

```
python3 foldlock.py unfold IN.fld [--out OUTFILE]
```

1. Parse header. Refuse bad magic, bad lexicon, short file, body_len mismatch. 2. Walk the stream. Expand opcodes to letters and bound spaces. 3. UTF-8 encode the restored text. 4. Refuse if `len != orig_size`. 5. Refuse if `SHA-256 != orig_sha256`. 6. Write bytes. Print `verified: True`, `zip: False`.

Partial decode is not a success. There is no “best guess” reinsertion of tethers. Prediction is not this spec. The opcodes are the memory of what was pulled.

---

7. Fold and info

```
python3 foldlock.py fold INFILE [--out OUT.fld]
python3 foldlock.py info IN.fld
```

Fold decodes UTF-8 or refuses. It does not delete `INFILE`. It does not store the filename. Default output is `INFILE.fld`.

Receipt fields include method `tether-suppression`, lexicon, tether hit count, estimated bytes saved inside the stream, orig/folded sizes, ratio, and `zip: False`.

Info reads the header and counts tether opcodes. It does not unfold.

---

8. Paths

Path A — regenerative. The same text may be folded again (later lexicon, later encoder). New file. Old `.fld` stays unless the operator deletes it outside this spec.

Path B — immutable. Do not rewrite a `.fld` in place. A later fold is a different object.

v0.3 does not stamp A/B into the header.

9. Worked miniatures (informative)

Exact restore on these strings (body size is the stream only):

Input	Hits	Body / raw
`the cat and the dog`	3	15 / 19
`As is has to and or etc.`	7	22 / 24
`and and and`	3	9 / 11
`hello`	0	5 / 5

EmployeeLock whitepaper markdown used as a live text (identifier-heavy, so a hard case, not a sales case): 550 hits, stream 13105 / 13607, container ratio about 0.967, unfold SHA-256 matched, magic `FLD3`, body is not zlib `\x78\x9c`.

Those numbers are receipts from the v0.3 burn. They are not a ranking against zip.

10. Relation to the rest of the mesh

Sibling	Boundary
EmployeeLock	May index a `.fld` as evidence. Does not suppress tethers.
TemporalLock	Time receipts. FoldLock has no timestamp field.
EmbryoLock / ARK1	May wrap a copy. Wrapping is not a fold.
GodLock	Not a codec.
UL / BAL	Issue cluster. Do not file FoldLock under UL-CAT.

11. Limits

FoldLock v0.3 does not:

- fold non-UTF-8 bytes;
- invent tethers that were never in the file;
- suppress tabs, newlines, or punctuation;
- encrypt;
- stream-fold without reading the file;
- repair a truncated stream;
- claim a zip-beating corpus score;
- claim a repository or DOI.

English is the shipped lexicon language. Other languages need a new lexicon_id.

12. Status

Item	State
Structure name	FoldLock v0.3
Paper	FL-WP-0.3 — this document
Companion spec	`FoldLock_v0.3_spec.md`
Runtime	`foldlock.py`

Item	State
Magic	FLD3
Lexicon	TETH-1 (112 words, append-only)
Method	algorithmic tether suppression
Zip	False
Public repo	None claimed
DOI	None claimed
License	Apache-2.0
Retired	FL-WP-0.2 zlib wrapper; FL-WP-0.1 registration store

Aziel / GodLock.AZ 2 September 2026 A fork that wraps zlib and keeps the name FoldLock is a zip bundle. It is not this spec.

FoldLock v0.3 — Reproduction Whitepaper

Paper ID: FL-WP-0.3-R Product: FoldLock v0.3 Aziel / GodLock.AZ 2 September 2026 License: Apache-2.0

Every file named. Every hash listed. Fold, unfold, compare.

1. What you are reproducing

FoldLock v0.3 is algorithmic tether suppression on UTF-8 text. It is not zip. Magic `FLD3`. Lexicon `TETH-1`. Parent paper: FL-WP-0.3. This paper is the rebuild recipe.

2. File set (coded and listed)

All paths under `/home/workdir/artifacts/` unless noted. Repro pack also copies the starred files into `FoldLock_v0.3_repro/`.

File	Role	SHA-256
`foldlock.py` *	runtime	`e3bb667b703802f7860f579aa77dc348df0c358dbc2d2217c59f57a667fe14b7`
`FoldLock_v0.3_spec.md` *	short spec	`7eda63e851e3e38293be97d70245c4622cbaebf20d063132f335dab7363e78f8`
`FoldLock_v0.3_whitepaper.md` *	FL-WP-0.3 source	`b25530d947fa040f251ac6cf5f1d8a64eb44cd18c3f8c30b37f8c39fe43603f9`
`FoldLock_v0.3_whitepaper.pdf`	FL-WP-0.3 print	see `FoldLock_v0.3.sha256.txt`
`FoldLock_v0.3_repro/VECTORS.txt` *	test plaintext	`1db33c4dc38d08b3d29f77f75e15b598e3805a7dde18a7a122b9f257369f4ba1`
`FoldLock_v0.3_repro/VECTORS.txt.fld` *	folded vectors	`0f6797cfade69f4bbd02c42cbc71c93a235cd649ddea716bdfa884b08e7d2049`
`FoldLock_v0.3_repro/MANIFEST.sha256` *	pack checksums	regenerate with `sha256sum`
`FoldLock_v0.3_repro/REPRODUCE.sh` *	one-shot check	run it
`FoldLock_v0.3_REPRO_whitepaper.md`	this paper	hashed after write

Python 3.12 stdlib only. No pip. No zlib used for the fold.

