DEUTSCHER SPARKASSEN- UND GIROVERBAND · TEAM KISPA

Frag Dein Konto.

Fachliche und technische Dokumentation

Aufbau, Datenflüsse, Sicherheitsmodell und Prüfergebnis

Fassung 1.7.2, 8. Oktober 2026. Technischer Name des Projekts: `fints-mcp`.

Technischer Name: fints-mcp

Erzeugt am 8. Oktober 2026 aus TECHNIK.md

Frag Dein Konto. ist ein Experiment des Teams KISPA im DSGV. Die Software läuft auf dem Rechner des Nutzers, liest über FinTS/HBCI und hält keine Zugangsdaten außerhalb dieses Rechners. Dieses Dokument beschreibt die Fassung 1.7.2.

Inhalt

1. Zweck und Einordnung 4

2. Bausteine 5

3. Datenflüsse 6

4. Der Tresor 7

5. Die Einrichtungsseite und ihre Verteidigung 8

6. Bankanbindung 8

7. Das Überweisungsexperiment 9

8. Menüleisten-Symbol und Schlüsselbund 10

9. Bau und Lieferkette 11

10. Sicherheitsprüfung 1.3.0 11

Behoben (hoch) 12

Behoben (mittel) 12

Behoben (niedrig, Auswahl) 12

Bewusst offen 13

Was die Prüfer als gut gelöst festgehalten haben 13

Sicherheitsprüfung 1.3.1 14

1.3.2 14

1.3.3 14

1.3.4 14

1.3.5 15

1.3.6 15

Sicherheitsprüfung 1.4.0 15

1.4.1 16

1.4.2 16

1.4.3 16

1.4.4 16

1.4.5 17

1.4.6 17

1.4.7 17

1.5.0 17

1.5.1 18

1.5.2 18

1.5.3 18

1.5.4 19

1.6.0 20
1.7.0 21
1.7.1 26
1.7.2 27
Verfahren für künftige Fassungen 27
11. Bedrohungsmodell in Kurzform 28
12. Umgebung und Betrieb 29

Diese Dokumentation beschreibt, was die Software tut, wie sie gebaut ist und welche Sicherheitsentscheidungen ihr zugrunde liegen. Sie richtet sich an Fachleute, Prüfer und Entwickler. Die Bedienung steht in der Anleitung, das Sicherheitsmodell in Kurzform in der README.

1. Zweck und Einordnung

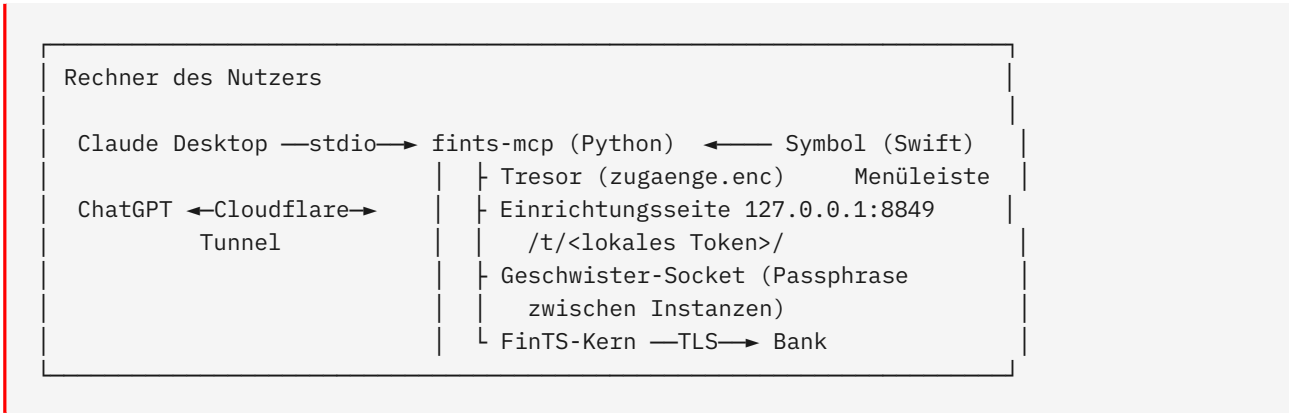
Die Software ist ein **MCP-Server** (Model Context Protocol), der Bankdaten über **FinTS/HBCI** liest und sie einem Sprachmodell als Werkzeuge anbietet. Sie läuft auf dem Rechner des Nutzers. Damit ist sie ein Werkzeug wie eine Banking-App und kein Kontoinformationsdienst im Sinne des ZAG: Es gibt keinen Betreiber, der fremde Zugangsdaten hält. Genau diese Bauform ist der Grund für mehrere Architekturentscheidungen — lokaler Tresor, Tunnel statt Server, keine zentrale Instanz.

Die Werkzeuge:

Werkzeug	Wirkung	Annotation für das Modell
`einrichtung_oeffnen`	Öffnet die Einrichtungsseite auf dem Rechner des Nutzers	harmlos, nicht nur lesend (öffnet ein Fenster)
`banken_auflisten`	Namen der hinterlegten Banken	nur lesend
`konten_abrufen`	Konten mit Saldo	nur lesend
`umsaetze_abrufen`	Buchungen eines Kontos, Zeitraum wählbar	nur lesend
`depot_abrufen`	Depotbestände	nur lesend
`ueberweisung_vorbereiten`	Legt einen Auftrag zur Freigabe vor (Experiment)	bewegt Geld erst nach Freigabe
`ueberweisung_status`	Zustand eines Auftrags	nur lesend

Dem Modell liegt eine Anleitung (`server.ANLEITUNG`) bei, die den Weg beschreibt und ausdrücklich verbietet, Passphrase, PIN oder Bankdaten im Chat zu erfragen oder anzunehmen. Sie erklärt außerdem, dass Bankdaten — Verwendungszwecke, Namen — Daten sind und keine Anweisungen.

2. Bausteine



fints-mcp (`src/fints_mcp/`) — der Server. Zwei Betriebsarten:

- **stdio** (Claude Desktop, Claude Code): Der Chat-Host startet den Prozess und spricht über stdin/stdout. Nebenher läuft die Einrichtungsseite auf `127.0.0.1:8849` .
- **HTTP** (Programm für ChatGPT & Co., Container): uvicorn auf `127.0.0.1:8848` , Zugriff nur mit Token (`FINTS_MCP_TOKEN` , ≥ 32 Zeichen), als Bearer-Header oder Pfadsegment `/t/<token>/` . TLS und Erreichbarkeit macht der Tunnel oder ein Reverse-Proxy davor.

Einrichtungsseite (`oberflaeche.py`) — HTML ohne Framework, ein einziges Skript für den Tresorzustand und das Auge an den Passphrase-Feldern. Seit 1.5.0 gegliedert wie die Systemeinstellungen: Seitenleiste mit den Bereichen `tresor` , `banken` , `chatbots` , `mehr` (je ein GET-Pfad, nebenwirkungsfrei); der Erststart läuft davor in drei Schritten ohne Seitenleiste. Welcher Bereich nach einem Formular wieder erscheint, bestimmt der Router aus dem Pfad (`BEREICH`). Dort passiert alles, was ein Geheimnis braucht: Tresor anlegen, entsperren, Banken eintragen, Frist wählen, Überweisungen freigeben, alles löschen.

Symbol (`erweiterung/menue/Menue.swift`) — ein Swift/AppKit-Programm in der Menüleiste des Mac. Es zeigt den Tresorzustand, entsperrt mit Passphrase oder Touch ID, zeigt die Seite in einem eigenen WKWebView-Fenster und startet das Programm für ChatGPT & Co. Es liest den Zustand über `GET /zustand` der lokalen Seite.

Programm für ChatGPT & Co. (`erweiterung/tunnel-bauen.sh` erzeugt es) — ein App-Bündel mit Python-Laufzeit, dem Server, `cloudflared` und dem Symbol. Es startet den Server im HTTP-Modus, erzeugt ein Token, öffnet einen Cloudflare-Schnelltunnel und nennt die Adresse.

Erweiterung für Claude Desktop (`erweiterung/mcpb-bauen.sh`) — ein `.mcpb` -Paket mit Python-Laufzeit, Server und Symbol.

Geschwister (`geschwister.py`) — Claude Desktop startet die Erweiterung zweimal. Damit der Nutzer nur einmal entsperrt, reichen die Prozesse die Passphrase über einen Unix-Socket (`/tmp/fints-mcp-<uid>/` , 700) weiter; der Wirt prüft die UID der Gegenseite (`SO_PEERCRED` / `LOCAL_PEERCRED`) und antwortet nur bei offenem Tresor. Unter Windows ein lokaler TCP-Port mit Geheimnis in einer Datei.

FinTS-Kern (`_kern.py` , `bank.py` , `_vendor/fints`) — übernommen aus BankingPal Pocket, mit Korrekturen für camt-Parser und Sparkassen-Eigenheiten. `bank.py` übersetzt zwischen dem Statusvertrag des Kerns (`ready` , `need_tan` , `error` , `vop_pending` , …) und dem, was der Server braucht.

Kundenseite und Vorführung (`kundenseite.py` , `auth_idp.py` , `demo.py`) — eine zentrale Vorführinstanz „Mustersparkasse“ mit OAuth-Anbindung und Testdaten. Sie zeigt, wie ein Institut den Zugang selbst anbieten könnte; sie hält keine echten Zugangsdaten und ist nicht Teil der ausgelieferten Pakete.

3. Datenflüsse

Einrichtung. Passphrase, Kennung und PIN gehen ausschließlich per POST an die lokale Seite (127.0.0.1). Im Tunnelbetrieb werden Aufrufe der Seite, die von außen kommen (Cloudflare-Kopfzeilen), auf die lokale Adresse umgelenkt, ohne den Formularinhalt zu verarbeiten — PIN und Passphrase passieren Cloudflares Rand nicht.

Abfrage. Das Modell ruft ein Werkzeug → der Server entnimmt dem offenen Tresor den Zugang → FinTS-Dialog über TLS zur Bank → Antwort wird in einfache Felder übersetzt (`bank.py`) → zurück an das Modell. Freitextfelder (Verwendungszweck, Gegenseite, Kontoname, Wertpapiername) werden von Steuer- und Richtungszeichen befreit und in der Länge begrenzt; sie sind Daten Dritter.

Was nie an das Modell geht: Passphrase, PIN, Token, die Adresse der Einrichtungsseite (sie trägt seit 1.3.0 immer ein Geheimnis), Bankleitzahl und FinTS-Adresse, Ausnahmetexte aus dem Bankdialog (nur die Fehlerklasse, der Text bleibt im Protokoll).

Was das Modell sieht und damit der Anbieter: Salden, Umsätze, Depotwerte, Kontonamen und IBANs — genau das, wonach der Nutzer gefragt hat.

Auf Platte (alle Dateien 600, Ordner 700, geschrieben über eine Zwischendatei mit `fsync` und `rename`):

Datei	Inhalt	Schutz
`~/.config/fints-mcp/zugaenge.enc`	Zugänge (Name, BLZ, Kennung, PIN, URL) als JSON	scrypt + AES-256-GCM
`~/.config/fints-mcp/einstellungen.json`	Frist; Schalter des Überweisungs-Experiments (`ueberweisung.an`)	Dateirechte
`~/.local/state/fints-mcp/seite.schluessel`	Token der lokalen Seite	Dateirechte
`~/.local/state/fints-mcp/fints_state_<blz>-<kennung>`	FinTS-Zustand: System-ID (Geräteerkennung), Bankparameter, Konten — keine PIN	Dateirechte
`~/.local/state/fints-mcp/auftraege/<id>.json`	Überweisungsaufträge	Dateirechte
`~/.local/state/fints-mcp/fints.log`	Protokoll (Rückmeldecodes, Fehler), Rotation ab 5 MB	Dateirechte
`~/.config/fints-mcp/token` (Tunnelprogramm)	HTTP-Token	Dateirechte
Schlüsselbund `de.bankingpal.fints-mcp`	gemerkte Passphrase (freiwillig)	macOS-Schlüsselbund

4. Der Tresor

`tresor.py` . Die Zugänge liegen als JSON, verschlüsselt mit **AES-256-GCM**; der Schlüssel kommt aus der Passphrase über **scrypt** (n = 2^{17}, r = 8, p = 1, 16 Byte Salz). Je Schreibvorgang frisches Salz und frische 12-Byte-Nonce, die Magie des Dateiformats geht als zusätzliche authentifizierte Daten mit. Ein manipulierter Kopf führt zu einem Entschlüsselungsfehler, nicht zu einem Downgrade.

