LogIQ VDA5050 Analyzer — LogistiXpert

TECHNICAL NOTE

Verifying a Report Package

For the recipient of a report package

For anyone who is handed a LogIQ conformance report and has to decide whether to believe it.

As of 8 August 2026. Package format **1.0**.

What you were given

A **report package** is a single `.tar.gz` holding a conformance report and every file the report was made from. It exists because a verdict on its own is not evidence: without the log it was made from and the documents it was measured against, nobody can retrace how the verdict came about — and nobody can tell whether it was edited on the way to them.

```
logiq-report-42-2026-08-08/
  MANIFEST.json          what is inside, with the SHA-256 of every file
  MANIFEST.json.sig      Ed25519 signature over MANIFEST.json
  SHA256SUMS             the same hashes, for `sha256sum -c`
  README.txt             the short version of this document
  report/report.json     the report, machine-readable
  report/report.md       the report, readable
  source/<upload>        the file that was analysed, exactly as it arrived
  source/cleaned.json    the messages the checks actually read
  source/cleaned.ndjson  the same, one message per line
  references/rulesets.json     the custom rulesets it was judged with
  references/factsheet-*.json  the factsheets the capabilities were measured against
  references/lif-*.json        the layout the order paths were checked against
```

Only files that exist for that run are in the archive. A log analysed without a factsheet has no `references/factsheet-*.json`, and that is not a defect.

What checking it proves, and what it does not

Two separate questions, and both have to be answered yes:

1. **Is the manifest ours, and unchanged?** That is the signature. It is made with a private key held only by LogistiXpert, on the instances we operate.
2. **Does every file still match what the manifest says it was?** That is the hashes. The manifest records the SHA-256 of every file, so one signature covers the whole archive.

Together they mean: this is a report we produced, and nothing in it has been touched since.

They do **not** mean the report's verdict is correct for your fleet, that the log is a truthful recording of your traffic, or that the person who sent it to you is entitled to it. A signature says who produced a file, never whether the file is the right one to be looking at.

In a browser, in ten seconds

Open <https://logistixpert.de/verify> and drop the `.tar.gz` on the page.

The check runs in the page, on your own machine: the archive is unpacked, the signature checked and every hash recomputed by your browser. Nothing is uploaded. A report package holds your traffic, and you should not have to send it to us to find out whether we signed it — the page has no server side and stores nothing.

A browser too old for it says so rather than passing the package silently; then use one of the ways below.

The short way, on the command line

Download `verify_report_package.py` from <https://logistixpert.de/verify> and run it:

```
pip install cryptography
python3 verify_report_package.py logiq-report-42-2026-08-08.tar.gz
```

```
Package  logiq-report-42-2026-08-08.tar.gz
Report   42 · hall2_capture.ndjson
Made     2026-08-08T09:14:22Z by LogIQ VDA5050 Analyzer 0.2.0
Verdict  FAIL  score 93.73

✓ signature valid — issued by LogistiXpert
✓ 9 files, every hash matches the manifest

The package is authentic and complete.
```

It exits **0** only when both questions were answered yes. Other exit codes:

Code	Meaning
0	signed by LogistiXpert, and every file intact
1	something is wrong — do not rely on the report
2	not a readable report package
3	contents intact, but unsigned: the origin is not established

The script is one file with one dependency and needs no part of the Analyzer, because whoever checks a report is usually not the person who produced it.

By hand

1. The signature

Fetch the public key from <https://logistixpert.de/verify> — it is the same key for every package we sign — and check the manifest against it:

```
openssl pkeyutl -verify -pubin -inkey logiq-report-key.pem \
  -rawin -in MANIFEST.json -sigfile MANIFEST.json.sig
```

```
Signature Verified Successfully
```

This needs **OpenSSL 3.0 or newer**; Ed25519 is not in older versions. The `openssl` that ships with macOS is LibreSSL and will fail with *unsupported algorithm* — install a real one (`brew install openssl@3`) and call `/opt/homebrew/opt/openssl@3/bin/openssl`, or use the script above.

2. The contents

```
cd logiq-report-42-2026-08-08
sha256sum -c SHA256SUMS          # macOS: shasum -a 256 -c SHA256SUMS
```

Every line must say `OK`. `SHA256SUMS` deliberately does not list itself; the manifest carries *its* hash instead, so the chain from the signature down to the last file has no gap in it.

If you would rather not install anything

Five lines of Python, using only what a current Python has plus `cryptography`:

```
import base64, hashlib, json, pathlib
from cryptography.hazmat.primitives.asymmetric.ed25519 import Ed25519PublicKey

folder = pathlib.Path("logiq-report-42-2026-08-08")
key = Ed25519PublicKey.from_public_bytes(base64.b64decode(PUBLIC_KEY_FROM_WEBSITE))
key.verify((folder / "MANIFEST.json.sig").read_bytes(), (folder /
"MANIFEST.json").read_bytes())
for f in json.loads((folder / "MANIFEST.json").read_text())["files"]:
    assert hashlib.sha256((folder / f["path"]).read_bytes()).hexdigest() == f["sha256"],
f["path"]
print("authentic and complete")
```

An exception means the package failed. No output but the last line means it passed.

What the manifest says

`MANIFEST.json` is plain JSON and worth reading — it is the part that is signed, so everything in it is covered:

```
{
  "package_format": "1.0",
  "product": "LogIQ VDA5050 Analyzer",
  "tool_version": "0.2.0",
  "report_id": "42",
  "source_name": "hall2_capture.ndjson",
  "created_at": "2026-08-08T09:14:22Z",
  "signed_by": "LogistiXpert",
  "signature_algorithm": "Ed25519",
  "files": [{"path": "report/report.json", "sha256": "…", "bytes": 311015, "role":
"report"}],
  "run": {"target_version": "2.1.0", "verdict": "FAIL", "score": 93.73,
          "vehicles": ["Balyo/AGV-1"], "software_integrity": "hosted service · …"}
}
```

`run.software_integrity` is the same line the report carries: what guaranteed the software that produced the verdict. On a delivered installation it names the seal over the application files; on our hosted service it names us as the operator.

When something fails

`SIGNATURE DOES NOT MATCH` — the manifest was altered after signing, or the package was not signed with the key you checked against. Ask the sender for the package again, from the source, and check the key you used is the one on our website.

`altered: <file>` — the manifest is genuine but that file no longer matches it. Something changed the file after the package was made. The report cannot be relied on.

`missing: <file>` — a file the manifest lists is not in the archive. Often an archive that was unpacked and repacked; sometimes a file that was removed on purpose. Either way it is no longer the package we produced.

`unlisted: <file>` — a file in the archive that the manifest does not vouch for. Treat anything in it as unverified.

`SIGNATURE MISSING` — the manifest names a signer but the signature file is gone. This is not the same as an unsigned package: a package that was never signed says `"signed_by": null`. Here somebody removed it.

`unsigned` (exit 3) — the package came from an installation with no signing key, which is every on-premises installation. The hashes still show the archive is internally consistent; they cannot show where it came from. If you need a signed package, ask for the run to be repeated on the hosted service.

Keeping one

The archive is byte-for-byte reproducible: exporting the same report twice gives two identical files. Two people who were sent "the report" can compare checksums and see whether they are holding the same thing.

For an audit trail, keep the `.tar.gz` as it was received. Unpacking and repacking it changes the archive — the files inside stay verifiable, the archive as a whole does not.

Questions about a package you were sent: <vda5050@logistixpert.de>