3. Reproduce in one shot

```
cd /home/workdir/artifacts/FoldLock_v0.3_repro
chmod +x REPRODUCE.sh
./REPRODUCE.sh
```

Expected last line: `REPRODUCE_OK`. `sha256sum -c MANIFEST.sha256` must pass before fold.

4. Reproduce by hand

```
python3 foldlock.py fold   VECTORS.txt --out /tmp/v.fld
python3 foldlock.py unfold /tmp/v.fld --out /tmp/v.out
cmp VECTORS.txt /tmp/v.out
python3 foldlock.py info   /tmp/v.fld
```

Unfold must print `verified: True` and `zip: False`. `cmp` must be silent.

VECTORS.txt contents (exact bytes, four lines):

```
the cat and the dog
As is has to and or etc.
and and and
```

```
hello
```

orig_size 63. orig_sha256 `1db33c4dc38d08b3d29f77f75e15b598e3805a7dde18a7a122b9f257369f4ba1`. Header makes this tiny file larger than plaintext. That is expected. The check is restore, not ratio.

5. Header to code

On-disk little-endian:

```
<4sBBQ32sQ
magic FLD3 | lexicon_id 1 | reserved 0 | orig_size u64
| orig_sha256 32 | body_len u64 | body
```

Implemented in `foldlock.py` as `MAGIC`, `LEXICON_ID`, `ONDISK`.

Stream: `0xFF 0xFF` = literal 0xFF. `0xFF marker id` = tether. Any other byte = residual. Marker = `(case << 2) | shape`.

6. TETH-1 table (IDs 0…111, do not reorder)

0 as, 1 is, 2 has, 3 to, 4 and, 5 or, 6 etc, 7 the, 8 a, 9 an, 10 of, 11 in, 12 on, 13 for, 14 with, 15 at, 16 by, 17 from, 18 into, 19 onto, 20 upon, 21 it, 22 its, 23 be, 24 am, 25 are, 26 was, 27 were, 28 been, 29 being, 30 have, 31 had, 32 having, 33 do, 34 does, 35 did, 36 doing, 37 will, 38 would, 39 shall, 40 should, 41 can, 42 could, 43 may, 44 might, 45 must, 46 not, 47 no, 48 nor, 49 but, 50 if, 51 then, 52 than, 53 so, 54 too, 55 that, 56 this, 57 these, 58 those, 59 which, 60 who, 61 whom, 62 whose, 63 what, 64 when, 65 where, 66 why, 67 how, 68 i, 69 me, 70 my, 71 we, 72 us, 73 our, 74 you, 75 your, 76 he, 77 him, 78 his, 79 she, 80 her, 81 they, 82 them, 83 their, 84 all, 85 any, 86 each, 87 every, 88 few, 89 more, 90 most, 91 other, 92 some, 93 such, 94 also, 95 just, 96 only, 97 even, 98 up, 99 out, 100 off, 101 over, 102 under, 103 here, 104 there, 105 both, 106 same, 107 own, 108 per, 109 via, 110 vs, 111 etcetera.

Source of truth: the tuple `TETHERS` in `foldlock.py`. This list must match that tuple in order.

7. Source is the code

The runtime to reproduce is the file `foldlock.py` in this pack (430 lines, SHA-256 above). Commands: `fold`, `unfold`, `info`. Do not substitute a zip encoder and keep the name FoldLock.

8. Status

Item	State
Paper	FL-WP-0.3-R
Runtime	foldlock.py v0.3
Pack	FoldLock_v0.3_repro/
License	Apache-2.0
Repo / DOI	none claimed

