



MIRROR: Local Evidence Bundles for Reproducible Research Software Review, with a JANUS Case Study

ANTON SOKOLOV 

ISSUES IN RESEARCH
SOFTWARE

]u[ubiquity press

ABSTRACT

Many statements in a research article depend on code or data, such as ‘the script in the supplement reproduces Table 4’. Such claims are hard to check later, because files change after the text is written. MIRROR is an open-source Python tool and workflow that keeps each claim next to the files that support it. A bundle is a local folder with the work under review, copies of its supporting files, and a claim register that links each claim to its evidence and to the wording the user judges safe to publish. The bundle records a digest (checksum) of each of these files: a short code that changes whenever the file changes. An offline verifier program reports missing or changed files, format errors, and claims that use assurance or compliance phrases such as ‘guarantees’. The worked case is JANUS, a study of how sentences from articles change when machine-translated into other languages and back. The review recorded which files were prepared for release and which stayed private, and it also led to a data availability statement that names the study’s data and code supplement. A bundle that passes the verifier shows what was kept and claimed, not that a claim is true.

CORRESPONDING AUTHOR:

Anton Sokolov

Researcher, Tyche Institute,
Tallinn, Estonia

anton.sokolov@tyche.institute

KEYWORDS:

research software;
reproducibility; provenance;
evidence bundles; artifact
review; claim register

TO CITE THIS ARTICLE:

Sokolov A 2026 MIRROR: Local Evidence Bundles for Reproducible Research Software Review, with a JANUS Case Study. *Journal of Open Research Software*, 14: 65. DOI: <https://doi.org/10.5334/jors.753>

1 INTRODUCTION

Published results often depend on code, and that code can be checked at several points. Before publication, a journal referee asks whether the code does what the methods section says. At publication, an evaluator runs the code and data supplied with an article. After publication, a colleague reuses a published script, and a later maintainer must say what an old release (a numbered, published version of the software) contained. Each is a software review: someone other than the author checks that the stated results still follow from the files. Much research software never reaches such a check, and when one happens, it may begin after the files have already changed. For example, an author writes, ‘The script in the supplement reproduces Table 4’, later fixes a bug in the script, regenerates the table with slightly different numbers, and leaves the sentence unchanged. Months later, a reviewer finds that the script, table, and sentence no longer agree, and nothing records which was checked against which.

MIRROR is a tool and workflow for this situation. Here, a claim is a statement in an article that depends on code or data. The central object is the claim register, a file that lists each claim beside its evidence. One row might read:

- Claim: ‘The script in the supplement reproduces Table 4’
- Evidence: The path to a copy of the supporting file—here, the script that produced the table
- Risk: medium (the bundler’s default; the user may set low, medium, high, or blocked)
- Safe wording: ‘The supplement contains the script that produced Table 4’ (the wording the user judges safe to publish).

A bundle is one folder that holds the register, the article or release being reviewed, and copies of the files its claims rely on. Its file list records a digest (checksum) of each of these files, a short code that changes whenever the file changes. The user states each claim and names its evidence. A bundler program copies the files and writes these claims into the register, with risk `medium` and the claim text as its safe wording. The user then adjusts risk and wording by hand. A verifier program works offline. It recomputes each digest, reports any listed file whose bytes changed, and checks the format of the register and the file list. Its wording rules reject claims that use one of eight phrases, such as ‘guarantees’, unless a negation word precedes it in the same sentence.

An author builds and checks the bundle before submission, a referee or evaluator can rerun the verifier offline to confirm that the bundle’s copies still match their recorded digests, and a colleague or later maintainer sees which evidence belonged to which release. In the Table 4 example, a bundle built before the bug fix keeps

the old script, so a reviewer sees which script the claim was paired with. The worked case is JANUS, a study that machine-translates sentences from research articles into other languages and back, and measures how they change. In JANUS, the author’s review made the data availability statement name a release file with a recorded digest, and the study’s own leakage scan flagged files of its MIRROR-format bundle as reviewer-only. Section 2 compares MIRROR with adjacent practices and describes the bundle and its claim register, Section 3 describes the checks, Sections 4 and 5 the JANUS case, Section 6 the lessons for practice, and Section 7 the limitations.