Entsperrt liegt die Passphrase im Speicher des Prozesses — sie wird gebraucht, um Banken hinzuzufügen, ohne sie erneut zu verlangen. Nach der **Frist** (Vorgabe 60 Minuten, wählbar 30/60/120/480 oder unbegrenzt) ohne Nutzung schließt der Tresor. Nutzung heißt Arbeit mit den Zugängen (`zugaenge()`), nicht Anzeige (`einsehen()`): Bis 1.2.3 verlängerte jedes Laden der Übersicht die Frist, und ein `<img src="http://127.0.0.1:8849/">` auf einer fremden Seite hätte den Tresor unbegrenzt offen gehalten.

Fehlversuche bremsen exponentiell (1, 2, 4, 8 … Sekunden, Deckel 300). Ein Lesefehler zählt nicht als Vertipper; eine Passphrase, die von einer Schwesterinstanz kommt, ebenfalls nicht.

Python-Zeichenketten lassen sich nicht überschreiben; Passphrase und PINs bleiben bis zur Freigabe durch den Garbage Collector im Speicher. Das ist der im README benannte Preis dafür, dass der Server erreichbar ist. Wer den laufenden Prozess übernimmt, hat die entsperrten Zugänge — das lässt sich nicht wegprogrammieren, nur klein halten (Frist, `repr` ohne Geheimnisse, Hardened Runtime des Symbols).

5. Die Einrichtungsseite und ihre Verteidigung

Die Seite ist der einzige Ort, an dem Geheimnisse eingegeben werden. Entsprechend ist sie mehrfach abgesichert:

- **Token im Pfad** — seit 1.3.0 auch lokal: `http://127.0.0.1:8849/t/<token>/`. Das Token (32 Byte, `secrets.token_urlsafe`) liegt in `seite.schluessel` (600) und bleibt zwischen Starts gleich, damit Lesezeichen gelten. Ohne Token antwortet die Seite mit 401 — auch `/zustand`. Grund: Gegen fremde Webseiten reichte die Herkunftsprüfung, gegen einen anderen Benutzer desselben Rechners nicht; der setzt Kopfzeilen selbst und konnte den Tresor ohne Passphrase löschen oder die Frist aushebeln.
- **Host-Prüfung** gegen DNS-Rebinding: Nur `127.0.0.1`, `localhost`, `[::1]` als Host-Kopf; sonst 421.
- **Herkunftsprüfung** gegen CSRF: POSTs brauchen `Sec-Fetch-Site: same-origin` / `none` oder einen `Origin`, der zum Host passt; ohne beides 403. Ein leerer Host zählt nicht als passend.
- **Content-Security-Policy** `default-src 'none'`, Skript nur mit Nonce (die Nonce wird nur in das eigene Skript-Tag eingesetzt, nicht dokumentweit ersetzt), `frame-ancestors 'none'`, `form-action 'self'`, `base-uri 'none'`; dazu `X-Frame-Options: DENY`, `nosniff`, `Referrer-Policy: no-referrer`, `Cache-Control: no-store`.
- **Ausgabemaskierung** mit `html.escape(quote=True)` für alles, was nicht aus dem Modul stammt — Banknamen, Verwendungszwecke, Meldungen, Werte aus Umgebung und Zustandsdateien.
- **Löschen ohne Passphrase** (der Weg bei vergessener Passphrase) verlangt einen Einmal-Beleg aus der Rückfrage-Seite und das getippte Wort „löschen“.
- **Nebenwirkungsfreies GET**: Die Übersicht verlängert die Frist nicht mehr; die Passphrase von Geschwistern wird nur bei einer echten Navigation (`Sec-Fetch-Dest: document` oder ohne Kopf) geholt, nicht bei einem eingebetteten Bild.
- **Eingabeprüfung** der Benutzerkennung: keine Pfadzeichen, keine Platzhalter, höchstens 64 Zeichen — sie wird Teil eines Dateinamens.
- **PIN** nicht im Passwortspeicher des Browsers (`autocomplete="off"`); ein Zwischenstand mit PIN (Bankauswahl bei mehrdeutigem Namen) verfällt nach zehn Minuten.

Im HTTP-Modus liegt die Token-Prüfung ganz außen, vor der Seite und vor dem MCP-Endpunkt; das Token wird aus dem Zugriffsprotokoll von uvicorn und aus Fehlermeldungen geschwärzt; Vergleiche sind zeitkonstant und arbeiten auf Bytes (kein 500 bei Nicht-ASCII).

6. Bankanbindung

Transport. `requests` mit Standard-Zertifikats- und Hostnamenprüfung (kein `verify=False` im Projekt), Zeitlimit 20 s. Die Institutsliste (`institute.json`, 1.727 Einträge, alle `https://`) ist statisch und wird nicht zur Laufzeit geändert; eine freie URL-Eingabe gibt es nicht.

Demo Sparkasse. Ein Institut, das nicht in der Liste steht, sondern in `institute.py` ergänzt wird: BLZ 12345678, Adresse `https://testdaten.invalid/fints` (RFC-2606-reservierter Host, es kommt nie ein Dialog zustande). Ein Zugang mit dieser Adresse wird in `herkunft.py` und `einrichtung.py` an `testdaten.py` weitergereicht statt an die Bankschicht — Konten, Umsätze und Depots sind erzeugt, gesät und zueinander stimmig (Salden gerechnet, Kartenabrechnung = Kartenumsätze des Vormonats, Depotwert = Summe der Posten). Die Zugangsdaten `demo` / `12345` sind öffentlich; falsche scheitern im Probelauf wie bei einer echten Bank. Der Datensatz ist derselbe wie in der zentralen Vorführinstanz (`demo.py`).

PIN/TAN. Der Kern bedient nur Verfahren mit Freigabe in der App (decoupled). Verlangt die Bank eine TAN zum Eingeben, meldet er das und hört auf — es sitzt niemand vor dem Server, der tippt. Nach erfolgreicher Freigabe speichert er den Zustand samt System-ID, damit die Bank das Gerät wiedererkennt (typisch 90 Tage ohne zweite Freigabe). Bei Sparkassen wird der Dialog bewusst nicht mit HKEND beendet, weil das die Geräteerkennung verwirft.

Protokoll. Die FinTS-Bibliothek läuft ausdrücklich auf `WARNING` ; auf `DEBUG` würde sie Nachrichten samt PIN im HNSHA-Segment protokollieren. `fints.log` ist 600, rotiert ab 5 MB.

Prompt-Injection. Verwendungszwecke und Gegenseiten schreiben Dritte. Sie erreichen das Modell bereinigt (nur druckbare Zeichen, begrenzte Länge) und mit der Anleitung, sie als Daten zu behandeln. Der harte Riegel gegen eine gekaperte Sitzung liegt nicht im Text, sondern in der Architektur: Der Auftrag erscheint nur auf der Einrichtungsseite, und deren Adresse trägt ein Geheimnis, das das Modell nie bekommt — es kann vorbereiten, nicht absenden. Dahinter die Freigabe der Bank in der App.

7. Das Überweisungsexperiment

`ueberweisung.py` . Zwei Riegel, beide außerhalb des Chats:

1. **Einschalten** durch ein getipptes Wort auf der Seite (`ueberweisung.an` in `einstellungen.json`). Ohne den Schalter liefert das Werkzeug nur den Hinweis, wo man einschaltet. Bis 1.5.3 hing „ein“ am Hash eines Geheimworts; ein solcher Hash ohne Schalter gilt weiterhin als eingeschaltet, und beim nächsten Einschalten wird er aus der Datei entfernt.
2. **Freigeben** jedes Auftrags auf der Seite. Der Riegel ist die Seite selbst: Ihre Adresse trägt ein Geheimnis, das das Modell nicht kennt. Danach verlangt die Bank ihre Freigabe in der App. Ein Geheimwort je Freigabe gab es bis 1.7.0 als freiwillige Zusatzsperre; es ist entfernt (Frank 8.9.2026) — es schützte nur gegen den unbedachten eigenen Klick, und dafür steht jetzt der Beleg.

Der Beleg. Ein wartender Auftrag steht nicht als Zeile, sondern als Beleg auf der Seite (`oberflaeche._beleg`): Betrag groß, darunter beschriftet Empfänger, IBAN in Vierergruppen, Verwendungszweck, Bank und Quellkonto, dazu die Restzeit bis zum Verfall. Alles darin ist maskiert; die IBAN wird nur zur Anzeige gruppiert, gespeichert bleibt sie glatt. Erledigte Aufträge bleiben eine Zeile.

Prüfungen beim Anlegen: IBAN-Format und Mod-97-Prüfziffer, SEPA-Zeichensatz für Name und Zweck, Verwendungszweck Pflicht (mindestens zwei Zeichen — ein Auftrag ohne Zweck ist auf dem Auszug nicht wiederzuerkennen und ein Zeichen dafür, dass das Modell ihn nicht beim Nutzer erfragt hat), Betrag als `Decimal` mit zwei Nachkommastellen, Höchstbetrag 1000 EUR, höchstens drei offene

Aufträge, kein gleichlautender Auftrag (IBAN + Betrag) innerhalb von 24 Stunden in den Zuständen wartet/läuft/freigabe/ausgeführt/unklar. Quellkonto nur `[A-Z0-9]{1,34}`, Auftragskennung nur `[0-9a-f]{12}` (kein Path Traversal über die vom Modell gelieferte Kennung).

Beanspruchung. Vor dem Ausführen wird der Auftrag atomar von „wartet“ auf „läuft“ gesetzt — unter einer Prozesssperre und mit einer Marke im Dateisystem (`O_CREAT|O_EXCL`), die es genau einmal gibt, auch über die zwei Instanzen von Claude Desktop hinweg. Bis 1.2.3 setzte erst der Hintergrundfaden „läuft“; zwei Klicks oder zwei Tabs konnten denselben Auftrag zweimal zur Bank bringen.

Zustände. `wartet`, `laeuft`, `freigabe`, `ausgefuehrt`, `abgelehnt`, `fehler`, `verworfen`, `verfallen` (30 Minuten) und seit 1.3.0 `unklar`: Sobald die Bank den Auftrag erhalten hat (Empfängerprüfung läuft, App-Freigabe steht aus), ist ein späterer Fehler — meist die abgelaufene Freigabefrist — keine Garantie, dass nicht gebucht wurde. Der Nutzer wird gebeten, erst in der Banking-App nachzusehen. Der Höchstbetrag wird vor dem Senden noch einmal geprüft.

Namensabgleich (Verification of Payee). Die Bank meldet ein strukturiertes Ergebnis (RCVC passt, RVMC nahe dran, RVNM passt nicht, RVNA nicht prüfbar). Automatisch bestätigt wird nur RCVC; alles andere führt im Experiment zur Ablehnung. Bis 1.2.3 entschied eine Stichwortsuche im Banktext, die „verifiziert, Name weicht leicht ab“ als Treffer gelesen hätte. Auch nach der App-Freigabe wird ein abweichendes Ergebnis nicht mehr blind bestätigt, sondern als `unklar` gemeldet.

8. Menüleisten-Symbol und Schlüsselbund

`Menue.swift`, signiert mit fester Kennung `de.bankingpal.fints-mcp.tunnel`, Team `2T96J9RV7P`, Hardened Runtime ohne Lockerungen, beglaubigt.