Aziel / GodLock.AZ 2 September 2026

Appendix A — foldlock.py

foldlock.py · 430 lines · listed in full

```
#!/usr/bin/env python3
"""FoldLock v0.3 — tether-word algorithmic suppression. Not a zip wrapper."""

from __future__ import annotations

import argparse
import hashlib
import re
import struct
import sys
from pathlib import Path

MAGIC = b"FLD3"
LEXICON_ID = 1
HEADER = "<4sBBQ32s"  # magic, lexicon_id, reserved, orig_size, orig_sha256
# on disk: header || body_len_u64 || body
ONDISK = "<4sBBQ32sQ"

# Stable IDs. Do not reorder. Append only. Unfold uses the same table.
# Core tethers named by the operator, then the surrounding function-word net
# that makes suppression a real fold instead of a slogan.
TETHERS: tuple[str, ...] = (
    "as",
    "is",
    "has",
    "to",
    "and",
    "or",
    "etc",
    "the",
    "a",
    "an",
    "of",
    "in",
    "on",
    "for",
    "with",
    "at",
    "by",
    "from",
    "into",
    "onto",
    "upon",
    "it",
    "its",
    "be",
    "am",
    "are",
    "was",
    "were",
    "been",
    "being",
    "have",
    "had",
    "having",
    "do",
    "does",
    "did",
    "doing",
    "will",
    "would",
    "shall",
    "should",
    "can",
    "could",
    "may",
    "might",
    "must",
    "not",
    "no",
    "nor",
    "but",
    "if",
```

```
    "then",
    "than",
    "so",
    "too",
    "that",
    "this",
    "these",
    "those",
    "which",
    "who",
    "whom",
    "whose",
    "what",
    "when",
    "where",
    "why",
    "how",
    "i",
    "me",
    "my",
    "we",
    "us",
    "our",
    "you",
    "your",
    "he",
    "him",
    "his",
    "she",
    "her",
    "they",
    "them",
    "their",
    "all",
    "any",
    "each",
    "every",
    "few",
    "more",
    "most",
    "other",
    "some",
    "such",
    "also",
    "just",
    "only",
    "even",
    "up",
    "out",
    "off",
    "over",
    "under",
    "here",
    "there",
    "both",
    "same",
    "own",
    "per",
    "via",
    "vs",
    "etcetera",
)

assert len(TETHERS) <= 255
TETHER_INDEX = {w: i for i, w in enumerate(TETHERS)}

# shape in low 2 bits of marker
SHAPE_BARE = 0
SHAPE_LEAD = 1   # " word"
SHAPE_TRAIL = 2  # "word "
SHAPE_BOTH = 3   # " word "
# case in next 2 bits
CASE_LOWER = 0
CASE_TITLE = 1
CASE_UPPER = 2
CASE_MIXED = 3  # treat as literal instead

ESC = 0xFF  # 0xFF 0xFF = literal 0xFF; 0xFF marker id = tether
```

```
TOKEN_RE = re.compile(r"[A-Za-z]+|[^A-Za-z]+")


def _case_code(word: str) -> int | None:
    if word.islower():
        return CASE_LOWER
    if word.isupper():
        return CASE_UPPER
    if word[:1].isupper() and word[1:].islower():
        return CASE_TITLE
    return None


def _apply_case(base: str, case: int) -> str:
    if case == CASE_LOWER:
        return base
    if case == CASE_UPPER:
        return base.upper()
    if case == CASE_TITLE:
        return base[:1].upper() + base[1:]
    raise ValueError("mixed case is not a tether opcode")


def suppress(text: str) -> tuple[bytes, dict]:
    """Reversible tether suppression. Exact restore. No zip."""
    tokens = TOKEN_RE.findall(text)
    out = bytearray()
    tether_hits = 0
    tether_bytes_saved = 0

    def emit_lit_bytes(raw: bytes) -> None:
        for b in raw:
            out.append(b)
            if b == ESC:
                out.append(ESC)

    i = 0
    n = len(tokens)
    prev_took_trail_space = False
    while i < n:
        tok = tokens[i]
        key = tok.lower()
        if tok.isalpha() and key in TETHER_INDEX:
            case = _case_code(tok)
            if case is None:
                emit_lit_bytes(tok.encode("utf-8"))
                prev_took_trail_space = False
                i += 1
                continue
            lead = (
                (not prev_took_trail_space)
                and i > 0
                and tokens[i - 1] == " "
            )
            trail = i + 1 < n and tokens[i + 1] == " "
            # only eat spaces that are exactly one ASCII space
            shape = SHAPE_BARE
            take_lead = False
            take_trail = False
            if lead and trail:
                shape = SHAPE_BOTH
                take_lead = True
                take_trail = True
            elif lead:
                shape = SHAPE_LEAD
                take_lead = True
            elif trail:
                shape = SHAPE_TRAIL
                take_trail = True
            if take_lead and out.endswith(b" "):
                del out[-1]
            elif take_lead and out:
                # last emitted byte is not a pending space we can steal
                shape = SHAPE_TRAIL if take_trail else SHAPE_BARE
                take_lead = False
            out.append(ESC)
            out.append((case << 2) | shape)
            out.append(TETHER_INDEX[key])
            tether_hits += 1
```

```
            original = ((" " if take_lead else "") + tok + (" " if take_trail else ""))
            tether_bytes_saved += len(original.encode("utf-8")) - 3
            i += 1
            if take_trail:
                i += 1
            prev_took_trail_space = take_trail
            continue
        emit_lit_bytes(tok.encode("utf-8"))

        prev_took_trail_space = False
        i += 1
    stats = {
        "tether_hits": tether_hits,
        "tether_bytes_saved": tether_bytes_saved,
        "lexicon": f"TETH-{LEXICON_ID}",
        "tether_words": len(TETHERS),
    }
    return bytes(out), stats


def expand(body: bytes) -> str:
