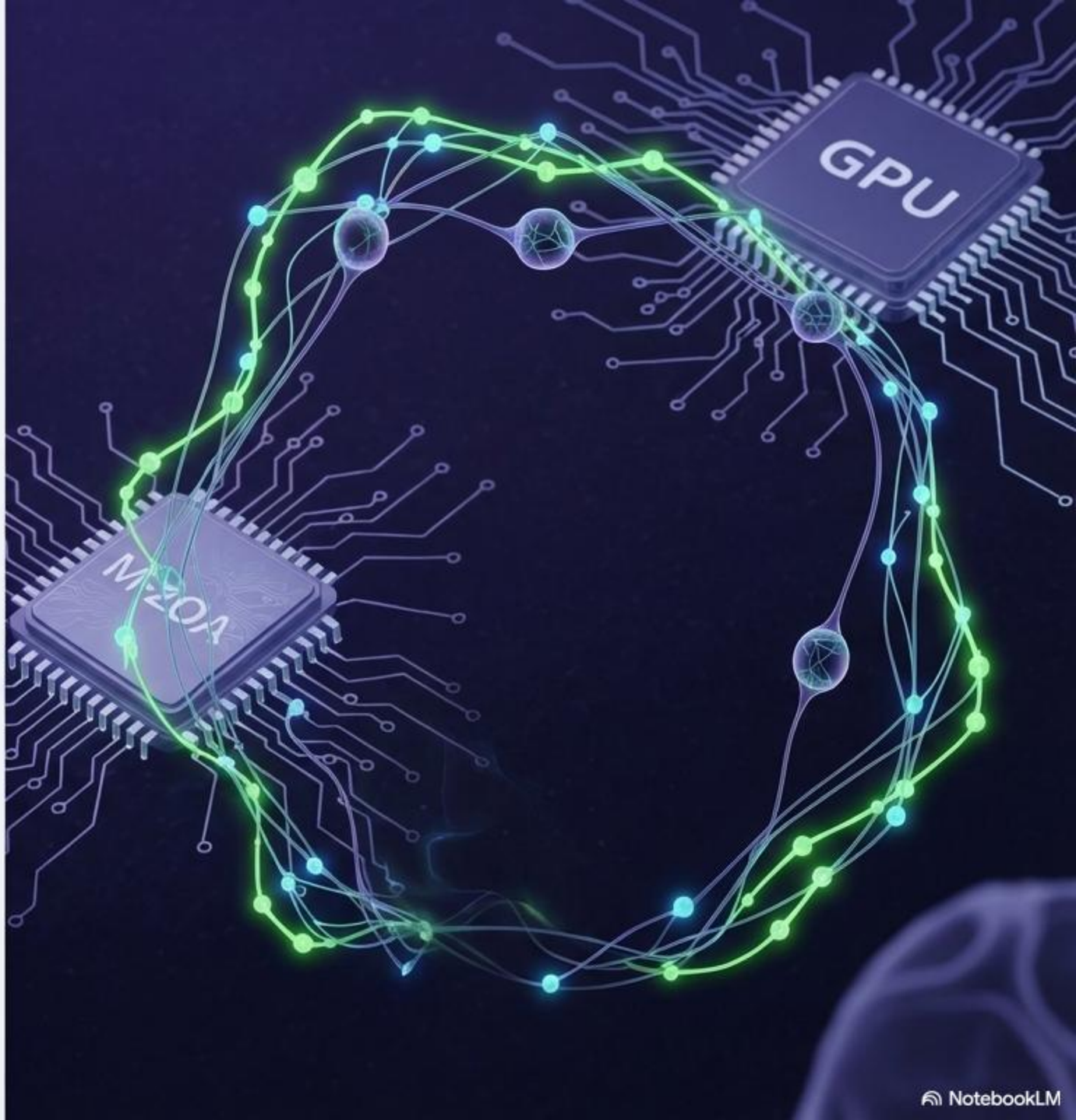