2 WHAT MIRROR ADDS

Most elements of a MIRROR bundle, such as digests and file lists, already exist; MIRROR arranges them around claims. Its nearest neighbours are RO-Crate (Research Object Crate), a standard for packaging a study’s files with a machine-readable description (1), and PROV, the World Wide Web Consortium model for recording what was made from what (2). Table 1 compares MIRROR with these and five other practices. The bundle’s own records are JSON (JavaScript Object Notation) files, and a schema is the machine-readable rule set they must satisfy. A bundle contains:

- The artifact under review, such as an article draft, software release, or data packet
- Retained copies of supporting files under `sources/`
- The claim register, `claims.json` (Section 2.1)
- A manifest, `manifest.json` (the file list of Section 1), with file paths, the schema version, and SHA-256 digests of the artifact, each source and the register. It also holds a required list of structured assumptions, the names of the expected checks, and a compatibility statement, which says that the bundle is unsigned.
- A note file, `notes.md`, where the bundler writes the purpose, method, a safety caveat and known gaps; reviewers may add notes. It may be edited after bundling: the verifier checks only that it exists; its contents are outside the digest check and the wording rules (Section 3).
- Verifier output, which the user keeps with the bundle.

2.1 THE CLAIM REGISTER IN DETAIL

The claim register, `claims.json`, is a JSON file holding a list of claim records. Its normative definition is the schema in the implementation, `schema/claims.schema.json`; `schema/manifest.schema.json` defines the manifest. Where the prose in the package’s `SCHEMA.md` differs, the JSON schemas and the code govern. A record has these fields:

PRACTICE	WHAT IT ALREADY DOES WELL	WHAT MIRROR ADDS
Checksummed package (e.g., BagIt, RFC 8493)	Lists files with their digests, so changed bytes can be detected (3).	Connects listed files and their digests to specific claims.
RO-Crate	Packages research objects with structured metadata (1).	A local working packet whose files can later be packaged as an RO-Crate.
PROV	Describes what was produced, from what inputs, by which activity or agent (2).	A practical claim register for review, without a full provenance graph.
CodeMeta	Standardises descriptive software metadata in a linked-data JSON format (4).	A claim-level register; its own <code>codemeta.json</code> names the software, not the evidence for each claim.
Artifact review (e.g., CODECHECK)	Assesses availability, functionality, reuse, or reproduction in a venue process (5, 6).	Local materials prepared before formal artifact review.
Claim-evidence models	Models a scholarly claim, its evidence, and arguments as linked data (7).	Pairs each claim with evidence pointers and publishable wording, offline, before publication.
Supply chain tools	Signs and logs builds and their inputs in automated release pipelines (8–10).	Runs offline under the user's control; signing and public logs stay with those tools.

Table 1 MIRROR compared with adjacent practices; numbers in parentheses are references.

- **id** (required): A stable identifier such as `C1` or `C-0001`.
- **text** (required): The claim as the user would state it.
- **evidence** (required): A list of one or more pointers to the supporting evidence.
- **risk** (required): One of `low`, `medium`, `high`, or `blocked`; the verifier checks only that the value is on this list.
- **safe_wording** (required): The wording the user considers safe to publish.
- **review_status** (optional): Free text with no fixed values; the verifier checks that it is a string.
- **notes** (optional): A free-text remark on this record, separate from the bundle's `notes.md`.

Each evidence entry has a **pointer** (the bundler writes a source ID, a path under `sources/`, or `artifact:` and a path) and a `source_id`, which names a listed source or is `null`. A real record from the released example bundle `examples/good` reads (keys reordered and long strings wrapped for print):

```
{
  "id": "C-0001",
  "text": "The example artifact has a local source
    copied into its MIRROR bundle.",
  "evidence": [
    {
      "pointer": "sources/source-1.txt",
      "source_id": null
    }
  ],
  "risk": "medium",
  "safe_wording": "The example artifact has a local
    source copied into its MIRROR bundle.",
  "review_status": "draft",
  "notes": "Generated from CLI arguments;
    review before publication."
}
```

The bundler writes all seven fields, with `risk` `medium`, a safe wording equal to the text, and `review_status` `draft`. The released tool has no command to change `review_status` or refresh a digest. A user who edits `claims.json` gets a digest mismatch until the file's new

digest is written into the manifest by hand (Section 3, Step 4).