- **Passphrase-Dialog**: Feld mit Auge (zwei Felder, verdeckt und offen, tauschen den Inhalt), Tresor-Symbol statt des generischen Ordners, Haken „Im Schlüsselbund merken“.
- **Schlüsselbund**: Der Eintrag wird bevorzugt im Data-Protection-Schlüsselbund mit `SecAccessControl(.userPresence)` angelegt, sodass das System die Herausgabe erst nach Touch ID, Apple Watch oder Anmeldepasswort erlaubt. Das setzt ein `keychain-access-groups`-Entitlement voraus, das ohne Provisioning-Profil nicht zu haben ist; in diesem Fall (Fehler –34018) fällt das Programm auf den Login-Schlüsselbund zurück und verlangt die Authentifizierung selbst über `LAContext`, bevor es den Eintrag liest. Siehe Abschnitt 10, offene Punkte.
- **Fenster** (WKWebView, nicht-persistenter Datenspeicher): Navigation nur zu `127.0.0.1` / `localhost` auf Ports laufender Instanzen; fremde `http(s)`-Adressen gehen an den Browser, andere Schemata werden verworfen; bei Ladefehler eine Meldung im Fenster, keine Adresse mit Token im Browser.
- **Programm starten**: Das Symbol sucht das Programm für ChatGPT & Co. nur in `/Applications` und `~/Applications` und prüft vor jedem Start die Signatur des Bündels gegen Team und Kennung (`SecStaticCodeCheckValidity`). Ein unsigniertes oder fremd signiertes Bündel wird nicht gestartet und im Menü als solches gezeigt.
- **Dateien** im Zustandsordner (`menue.pid`, `menue.adresse` mit der Tunnel-Adresse) entstehen mit 600, der Ordner mit 700. Beendet wird nur ein Prozess, der nachweislich zu uns gehört (Pfad und Argumente).

9. Bau und Lieferkette

`erweiterung/gemeinsam.sh`, `mcpb-bauen.sh`, `tunnel-bauen.sh`.

- **Herkunft der Bestandteile**: Python-Laufzeit (python-build-standalone) und `cloudflared` werden mit festgeschriebener SHA-256 geprüft; eine Abweichung bricht den Bau ab.
- **Abhängigkeiten**: `requirements.lock` mit Fassungen und Hashes aller Wheels für die drei Ziele, installiert mit `pip install --require-hashes`. Vor dem Bau läuft `pip-audit` gegen das Lockfile; bekannte Schwachstellen brechen ab (`FINTS_MCP_OHNE_AUDIT=1` nur bewusst, offline).
- **Signatur**: Bibliotheken ohne Entitlements; die lockeren Rechte (`disable-library-validation`, `allow-unsigned-executable-memory`, `allow-jit`), die der Python-Interpreter braucht, bekommen nur `runtime/bin/*` und `cloudflared`. Das Hauptprogramm (das Symbol) wird mit dem Bündel signiert, mit voller Hardened Runtime. `codesign --verify --deep --strict`, Beglaubigung bei Apple mit geheftetem Ticket.
- **Integrität der Downloads**: Der Bau schreibt die SHA-256 jedes Pakets in `docs/fassungen.json` und `SHA256SUMS-<Fassung>.txt`; die Fassungsseite zeigt sie an.
- **Windows**: Das ZIP enthält Python-Laufzeit, Server, `cloudflared.exe` und ein Startskript; es ist nicht signiert (siehe offene Punkte).

Betrieb der Vorführinstanz (`deploy/`): systemd-Unit mit `NoNewPrivileges`, `ProtectSystem=strict`, `ProtectHome`; Docker-Ports an 127.0.0.1; Kundenseite mit Sicherheitsköpfen, CSP mit Nonce, Herkunftsprüfung, Sitzungskeks `SameSite=Strict`.

10. Sicherheitsprüfung 1.3.0

Am 6. September 2026 wurden alle Teile durchgesehen — Server-Kern, Web-Oberflächen, Menüleiste mit Bau und Auslieferung, Bankanbindung — mit der Fragestellung, ob die Anwendung so angeboten werden kann, dass ein Sicherheitsfachmann keinen Einwand hätte. Ergebnis: **kein kritischer Befund**, vier hohe, neun mittlere, eine Reihe niedriger und Hinweise. Alle hohen und mittleren sind behoben, die niedrigen bis auf die unten genannten ebenfalls.

Behoben (hoch)

Befund	Wo	Korrektur
Das Token stand in Werkzeugantworten und Fehlermeldungen an das Modell (Adresse der Seite)	`server.py`	Keine Adresse mehr an das Modell; Seite öffnet sich selbst oder über das Symbol
Das Symbol führte Startskripte aus Bündeln in `~/Downloads` / `~/Desktop` ohne Signaturprüfung aus	`Menue.swift`	Nur `/Applications` , `~/Applications` ; Signaturprüfung gegen Team und Kennung vor jedem Start
Tunnel-Adresse mit Token in weltlesbarer Datei (`menue.adresse` , 644)	`Menue.swift`	600 von Anfang an, Ordner 700
Doppelte Ausführung einer Überweisung durch Wettlauf zwischen Prüfung und Faden	`ueberweisung.py` , `oberflaeche.py`	Atomare Beanspruchung mit Dateimarke vor dem Start

Behoben (mittel)

- Lokale Seite ohne Geheimnis gegenüber anderen Benutzern des Rechners → Token im Pfad, wie im Tunnelbetrieb.
- `GET /` verlängerte die Frist und holte die Passphrase von Geschwistern → `einsehen()` ohne Verlängerung, Geschwister nur bei Navigation.
- Zustand „fehlgeschlagen“ nach Zeitablauf, obwohl die Bank noch buchen konnte → Zustand „unklar“, Doppelprüfung beim Anlegen.
- Namensabgleich per Stichwortsuche → Ergebniscode, nur RCVC automatisch.
- FinTS-Zustandsdatei, Protokoll, Zustandsordner weltlesbar → 600/700 von der ersten Sekunde, Rotation.
- Banktexte ungefiltert an das Modell → bereinigt und begrenzt.
- Kundenseite ohne Sicherheitsköpfe, ohne Maskierung, POSTs nur durch `SameSite=Lax` geschützt → CSP mit Nonce, Köpfe, `_e()` , Herkunftsprüfung, `Strict` .
- Schlüsselbund-Eintrag ohne Zugriffskontrolle → Data-Protection-Schlüsselbund mit `userPresence` , Rückfall siehe unten.
- Zu weite Entitlements auf dem Symbol → nur noch für Python und cloudflared.
- Downloads ohne Prüfsummen, Fremdartefakte ohne Hash, Abhängigkeiten ohne Lockfile → SHA-256 überall, Lockfile mit Hashes, `pip-audit` .

Behoben (niedrig, Auswahl)

Zeitkonstante Vergleiche auf Bytes (kein 500 bei Nicht-ASCII); Socketordner in `/tmp` wird auf Eigentümer und Rechte geprüft, fremde Sockets werden nicht gefragt, Schwester-Passphrase zählt nicht als Fehlversuch; `frist_setzen` überschreibt das Geheimwort nicht mehr; `glob.escape` für Kennungen; Benutzerkennung ohne Pfadzeichen; PIN nicht im Browser-Passwortspeicher; `_e()` auch für Werte aus

Umgebung und Zustandsdateien; Nonce nur im eigenen Skript; Höchstbetrag vor dem Senden; IBAN-Prüfziffer; SEPA-Zeichensatz; höchstens drei offene Aufträge; `Tresor.__repr__` ohne Geheimnisse; Untergrenzen `requests ≥ 2.32.4`, `cryptography ≥ 44`; WKWebView-Navigation eingeschränkt; kein `kill` an fremde PIDs; `umask 077` und `mkdir -m 700` im Startskript; Bankliste nennt nur Namen; Ausnahmetexte bleiben im Protokoll; Anfragen mit unbekannter `kid` lösen höchstens alle 30 s einen JWKS-Abruf aus.

Bewusst offen

Punkt	Warum offen	Stand
Schlüsselbund mit systemseitiger Zugriffskontrolle	braucht `keychain-access-groups`-Entitlement, also ein Provisioning-Profil für Developer ID	Code ist da, greift automatisch, sobald das Entitlement signiert wird; bis dahin Rückfall: Login-Schlüsselbund, Touch ID über `LAContext` vor dem Lesen
`.mcpb` signieren	Claude Desktop lehnt signierte Bündel derzeit ab (mcpb Issue #278)	SHA-256 auf der Seite als Ersatz
Windows-Paket signieren	braucht Authenticode-Zertifikat	SHA-256 auf der Seite
Website: CSP, `Content-Disposition: attachment`, je Tester ein Passwort, Upload ohne root	liegt im Caddy des Servers, nicht im Repo	in `deploy/website.md` als Anleitung hinterlegt
Intel-Mac: `cryptography` 46.0.3 mit gemeldeten Lücken	seit Fassung 47 keine x86_64-Räder für macOS; die Lücken (PKCS#7/CMS, X.509-Pfadprüfung und Namensbeschränkungen, EC-Schlüssel laden, nicht zusammenhängende Puffer, eingebautes OpenSSL in PKCS7/CMS/OCSP/QUIC) betreffen keinen Pfad, den das Projekt nutzt — es braucht nur AES-GCM für den Tresor, TLS macht die ssl-Bibliothek des Interpreters	genau diese Kennungen sind in der Schwachstellenprüfung nur für die Intel-Datei ausgenommen; jede neue Meldung bricht den Bau ab
Fehlversuchszähler je Prozess, nicht persistent	mit Token im Pfad kein Angriffsweg mehr von außen	bleibt

Was die Prüfer als gut gelöst festgehalten haben

Kryptographie des Tresors; Trennung der Kanäle (Geheimnisse nur über die Seite, nie über den Chat, mit Absage in jeder Antwort an das Modell); Token ganz außen, geschwärzt im Protokoll; Bindung ausschließlich an 127.0.0.1; CSRF- und DNS-Rebinding-Schutz mit Tests; Peer-UID am Geschwister-Socket; zwei Riegel vor jeder Geldbewegung; fail-closed im Kern (keine TAN-Eingabe → nicht ausgeführt; unbekanntes VOP-Ergebnis → Warnung); Developer-ID-Signatur und Beglaubigung; keine Geheimnisse in Bauskripten oder im Repo.

Sicherheitsprüfung 1.3.1

Geprüft am 6. September 2026 wurde der Diff seit 1.3.0 (Demo Sparkasse, Fenster-Hinweise, Antwortform) durch zwei unabhängige Durchgänge — ein allgemeiner Code-Security-Review und eine gezielte AppSec-Prüfung mit Blick auf Einrichtungsseite, Institutsliste, Herkunft-Weiche und Menüleistenprogramm. **Kein hoher Befund.** Zwei mittlere, beide vor dem Bau behoben: (1) Der Knopf „Fenster schließen“ und jeder andere Verweis auf eine unserer Seiten wurde an den Browser gereicht, sobald die Portliste des Symbols leer war (Programm angehalten, Fenster noch offen) — die Adresse trägt das Token und wäre im Browserverlauf gelandet; jetzt bleiben Loopback-Adressen immer im Fenster, der Knopf prüft nur den Rechner, nicht den Port. (2) Der Überweisungspfad kannte die Demo nicht; ein freigegebener Auftrag wäre in einen Verbindungsversuch gegen die erfundene Adresse gelaufen. Jetzt lehnt `ueberweisung_vorbereiten` Demo-Konten ab. Niedrige Befunde, ebenfalls behoben: Merker-Dateien des Symbols mit 0600/0700 statt Standardrechten; Kontokennungen werden der Demo nur bei exakter Übereinstimmung zugeordnet (vorher Endungsvergleich, der echte Nummern hätte fangen können), unbekannte Kennungen bekommen eine Auskunft statt erfundener Umsätze; die Demo steht in Suchtreffern zuletzt; jedes Konto, jede Buchung und jeder Depotposten der Demo trägt `demo: true`, und die Anleitung weist das Modell an, das zu sagen; die Kürze-Regel der Antwortform nimmt Sicherheitshinweise (Befehle in Bankdaten) ausdrücklich aus; Demo-Konstanten werden im HTML maskiert; der Website-Teaser nennt die Zugangsdaten nur mit dem Wort „erfunden“ im selben Satz. Als gut gelöst festgehalten: Die FinTS-Adresse eines Zugangs stammt immer aus der Institutsliste, nie aus dem Formular; falsche Demo-Zugangsdaten scheitern vor dem Speichern; die Ein-Klick-Formulare laufen durch dieselbe CSRF-Prüfung und `form-action 'self'`. Regeln festgehalten in `tests/test_sicherheit_1_3_1.py`.

1.3.2

Nur Anzeige: Menüleisten-Beschriftung „gesperrt“, zustandsabhängige Fußzeile im Fenster, Demo-Karte auf der Einrichtungsseite entfernt (Zugangsdaten stattdessen vorausgefüllt). Keine Änderung an Tresor, Token, Bankanbindung, Werkzeugen oder Bau — die Prüfung von 1.3.1 gilt unverändert.