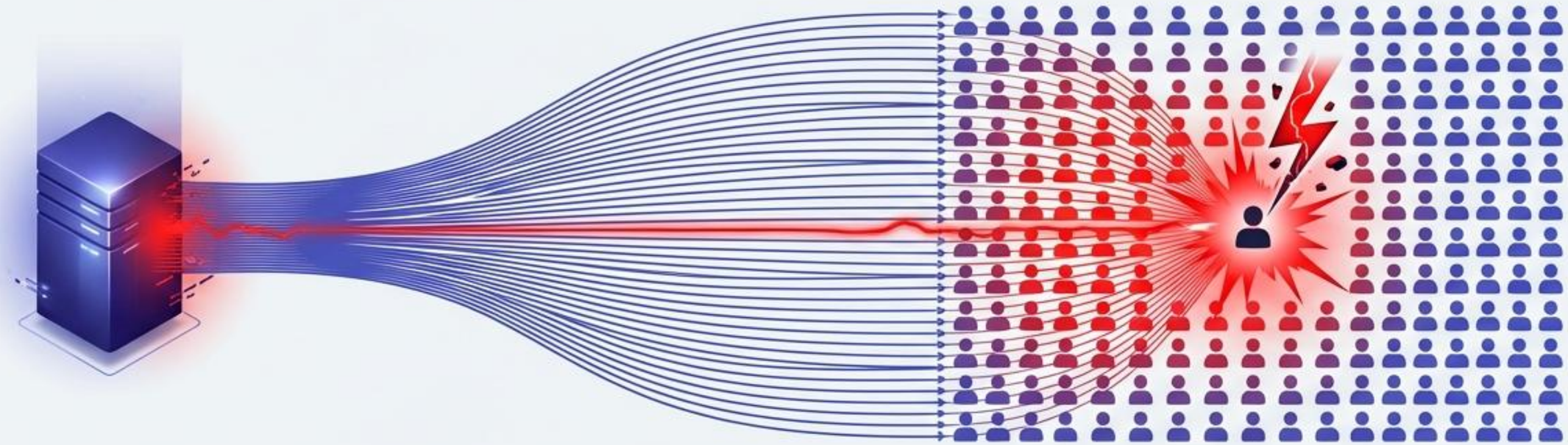
M-ZOA PolyGuard