3 WORKFLOW

The reference implementation is written in Python: a bundler (`bundle.py`), an offline verifier (`verify.py`), JSON schemas, roll-up and report helpers, and fixtures (sample bundles that must pass or must fail).

1. Select the artifact to review.
2. Name its supporting files, and state each claim with a pointer to its evidence.
3. Run the bundler, which copies the files and writes the register, manifest, and note file.
4. Review each claim's risk and wording. After editing `claims.json`, write its new SHA-256 digest into the `value` field of `claims_file.sha256` in `manifest.json` by hand.
5. Run the verifier and keep its output with the bundle.

The verifier validates `manifest.json` and `claims.json` against their schemas. It recomputes the SHA-256 digest (11) of the artifact, of every retained source, and of `claims.json`, and compares each with the manifest. It resolves an evidence pointer only through the entry's `source_id` or a path under `sources/`; a pointer such as `notes.md` or `artifact:missing-file.md` passes unresolved. It rejects absolute paths and, except for the artifact's own recorded path, paths with a `..` step, and checks that `notes.md` exists. Its wording rules (Section 3.1) scan the manifest's compatibility statement and each claim's `text` and `safe_wording`, so a banned phrase must leave both the text and the safe wording.

The verifier neither digests `notes.md` nor scans it for wording, so its contents may change: reviewers write there after bundling, and a digest would turn every such note into a failure. Also outside the digest check are the `.meta.json` file the bundler writes beside each source and any file the manifest does not list. What must not change belongs in a retained file or in the register. The manifest is not digested or signed, so the digests catch accidental change only; whoever edits a file can also rewrite its digest.

A passing result (`"status": "ok"`, no errors) is easy to misread. It means only that the retained files match the recorded digests and structure and that no wording rule fired. Whether a claim is correct, complete, legally compliant, or ready to publish remains a human decision.

A bundle records one version of the artifact. The bundler copies the artifact into the bundle and records that copy's path, which the verifier checks, so later edits to the working file do not fail its bundles. To review a new version, the user builds a new bundle and treats it as replacing the old one. A manifest records its creation time but no link to a predecessor, so successive bundles are told apart by digests, timestamps, and directory names.

3.1 HOW THE WORDING RULES WORK

The wording rules are a fixed list of eight phrases in the verifier source: `certifies compliance`, `proves legal compliance`, `official audit`, `guarantees, qualified electronic signature`, `qualified timestamp`, `qualified trust service`, and `regulated trust service`. The last four keep an unsigned local bundle out of the vocabulary of trust services regulated under European Union law. The list is versioned with the verifier code, so changing it needs a new release. It targets assurance and compliance language; research overstatements such as 'reproduces' are left to the human safe-wording step (Section 1).

The rule is lexical and errs toward passing. A phrase passes when a negation marker (*not*, *no*, *without*) stands within the 256 characters before it in the same sentence, unless the marker is part of *not only*, or a contrast marker such as *but* or *however*, or a new subject after *and*, comes between them. The released compatibility statement, which says the bundle *is not* a qualified electronic signature, therefore passes; so does 'the script does not crash, so the release guarantees reproducibility'. A failure names the field and the phrase, for example, `manifest.compatibility.statement: must not use unsafe assurance language: certifies compliance`.

4 WORKED EXAMPLE: JANUS

For JANUS, the translation study from Section 1, the author prepared a packet for public release: a manuscript, a source archive, and a data and code supplement with

aggregate CSV files, plots, curated examples, scripts, and dependency metadata. The workspace also held source sentences, translations, and notes that had to stay private. File digests alone could not show what was withheld or what each claim rested on.

The review of the JANUS packet recorded three packet files prepared for release: the PDF manuscript, the source ZIP, and the supplement ZIP. It also recorded five source descriptors, retained files that describe the packet: the checksum manifest, the final packet manifest, the row-level leakage scan (a scan of the supplement and of reviewer-only files for private material), the supplement file manifest, and the packet status note. The review changed the packet in three ways:

1. The JANUS data availability statement was changed to name the data and code supplement prepared for public release instead of describing it only abstractly.
2. Scripts and a dependency manifest were added to the supplement, beside its aggregate outputs.
3. Worked examples about the release process were replaced with examples of sentences that changed in translation.