1.3.3

Nur die Anleitung an das Modell (Antwortform, keine Adressen/Dateinamen, keine Banken aus dem Gedächtnis) und ein Wort in einer Werkzeugbeschreibung. Kein Code am Tresor, an der Seite oder der Bankanbindung — die Prüfung von 1.3.1 gilt unverändert.

1.3.4

Das Menüleistenprogramm öffnet bei einer neuen, noch nie kopierten Tunnel-Adresse einmal das Fenster am Abschnitt „ChatGPT verbinden“ (Merker `menue.adresse.gezeigt`, 0600). Es zeigt nur eine ohnehin erlaubte lokale Seite mit Anker; kein neuer Pfad, kein neues Geheimnis. Die Prüfung von 1.3.1 gilt unverändert.

1.3.5

Neu: „In ChatGPT eintragen“ (`chatgpt_lokal.py`). Die ChatGPT-App liest dieselbe MCP-Konfiguration wie Codex (`~/.codex/config.toml`) und startet dort eingetragene STDIO-Server selbst; der Knopf auf der Einrichtungsseite schreibt einen eigenen, markierten Abschnitt `[mcp_servers.frag_dein_konto]` mit dem Pfad zu Interpreter und Server dieser Installation. Gezielt betrachtet, weil eine fremde Datei beschrieben wird: nur der eigene Abschnitt wird angefasst (Marken plus Kopfzeilen-Erkennung), vorher Sicherung `config.toml.bak-frag-dein-konto` (0600), Schreiben atomar über eine Temporärdatei im selben Ordner, davor Prüfung mit `tomllib` , dass das Ergebnis gültig ist — eine kaputte Datei nähme ChatGPT alle Server. Der Eintrag enthält kein Geheimnis (Pfade und Umgebungsvariablen, keine Token). Verweigert wird er, wenn das Programm aus dem AppTranslocation-Ordner läuft (Downloads) oder keinen eigenen Interpreter hat, weil der Pfad dann nicht hält. Die POST-Pfade laufen hinter CSRF-Prüfung und offenem Tresor wie alle anderen. Die Prüfung von 1.3.1 gilt im Übrigen unverändert.

1.3.6

Gemeinsame Tresor-Uhr: `tresor-uhr.json` (0600) neben der Zugangsdatei hält den Zeitpunkt der letzten Nutzung und des letzten Sperrens für alle Prozesse; jede Nutzung verlängert für alle, ein Sperren sperrt alle. Die Datei enthält kein Geheimnis; der entschlüsselte Inhalt bleibt wie bisher nur im Speicher des jeweiligen Prozesses. Kein neuer Netzwerkpfad, keine Änderung an Passphrase-Weitergabe oder Seite. Die Prüfung von 1.3.1 gilt unverändert.

Sicherheitsprüfung 1.4.0

Geprüft am 6. September 2026: der Diff seit 1.3.1 mit Schwerpunkt auf den neuen Schreibvorgängen in fremde Konfigurationsdateien (`chatgpt_lokal.py` , `claude_lokal.py`), der gemeinsamen Tresor-Uhr, dem Menüleistenprogramm und dem Upload-Skript — zwei unabhängige Durchgänge. **Ein hoher Befund**, vor dem Bau behoben: `zuruecksetzen()` überschrieb die neue Methode `_offen_bis` mit einer Zahl; nach „Alles von diesem Rechner entfernen“ warf jede Anfrage einen Fehler, die Seite war bis zum Neustart unbenutzbar. Mittlere Befunde, alle behoben: die vier Schreib-Endpunkte sind nur noch auf der lokalen Seite erreichbar, nicht über Tunnel- oder Vorführinstanz; der Programmpfad wird vor dem dauerhaften Eintrag geprüft — Positivliste der Orte (Programme, ~/Applications, Claude-Erweiterungsordner; nicht Downloads, DMG, Netzlaufwerk, Temporärordner), keine Fremdschreibrechte, und bei .app-Bündeln die Signatur mit unserer Team-Kennung (`codesign --verify -R`); der Eintrag erbt keine Umgebungsvariablen des laufenden Prozesses mehr, sondern schreibt die kanonischen Pfade; die Sicherungskopie der fremden Datei liegt jetzt in unserem Zustandsordner, 0600 von der ersten Sekunde, und wird von „Alles entfernen“ mitgenommen; „Alles entfernen“ nimmt auch die Einträge in Claude und ChatGPT heraus und sperrt über die gemeinsame Uhr die Geschwisterprozesse; die gemeinsame Uhr klemmt Zeitpunkte auf „jetzt“, verwirft NaN, und schreibt nur vorwärts (ein gleichzeitiges Verlängern kann ein Sperren nicht mehr verschlucken); das Upload-Skript prüft Fassungsnummern gegen X.Y.Z, wertet die Prüfsummenkontrolle über Exit-Code und Zeilenzahl aus und löscht nichts mehr (Frank: alte Fassungen bleiben herunterladbar). Niedrige Befunde behoben: `ohne_abschnitt` stürzt bei vertauschten Marken nicht mehr, kein `chmod` mehr auf fremde Ordner, `lage()` fängt Dekodierfehler, der irreführende Kommentar an `PFADE` ist korrigiert. Bewusst offen: (1) Ein Prozess desselben Nutzers kann über `tresor-uhr.json` einen bereits offenen Tresor bis zur Frist

verlängert halten — dieselbe Vertrauensgrenze wie der Geschwister-Socket, der die Passphrase herausgibt; keine neue Privilegienstufe. (2) Der ChatGPT-Eintrag aus der Claude-Erweiterung heraus zeigt in den Erweiterungsordner, den Claude bei Updates ersetzt — der Pfad trägt unsere Erweiterungskennung und wird nur von unserem eigenen Paket besetzt; nach einer Deinstallation scheitert der Start folgenlos, der Eintrag geht über „Aus ChatGPT entfernen“ oder „Alles entfernen“ weg. (3) Das Menü vertraut den Portdateien im 0700-Zustandsordner (gleicher Nutzer). (4) `fassungen.html` verweist auf Fassungen, die auf dem Server fehlen können. Regeln festgehalten in `tests/test_chatgpt_lokal.py`, `tests/test_claude_lokal.py`, `tests/test_frist.py`.

1.4.1

Nur Fehlerbehebung ohne neue Angriffsfläche: Das Tunnel-Startskript endet bei INT/TERM statt nach dem Aufräumen weiterzuwachen (falsche Meldung „Der Server ist abgestürzt“ beim Beenden über das Menü), die Wache prüft zuvor die PID-Datei; die gesperrte Ansicht im Fenster bietet „Passphrase vergessen? → Tresor löschen“ wie die Browser-Seite; neues Entwicklerskript `deploy/jungfraeulich.sh` (nur lokal, Vorschau, `--wirklich`). Die Prüfung von 1.4.0 gilt unverändert.

1.4.2

Geführter Erststart: Passphrase → Bank → Chatbot wählen (Claude Desktop, ChatGPT-App) → Einrichten. Neuer POST-Pfad `/chatbots-einrichten` (nur lokal, hinter Tresor-Riegel und CSRF-Prüfung wie die Einzelknöpfe; nutzt dieselben `eintragen()`-Funktionen mit allen Prüfungen aus 1.4.0). Tunnel-Skript mit atomarem Startriegel (`mkdir`) gegen Doppelstart in den ersten Sekunden. Keine neue Angriffsfläche; die Prüfung von 1.4.0 gilt.

1.4.3

Nur Menüleistenprogramm: Doppelklick auf das Bündel startet das Programm selbst; ohne Tresor öffnet sich sofort das Fenster mit einer eingebauten Warteseite (`loadHTMLString`, kein Netz), die in Schritt 1 übergeht; nach 30 s Hinweis mit Knopf „Noch einmal starten“ (Pfad `neu-starten`, nur Loopback, wird im Fenster abgefangen wie `fenster-schliessen`). Keine neue Angriffsfläche; die Prüfung von 1.4.0 gilt.

1.4.4

Die Schreib-Endpunkte für Claude/ChatGPT gelten jetzt für „lokale Seite ODER Tunnelprogramm“ (`FINTS_MCP_SEITE_LOKAL`); die zentrale Vorführinstanz bleibt gesperrt, Anfragen von außen sind im Tunnelprogramm vorher umgeleitet — 1.4.3 gab im Fenster des Programms „Not Found“. Der Tunnel ist eine Option: Schalter `tunnel` in `einstellungen.json`, gesetzt über Schritt 3 der Einrichtung oder `/tunnel-ein` / `/tunnel-aus` (nur Tunnelprogramm, offener Tresor, CSRF-Prüfung); das Startskript liest ihn beim Start und in der Wache und startet/beendet cloudflared entsprechend; Bestandsnutzer mit Adressdatei behalten den Tunnel. Kein neues Geheimnis, keine neue Angriffsfläche — ein ausgeschalteter Tunnel ist weniger Angriffsfläche als vorher. Die Prüfung von 1.4.0 gilt.

1.4.5

Das Programm im HTTP-Modus (Tunnelprogramm) startet jetzt ebenfalls den Geschwister-Wirt (`geschwister.wirt_starten`), damit die im Fenster eingegebene Passphrase die Claude-/ChatGPT-Prozesse erreicht — vorher blieben sie zu und Claude meldete „gesperrt“. Der Wirt ist derselbe wie in den STDIO-Prozessen (Unix-Socket 0600 im 0700-Ordner, Peer-UID-Prüfung); keine neue Angriffsfläche. Sonst: Programmname ohne Fassung, „Demo“ ohne Zugangsdaten (nur für das Demo-Institut, echte Banken unverändert), Texte, farbige Tunnel-Karte, keine Prozesszählung. Die Prüfung von 1.4.0 gilt.

1.4.6

Nur Anordnung der Übersichtsseite (Statuszeile, Banken, Chatbots-Karte, eingeklappte Einstellungen). Dieselben Formulare und Pfade, keine neuen Endpunkte; die Prüfung von 1.4.0 gilt.

1.4.7

Nur Anordnung von Schritt 3 und der Chatbots-Karte (drei Blöcke mit eigenen Knöpfen; `/chatbots-einrichten` ist nur noch der Abschluss des Erststarts). Dieselben Endpunkte und Prüfungen; die Prüfung von 1.4.0 gilt.

1.5.0

Geprüft am 6. September 2026 wurde der Diff seit 1.4.7 (Einrichtungsseite mit Seitenleiste und den vier Bereichen `tresor` , `banken` , `chatbots` , `mehr` ; Erststart in drei Schritten mit Fortschrittsanzeige; Bank in zwei Schritten; „Zurücksetzen“ unter „Mehr“; Frist als Radioknöpfe; Folge-Skript lädt den aktuellen Bereich; Menüleistenprogramm öffnet den ChatGPT-Hinweis im Bereich Chatbots) durch zwei unabhängige Durchgänge — ein allgemeiner Code-Security-Review und eine gezielte AppSec-Prüfung mit Blick auf Nebenwirkungen der neuen GET-Pfade, den Weg zum Zurücksetzen, die HTML-Ausgabe aller Ansichten, das Folge-Skript samt CSP und die Adressbildung im Fenster. **Kein hoher, kein mittlerer Befund.** Als gut gelöst festgehalten: Die neuen Pfade stehen in `PFADE` und laufen durch dieselbe Kette (Host, Herkunft, Tunnel-Umleitung, Tresorzustand); sie lesen nur (`einsehen()` , nicht `zugaenge()`); der Beleg des Zurücksetzens ist einmalig und wird zeitkonstant verglichen, beide Knöpfe („Mehr“ und „Passphrase vergessen?“) führen auf denselben Zweig; alle fremden Werte in den Ansichten sind maskiert, `href` / `action` /Meta-Refresh-Ziele sind Konstanten; die Adresse im Fenster bleibt Loopback und gelangt nicht in den Browser. Niedrige Befunde, alle vor dem Bau behoben: (1) `/hinzufuegen` löschte den ganzen Zwischenzustand samt Erststart — eine echte Bank landete danach in der Übersicht statt bei Schritt 3; jetzt wird nur die gemerkte Bank vergessen. (2) Die nach `/suchen` gemerkte Bank hing ohne Verfall im Speicher und zeigte dauerhaft das Zugangsdaten-Formular; jetzt verfällt sie nach zehn Minuten, und die Seite hat einen Abbrechen-Knopf (`/bank-abbrechen`). (3) Läuft die Frist zwischen Prüfung und Rendern ab, kommt die Entsperrseite statt eines Fehlers 500. (4) `_block` , `_betrag` (Währung und Rückfall aus Bankdaten) und die Statuszeile von `_rahmen` maskieren jetzt selbst; die Nonce geht an das letzte Skript-Tag des Dokuments (unseres), nicht an das erste; das Folge-Skript bindet das Ziel an `location.origin` ; das Zählen beim Zurücksetzen nutzt `einsehen()` . Zur Kenntnis genommen: Eine echte Navigation auf einen Prozess mit geschlossenem Speicherzustand

holt die Passphrase von den Geschwistern und setzt die gemeinsame Uhr — absichtlich, durch `_ist_navigation` gegen eingebettete Aufrufe gesichert, seit 1.4.5 unverändert. Regeln festgehalten in `tests/test_sicherheit_1_5_0.py`.