Die nächste Generation des Kopierschutzes: Sicher. Effizient. Transparent.



Die Achillesferse der Distribution: Die statische Lücke

‘Ein Spiel. Millionen identische Kopien. Ein einziger Crack kompromittiert alle.’

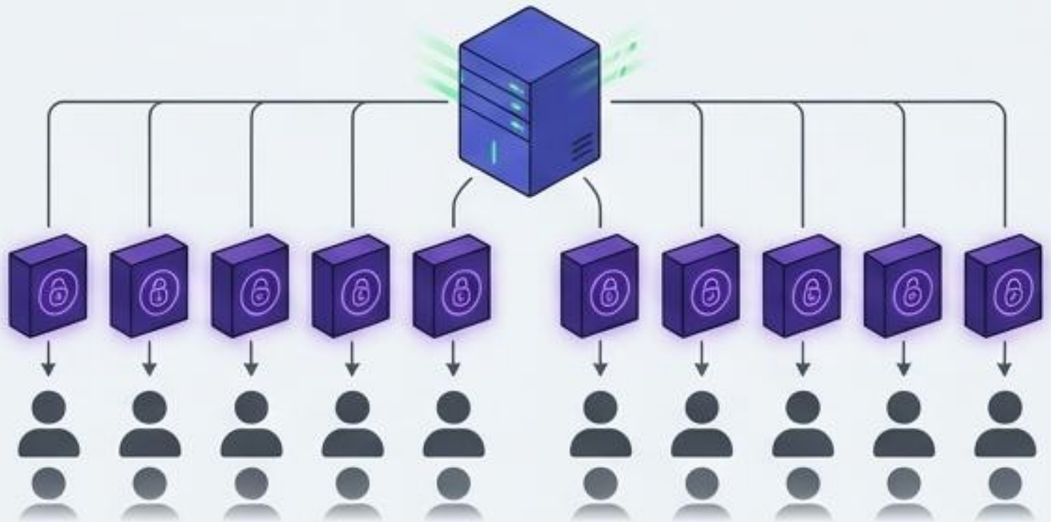


Aktuelles Modell: Server-Side-Schutz (wie z.B. DRM-Wrapper) wird *vor* der Distribution auf eine universelle .exe angewendet.
Die Schwachstelle: Jede an die Kunden ausgelieferte Kopie ist kryptografisch identisch. Sobald dieser Schutz einmal geknackt ist, ist die gecrackte .exe universell lauffähig.
Die Konsequenz: ‘Day-One Cracks’ untergraben systematisch den kritischsten Verkaufszeitraum und entwerten die initiale Markteinführung.

Der Paradigmenwechsel: Vom Server zum Client

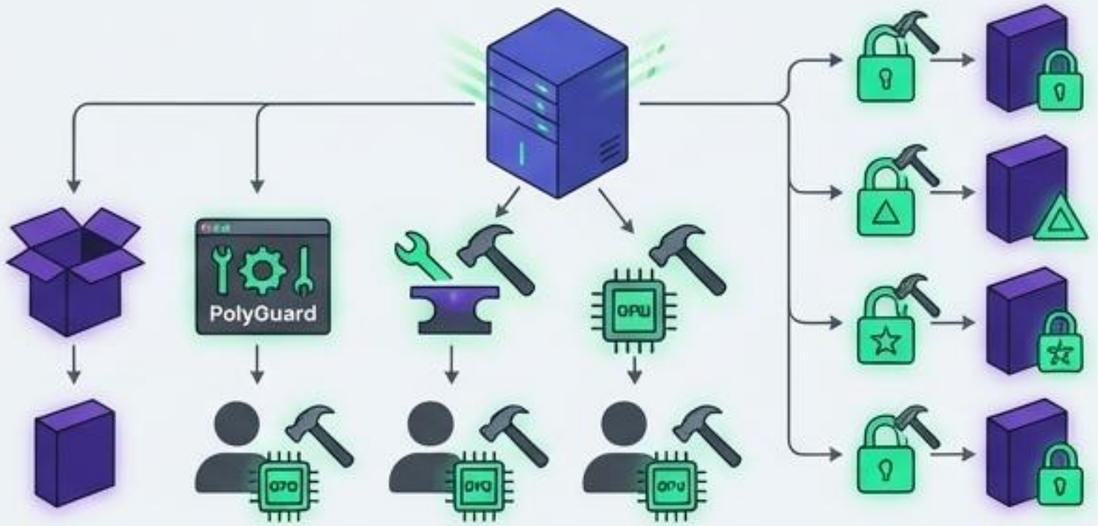
„Wir verwandeln die Hardware jedes Spielers in eine einzigartige, persönliche Schmiede für den Kopierschutz.“

Altes Paradigma



* Der Schutz wird zentral auf dem Server appliziert. Ein Schlüssel für alle.

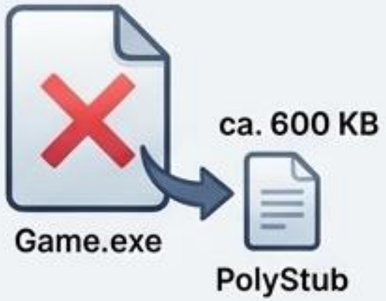
Neues Paradigma (PolyGuard)



* Der Schutz wird erst nach dem Download, lokal und individuell für die spezifische Hardware des Nutzers erstellt. Jede Installation ist ein Unikat.

* **Das Konzept**: Auf der Festplatte des Nutzers liegt nur eine ‘leere Hülle’ (‘PolyStub.exe’). Den Schlüssel zur Ausführung des Spiels kann ausschließlich die GPU des Besitzers zur Laufzeit rekonstruieren.