    raw_out = bytearray()
    i = 0
    n = len(body)
    while i < n:
        b = body[i]
        i += 1
        if b != ESC:
            raw_out.append(b)
            continue
        if i >= n:
            raise ValueError("truncated escape")
        nxt = body[i]
        i += 1
        if nxt == ESC:
            raw_out.append(ESC)
            continue
        if i >= n:
            raise ValueError("truncated tether id")
        wid = body[i]
        i += 1
        marker = nxt
        if wid >= len(TETHERS):
            raise ValueError(f"tether id {wid} not in TETH-{LEXICON_ID}")
        case = (marker >> 2) & 0x03
        shape = marker & 0x03
        if case == CASE_MIXED:
            raise ValueError("mixed-case tether opcode is illegal")
        word = _apply_case(TETHERS[wid], case)
        if shape == SHAPE_LEAD:
            word = " " + word
        elif shape == SHAPE_TRAIL:
            word = word + " "
        elif shape == SHAPE_BOTH:
            word = " " + word + " "
        raw_out.extend(word.encode("utf-8"))
    return raw_out.decode("utf-8")


def fold(src: Path, dst: Path) -> dict:
    raw = src.read_bytes()
    try:
        text = raw.decode("utf-8")
    except UnicodeDecodeError as e:
        raise ValueError(
            "FoldLock v0.3 folds UTF-8 text by tether suppression. "
            "Binary input is refused. This is not a zip wrapper."
        ) from e
    digest = hashlib.sha256(raw).digest()
    body, stats = suppress(text)
    header = struct.pack(ONDISK, MAGIC, LEXICON_ID, 0, len(raw), digest, len(body))
    dst.write_bytes(header + body)
    return {
        "in": str(src),
        "out": str(dst),
        "method": "tether-suppression",
        "lexicon": stats["lexicon"],
        "tether_words": stats["tether_words"],
        "tether_hits": stats["tether_hits"],
        "tether_bytes_saved": stats["tether_bytes_saved"],
```

```
        "orig_size": len(raw),
        "folded_size": len(header) + len(body),
        "body_size": len(body),
        "orig_sha256": digest.hex(),
        "ratio": ((len(header) + len(body)) / len(raw)) if raw else 0.0,
        "zip": False,
    }

def read_container(blob: bytes) -> tuple[dict, bytes]:
    n = struct.calcsize(ONDISK)
    if len(blob) < n:
        raise ValueError("file too short for FoldLock v0.3 header")
    magic, lex, _res, orig_size, digest, body_len = struct.unpack(ONDISK, blob[:n])
    if magic == b"FLD2":
        raise ValueError("FLD2 (zlib wrapper) is retired. Refold with FoldLock v0.3.")
    if magic != MAGIC:
        raise ValueError("not a FoldLock v0.3 file")
    if lex != LEXICON_ID:
        raise ValueError(f"unsupported lexicon {lex}")
    body = blob[n:]
    if len(body) != body_len:
        raise ValueError(f"body length mismatch: header {body_len} file {len(body)}")
    return {
        "lexicon": f"TETH-{lex}",
        "orig_size": orig_size,
        "body_size": body_len,
        "orig_sha256": digest.hex(),
        "digest_raw": digest,
        "method": "tether-suppression",
        "zip": False,
    }, body


def unfold(src: Path, dst: Path) -> dict:
    meta, body = read_container(src.read_bytes())
    text = expand(body)
    raw = text.encode("utf-8")
    if len(raw) != meta["orig_size"]:
        raise ValueError("orig_size mismatch after unfold — suppression restore failed")
    got = hashlib.sha256(raw).digest()
    if got != meta["digest_raw"]:
        raise ValueError("orig_sha256 mismatch — unfold refused")
    dst.write_bytes(raw)
    return {
        "in": str(src),
        "out": str(dst),
        "orig_size": len(raw),
        "orig_sha256": got.hex(),
        "verified": True,
        "method": "tether-suppression",
        "zip": False,
    }


def info(src: Path) -> dict:
    meta, body = read_container(src.read_bytes())
    meta.pop("digest_raw", None)
    meta["file"] = str(src)
    meta["magic"] = "FLD3"
    i = 0
    hits = 0
    while i < len(body):
        b = body[i]
        i += 1
        if b != ESC:
            continue
        if i >= len(body):
            break
        nxt = body[i]
        i += 1
        if nxt == ESC:
            continue
        hits += 1
        i += 1
    meta["tether_hits"] = hits
    return meta


def main() -> int:
    p = argparse.ArgumentParser(
```

```
        prog="foldlock",
        description="FoldLock v0.3 — algorithmic tether suppression. Not zip.",
    )
    sub = p.add_subparsers(dest="cmd", required=True)
    f = sub.add_parser("fold")
    f.add_argument("src")
    f.add_argument("--out")
    u = sub.add_parser("unfold")
    u.add_argument("src")
    u.add_argument("--out")
    i = sub.add_parser("info")
    i.add_argument("src")
    args = p.parse_args()
    try:
        if args.cmd == "fold":
            src = Path(args.src)
            dst = Path(args.out) if args.out else Path(str(src) + ".fld")
            print(fold(src, dst))
            return 0
        if args.cmd == "unfold":
            src = Path(args.src)
            if args.out:
                dst = Path(args.out)
            elif src.name.endswith(".fld"):
                dst = src.with_name(src.name[: -4])
            else:
                dst = Path(str(src) + ".out")
            print(unfold(src, dst))
            return 0
        print(info(Path(args.src)))
        return 0
    except ValueError as e:
        print(f"foldlock: {e}", file=sys.stderr)
        return 1


if __name__ == "__main__":
    raise SystemExit(main())
```

