TECHNISCHES MERKBLATT

Ein Berichtspaket prüfen

Für Empfänger eines Berichtspakets

Für alle, denen ein LogIQ-Konformitätsbericht vorgelegt wird und die entscheiden müssen, ob sie ihm glauben.

Stand: 16. August 2026. Paketformat **1.0**.

Was Sie bekommen haben

Ein **Berichtspaket** ist ein einzelnes `.tar.gz` und enthält einen Konformitätsbericht samt jeder Datei, aus der er entstanden ist. Es existiert, weil eine Bewertung für sich genommen kein Nachweis ist. Ohne das Log, aus dem sie stammt, und ohne die Dokumente, gegen die gemessen wurde, kann niemand nachvollziehen, wie es zustande kam. Niemand kann dann erkennen, ob unterwegs etwas geändert wurde.

```
logiq-report-42-2026-08-08/
  MANIFEST.json          was enthalten ist, mit dem SHA-256 jeder Datei
  MANIFEST.json.sig      Ed25519-Signatur über MANIFEST.json
  SHA256SUMS             dieselben Prüfsummen, für sha256sum -c
  README.txt             die Kurzfassung dieses Dokuments
  report/report.json     der Bericht, maschinenlesbar
  report/report.md       der Bericht, lesbar
  report/requirements-*.md  je Anforderungsdatei eines Partners, gegen die
                         gemessen wurde, die Antwort als eigenes Dokument
  report/dispositions.json  was die abnehmende Seite zu einzelnen Befunden
                         festgehalten hat, sofern etwas festgehalten wurde
  report/waivers.json    die stehenden Regeln je Check, die Befunde dieses
                         Laufs abgedeckt haben, sofern welche griffen
  source/<Upload>        die analysierte Datei, exakt wie sie ankam
  source/cleaned.json    die Nachrichten, die die Prüfungen wirklich gelesen
  source/cleaned.ndjson  haben, sofern die Bereinigung etwas verändert hat;
                         ein Mitschnitt ist bereits die bereinigte Form,
                         dort fehlen die beiden
  references/rulesets.json     die Custom Rulesets, mit denen bewertet wurde
  references/factsheet-*.json  die Factsheets, gegen die die Fähigkeiten gemessen wurden
  references/lif-*.json        das Layout, gegen das die Auftragswege geprüft wurden
```

Der `input_sha256` im Audit-Block des Berichts ist die SHA-256 von `source/<Upload>`, derselbe Wert, den das Manifest für diese Datei führt. Das ist die Zeile, die man nachrechnet, um sicher zu sein, dass man die Eingabe in der Hand hält, die bewertet wurde.

Enthalten ist nur, was es für diesen Lauf auch gibt. Ein Log, das ohne Factsheet analysiert wurde, hat kein `references/factsheet-*.json`. Das ist kein Mangel.

Was die Prüfung beweist und was nicht

Zwei getrennte Fragen, beide müssen mit Ja beantwortet sein:

1. **Stammt das Manifest von uns und ist es unverändert?** Das ist die Signatur. Sie entsteht mit einem privaten Schlüssel, den nur LogistiXpert hält, auf den von uns betriebenen Instanzen.
2. **Entspricht jede Datei noch dem, was das Manifest über sie sagt?** Das sind die Prüfsummen. Das Manifest führt den SHA-256 jeder Datei, deshalb deckt eine einzige Signatur das ganze Archiv ab.

Zusammen heißt das: Dies ist ein Bericht, den wir erzeugt haben, und seither wurde nichts daran angefasst.

Es heißt **nicht**, dass die Bewertung für Ihre Flotte richtig ist, dass das Log den Verkehr wahrheitsgemäß aufgezeichnet hat, oder dass die absendende Person berechtigt war, es weiterzugeben. Eine Signatur sagt, wer eine Datei erzeugt hat, nie ob es die richtige Datei ist.

Im Browser, in zehn Sekunden

https://logistixpert.com/de/pruefen öffnen und das `.tar.gz` auf die Seite ziehen.

Die Prüfung läuft in der Seite, auf Ihrem eigenen Rechner: Ihr Browser entpackt das Archiv, prüft die Signatur und rechnet jede Prüfsumme neu. Nichts wird hochgeladen. Ein Berichtspaket enthält Ihren Verkehr und Sie sollten es uns nicht schicken müssen, um zu erfahren, ob wir es signiert haben. Die Seite hat keine Serverseite und speichert nichts.

Ein zu alter Browser sagt das, statt das Paket stillschweigend durchzuwinken; dann einen der Wege unten nehmen.

Der kurze Weg auf der Kommandozeile

`verify_report_package.py` von https://logistixpert.com/verify laden und ausführen:

```
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 Fleets 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.
```

Der Exitcode ist **0** nur dann, wenn beide Fragen mit Ja beantwortet wurden. Die übrigen:

Code	Bedeutung
0	von LogistiXpert signiert, jede Datei unversehrt
1	etwas stimmt nicht, dem Bericht nicht vertrauen
2	kein lesbares Berichtspaket
3	Inhalt unversehrt, aber unsigniert: Herkunft nicht belegt

Das Skript ist eine Datei mit einer Abhängigkeit und braucht keinen Teil des Analyzers. Wer einen Bericht prüft, ist meist nicht die Person, die ihn erzeugt hat.