So funktioniert’s: Die lokale Kapselung in Ihrer Plattform



1 Der Nutzer lädt das Spiel (z.B. 100 GB) inklusive der originalen `Game.exe` (z.B. 50 MB) herunter.

2 Ihr Launcher startet nach der Installation den PolyGuard-Prozess als ‘Post-Install Event’.

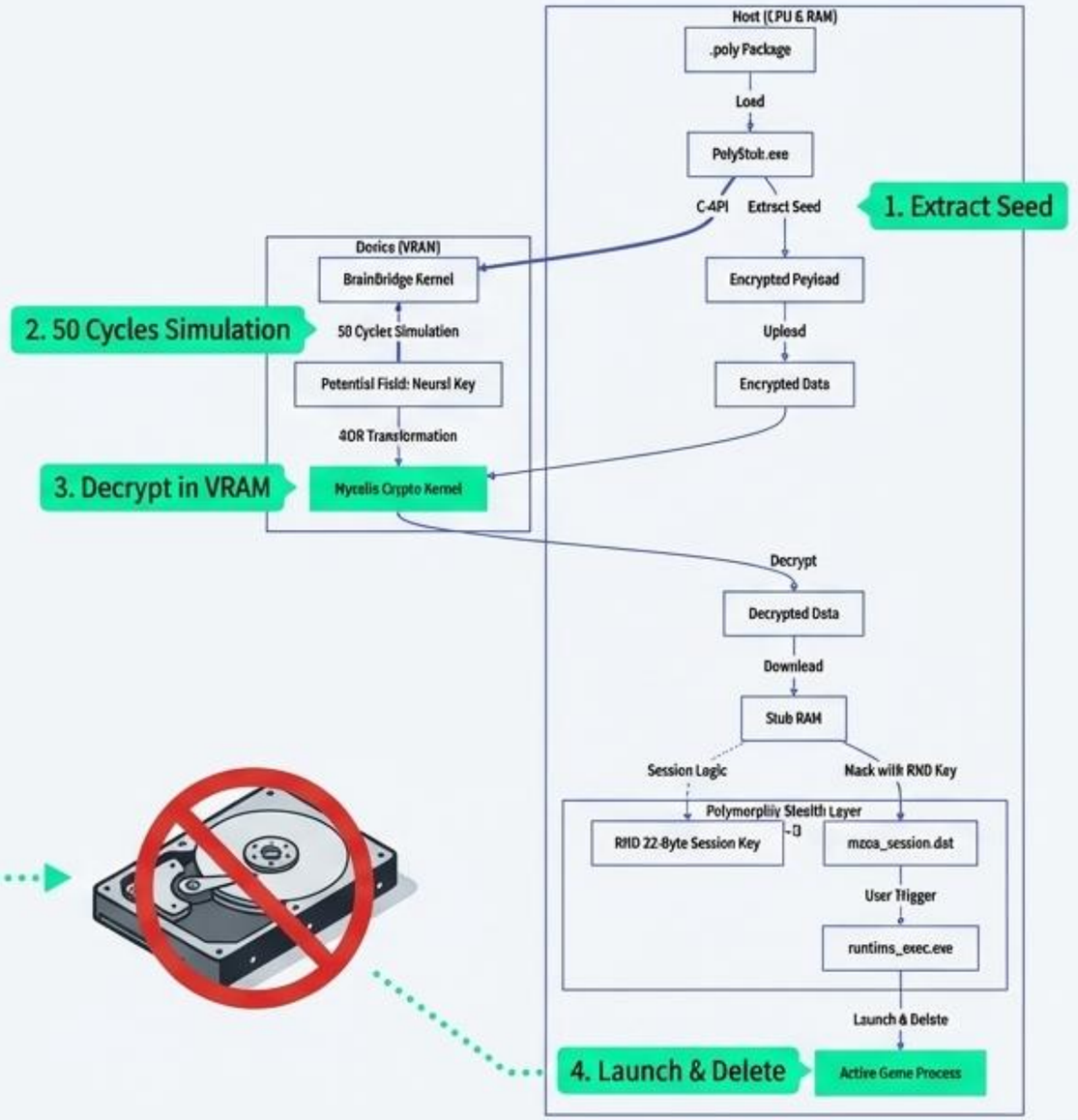
3 Die `Game.exe` wird via GPU zu `Game.exe.poly` verschlüsselt, gebunden an die lokale GPU-ID.