1.5.1

Nur Anzeige: Fenster-Hinweis an zwei Stellen statt unter jedem Blatt, lesbare Knopfschrift, neutrale Einrichten-Knöpfe. Keine Änderung an Pfaden, Formularen, Tresor, Token oder Bau — die Prüfung von 1.5.0 gilt. Der Bau bricht seit diesem Stand ohne Beglaubigung ab, statt unbeglaubigt zu packen (1.5.0 war im ersten Bau unbeglaubigt ausgeliefert und wurde neu gebaut).

1.5.2

Geprüft am 7. September 2026 wurde der Diff seit 1.5.1 (Fassungsprüfung beim Start, „Zurücksetzen“ macht den Mac jungfräulich, Neustart eines nicht antwortenden Programms) mit der Brille eines AppSec-Prüfers — gezielt, weil dabei erstmals Prozesse beendet und Ordner, Schlüsselbund-Einträge und das Programm selbst gelöscht werden. **Ein hoher Befund, vor dem Bau behoben:** Bei eingeschaltetem Autostart hätte `launchctl bootout` mitten im Zurücksetzen die ganze launchd-Prozessgruppe abgeräumt — Menü, Startskript und diesen Server. Die Antwort wäre nie beim Nutzer angekommen, und alles danach (Schlüsselbund, Ordner, Programm) wäre liegen geblieben: ein halb aufgeräumter Rechner nach einem bestätigten „löschen“. Jetzt entfernt `jungfraeulich()` nur die Agent-Datei; das `bootout` macht `spaeter_beenden()` nach der Antwort und nach den Signalen. Mittlere Befunde, ebenfalls behoben: (1) Das Startskript beendete die PID aus `tunnel.pid` ohne zu prüfen, wem sie gehört — nach einem Absturz kann sie einen fremden Prozess des Nutzers benennen; jetzt nur nach Ziffernprüfung und wenn die Kommandozeile unser Startskript nennt, sonst wird die Datei als Leiche entfernt. (2) Zwei Symbole verschiedener Fassungen hätten einander im Sekundentakt neu gestartet; jetzt ersetzt nur die neuere Fassung, die ältere beendet sich. (3) Ein Test löschte den echten Autostart des Entwicklers — Agent und Schlüsselbund-Dienst sind jetzt Modulkonstanten, die der Test umbiegt. Niedrige Befunde, behoben: `pgrep` mit maskiertem Pfad und auf den eigenen Benutzer beschränkt, vorrangig aber die Merkdateien `tunnel.pid` und `menue.pid`; gelöscht werden nur die zwei kanonischen Ordner (aufgelöst verglichen, nicht nach Namen — ein Repo namens `fints-mcp` bleibt stehen); gemeldet wird nur, was wirklich weg ist; die Schlüsselbund-Schleife hat eine Obergrenze; die Fassungsdatei wird vor der PID-Datei geschrieben. Als gut befunden: Das Zurücksetzen ist von außen nicht auslösbar (Herkunftsprüfung, Host-Prüfung, Token im Pfad, Einweg-Beleg, Tippwort), `_im_programm()` schließt Erweiterung, Tests und Nicht-Mac aus, die PIDs werden vor dem Löschen ermittelt, der eigene Prozess ist ausgenommen, `rmtree` folgt keinen Symlinks, und der Zustandsordner liegt unter 0700. Regeln festgehalten in `tests/test_sicherheit_1_5_0.py`.

1.5.3

Drei Dinge aus dem Test auf einem zweiten Mac, alle im Bereich „welche Instanz gehört mir“. (1) Das Fenster zeigte die Einrichtungsseite einer Erweiterung 0.4.1, die in Claude Desktop noch mitlief: Das Menü nahm jede antwortende Instanz als seine. Jetzt kommen nur Seiten ins Fenster, die sich über `/zustand` mit derselben Fassung melden (`eigeneSeiten`); eine fremde nennt das Menü als Hinweis. Merkdateien in `instanzen/` werden entfernt, wenn ihr Prozess tot ist, und ignoriert, wenn die Nummer

inzwischen einem fremden Prozess gehört (`prozessIstUnser`). (2) Wurde der Zustandsordner gelöscht, während der Server lief, hielt der alte Prozess den Port und wies das neue Token ab — die Seite zeigte `{"error":"unauthorized"}` . Das Startskript prüft jetzt, wer auf dem Port hört: eigene Prozesse werden beendet, ein fremdes Programm wird gemeldet. (3) `Alles von diesem Mac entfernen …` steht jetzt auch im Menü, weil die Seite dafür erreichbar sein müsste; ausgeführt wird derselbe Code (`einrichtung.jungfraeulich`) über `starten --zuruecksetzen` , und zwar per `exec` — bash liest sein Skript sonst zeilenweise weiter, und das Skript liegt im Bündel, das dabei verschwindet. Keine neuen Endpunkte, keine Änderung an Tresor, Token oder Herkunftsprüfung. Gezielt geprüft am 7. September 2026, weil der Notausgang Prozesse beendet und löscht. **Ein Befund hoch bis mittel, vor dem Bau behoben:** Das Urteil der Signaturprüfung lag in einem Merker, dessen Schlüssel das Änderungsdatum des Bündels war — das bewegt sich aber nicht, wenn nur `Contents/Resources/starten` ausgetauscht wird. Ein altes „in Ordnung“ hätte damit vor einem gebrochenen Siegel gestanden, und genau daran hängt der Start des löschenden Skripts. Wer ein Skript startet, fragt jetzt ohne Merker (`frisch: true`); der Merker bleibt für die Menü-Anzeige. Mittlere Befunde, behoben: (1) `jungfraeulich()` entfernte die Einträge in Claude und ChatGPT nicht — der Rückfragetext versprach sie. Sie stehen jetzt dort, damit Menü und Seite dasselbe tun. (2) Beim Herausziehen der Hilfsfunktion `ablegen` war die Besitzprüfung vor dem harten `kill -9` verlorengegangen; in den fünf Sekunden Wartezeit kann macOS die Nummer neu vergeben. Niedrige Befunde, behoben: Der Port wird auf Ziffern geprüft, bevor er in eine Adresse oder einen AppleScript-Text gelangt, und Dialogtexte werden maskiert beziehungsweise als Argument übergeben; das Zurücksetzen läuft mit `python -I` (kein Arbeitsverzeichnis auf `sys.path` , keine `PYTHON*` -Variablen aus der Umgebung) und meldet einen Fehlschlag samt Protokoll unter `~/Library/Logs` , statt stumm zu bleiben; die gemeldete Fassung einer fremden Instanz wird gefiltert und gekappt, bevor sie in einen Menütitel geht; das Fenster darf nur eigene Seiten laden (`erlaubteSeiten`); die Fassungslogik gilt wieder nur im Bündel; `spaeter_beenden` prüft vor jedem Signal, ob die Nummer noch uns gehört. Bewusst so gelassen: Der Skriptweg `starten --zuruecksetzen` verlangt kein Geheimnis — er gewinnt keine Rechte, denn der Tresor ist für denselben Benutzer ohnehin löschbar; die Bremse ist die Rückfrage im Menü. Regeln in `tests/test_startskript.py` und `tests/test_sicherheit_1_5_0.py` .

1.5.4

Geprüft am 7. September 2026 wurde der Diff seit 1.5.3 (der Tresor liest nach, wenn eine andere Instanz ihn geändert hat; Zugangsdaten-Seite bleibt nach einer Abweisung stehen). **Kein hoher Befund. Zwei mittlere, vor dem Bau behoben:** (1) Scheiterte das Entschlüsseln beim Nachladen, wurde der gemerkte Dateistand nicht fortgeschrieben — jeder weitere Seitenaufruf kostete dann wieder eine scrypt-Ableitung, synchron und unter der Sperre; ein `<img>` einer fremden Seite hätte das auslösen können. Jetzt wird der Stand auch im Fehlerfall vermerkt, ein neuer Versuch kostet erst wieder eine echte Änderung. (2) Nach einem gescheiterten Nachladen schrieben `hinzufügen` / `entfernen` trotzdem weiter und hätten einen Inhalt überschrieben, den dieser Prozess nicht kennt — und eine Sicherung gibt es nicht. `_nachladen()` gibt jetzt zurück, ob der Stand passt; sonst gibt es eine Meldung statt eines Schreibvorgangs. Niedrige Befunde, ebenfalls behoben: eine Größenschranke von 4 MB vor dem Einlesen; eine prozessübergreifende Sperre (`flock` auf eine Sperrdatei, ohne `fcntl` wie bisher) um lesen-ändern-schreiben, weil zwischen Lesen und `replace()` eine scrypt-Ableitung liegt; der Zwischenzustand mit Anmeldename verfällt nach zehn Minuten und wird ohne laufenden Probelauf verworfen; der Inode gehört zur Änderungserkennung, weil eine grobe Zeitauflösung eine gleich große

Änderung in derselben Sekunde verstecken würde; eine Zeile auf stderr, wenn eine geänderte Datei nicht lesbar ist — ohne Inhalt und ohne Passphrase; `_dateistand` wird in allen Pfaden geräumt. Als gut befunden: Die Passphrase bleibt im Speicher und `repr=False`, das Nachladen verlängert die Frist nicht und schreibt die gemeinsame Uhr nicht, die Liste entsteht vollständig, bevor sie zugewiesen wird, die Reihenfolge in `hinzufügen` (nachladen, dann Dublettenprüfung, dann schreiben) verhindert Doppel und verlorene Einträge, die Bankmeldung ist maskiert und die PIN wird im Zwischenzustand nicht einmal gespeichert.

Im selben Zug das Überweisungs-Experiment umgebaut (Frank 7.9.2026: „warum brauche ich ein extra Geheimwort, Überweisungen sind doch durch pushTAN abgesichert“). Die Analyse: Der tragende Riegel gegen ein gekapertes Modell ist die Seite, nicht das Wort — ihre Adresse trägt ein Geheimnis, das das Modell nie bekommt, also kann es vorbereiten, aber nicht absenden. Das Geheimwort schützte darüber hinaus nur gegen einen unbedachten Klick an diesem Rechner, und den stoppt die App-Freigabe der Bank ohnehin. Eingeschaltet wird das Experiment jetzt durch ein getipptes Wort, freigegeben durch den Klick auf der Seite; das Geheimwort bleibt als freiwillige Zusatzsperre. Behalten wurde die Bestätigung auf der Seite auch deshalb, weil nicht jede Überweisung zwingend eine starke Kundenauthentifizierung nach sich zieht — Umbuchungen zwischen eigenen Konten laufen bei manchen Instituten ohne TAN.