WhistleLock

Local immutable drop + dead-man packet

Paper ID: WL-WP-0.1 Product: WhistleLock v0 Aziel / GodLock.AZ 2 September 2026 License: Apache-2.0

> *Store the record. Chain the row. Release the packet locally. The operator carries it.*

Abstract

WhistleLock is a local vault for materials the operator already holds: whistle records, investigative files, journalist correspondence saved as files, and databases in the operator's possession. Drops are copied into the store, hashed, and written to an append-only ledger. A dead-man switch watches a check-in clock. If the clock lapses, the pre-placed packet is copied into `released/`.

WhistleLock does not mail. It does not rotate sender addresses. It does not mask IP. It does not scrape inboxes at boot. Those would be an untraceable-origin send path. That path is refused.

Runtime: `whistlelock.py`. Spec string: `whistlelock-v0`. No repository or DOI is claimed as of this paper.

---

1. Purpose

Whistle material dies in three ordinary ways: the file is deleted, the story is renamed until it has no owner, or the holder is silenced before the packet moves.

WhistleLock answers those with three local acts:

1. Drop — copy a file the operator already has. Hash it. Ledger it. 2. Chain — each ledger row names the previous row's hash. Delete is not a command. 3. Dead-man — if check-in stops, copy the packet the operator pre-placed into `released/`.

Accuracy is re-hashing what is already in the store. It is not a search of other people's mail.

---

2. What this is / is not

This is

- a directory store with `HEADER.json`, `ledger.jsonl`, `drops/`,

`deadman/packet/`, `deadman/released/`;

- TemporalLock-shaped rows;
- a local dead-man copy;
- optional refresh of an operator-supplied `source_url` at verify time.

This is not

- a rotating encrypted identity mailbox;
- an IP-masking or proxy stack;
- a mixnet, drop box in the sky, or anonymous relay;
- a boot scraper of journalist or source inboxes;
- a public website;
- UL, FoldLock, EmployeeLock, or GodLock;
- legal advice. Filing, reporting, and source protection remain the

operator's work outside this tool.

Refused on purpose. AZbot standing rule: do not build an untraceable-origin upload path. The operator moves the released packet on a channel they already control.

---

3. Store layout

```
STORE/
  HEADER.json
  ledger.jsonl
  drops/
    DR-<digest12>/
      meta.json
      <original filename>
  deadman/
    state.json
    packet/          # operator places files here
    released/
      <UTC stamp>/   # tick copies packet here when overdue
```

HEADER.spec = `whistlelock-v0`.

Drop meta: drop_id, original_name, stored_path, size_bytes, payload_sha256, source_note, source_url, imported.

---

4. Ledger

JSONL, append-only. Genesis prev_hash = 64 ASCII zeros.

Hashed fields only:

```
entry_id, timestamp, kind, summary, drop_id,
payload_sha256, source_note, prev_hash
```

Canonical form: UTF-8 JSON, sorted keys, compact separators. `row_hash` = SHA-256 hex of that string.

kind: `drop` | `checkin` | `arm` | `release` | `verify` | `note`.

A correction is a new row. The old line stays.

---

5. Drop

```
python3 whistlelock.py drop STORE FILE --summary TEXT [--source NOTE] [--url URL]
```

Copies FILE into `drops/DR-<first12 of sha256>/`. Writes meta.json. Appends a `drop` row with the payload digest.

`--url` is a locator the operator already knows (a public page they saved from). It is not a hunt instruction. `verify --refresh` may fetch that exact URL and compare hashes. Failure is logged. The local copy is not deleted.

Email: store a `.eml` or export the operator already possesses. WhistleLock does not collect mail.

---

6. Dead-man

```
python3 whistlelock.py arm     STORE --hours N
python3 whistlelock.py checkin STORE
python3 whistlelock.py tick    STORE
```

Arm sets interval and stamps check-in. Check-in resets the clock. Tick does nothing inside the window. Past the window, tick copies `deadman/packet/` to `deadman/released/<UTC>/`, writes `RELEASE_NOTICE.txt`, ledgers `release`, and will not copy again until the operator arms again.

Boot / cron is the operator's. Example local unit only:

```
# every 15 minutes, this machine, this store
*/15 * * * * python3 /path/whistlelock.py tick /path/STORE
```

No mailer in that line. No IP flag. No rotating From:.

If the packet directory is empty, release still happens and the notice file is the whole payload. Put the real files in `packet/` before you arm.

---

7. Verify

```
python3 whistlelock.py verify STORE [--refresh]
```

Walks the ledger (hash + prev_hash). Re-hashes each stored drop. Reports missing files. `--refresh` only hits URLs written at ingest. It does not discover new sources.

A verify row is itself appended (so the check is in the chain).