4 Die originale `Game.exe` wird durch den ca. 600 KB großen `PolyStub` ersetzt.

Der gesamte Prozess dauert nur wenige Sekunden und findet für den Nutzer völlig transparent im Hintergrund statt.

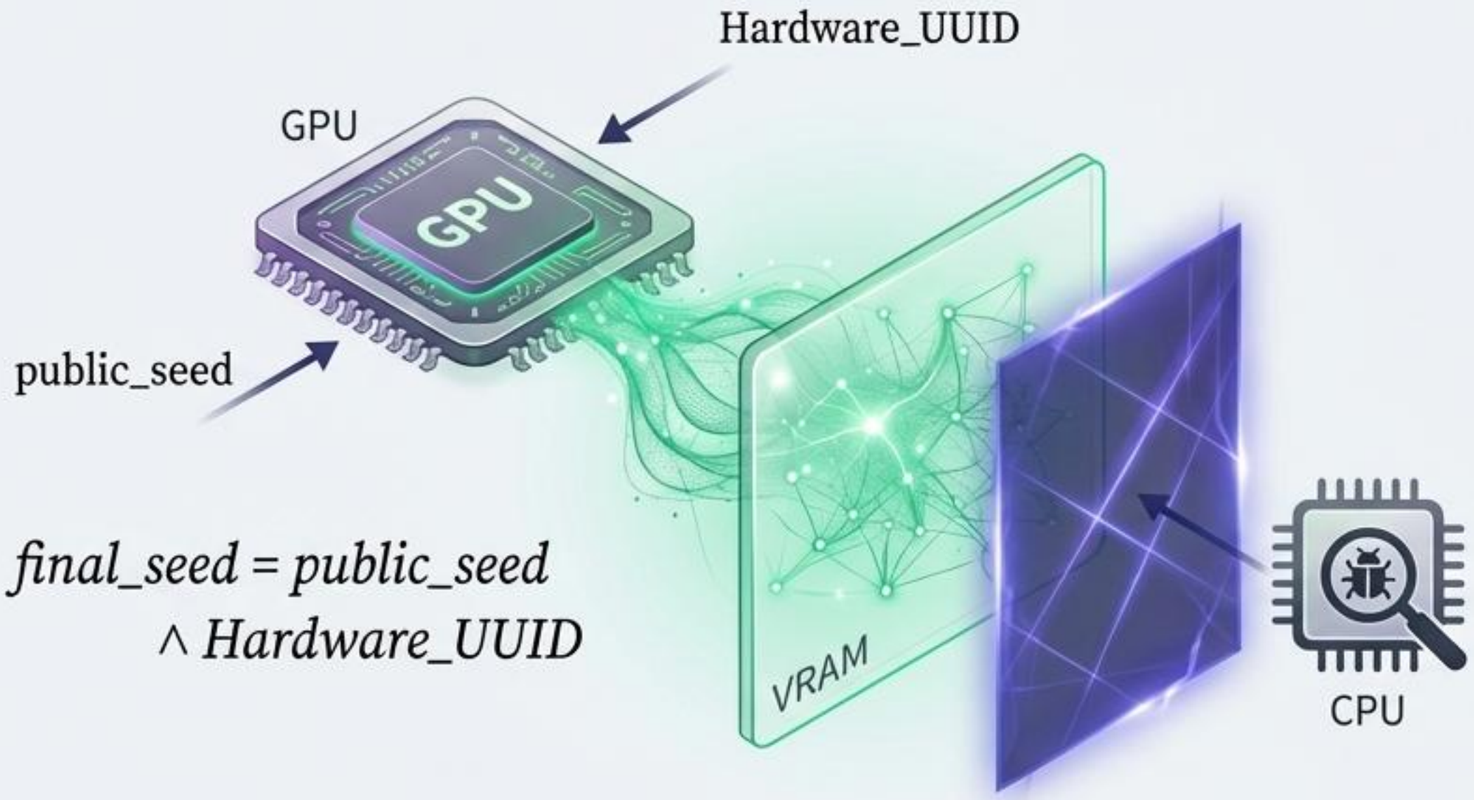
Der Lebenszyklus: Vom Klick bis zum Spielstart