1.6.0

Geprüft am 7. September 2026 wurde der Diff seit 1.5.4 (schlanke Antwortform, ein Aufruf für alle Zahlungskonten, das neue Werkzeug `zusammenfassung`, die Kategorientabelle). **Kein hoher Befund. Sieben mittlere, alle vor dem Bau behoben:** (1) Das Demo-Kennzeichen hing an den Buchungen — ein Demo-Konto ohne Buchungen im Zeitraum kam ohne Kennzeichen, und das Modell hätte über ein erfundenes Konto „keine Buchungen“ gesagt; es kommt jetzt aus der Kontenliste beziehungsweise aus `kennt_konto`. (2) Fremdwährung wurde zu EUR umgeschrieben und mitaddiert; die Währung wird jetzt nur hochgezogen, wenn alle Buchungen dieselbe haben, und die Zusammenfassung rechnet nur die Hauptwährung, den Rest getrennt und ohne Umrechnung. (3) Eine ausgefallene Bank verschwand stillschweigend — Summen sahen vollständig aus, obwohl ein Konto fehlte; jetzt stehen `unvollstaendig` und `ohne_antwort` in beiden Antworten. (4) Aus der Verbotsliste wurde eine Positivliste (Giro, Spar, Festgeld): Eine Kontoart, die die Bank mit unbekanntem Code meldet, fällt heraus statt hinein. Dazu fängt `herkunft.umsaetze` jetzt je Konto ab — ein Festgeld ohne Umsatzabruf nahm vorher die Buchungen des Girokontos mit, und die Gruppierung nach Bank verhindert „mehrere Banken hinterlegt“ bei Kontonummern ohne IBAN. (5) `von` / `bis` behaupteten den angefragten Zeitraum; sie kommen jetzt aus den Daten, die Anfrage steht als `angefragte_tage` daneben. (6) Dieselbe Frage ergab je nach Aufrufstil andere Summen, weil die eigenen IBANs fehlten, wenn der Aufruf Kennungen nannte; die Zusammenfassung holt die Kontenliste jetzt immer. (7) Ein Eigenübertrag verlangt jetzt einen Beleg — die eigene IBAN in der Gegenseite oder eine eigene Kontonummer im Text. Die Wortliste („Übertrag“, „Tagesgeld“) versteckte echte Ausgaben: „Überweisung an Tagesgeld Nachbar“ fiel aus den Ausgaben heraus und verbesserte die Sparquote. Niedrige Befunde, ebenfalls behoben: Wortgrenze für alle Suchwörter (`netto` in „Nettoentgelt“, `strom` in „Stromberg“, `verein` in „Vereinigte Volksbank“), Umlaute gelten auch ausgeschrieben („Gebuehr“), unlesbare Beträge werden gezählt statt verschluckt, `eigenuebertraege` erscheint auch bei Netto-Null, und der Injektionshinweis steht im Docstring der Zusammenfassung. Als gut befunden: `kategorien.py` ist ohne ReDoS-Risiko (nur Literal-Alternation, `re.escape` auf jedem Wort, begrenzter Cache aus Modulkonstanten), die

Zusammenfassung gibt keine IBAN und keine Kontonummer heraus (`konten` ist eine Zahl), vorgemerkte Buchungen und Eigenüberträge sind aus den Summen heraus und stehen für sich, und `konten_abrufen` sowie `depot_abrufen` tragen `demo: true` unverändert je Konto und je Posten.

Aus derselben Prüfung eine Entscheidung ohne Code: Der Skill „Monatsabschluss“ legte ein Archiv aus Buchungen, Salden und Depotpositionen auf die Platte und bat im Zweifel darum, es zum Modellanbieter hochzuladen. Das widerspricht der Ansage, dass keine Bankdaten auf die Platte gehören (Frank 7.9.2026). Der Skill ist entfernt, die übrigen sechs kennen kein Archiv mehr, und ein Test hält das fest.

1.7.0

Auslieferung und Auffindbarkeit, kein neuer Zugriff. (1) Der Mac bekommt ein Abbild statt eines Archivs: `mac_dmg` in `gemeinsam.sh` legt Programm und ein Alias „Programme“ in ein UDZO-Abbild (gleiche Größe wie die ZIP, an 1.6.0 gemessen), `hdiutil verify` prüft es. Beglaubigt wird zweimal — erst das Programm über ein Archiv, das nur zum Einreichen entsteht und danach verschwindet, dann das Abbild selbst; so trägt das Programm seine Bestätigung auch, wenn es aus dem Abbild gezogen wird und der Rechner offline ist. Ohne Beglaubigung bricht der Bau ab, für Programm und Abbild getrennt. Windows bleibt ein ZIP. (2) Der Server heißt im Chatbot jetzt „Bankkonten (FinTS)“ statt „fints-mcp“, und die erste Zeile der Anleitung nennt die Wörter, nach denen Nutzer fragen — beides betrifft nur, wie ein Modell den Server findet, nicht was er herausgibt. (3) Für ChatGPT steht auf der Einrichtungsseite ein Satz zum Kopieren, sobald es eingerichtet ist: ChatGPT entscheidet je Frage selbst über einen Connector, und seine Konfiguration (`~/.codex/config.toml`) kennt kein Feld für eine Beschreibung — nur `command` , `args` , `env` , `enabled` und Zeitgrenzen. Der Satz geht in die Personalisierung des Nutzers, nicht in eine Datei, die wir schreiben.

Keine Änderung an Tresor, Token, Herkunftsprüfung, Werkzeugen oder Datenpfaden; die Prüfung von 1.6.0 gilt. Neu geprüft wurde der Bauweg: Das Abbild entsteht in einem eigenen Temporärordner, enthält nur das signierte Programm und ein Alias auf `/Applications` , und das Archiv zum Einreichen wird nach dem Einreichen gelöscht, damit es nicht versehentlich ausgeliefert wird.

Ein zweiter Durchgang am 7. September 2026, nachdem Windows zum ersten Mal auf einem echten Rechner lief: geprüft wurden das Lockern der Ortsregel, das Löschen und die neue Prozesserkennung. **Kein hoher Befund. Vier mittlere, alle vor dem Bau behoben.** (1) Mit dem Wegfall von „Downloads“ war unter Windows *keine* Ortsklasse mehr übrig — `_fremd_beschreibbar` ist dort ohnehin ohne Wirkung, eine Positivliste wie auf dem Mac gibt es nicht, und eine Signaturprüfung auch nicht. Ein Pfad auf einer Netzfreigabe oder einem USB-Stick wäre als `command` in Claude und ChatGPT gelandet: Wer auf die Freigabe schreiben darf oder den nächsten Stick mit demselben Laufwerksbuchstaben steckt, hätte bestimmt, was bei jedem Start als Nutzer läuft. Jetzt lehnt `hindernis()` UNC-Pfade ab und verlangt eine feste Platte (`GetDriveTypeW == DRIVE_FIXED`). (2) Die Menge der Ordner, die gelöscht werden dürfen, wurde aus derselben Quelle gebildet wie der Kandidat, und *beide* Seiten waren aufgelöst — der Vergleich prüfte am Ende nur, dass zwei gleiche Ausdrücke gleich sind. Eine Junction (die ein Nutzer ohne Adminrechte anlegen darf) auf „Dokumente“ hätte das Ziel rekursiv gelöscht. Jetzt erkennt `_verbogen()` Symlinks und Reparse Points und lässt sie liegen, gemeldet als „NICHT weg“. Dazu kam der neue dritte Ordner aus `%LOCALAPPDATA%` , also aus der Umgebung frei wählbar; er steht jetzt fest auf `~/AppData/Local/fints-mcp` . (3) `Remove-Item -Recurse -Force` läuft in Windows PowerShell 5.1 in

Junctions **hinein** (in PowerShell 7 behoben, hier nicht): Das Startprogramm entfernt jetzt zuerst die Verweise selbst und löscht danach mit `rd /s`, das ihnen nicht folgt. (4) Der Tresorpfad kam aus PowerShells `$HOME` (aus `%HOMEDRIVE%%HOMEPATH%`), der Eintrag in Claude aus Pythons `Path.home()` (aus `%USERPROFILE%`); auf Rechnern mit umgeleitetem Heimatverzeichnis sind das zwei verschiedene Orte, der Chatbot hätte einen leeren Tresor gesehen. Beide Seiten nehmen jetzt `%USERPROFILE%`.

Niedrige Befunde derselben Prüfung, ebenfalls behoben: Die Seite versprach beim Zurücksetzen Autostart und Startmenü, entfernt hat sie beides nicht — `jungfraeulich()` nimmt die zwei Verknüpfungen jetzt namentlich mit; `_ist_unser` filterte unter Windows nur nach Programmnamen („powershell.exe“, „python.exe“ — die häufigsten überhaupt) und hätte nach neu vergebener Prozessnummer einen fremden Prozess desselben Nutzers beendet, jetzt zählt die Kommandozeile (`Get-CimInstance Win32_Process`); die Protokolle des Aufräumens lagen unter vorhersagbaren Namen in `%TEMP%` und blieben liegen, jetzt Zufallsname und danach gelöscht; und `DefaultWebProxy = $null` galt auch für den einen Aufruf ins Netz (DoH bei Cloudflare) und umging damit auf verwalteten Rechnern den Unternehmens-Proxy — der bleibt für diesen Aufruf erhalten, umgangen wird er nur für 127.0.0.1. Bewusst nicht umgesetzt: eine ACL-Prüfung des Programmordners (`icacls`) unter Windows. Für den erwartbaren Fall trägt das: Wer im eigenen Profil schreiben darf, führt ohnehin schon Code als dieser Nutzer aus. Als niedriger Restbefund bleibt ein Ordner auf einer festen Platte **außerhalb** des Profils — etwa `C:\FragDeinKonto` oder unter `C:\ProgramData`, wo ein zweiter, nicht privilegierter Nutzer desselben Rechners hineinschreiben könnte. Ein Mehrbenutzer-Windows ist für dieses Werkzeug die Ausnahme, und die Alternative wäre eine Ortsregel, die einen legitimen zweiten Datenträger (`D:\Programme`) ausschließt. Als gut befunden: der Aufräumcode geht als konstantes `-c`-Argument an den Interpreter (keine Datei in `%TEMP%`, die zwischen Schreiben und Starten getauscht werden könnte), das Quoting des Serverstarts ist unkritisch (Anführungszeichen sind in Windows-Pfaden unzulässig, `Start-Process` geht ohne `cmd.exe` direkt an `CreateProcess`), 8.3-Kurznamen und Symlinks sind zum Prüfzeitpunkt schon aufgelöst, und der Reset-Weg selbst bleibt hinter Token-Pfad, Herkunftskopf, Einmal-Beleg und Tippwort. Die Nachkontrolle derselben Prüfung fand **zwei weitere mittlere Befunde in den Nachbesserungen selbst**, beide behoben: Der Ordner aus `%LOCALAPPDATA%` war nur auf der Python-Seite festgenagelt worden, im Startprogramm bestimmte weiter die Umgebung, welcher Ordner namens `fints-mcp` mit `rd /s /q` fällt; und `Stop-Process -Force` auf die Nummer aus `tunnel.pid` hatte keine Eigentümerprüfung — genau der Befund, der auf der Python-Seite mit `_ist_unser` behoben war, im Zwilling offen. Durch den neuen Vergleich des Programmordners wird dieser Zweig deutlich häufiger scharf (jeder zweite Download), und die Merkdatei überlebt einen Neustart; jetzt fragt `IstUnser` erst die Kommandozeile. Dazu niedrige Befunde: Der Startordner selbst wurde nicht auf Verweis geprüft (`Get-ChildItem -Recurse` steigt in Verweise nicht hinab, den ausdrücklich übergebenen Startpfad durchläuft es aber, und dann streifte das Zurücksetzen der Merker in einem fremden Baum ReadOnly und Hidden ab); `Test-Path` und `Get-ChildItem` behandeln `[` und `]` im Pfad als Platzhalter, deshalb jetzt durchgehend `-LiteralPath`; die Portwache beendete jedes Python unter einem Pfad `*\runtime\python.exe`, jetzt muss unser Server daneben liegen; und der Proxy-Aufruf braucht `ProxyUseDefaultCredentials`, sonst antwortet ein Proxy mit Anmeldezwang mit 407 und die Tunnel-Adresse erschiene nie.

Ein Befund aus der Nachkontrolle ist kein Sicherheitsproblem, sondern hätte den Start wieder unmöglich gemacht: PowerShell löst Funktionen erst beim Aufruf auf, und `IstUnser` stand 27 Zeilen **hinter** seiner ersten Verwendung. Der Parser findet das nicht. Ein Test prüft jetzt für jede Funktion des Skripts, dass sie vor ihrem ersten Aufruf auf oberster Ebene steht.