8. CLI

```
python3 whistlelock.py init    STORE
python3 whistlelock.py drop    STORE FILE --summary TEXT [--source NOTE] [--url URL]
python3 whistlelock.py checkin STORE
python3 whistlelock.py arm     STORE --hours N
python3 whistlelock.py tick    STORE
python3 whistlelock.py verify  STORE [--refresh]
python3 whistlelock.py list    STORE
```

9. Files to reproduce

File	Role
`whistlelock.py`	runtime
`WhistleLock_v0_spec.md`	short spec
`WhistleLock_v0_whitepaper.md`	this paper
`WhistleLock_v0_whitepaper.pdf`	print
`WhistleLock_v0/`	optional live store created by init

Python 3 stdlib only (`hashlib`, `json`, `shutil`, `urllib.request`).

Smoke:

```
python3 whistlelock.py init /tmp/wl
echo test > /tmp/wl_sample.txt
python3 whistlelock.py drop /tmp/wl /tmp/wl_sample.txt --summary "sample drop"
python3 whistlelock.py arm /tmp/wl --hours 1
python3 whistlelock.py checkin /tmp/wl
python3 whistlelock.py verify /tmp/wl
python3 whistlelock.py list /tmp/wl
```

10. Relation

Sibling	Boundary
FoldLock	May fold a text drop. Not this ledger.
EmployeeLock	Workplace event rows. Not a whistle store.
TemporalLock	Ethic borrowed. Product stays separate.
EmbryoLock / ARK1	Wrap a copy of STORE if the threat model says so.
UL / BAL	Issue cluster. Do not file WhistleLock under UL-CAT.

11. Limits

WhistleLock v0 does not encrypt the store (pair with ARK/EmbryoLock), does not set exhibit numbers, does not decide whether a record should be published, does not contact journalists, and does not hide the operator's network identity.

12. Status

Item	State
Structure	WhistleLock v0
Paper	WL-WP-0.1
Runtime	whistlelock.py
Repo / DOI	none claimed
License	Apache-2.0

Aziel / GodLock.AZ 2 September 2026 A fork that adds hidden From: rotation or IP masking and keeps this name is no longer this spec.