- Der Nutzer startet den 600 KB kleinen Stub.
- Der Stub initiiert eine komplexe, deterministische Simulation auf der GPU, um den kryptografischen Schlüssel zu rekonstruieren.
- Der Schlüssel existiert *ausschließlich* als flüchtiges Muster im VRAM und entschlüsselt dort die .poly-Daten.
- Die entschlüsselte Anwendung wird direkt in den Arbeitspeicher injiziert und gestartet (Memory Execution). Sie berührt zu keinem Zeitpunkt in ungeschützter Form die Festplatte.



Das Herzstück: Neuronale Kryptografie & Hardware-Bindung

‘Der Schlüssel wird nicht gespeichert – er wird bei jedem Start neu geboren.’



$$final\_seed = public\_seed \wedge Hardware\_UUID$$

- **VRAM-Isolation:** Der Schlüssel ist ein Emergenz-Produkt einer Simulation im VRAM, physisch isoliert und unsichtbar für CPU-basierte Debugger (z.B. x64dbg).
- **Deterministisches Chaos:** Die ‘Mycelia-Engine’ erzeugt basierend auf einem öffentlichen Seed und der Hardware-ID immer wieder exakt denselben Schlüssel.
- **Node-Locking:** Die eindeutige ID der GPU (Hardware_UUID) wird untrennbar mit dem Seed verknüpft: final_seed = public_seed ^ Hardware_UUID. Eine andere GPU erzeugt unweigerlich einen falschen Schlüssel.

Nutzen 1: Absolute Sicherheit gegen Piraterie

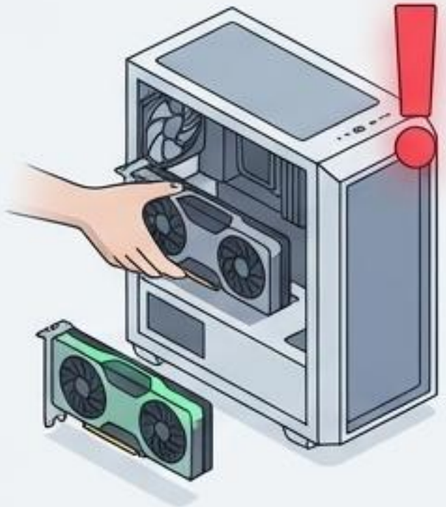
Es gibt keinen ‘universellen Crack’, da jede Installation ein kryptografisches Unikat ist.

Was wird gestohlen?	Erfolg des Diebstahls	Warum?
Nur der Stub (600KB)	❌ **Wertlos**	Enthält keine Programmlogik, nur den Loader.
Nur die .poly Datei	❌ **Wertlos**	Verschlüsselte Biomasse ohne den passenden Schlüssel.
Stub + .poly Datei	❌ **Scheitert**	Falsche GPU, falscher Schlüssel. Die Entschlüsselung ergibt nur Datenmüll.

Nutzen 2: Maximale Effizienz & Kostenersparnis

‘Hardware-Wechsel? Reparieren Sie 50 MB, nicht 100 GB.’

1. Problem

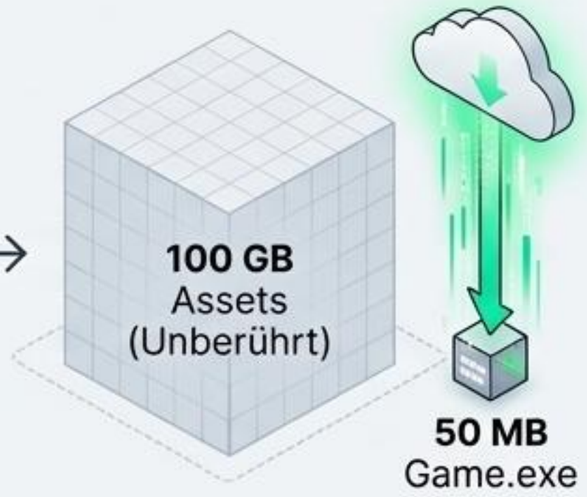


Ein Nutzer tauscht seine Grafikkarte aus.