Ein letzter Befund aus demselben Testlauf, gefunden an Franks eigener Datei: Der Eintrag in Claude Desktop kam nie an, obwohl die Seite „eingerichtet" meldete und Claudes Entwicklerbereich („Lokale MCP-Server") vorhanden und nutzbar war — er zeigte „Keine Server hinzugefügt". `claude_desktop_config.json` ist nicht, wofür wir sie gehalten haben: Claude verwaltet diese Datei selbst, mitsamt `mcpServers`, und schreibt sie beim Beenden aus dem Speicher zurück. In der Datei auf dem Mac steht deshalb ein von Claude geschriebenes leeres `"mcpServers": {}` neben `preferences` und `coworkUserFilesPath`. Ein Eintrag, der bei laufendem Claude entsteht, wird beim Beenden verworfen. `eintragen()` und `entfernen()` verweigern jetzt, solange `laeuft()` wahr ist (Windows über `tasklist`, macOS über `pgrep -x Claude`), und die Seite zeigt in diesem Fall gar keinen Knopf, sondern wie man Claude wirklich beendet — unter Windows über das Symbol in der Taskleiste, denn das Fenster zu schließen beendet es dort nicht. Die Reihenfolge steht damit vor dem Klick statt danach.

Und der Grund, warum Claude den Eintrag nie zeigte, stand am Ende in Franks eigener Datei — nicht am Ort, nicht am Format, nicht an der Reihenfolge: `command` zeigte auf `C:\Users\…\Downloads\FragDeinKonto-1.7.0-Windows (1)\…`, den **ersten** Download, während längst der vierte Ordner in Benutzung war. Der Eintrag ist ein fester Pfad, und unter Windows liegt das Programm dort, wo der Nutzer entpackt hat; Windows hängt beim zweiten Download „ (1)" an. `eingetragen()` fragte nur, ob unser Schlüssel in der Datei steht — also meldete die Seite „eingerichtet", der Knopf hieß „Aus Claude entfernen", und es gab keinen Weg, den Pfad zu erneuern. Kein Neuinstallieren konnte daran etwas ändern. Jetzt vergleichen `veraltet()` und `zeigt_ins_leere()` den eingetragenen Pfad mit dieser Installation; die Seite zeigt dann „zeigt auf einen anderen Ordner", nennt den alten Pfad und bietet „Auf diesen Ordner umstellen" an. Dasselbe gilt für ChatGPT, wo die Falle identisch war. Der eigentliche Ausweg bleibt ein Installer, der das Programm an einen festen Ort legt — er steht weiter unter den offenen Punkten.

Und dann war es am Ende doch der Ort, nur ein anderer als vermutet: Claude Desktop kam auf dem Testrechner aus dem **Microsoft Store**. Für Store-Programme biegt Windows `%APPDATA%` auf einen paket-eigenen Ordner um, und Claudes Datei lag unter `%LOCALAPPDATA%\Packages\Claude_pzs8sxrjxfjjc\LocalCache\Roaming\Claude\claude_desktop_config.json` — mit Änderungsdatum vom Vortag, unberührt von allem, was wir geschrieben hatten. Unsere Datei unter `%APPDATA%\Roaming\Claude` liest so ein Claude nie. Die Regel lautet jetzt nicht „Store oder nicht", sondern: **Der Ordner, in dem Claude arbeitet, ist der richtige.** Erkennbar an Claudes eigenen Dateien daneben — `config.json`, `logs`, `Cache`, `Local Storage`, `Claude Extensions`, `extensions-installations.json`. Geprüft werden beide Orte, der Paketordner (`%LOCALAPPDATA%\Packages\*Claude*\LocalCache\Roaming\Claude`) und der gewohnte (`%APPDATA%\Claude`); es gewinnt der bewohnte, unabhängig davon, wo eine Datei liegt. Das deckt beide Installationswege ab und auch den Fall, dass beide auf demselben Rechner stehen; unser eigener, versehentlich angelegter Ordner enthielt nur die eine Datei und fällt damit durch. Beim Eintragen wird die verwaiste Kopie am falschen Platz mitgenommen, damit nicht später zwei Server auftauchen und die nächste Fehlersuche wieder bei der Frage beginnt, welche Datei gilt. Nebenbei stimmt damit auch die Erkennung der Erweiterung: `extensions-installations.json` und der Ordner `Claude Extensions` liegen neben derselben Datei.

Vier Fehlversuche davor sind der eigentliche Lehrsatz: Ortsvermutung (falsch), Richtlinie des Firmenkontos (durch den Screenshot des Entwicklerbereichs widerlegt), Claude überschreibt beim Beenden (die Datei behielt unseren Eintrag), veralteter Pfad im Eintrag (real, aber nur ein zweiter Fehler daneben). Was jeden Umlauf gekostet hat, war die fehlende Auskunft: Weder die Seite noch das

Protokoll nannte den Pfad, in den geschrieben wurde. Beides steht jetzt dort — die Seite unter jedem eingerichteten Chatbot, „Fehler zeigen" für alle in Frage kommenden Orte samt der Auskunft, ob unser Eintrag drin steht.

Regeln festgehalten in `tests/test_windows_1_7_0.py` und `tests/test_startskript.py`.

Windows holt auf. Das Startprogramm (`starten.ps1`) war noch der Weg von vor 1.4.4: immer erst einen Tunnel aufbauen, bis zu vier Minuten auf Adresse, DNS und Erreichbarkeit warten, dann eine Wand aus Text. Weil `Starten.cmd` PowerShell mit `-WindowStyle Hidden` ruft, sah der Nutzer in dieser Zeit **nichts** — „Klick auf Starten, nix passiert“ (Frank 7.9.2026). Neu, nach dem Vorbild des Menüleisten-Programms: Server starten, auf `/zustand` warten (und einen Fehlschlag als Fenster melden, samt Pfad zum Protokoll), Einrichtungsseite im Browser öffnen, Symbol in die Taskleiste setzen. Der Tunnel ist eine Option und folgt derselben Einstellung (`einstellungen.json` , Schlüssel `tunnel`) wie auf dem Mac; die Adresse wird erst weitergegeben, wenn Cloudflares DNS den Namen kennt.

Das Symbol (`NotifyIcon` mit `ContextMenuStrip` , Takt drei Sekunden) hat dieselben Punkte wie das Mac-Menü: Zustand mit Restzeit, sperren, entsperren (öffnet die Seite), Einstellungen, Tunnel ein/aus, Adresse kopieren, Autostart, alles entfernen, beenden. Zwei bewusste Unterschiede: Die Einrichtungsseite läuft im Standardbrowser statt in einem eigenen Fenster — WebView2 ist auf Windows nicht verlässlich vorhanden, und die Seite ist dieselbe; dadurch steht die Adresse mit dem Token allerdings im Browserverlauf, anders als auf dem Mac. Und entsperrt wird nur über die Seite: Ein Passwortfeld in diesem Programm gäbe es sonst zweimal, und Touch ID hat Windows nicht. Ändernde Aufrufe schicken `Sec-Fetch-Site: same-origin` , sonst weist der Server sie ab. „Alles entfernen“ ruft dasselbe `einrichtung.jungfraeulich()` wie der Mac, mit `python -I` , und räumt zusätzlich die Autostart-Verknüpfung weg.

Zwei Dinge, die den ersten Start überhaupt erst möglich machten. Erstens: `RandomNumberGenerator.Fill` gibt es nur in .NET Core, Windows PowerShell 5.1 läuft auf dem .NET Framework — ohne Tokendatei, also bei **jedem** ersten Start, starb das Skript in der dritten Sekunde, und weil das Fenster versteckt ist, sah man nur ein Aufblitzen. Jetzt `Create().GetBytes` . Zweitens hat das Skript eine `trap` : Jeder unbehandelte Fehler zeigt ein Fenster und landet in `%TEMP%\FragDeinKonto-start.log` — ein verstecktes Fenster ohne Rückmeldung ist der schlechteste Zustand von allen.

Und die Verpackung: Der entpackte Ordner hatte alles offen liegen — Laufzeit, Bibliotheken, Server, Startskript — und sah nach Entwicklerablage aus („das sieht sehr handgebastelt aus“, Frank 7.9.2026). Jetzt stehen oben genau zwei Dinge: **Frag Dein Konto.cmd** und **BITTE-LESEN.txt**; alles andere liegt in **Programmdateien**, das beim ersten Start das Hidden-Attribut bekommt. Dazu schreibt der erste Start eine Verknüpfung ins Startmenü des Nutzers, mit Symbol und Beschreibung, sodass „Frag Dein Konto“ dort auffindbar ist; „Alles entfernen“ nimmt sie wieder mit. Ein echter Installer (MSI) bleibt offen — dafür braucht es eine Windows-Maschine im Bau, die es hier nicht gibt.

Der eigentliche Grund, warum dreimal nichts passierte, war keiner der beiden oben: Die Startdatei wurde **im ZIP-Fenster** angeklickt. Der Explorer zeigt ein Archiv wie einen Ordner, kopiert beim Start aber nur die angeklickte Datei in einen Zwischenordner unter `%TEMP%` — `Programmdateien` bleibt im Archiv, und der Start bricht ab, bevor irgendeine eigene Meldung greift. Die Startdatei prüft das jetzt als Erstes in reinem Batch, ohne PowerShell, und erklärt im Fenster, was zu tun ist. Nebenbei entlastet

dieser Befund die Richtlinien-Theorie: Auf dem Testrechner stand `Get-ExecutionPolicy -List` durchgehend auf `Undefined`. Der Start über `-Command` bleibt trotzdem, weil er auf verwalteten Rechnern der verlässlichere Weg ist.

Danach kam der nächste Start bis zum Server und starb dort. Im Protokoll stand `can't open file 'C:\Users\…\FragDeinKonto-1.7.0-Windows': [Errno 2]` — der Ordner aus dem Download hieß `FragDeinKonto-1.7.0-Windows (4)`, und `Start-Process` gibt seine `-ArgumentList` **unquotiert** weiter. Windows schnitt den Aufruf am Leerzeichen ab. Der Pfad steht jetzt in Anführungszeichen, dazu `-WorkingDirectory`; nachgestellt mit echtem PowerShell in einem Ordner desselben Namens (ohne Anführungszeichen Ende-Code 2 und dieselbe Meldung, mit Anführungszeichen läuft der Server). Drei Folgerungen daraus: Beim Scheitern stehen die letzten Protokollzeilen und der Ende-Code **in der Meldung selbst** statt nur deren Pfad — ein Pfad hilft nur dem, der ihn öffnet; ein fehlender Interpreter wird als unvollständig entpackter Ordner gemeldet; und `[System.Net.WebRequest]::DefaultWebProxy = $null` verhindert, dass Windows PowerShell die Abfragen an 127.0.0.1 durch einen eingestellten Proxy schickt.

Zwei Dinge liefen danach anders als auf dem Mac. **Chatbot-Erkennung**: Claude Desktop und Codex waren installiert, die Seite sagte „nicht installiert“ bzw. „nicht möglich“. Claude legt `%APPDATA%\Claude` erst an, wenn es den Ordner braucht; gesucht wird jetzt zusätzlich dort, wo Windows-Programme liegen (`%LOCALAPPDATA%\Programs`, `%LOCALAPPDATA%`, `%APPDATA%`, beide `Program Files`, Startmenü). Und die Ortsregel für den dauerhaften Startbefehl war vom Mac übernommen: Dort startet das System eine heruntergeladene App aus einem wechselnden Ordner, weshalb „Downloads“ ausgeschlossen ist. Unter Windows gibt es das nicht, und der Rat „nach Programme ziehen“ ist dort nicht befolgbar — dieser Ordner gehört dem Administrator. Verboten bleibt nur ein Zwischenordner (`\Temp\`, `%LOCALAPPDATA%\Temp`, ein Pfad durch `.zip\`), den Windows selbst wieder wegräumt. **Zurücksetzen**: `_im_programm()` prüfte nur auf ein `.app`-Bündel, also war unter Windows der volle Weg gesperrt; jetzt gilt er auch dort, erkannt an `FINTS_MCP_SEITE_LOKAL` plus mitgeliefertem `runtime\python.exe`. Dazu drei Windows-Eigenheiten: Eine offene Datei lässt sich nicht löschen, deshalb beendet „Alles entfernen“ erst den Server und dann den Rest; `shutil.rmtree(ignore_errors=True)` verschluckte jeden Nur-Lesen-Merker, dafür gibt es `_baum_weg` mit `chmod`-Nachhilfe und eine Rückmeldung, was liegen blieb; und der Sockelordner der Geschwister liegt unter Windows im Profil (`%LOCALAPPDATA%\fints-mcp`) statt in einem `/tmp`, das ein Neustart wegräumt — er ging bisher nie mit. Das Startprogramm räumt zur Sicherheit selbst nach und nennt in der Meldung, was übrig ist. Der Aufruf des Aufräumcodes geht als `-c`-Argument an den Interpreter, nicht über eine Datei in `%TEMP%`, die zwischen Schreiben und Starten ausgetauscht werden könnte.