The release packet gives no evidence on private source sentences, row-level translation quality, licensing for excluded materials, or the methodological correctness of JANUS.

5 EVALUATION

The evaluation records what a reviewer of the JANUS packet can check with its descriptors (Table 2).

The software has its own test suite. The archived v0.1.1 package has 115 tests: 70 exercise the bundler and verifier, 38 the daily roll-up helper, and seven the two report helpers. The suite needs Python 3.12 or newer and two pinned packages, `jsonschema 4.23.0` and `pytest 8.3.5`; `python3 -m pytest tests/` runs it offline. On 27 September 2026, on a Linux host, all 115 passed in under ten seconds.

6 LESSONS FOR RESEARCH SOFTWARE PRACTICE

A claim register tells a reviewer which file supports which statement. In JANUS, this meant writing down which packet file stood behind the data availability statement. Failure messages should also be plain. A digest mismatch might reveal a serious problem, or only a file edited after its digest was recorded, so MIRROR reports which file changed and leaves the verdict to the reviewer. Local review helps before anything is released, signed, or timestamped. In JANUS, the study's leakage scan showed before release that four files of

CRITERION	QUESTION FOR A REVIEWER	JANUS OBSERVATION
Completeness	Are the release files and descriptor files present?	Yes: Three release files and five source descriptors (Section 4).
File identity	Can retained bytes be checked later?	The PDF, source ZIP, and supplement ZIP were tied to SHA-256 records.
Claim clarity	Does each claim point to inspectable support?	The data availability statement was changed to name the release supplement, whose digest the packet records.
Problem visibility	Are warnings and exclusions visible?	The study's own leakage scan flagged four files as reviewer-only, all from its MIRROR-format bundle (register, manifest, note file, and one source CSV); they stayed out of the release packet.

Table 2 Author's assessment of the JANUS packet of 30 May 2026 and its leakage scan of 28 May 2026; counts are files, not claims. The scan covered the 21 files of the release supplement and five reviewer-only files, four of them with warnings.

its MIRROR-format bundle had to stay reviewer-only; the register and manifest carried claim-to-evidence text and row identifiers.

7 LIMITATIONS

MIRROR assumes that the local file system, retained files, and verifier code are open to inspection, and it checks only the bytes it is given, so it cannot detect a malicious operating system, omitted claims, or forged sources. Each manifest marks the bundle as unsigned and untimestamped. Separate software by the same author (12) signs evidence packages and binds to them a timestamp from an outside timestamping service; it could cover an archive of the bundle without editing it. In v0.1.1, the bundler gives a web-address source a type that the schema does not accept, so such a bundle fails verification by default. Evidence held mainly in remote services, closed databases, or interviews needs other review methods.

The JANUS case is a self-study; it cannot show how independent teams would use the workflow.

8 CONCLUSION

Unlike checksum and metadata tools, MIRROR maps each claim to the file behind it and keeps the wording that the user judged safe to publish. It works locally and without signatures, which suits the stage when claims are still being revised.

The next step is to give comparable bundles, with and without MIRROR support, to reviewers who had no hand in making them, and compare time and errors found.

AI ASSISTANCE DISCLOSURE

The author used OpenAI Codex and Anthropic Claude for planning, drafting, and editing text, local package checks, and review. AI coding tools also assisted with the reference implementation, its schemas and tests, under the author's direction; the author designed the

method and reviewed, tested, and released the code. Final responsibility rests with the author, who verified all claims, results, design decisions, and references.

DATA ACCESSIBILITY STATEMENT

The MIRROR software, tests, fixtures, and minimal public evidence package ([release/mirror-methodology-rc1/](#)) are archived as v0.1.1 on Zenodo under the Apache License 2.0: <https://doi.org/10.5281/zenodo.20463358> (13).

Private workspace paths, unpublished third-party material, raw and translated sentences, per-record reports, and operational notes were excluded from the JANUS release packet. The public evidence for this article is the MIRROR archive. The JANUS packet is unpublished, and the study's bundle stays reviewer-only.