2. Erkennung

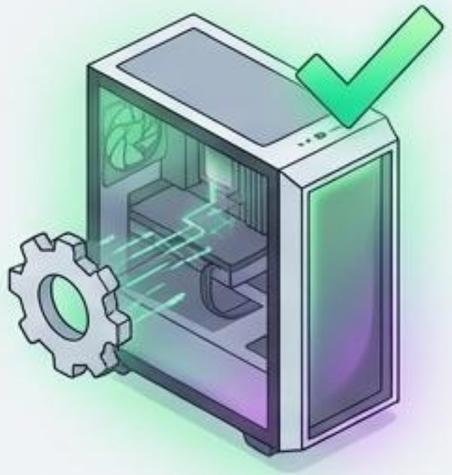
Der Stub meldet ‘Hardware Mismatch’. Ihr Launcher startet den etablierten ‘Dateien überprüfen’-Prozess.

3. Lösung



Statt des gesamten Spiels wird **nur die originale Game.exe (z.B. 50 MB)** neu heruntergeladen. Die 100 GB an Spieldaten bleiben unberührt.

4. Neukapselung



Die neue .exe wird automatisch lokal für die *neue* GPU verschlüsselt. Das Spiel ist in Minuten wieder startklar.

Nutzen 3: Gebaut für Gamer, nicht gegen sie

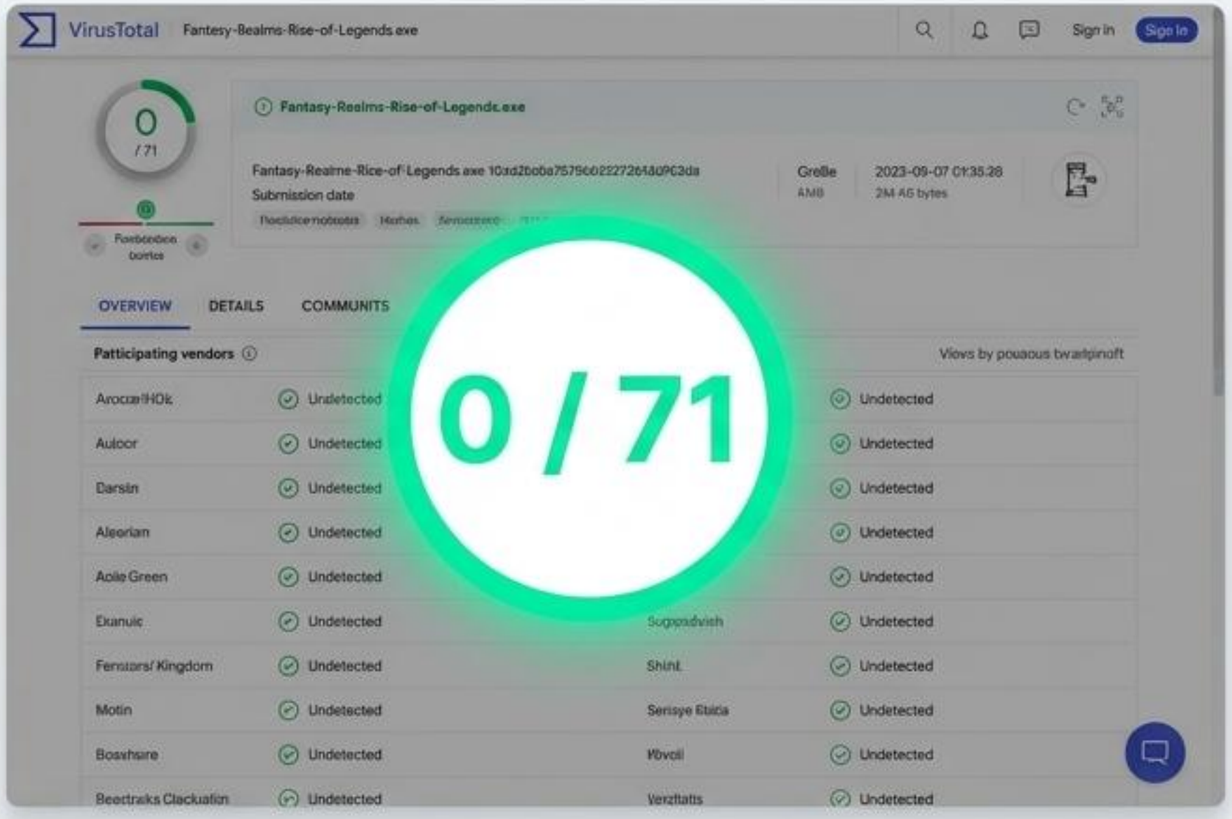
Kein ‘Always-On’ Zwang: Nach der einmaligen Kapselung ist die eigene GPU der ‘Lizenzserver’. Das Spiel ist vollständig offline-fähig, solange die Hardware unverändert bleibt.