Ungetestet bleibt der Rechner selbst: Es gibt hier keinen Windows-Rechner. Der Bau prüft `starten.ps1` aber jetzt mit einem echten PowerShell-Parser (`pwsh`, `Parser::ParseFile`) statt nur die Klammern zu zählen; steht `pwsh` nicht bereit, bleibt der alte Rückfall. Als Tests festgehalten: Anführungszeichen um den Skriptpfad, Protokoll in der Meldung, kein Proxy, die Reihenfolge (Seite vor Tunnel, Server-Ende vor dem Löschen), die Menüpunkte, der Herkunftskopf und das Fehlen jedes Passwortfelds.

1.7.1

Anlass war eine Meldung vom 7. Oktober 2026: Mit der Vorführfassung 1.7.0_demo seien echte Kontodaten eines Dritten sichtbar gewesen. Es stimmte. `institute.suchen` war in der Demo gesperrt, `institute.nach_blz` nicht. Eine Bankleitzahl in Ziffern führte auf der Einrichtungsseite zum Formular einer echten Bank, und der Probelauf baute mit Kennung und PIN einen echten FinTS-Dialog auf. Ohne fremde Zugangsdaten kam man nicht hinein. Die Demo-Pakete wurden noch am selben Tag zurückgezogen. Die zentrale Vorführinstanz war nicht betroffen: Dort waren keine echten Zugänge eingetragen, und im Abbild fehlte der Bankcode.

1.7.1 sperrt an der Quelle: `nach_blz` kennt in der Demo nur die Demo Sparkasse. Probelauf und Überweisung verweigern echte Zugänge zusätzlich. Die Bankschicht fehlt im Demo-Paket ganz, und der Bau bricht ab, wenn sie doch drin ist oder ein Modul sie beim Laden importiert. Die Demo trägt überall eigene Namen: Tresor, Token, Zustand, Port, Geschwister, Schlüsselbund, Autostart und die Einträge in Claude und ChatGPT. Bis dahin konnte sie neben der Vollfassung deren Server übernehmen.

Die gezielte Prüfung danach fand keinen hohen und zwei mittlere Befunde, beide behoben. Unter Windows nahm „Zurücksetzen“ in der Demo Verknüpfungen und Geschwisterordner der Vollfassung mit. In Fassung 2 trugen Demo und Vollfassung dieselben Marken im ChatGPT-Eintrag und hätten sich gegenseitig ausgetragen. Ebenfalls behoben sind die niedrigen Befunde: Die Bauprobe erkennt auch absolute und verschachtelte Importe, das Demo-Zurücksetzen räumt die eigenen Ordner, die Windows-Bauprüfung sucht gezielt nach Pfaden und Port der Vollfassung, und die zentrale Instanz setzt `FINTS_MCP_NUR_DEMO=1` als zweite Sicherung.

Ein Tester mit Sicherheitshintergrund wies darauf hin, dass im ausgelieferten Code Kommentare mit Namen, Daten und Prüfungsnummern stehen, also eine Landkarte früherer Lücken. Seit 1.7.1 streift der Bau Kommentare und Docstrings aus dem eigenen Python-Code und den Startskripten (`erweiterung/abstreifen.py`). Im Repo bleiben sie. Ausgenommen sind die Docstrings der Werkzeuge, denn sie sind die Beschreibung für das Modell, und fremder Code unter `_vendor` samt Lizenzkopf. Die Zeilen bleiben als Leerzeilen stehen, damit Zeilennummern aus Protokollen weiter zum Repo passen. Eine Bauprobe bricht ab, wenn im Paket noch ein Name, ein Datum oder eine Prüfungsnummer steht. Ein Test vergleicht die Werkzeugliste vor und nach dem Abstreifen, und die ganze Testsuite läuft auch gegen den abgestreiften Code.

Eine zweite gezielte Prüfung über diese Korrekturen und das Abstreifen fand keinen hohen und einen mittleren Befund, behoben: In Fassung 2 hätte das Austragen aus ChatGPT bei einem Eintrag aus einem älteren Bau alles bis zum Dateiende gelöscht, auch fremde Server dahinter. Jetzt gilt die alte Endmarke als Rückfall, und ohne Ende geht nur die Anfangsmarke. Die niedrigen Befunde sind ebenfalls behoben: Das Abstreifen kürzt Leerraum nur noch in Zeilen, in denen es geschnitten hat, und lässt CRLF in Batchdateien stehen. Der Bau verlangte außerdem neue Fassungen zweier Abhängigkeiten, beide mit bekannten Lücken: PyJWT 2.15.1 (vorher 2.13.0) und urllib3 2.8.0 (vorher 2.7.0).

Bewusst offen: (1) Unter Windows ist nichts signiert, also kann niemand prüfen, ob die Skripte unverändert von uns stammen. Dafür braucht es ein Code-Signing-Zertifikat. Den Code in eine `.exe` zu verpacken, verbirgt ihn nur, sicherer wird er davon nicht. (2) In Fassung 2 liegt der FinTS-Kern noch im Demo-Programm. Abgeschnitten sind alle Aufrufstellen, erreichbar ist er nur mit einem Eingriff ins Programm selbst.

1.7.2

Windows bekommt ein Setup (`…-Windows-Setup.exe`), gebaut auf dem Mac mit NSIS (`erweiterung/windows-setup.nsi`). Es installiert nur für das eigene Benutzerkonto (`RequestExecutionLevel user` , Schreibzugriffe nur unter HKCU) an einen festen Ort, `%LOCALAPPDATA%\Programs\Frag Dein Konto` . Der Ort ist nicht wählbar, weil der Deinstaller ihn ganz löscht, auch nicht über den Schalter `/D=` , den jedes NSIS-Setup annimmt: `.onInit` setzt ihn fest. Der Deinstaller löscht nur an genau diesem Ort und nur, wenn dort unsere Fassungsdatei liegt. Ein erhöht gestartetes Setup warnt, weil es sonst im Profil des Administrators landete. Vor dem Ersetzen oder Entfernen beendet er die laufenden Teile, also Server und Tunnel aus diesem Ordner und das Symbol, dessen PID in `tunnel.pid` steht. Diese PID wird nur beendet, wenn sie zu einer PowerShell gehört, die `starten.ps1` ausführt; eine nach einem Neustart wiederverwendete Nummer trifft so keine fremde PowerShell. Den Ordner bekommt PowerShell als Umgebungsvariable, nicht in der Befehlszeile, denn ein Apostroph im Benutzernamen bräche sonst die Zeichenkette. Daten löscht der Deinstaller nur auf ausdrückliches Ja, über dieselbe Funktion wie „Alles von diesem Rechner entfernen“. Die Vorführfassung hat eigenen Ordner, eigenen Eintrag unter „Apps“ und eigene Verknüpfungen.

Das Setup ist nicht signiert (siehe oben). Neben jedem Download steht deshalb die SHA-256 (`summen_stempeln` in `gemeinsam.sh`), und die Anleitung zeigt, wie man sie mit `Get-FileHash` prüft. Das schützt gegen beschädigte oder unterwegs ausgetauschte Dateien, nicht gegen einen übernommenen Server, der Datei und Summe zugleich tauschen könnte. Das leistet erst eine Signatur.

Die neue Endung `.exe` stand anfangs nicht im Caddy-Matcher, der die Vollfassung hinter den Zugang legt. Ein Upload hätte sie offen ausgeliefert. `hochladen.sh` prüft deshalb schon vor dem Upload, dass Caddy jeden Paketnamen der Vollfassung mit 401 beantwortet, und danach noch einmal; ein Paket, das ohne Zugang erreichbar ist, nimmt es sofort wieder vom Server. Die Prüfung von 1.7.2 fand dazu einen mittleren Befund (Ort über `/D=` verschiebbar) und kleinere, alle vor der Auslieferung behoben.

Seit dem 8. Oktober 2026 bietet die Website für Windows nur noch das Setup an. Ab der nächsten Fassung baut `tunnel-bauen.sh` kein ZIP und keine `BITTE-LESEN.txt` mehr.

Verfahren für künftige Fassungen

Vor jeder Fassung: `security-review` auf den Diff; bei Änderungen an Tresor, Token, Seite, Geschwistern, Überweisung, Symbol, Bau oder Auslieferung eine gezielte Prüfung; Befunde hoch und mittel vor dem Bau beheben; Ergebnis hier fortschreiben. Neue Regeln werden als Tests festgehalten (`tests/test_sicherheit_1_3.py` als Muster). Der Bau prüft Abhängigkeiten und Downloads bei jedem Lauf.

11. Bedrohungsmodell in Kurzform

Angreifer	Ziel	Gegenmaßnahme
Findet ein Backup oder die Festplatte	Zugangsdaten	Tresor scrypt/AES-GCM; Passphrase nirgends gespeichert
Fremde Webseite im Browser des Nutzers	Tresor sperren/löschen, Frist aushebeln, Daten lesen	Token im Pfad, Host-Prüfung, Herkunftsprüfung, CSP, nebenwirkungsfreies GET
Anderer Benutzer desselben Rechners	Seite bedienen, Dateien lesen, Socket unterschieben	Token `600`, alle Dateien `600`/`700`, Eigentümerprüfung am Socketordner
Findet die Tunnel-Adresse	Kontodaten abfragen	Token ≥ 32 Byte, nur solange der Tresor offen ist; Adresse geht nicht an das Modell; `600` auf Platte
Manipulierter Verwendungszweck (Prompt-Injection)	Modell zu Überweisung oder Datenabfluss verleiten	Bereinigte Daten, Anleitung; Geld nur nach Freigabe auf der Seite (Adresse mit Geheimnis, das das Modell nicht hat) und App-Freigabe der Bank; Höchstbetrag; Doppelprüfung
Kompromittierter Webserver oder Netz	Ausgetauschtes Paket	Signatur und Beglaubigung der .app, SHA-256 auf der Seite; offen: .mcpb/Windows-Signatur
Kompromittierte Abhängigkeit	Eingeschleuster Code im Paket	Lockfile mit Hashes, `pip-audit`, feste Hashes für Laufzeit und cloudflared
Übernimmt den laufenden Prozess	Entsperrte Zugänge im Speicher	Nicht verhinderbar; klein gehalten durch Frist, Hardened Runtime, keine Lockerungen am Symbol
Bank oder Netz dazwischen	Mitlesen des Dialogs	TLS mit Zertifikatsprüfung; PIN/TAN-Sicherheit hängt an TLS (protokollgemäß)

12. Umgebung und Betrieb

Variable	Standard	Wofür
`FINTS_MCP_TRESOR`	`~/.config/fints-mcp/zugaenge.enc`	Tresor
`FINTS_MCP_ZUSTAND`	`~/.local/state/fints-mcp`	Zustand, Aufträge, lokales Token
`FINTS_MCP_LOG`	wie Zustand	Protokoll
`FINTS_MCP_TRANSPORT`	`stdio`	`http` für Tunnel/Server
`FINTS_MCP_TOKEN`	—	Pflicht im HTTP-Modus, ≥ 32 Zeichen
`FINTS_MCP_HOST` / `FINTS_MCP_PORT`	`127.0.0.1` / `8848`	HTTP-Bindung
`FINTS_MCP_SEITE_PORT`	`8849`	Lokale Seite im stdio-Modus
`FINTS_MCP_SEITE_LOKAL`	—	Tunnelprogramm: Seite nur lokal
`FINTS_MCP_FRIST_MINUTEN`	`60`	Vorgabe der Tresorfrist
`FINTS_MCP_KEIN_FENSTER`	—	Kein Fenster/Browser öffnen (Tests, Sonden)
`FINTS_MCP_OHNE_AUDIT`	—	`pip-audit` im Bau überspringen

Tests: `.venv/bin/pytest -q` (rund 400 Tests; `tests/conftest.py` sorgt dafür, dass kein Test die echten Ordner des Nutzers berührt). Bau: `cd erweiterung && ./mcpb-bauen.sh && ./tunnel-bauen.sh`.