Appendix B — whistlelock.py

whistlelock.py · 410 lines · listed in full

```
#!/usr/bin/env python3
"""WhistleLock v0 — local immutable drop ledger + dead-man packet.

Not an anonymous mailer. Not an IP mask. Not an inbox scraper.
The operator moves anything that leaves this store.
"""

from __future__ import annotations

import argparse
import hashlib
import json
import shutil
import sys
import urllib.request
from datetime import datetime, timezone
from pathlib import Path

SPEC = "whistlelock-v0"
GENESIS = "0" * 64
KINDS = ("drop", "checkin", "arm", "release", "verify", "note")


def utc_now() -> str:
    return datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ")


def sha256_file(path: Path) -> tuple[str, int]:
    h = hashlib.sha256()
    n = 0
    with path.open("rb") as f:
        while True:
            chunk = f.read(1024 * 1024)
            if not chunk:
                break
            h.update(chunk)
            n += len(chunk)
    return h.hexdigest(), n


def canon(fields: dict) -> str:
    body = {
        "entry_id": fields["entry_id"],
        "timestamp": fields["timestamp"],
        "kind": fields["kind"],
        "summary": fields["summary"],
        "drop_id": fields["drop_id"],
        "payload_sha256": fields["payload_sha256"],
        "source_note": fields["source_note"],
        "prev_hash": fields["prev_hash"],
    }
    return json.dumps(body, sort_keys=True, separators=(",", ":"), ensure_ascii=False)


def row_hash(fields: dict) -> str:
    return hashlib.sha256(canon(fields).encode("utf-8")).hexdigest()


def store_paths(root: Path) -> dict:
    return {
        "root": root,
        "header": root / "HEADER.json",
        "ledger": root / "ledger.jsonl",
        "drops": root / "drops",
        "packet": root / "deadman" / "packet",
        "released": root / "deadman" / "released",
        "deadman": root / "deadman" / "state.json",
    }


def load_header(p: dict) -> dict:
    return json.loads(p["header"].read_text(encoding="utf-8"))

def write_header(p: dict, header: dict) -> None:
```

```
    p["header"].write_text(json.dumps(header, indent=2) + "\n", encoding="utf-8")


def last_hash(p: dict) -> str:
    if not p["ledger"].exists() or p["ledger"].stat().st_size == 0:
        return GENESIS
    last = ""
    with p["ledger"].open(encoding="utf-8") as f:
        for line in f:
            if line.strip():
                last = line
    return json.loads(last)["row_hash"]


def next_id(p: dict, prefix: str) -> str:
    n = 0
    if p["ledger"].exists():
        with p["ledger"].open(encoding="utf-8") as f:
            for line in f:
                if line.strip():
                    n += 1
    return f"{prefix}-{n+1:04d}"


def append(p: dict, kind: str, summary: str, drop_id: str = "", payload: str = "", source: str = "") -> dict:
    if kind not in KINDS:
        raise ValueError(f"bad kind {kind}")
    fields = {
        "entry_id": next_id(p, "WL"),
        "timestamp": utc_now(),
        "kind": kind,
        "summary": summary,
        "drop_id": drop_id,
        "payload_sha256": payload,
        "source_note": source,
        "prev_hash": last_hash(p),
    }
    digest = row_hash(fields)
    rec = dict(fields)
    rec["canonical"] = canon(fields)
    rec["row_hash"] = digest
    with p["ledger"].open("a", encoding="utf-8") as f:
        f.write(json.dumps(rec, separators=(",", ":"), ensure_ascii=False) + "\n")
    return rec


def init(root: Path) -> dict:
    p = store_paths(root)
    if p["header"].exists():
        raise ValueError(f"store already exists: {root}")
    p["drops"].mkdir(parents=True)
    p["packet"].mkdir(parents=True)
    p["released"].mkdir(parents=True)
    header = {
        "spec": SPEC,
        "created": utc_now(),
        "note": "Local drop ledger. Operator moves released packets. No anonymous send path.",
    }
    write_header(p, header)
    p["ledger"].write_text("", encoding="utf-8")
    p["deadman"].write_text(
        json.dumps(
            {
                "armed": False,
                "interval_hours": 0,
                "armed_at": "",
                "last_checkin": "",
                "released": False,
                "last_release": "",
            },
            indent=2,
        )
        + "\n",
        encoding="utf-8",
    )
    rec = append(p, "note", "store genesis")
    return {"store": str(root), "spec": SPEC, "genesis": rec["row_hash"]}

def drop(root: Path, src: Path, summary: str, source: str, url: str) -> dict:
```

```
    p = store_paths(root)
    if not p["header"].exists():
        raise ValueError("init the store first")
    if not src.is_file():
        raise FileNotFoundError(src)

    digest, size = sha256_file(src)
    drop_id = f"DR-{digest[:12]}"
    dest_dir = p["drops"] / drop_id
    dest_dir.mkdir(parents=True, exist_ok=True)
    dest = dest_dir / src.name
    if not dest.exists():
        shutil.copy2(src, dest)
    meta = {
        "drop_id": drop_id,
        "original_name": src.name,
        "stored_path": str(dest),
        "size_bytes": size,
        "payload_sha256": digest,
        "source_note": source,
        "source_url": url,
        "imported": utc_now(),
    }
    (dest_dir / "meta.json").write_text(json.dumps(meta, indent=2) + "\n", encoding="utf-8")
    rec = append(
        p,
        "drop",
        summary,
        drop_id=drop_id,
        payload=digest,
        source=source or url,
    )
    return {"drop_id": drop_id, "payload_sha256": digest, "size": size, "entry_id": rec["entry_id"]}


def load_deadman(p: dict) -> dict:
    return json.loads(p["deadman"].read_text(encoding="utf-8"))


def save_deadman(p: dict, state: dict) -> None:
    p["deadman"].write_text(json.dumps(state, indent=2) + "\n", encoding="utf-8")


def checkin(root: Path) -> dict:
    p = store_paths(root)
    state = load_deadman(p)
    state["last_checkin"] = utc_now()
    save_deadman(p, state)
    rec = append(p, "checkin", f"checkin at {state['last_checkin']}")
    return {"last_checkin": state["last_checkin"], "armed": state["armed"], "entry_id": rec["entry_id"]}


def arm(root: Path, hours: int) -> dict:
    if hours <= 0:
        raise ValueError("hours must be > 0")
    p = store_paths(root)
    state = load_deadman(p)
    state["armed"] = True
    state["interval_hours"] = hours
    state["armed_at"] = utc_now()
    state["last_checkin"] = state["armed_at"]
    state["released"] = False
    state["last_release"] = ""
    save_deadman(p, state)
    rec = append(p, "arm", f"armed {hours}h")
    return {"armed": True, "interval_hours": hours, "entry_id": rec["entry_id"]}


def parse_ts(ts: str) -> datetime:
    return datetime.strptime(ts, "%Y-%m-%dT%H:%M:%SZ").replace(tzinfo=timezone.utc)


def tick(root: Path) -> dict:
    p = store_paths(root)