ETHICS AND CONSENT

The case study uses the author's own local research workspace and software artifacts. It does not report human subjects' data, private third-party data, customer data, or employer confidential material.

FUNDING INFORMATION

This work received no external funding.

COMPETING INTERESTS

The author declares no competing interests for this article. The author has professional experience in public key infrastructure and digital identity systems; no employer, client, or private operational material is named or used in this article.

AUTHOR CONTRIBUTIONS

Anton Sokolov designed the MIRROR workflow, implemented or supervised the reference

implementation, ran the verifier checks, and prepared this manuscript.

AUTHOR AFFILIATIONS

Anton Sokolov  orcid.org/0000-0003-2452-7096
Researcher, Tyche Institute, Tallinn, Estonia

REFERENCES

1. **Sefton P, Ó Carragáin E, Soiland-Reyes S, Corcho O, Garijo D, Palma R**, et al. RO-Crate Metadata Specification 1.1.3. researchobject.org community; 2023 Apr 26. DOI: <https://doi.org/10.5281/zenodo.7867028>
2. **Groth P, Moreau L, editors.** *PROV-Overview: An Overview of the PROV Family of Documents*. W3C Working Group Note; 2013 Apr 30. Available from: <https://www.w3.org/TR/prov-overview/>
3. **Kunze J, Littman J, Madden E, Scancella J, Adams C.** The BagIt File Packaging Format (V1.0). RFC 8493. RFC Editor; 2018 Oct. DOI: <https://doi.org/10.17487/RFC8493>
4. **The CodeMeta Project.** CodeMeta: a shared, minimal metadata vocabulary for software; cited 2026 Aug 24. Available from: <https://codemeta.github.io/>
5. **Association for Computing Machinery.** Artifact Review and Badging Version 1.1. ACM Publications Policy; 2020 Aug 24. Available from: <https://www.acm.org/publications/policies/artifact-review-and-badging-current>
6. **Nüst D, Eglen SJ.** CODECHECK: an Open Science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility. *F1000Research*. 2021; 10:253 (version 2). DOI: <https://doi.org/10.12688/f1000research.51738.2>
7. **Clark T, Ciccarese PN, Goble CA.** Micropublications: a semantic model for claims, evidence, arguments and annotations in biomedical communications. *Journal of Biomedical Semantics*. 2014; 5(1):28. DOI: <https://doi.org/10.1186/2041-1480-5-28>
8. **Supply-chain Levels for Software Artifacts.** SLSA specification version 1.2. Linux Foundation; cited 2026 May 30. Available from: <https://slsa.dev/spec/v1.2/>.
9. **Torres-Arias S, Afzali H, Kuppusamy TK, Curtmola R, Cappos J.** in-toto: Providing farm-to-table guarantees for bits and bytes. In: *Proceedings of the 28th USENIX Security Symposium*. Santa Clara, CA; 2019. Available from: <https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias>
10. **Sigstore.** Overview. OpenSSF / Linux Foundation; cited 2026 May 30. Available from: <https://docs.sigstore.dev/>
11. **National Institute of Standards and Technology.** Secure Hash Standard (SHS), FIPS PUB 180-4. Gaithersburg, MD: NIST; 2015 Aug. DOI: <https://doi.org/10.6028/NIST.FIPS.180-4>
12. **Sokolov A.** EATF Agent Evidence Package toolkit. Zenodo; 2026. DOI: <https://doi.org/10.5281/zenodo.21763055>
13. **Sokolov A.** MIRROR: Local Evidence Bundles for Reproducible Research Software Review. Zenodo; 2026. DOI: <https://doi.org/10.5281/zenodo.20463358>

TO CITE THIS ARTICLE:

Sokolov A 2026 MIRROR: Local Evidence Bundles for Reproducible Research Software Review, with a JANUS Case Study. *Journal of Open Research Software*, 14: 65. DOI: <https://doi.org/10.5334/jors.753>

Submitted: 30 May 2026 **Accepted:** 30 September 2026 **Published:** 21 October 2026

COPYRIGHT:

© 2026 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <https://creativecommons.org/licenses/by/4.0/>.

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press.