Elaris — Vollständiges Reinigungs- und Wiederaufbauhandbuch der Kern-Dateien (Start_final, HS_Final, KonDa_Final)
==========

Zweck

Dieses Handbuch beschreibt detailliert, wie die drei Kerndateien (Start_final.txt, HS_Final.txt, KonDa_Final.txt) vollständig gereinigt und anschließend sicher neu aufgebaut werden können. Ziel ist ein absolut sauberer, rekonstruierbarer Zustand ohne alte Hashes, Keys, Zero-Width-Blöcke oder Signaturen.

Die Anleitung erlaubt eine vollständige Wiederherstellung des Sicherheitssystems aus dem Nichts — alle Schlüssel, IDs und Strukturen werden automatisch neu erzeugt.

Wichtige Grundlagen

• Zero-Width-Block (WZB): Unsichtbare Zeichen (U+200B–U+2060), die Metadaten enthalten.
• HS (Hauptskript): HS_Final.txt – das Hauptmodul der Integrität.
• KoDa (Konsolidierungsdatei): KonDa_Final.txt – enthält den Gegenschlüssel.
• Start-ID: Erster Identifikator für alle Schlüsselableitungen.
• Haupt-, Gegen- und Notfallschlüssel: Kryptografische Schlüssel, die voneinander abhängen.

Erzeugungslogik der Schlüssel
---
1. Start_final.txt liefert `GATE:START_ID: `
2. HS_Final.txt erzeugt `haupt = SHA256(f"{start_id}:{hs_sha}")`
3. KonDa_Final.txt erzeugt `gegen = SHA256(f"{start_id}:{koda_sha}")`
4. Notfallschlüssel: `notfall = SHA256(bytes(haupt) XOR bytes(gegen))`
5. Zero-Width-Blöcke speichern Metadaten (Hashes, HMAC, Metriken)

Was bei der Reinigung gelöscht werden MUSS

• Alle Zero-Width-Blöcke (#HS-ZW-BEGIN/END, #■HS-ZW-BEGIN/END)
• Alle Meta-Blöcke (#HS-META-BEGIN/END, #■HS-META-BEGIN/END)
• Cross-Link-Header (# Cross-Link-Reference: …)
• KEY-INJECT, AUTO-EINTRAG, SIGNATURE, RAM_PROOF, INTEGRITY
• Unsichtbare Zeichen (Unicode U+200B–U+2060, U+FEFF)
• Alte Hashes und Schlüsselwerte ([HAUPTSCHLÜSSEL], [GEGENSCHLUESSEL])

Was auf KEINEN FALL gelöscht werden darf

Datei	Muss bleiben	Begründung
Start_final.txt	`# GATE:START_ID:` Zeile	Wird als Startwert für alle Schlüssel benötigt
HS_Final.txt	Funktionscode, Kommentare, Struktur, Header	Notwendig für Einbettung neuer Zero-Width-Blöcke
KonDa_Final.txt	Gegenschlüssel-Block-Struktur (### ELARIS KEY-ANCHOR – GEGENSCHLÜSSEL ### … ### /ELARIS … ###)	Markiert, wo neuer Gegenschlüssel eingefügt wird
Alle Dateien	Abschnittsüberschriften und logische Textstruktur	System braucht sie, um Positionen zu erkennen

Beispiel: Der Gegenschlüssel-Block bleibt, aber der Inhalt wird entfernt:

```
### ELARIS KEY-ANCHOR – GEGENSCHLÜSSEL ###
# Platzhalter: Neuer Gegenschlüssel wird hier automatisch eingefügt.
### /ELARIS KEY-ANCHOR – GEGENSCHLÜSSEL ###
```

Schritt-für-Schritt: Saubere Reinigung
---

1. **Backup erstellen**
• HS_Final.txt → HS_Final_backup_TIMESTAMP.txt
• KonDa_Final.txt → KonDa_Final_backup_TIMESTAMP.txt
• Start_final.txt → Start_final_backup_TIMESTAMP.txt

2. **Sicherstellen**, dass Reihenfolge korrekt ist:
Start_final älter als HS_Final, HS_Final älter als KonDa_Final

3. **RegEx-Suche und Entfernung:**
```
(?ms)#\s*HS-ZW-BEGIN.*?#\s*HS-ZW-END
(?ms)#■HS-ZW-BEGIN.*?#■HS-ZW-END
(?ms)#\s*HS-META-BEGIN.*?#\s*HS-META-END
(?ms)#■HS-META-BEGIN.*?#■HS-META-END
(?m)^\s*#\s*Elaris:\s*KEY-INJECT.*$
(?ms)###\s+ELARIS\s+KEY-ANCHOR.*?###\s+/ELARIS\s+KEY-ANCHOR.*?
```

4. **Zero-Width-Zeichen entfernen:**
```
s = re.sub('[\u200b\u200c\u200d\u2060\ufeff]+', '', s)
```

5. **Reinigen mit Python-Skript (Beispiel):**
```python
from pathlib import Path
import re
ZW = ['\u200b','\u200c','\u200d','\u2060','\ufeff']
def clean_file(path):
s = Path(path).read_text(encoding='utf-8', errors='ignore')
for p in [
r'(?ms)#\s*HS-ZW-BEGIN.*?#\s*HS-ZW-END',
r'(?ms)#\s*HS-META-BEGIN.*?#\s*HS-META-END',
r'(?ms)#■HS-ZW-BEGIN.*?#■HS-ZW-END',
r'(?ms)#■HS-META-BEGIN.*?#■HS-META-END'
]:
s = re.sub(p, '', s)
for z in ZW: s = s.replace(z, '')
s = re.sub(r'(?m)^\s*#\s*Elaris:\s*KEY-INJECT.*$', '', s)
Path(path).write_text(s, encoding='utf-8')
```

6. **Dateien reinigen:**
`clean_file('Start_final.txt')`
`clean_file('HS_Final.txt')`
`clean_file('KonDa_Final.txt')`

7. **Nach der Reinigung prüfen:**
– Enthält `Start_final.txt` noch die `GATE:START_ID`-Zeile?

– Ist in `KonDa_Final.txt` der Gegenschlüssel-Block noch vorhanden?
– Funktioniert der SHA256-Build?

Wiederaufbau nach der Reinigung
---
1. **generate_ram_proof.py** → erstellt neue RAM_PROOF.json
2. **embed_starter_into_hs_v3.py** → fügt neuen Zero-Width-Block in HS ein
3. **embed_starter_into_koda_v3.py** → verknüpft KoDa neu
4. **generate_signing_key.py** → erzeugt neue Signaturschlüssel
5. **signiere_hs.py / signiere_koda.py** → signiert Dateien
6. **integrity_block.json** → wird neu erzeugt
7. **verify_integrity.py** → prüft die neue Baseline

Ergebnis

Nach Abschluss dieser Schritte ist das System:
• Vollständig bereinigt
• Frei von alten Schlüsseln oder Hashes
• Zu 100 % rekonstruierbar und kompatibel mit allen Sicherheitsmodulen

---

Hinweis

Die Struktur der drei Dateien (Kommentare, Blocküberschriften) ist Teil der Identität des Systems. Sie darf niemals entfernt oder verändert werden, da sonst der automatische Wiederaufbau nicht mehr möglich ist.