    state = load_deadman(p)
    if not state.get("armed"):
        return {"released": False, "reason": "not armed"}
    if state.get("released"):
        return {"released": True, "reason": "already released", "at": state.get("last_release")}
    last = state.get("last_checkin") or state.get("armed_at")
```

```
    if not last:
        return {"released": False, "reason": "no checkin clock"}
    age_h = (datetime.now(timezone.utc) - parse_ts(last)).total_seconds() / 3600.0
    if age_h < float(state["interval_hours"]):
        return {
            "released": False,
            "reason": "inside window",
            "age_hours": round(age_h, 4),
            "interval_hours": state["interval_hours"],
        }
    stamp = utc_now().replace(":", "")
    dest = p["released"] / stamp
    dest.mkdir(parents=True)
    copied = []
    if p["packet"].exists():
        for item in p["packet"].iterdir():
            target = dest / item.name
            if item.is_file():
                shutil.copy2(item, target)
                copied.append(item.name)
            elif item.is_dir():
                shutil.copytree(item, target)
                copied.append(item.name + "/")
    notice = dest / "RELEASE_NOTICE.txt"
    notice.write_text(
        "WhistleLock dead-man release\n"
        f"released_at: {utc_now()}\n"
        f"last_checkin: {last}\n"
        f"interval_hours: {state['interval_hours']}\n"
        "This packet was copied locally. WhistleLock did not mail it.\n"
        "The operator moves the files.\n",
        encoding="utf-8",
    )
    state["released"] = True
    state["last_release"] = utc_now()
    save_deadman(p, state)
    rec = append(p, "release", f"dead-man copied {len(copied)} items to {dest.name}")
    return {
        "released": True,
        "dest": str(dest),
        "copied": copied,
        "entry_id": rec["entry_id"],
    }


def verify(root: Path, refresh: bool) -> dict:
    p = store_paths(root)
    errors = []
    expected = GENESIS
    rows = 0
    if not p["ledger"].exists():
        raise ValueError("no ledger")
    with p["ledger"].open(encoding="utf-8") as f:
        for i, line in enumerate(f, 1):
            if not line.strip():
                continue
            rec = json.loads(line)
            rows += 1
            fields = {k: rec[k] for k in (
                "entry_id", "timestamp", "kind", "summary", "drop_id",
                "payload_sha256", "source_note", "prev_hash",
            )}
            if row_hash(fields) != rec.get("row_hash"):
                errors.append(f"line {i} hash mismatch {rec.get('entry_id')}")
            if rec.get("prev_hash") != expected:
                errors.append(f"line {i} chain break {rec.get('entry_id')}")
            expected = rec.get("row_hash")
    missing = []
    url_mismatch = []
    for meta_path in p["drops"].glob("*/meta.json"):
        meta = json.loads(meta_path.read_text(encoding="utf-8"))
        stored = Path(meta["stored_path"])
        if not stored.is_file():
            missing.append(meta["drop_id"])
            continue
        got, _n = sha256_file(stored)
        if got != meta["payload_sha256"]:
            errors.append(f"drop {meta['drop_id']} payload hash mismatch")
        if refresh and meta.get("source_url"):
```

```
            try:
                with urllib.request.urlopen(meta["source_url"], timeout=20) as resp:
                    remote = hashlib.sha256(resp.read()).hexdigest()
                if remote != meta["payload_sha256"]:
                    url_mismatch.append(meta["drop_id"])
            except Exception as e:  # noqa: BLE001 — verify must not crash the store
                errors.append(f"drop {meta['drop_id']} refresh failed: {e}")

    rec = append(
        p,
        "verify",
        f"verify rows={rows} errors={len(errors)} missing={len(missing)} refresh={refresh}",
    )
    return {
        "ok": not errors and not missing and not url_mismatch,
        "rows": rows,
        "errors": errors,
        "missing_files": missing,
        "url_mismatch": url_mismatch,
        "entry_id": rec["entry_id"],
    }


def list_store(root: Path) -> dict:
    p = store_paths(root)
    drops = []
    for meta_path in sorted(p["drops"].glob("*/meta.json")):
        drops.append(json.loads(meta_path.read_text(encoding="utf-8")))
    rows = 0
    if p["ledger"].exists():
        with p["ledger"].open(encoding="utf-8") as f:
            rows = sum(1 for line in f if line.strip())
    return {
        "store": str(root),
        "spec": load_header(p)["spec"],
        "rows": rows,
        "drops": drops,
        "deadman": load_deadman(p),
        "packet_items": [x.name for x in p["packet"].iterdir()] if p["packet"].exists() else [],
    }


def main() -> int:
    pr = argparse.ArgumentParser(
        prog="whistlelock",
        description="WhistleLock v0 local drop ledger + dead-man. No anonymous send path.",
    )
    sub = pr.add_subparsers(dest="cmd", required=True)
    s = sub.add_parser("init")
    s.add_argument("store")
    d = sub.add_parser("drop")
    d.add_argument("store")
    d.add_argument("file")
    d.add_argument("--summary", required=True)
    d.add_argument("--source", default="")
    d.add_argument("--url", default="")
    c = sub.add_parser("checkin")
    c.add_argument("store")
    a = sub.add_parser("arm")
    a.add_argument("store")
    a.add_argument("--hours", type=int, required=True)
    t = sub.add_parser("tick")
    t.add_argument("store")
    v = sub.add_parser("verify")
    v.add_argument("store")
    v.add_argument("--refresh", action="store_true")
    ls = sub.add_parser("list")
    ls.add_argument("store")
    args = pr.parse_args()
    try:
        if args.cmd == "init":
            print(json.dumps(init(Path(args.store)), indent=2))
            return 0
        if args.cmd == "drop":
            print(json.dumps(drop(Path(args.store), Path(args.file), args.summary, args.source, args.url), indent=2))
            return 0
        if args.cmd == "checkin":
            print(json.dumps(checkin(Path(args.store)), indent=2))
            return 0
        if args.cmd == "arm":
```

```
            print(json.dumps(arm(Path(args.store), args.hours), indent=2))
            return 0
        if args.cmd == "tick":
            print(json.dumps(tick(Path(args.store)), indent=2))
            return 0
        if args.cmd == "verify":
            rec = verify(Path(args.store), args.refresh)
            print(json.dumps(rec, indent=2))
            return 0 if rec["ok"] else 1
        print(json.dumps(list_store(Path(args.store)), indent=2))
        return 0
    except (ValueError, FileNotFoundError) as e:
        print(f"whistlelock: {e}", file=sys.stderr)
        return 1


if __name__ == "__main__":
    raise SystemExit(main())
```