Von Hand

1. Die Signatur

Den öffentlichen Schlüssel von https://logistixpert.com/verify holen (es ist derselbe für jedes von uns signierte Paket) und das Manifest dagegen prüfen:

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

```
Signature Verified Successfully
```

Dafür ist **OpenSSL 3.0 oder neuer** nötig; ältere Versionen kennen Ed25519 nicht. Das mit macOS gelieferte `openssl` ist LibreSSL und scheitert mit *unsupported algorithm*. Dann ein echtes installieren (`brew install openssl@3`) und `/opt/homebrew/opt/openssl@3/bin/openssl` aufrufen oder das Skript oben verwenden.

2. Der Inhalt

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

Jede Zeile muss `OK` sagen. `SHA256SUMS` führt sich absichtlich nicht selbst auf; stattdessen trägt das Manifest *dessen* Prüfsumme, damit die Kette von der Signatur bis zur letzten Datei keine Lücke hat.

Wenn Sie nichts installieren möchten

Fünf Zeilen Python, nur mit `cryptography` zusätzlich:

```
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_VON_DER_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("authentisch und vollständig")
```

Eine Ausnahme bedeutet, das Paket ist durchgefallen. Nur die letzte Zeile bedeutet, es hat bestanden.

Was im Manifest steht

`MANIFEST.json` ist schlichtes JSON und lohnt das Lesen. Es ist der signierte Teil, alles darin ist also abgedeckt:

```
{
  "package_format": "1.0",
  "product": "LogIQ Fleets",
  "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` ist dieselbe Zeile, die auch der Bericht trägt: was die Software verbürgt, die die Bewertung erzeugt hat. Auf einer ausgelieferten Installation benennt sie das Siegel über den Anwendungsdateien, auf unserem gehosteten Dienst uns als Betreiber.

Wenn etwas fehlschlägt

`SIGNATURE DOES NOT MATCH` : das Manifest wurde nach dem Signieren geändert oder das Paket wurde nicht mit dem Schlüssel signiert, gegen den Sie geprüft haben. Das Paket erneut von der Quelle anfordern und prüfen, ob der verwendete Schlüssel der von unserer Website ist.

`altered: <Datei>` . Das Manifest ist echt, diese Datei passt aber nicht mehr dazu. Nach dem Packen hat etwas die Datei verändert. Dem Bericht ist nicht zu trauen.

`missing: <Datei>` . Eine im Manifest geführte Datei fehlt im Archiv. Oft ein Archiv, das ausgepackt und neu gepackt wurde; manchmal eine absichtlich entfernte Datei. So oder so ist es nicht mehr das von uns erzeugte Paket.

`unlisted: <Datei>`. Eine Datei im Archiv, für die das Manifest nicht bürgt. Alles darin gilt als ungeprüft.

`SIGNATURE MISSING`: das Manifest nennt einen Unterzeichner, die Signaturdatei fehlt aber. Das ist nicht dasselbe wie ein unsigniertes Paket: ein nie signiertes sagt `"signed_by": null`. Hier hat jemand sie entfernt.

`unsigned` (Exitcode 3): Das Paket stammt aus einer Installation ohne Signierschlüssel, also aus jeder dedizierten Installation. Die Prüfsummen zeigen weiterhin, dass das Archiv in sich stimmig ist; sie zeigen nicht, woher es kommt. Wird ein signiertes Paket gebraucht, den Lauf auf dem gehosteten Dienst wiederholen lassen.

Aufbewahren

Das Archiv ist ab Werkzeugversion **0.2.2** Byte für Byte reproduzierbar: Wird derselbe Bericht beim selben Bewertungsstand zweimal exportiert, entstehen zwei identische Dateien. Davor schrieb die gzip-Hülle die Uhrzeit des Exports in ihren Kopf, zwei Exporte desselben unveränderten Berichts unterschieden sich also in vier Bytes und ihre SHA-256 mit ihnen. Inhalt und Signatur über `MANIFEST.json` waren nie betroffen; ein älteres Paket prüft sich wie eh und je, nur die Prüfsumme des `.tar.gz` selbst war nicht stabil. Eine neue Bewertung ist das Einzige, was einen späteren Export verändert. Sie gehört zum Protokoll; Manifest und Signatur decken sie ab wie alles andere. Zwei Personen, denen „der Bericht" geschickt wurde, können Prüfsummen vergleichen und sehen, ob sie dasselbe in Händen halten.

Für einen Prüfpfad das `.tar.gz` so aufbewahren, wie es angekommen ist. Auspacken und neu packen verändert das Archiv. Die Dateien darin bleiben prüfbar, das Archiv als Ganzes nicht.

Fragen zu einem Paket, das Ihnen vorgelegt wurde: support@logiq-fleets.com