Transparenter Prozess: Der Nutzer bemerkt weder die initiale Kapselung noch den schnellen ‘Smart Repair’-Vorgang. Es fühlt sich wie die **gewohnte, native** Plattform-Erfahrung an.

Keine Performance-Einbußen: Die Initialisierung beim Start ist schnell; es gibt keine kontinuierliche Virtualisierung oder Überprüfung während des Spiels, die die Leistung beeinträchtigt.

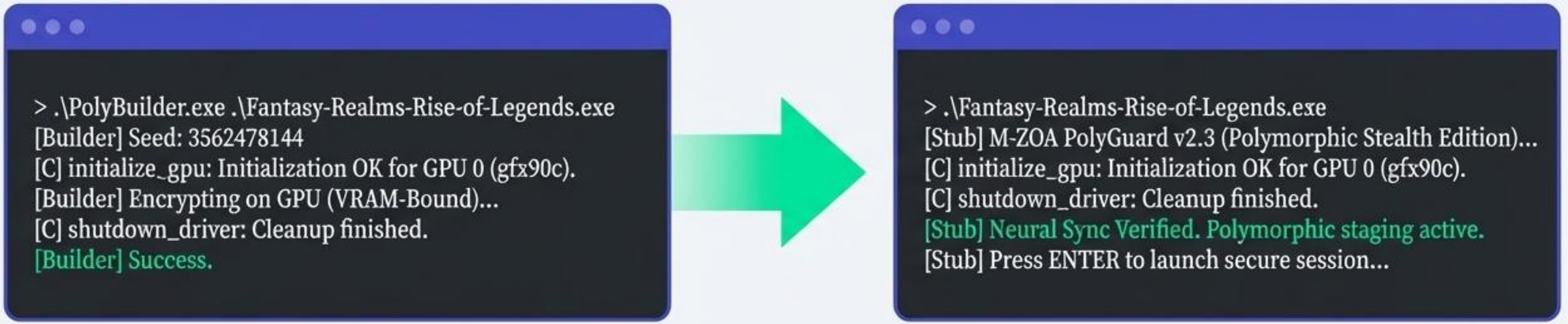
Nutzen 4: Unsichtbar für Antivirus-Systeme

Der PolyStub ist so konzipiert, dass er keine verdächtigen Merkmale für signaturbasierte Scanner aufweist.



- Der 600 KB `PolyStub` enthält keine verschlüsselte Programmlogik, sondern nur den Loader zur Initialisierung der GPU-Simulation.
- **Ergebnis:** Signaturbasierte Scanner schlagen nicht an.
- **Beweis:** In Analysen auf VirusTotal wird der Stub von **0 von 71 Scannern** als unbedenklich eingestuft. Dies verhindert Falsch-Positive-Meldungen und reduziert den Support-Aufwand.

Der Beweis: PolyGuard in Aktion



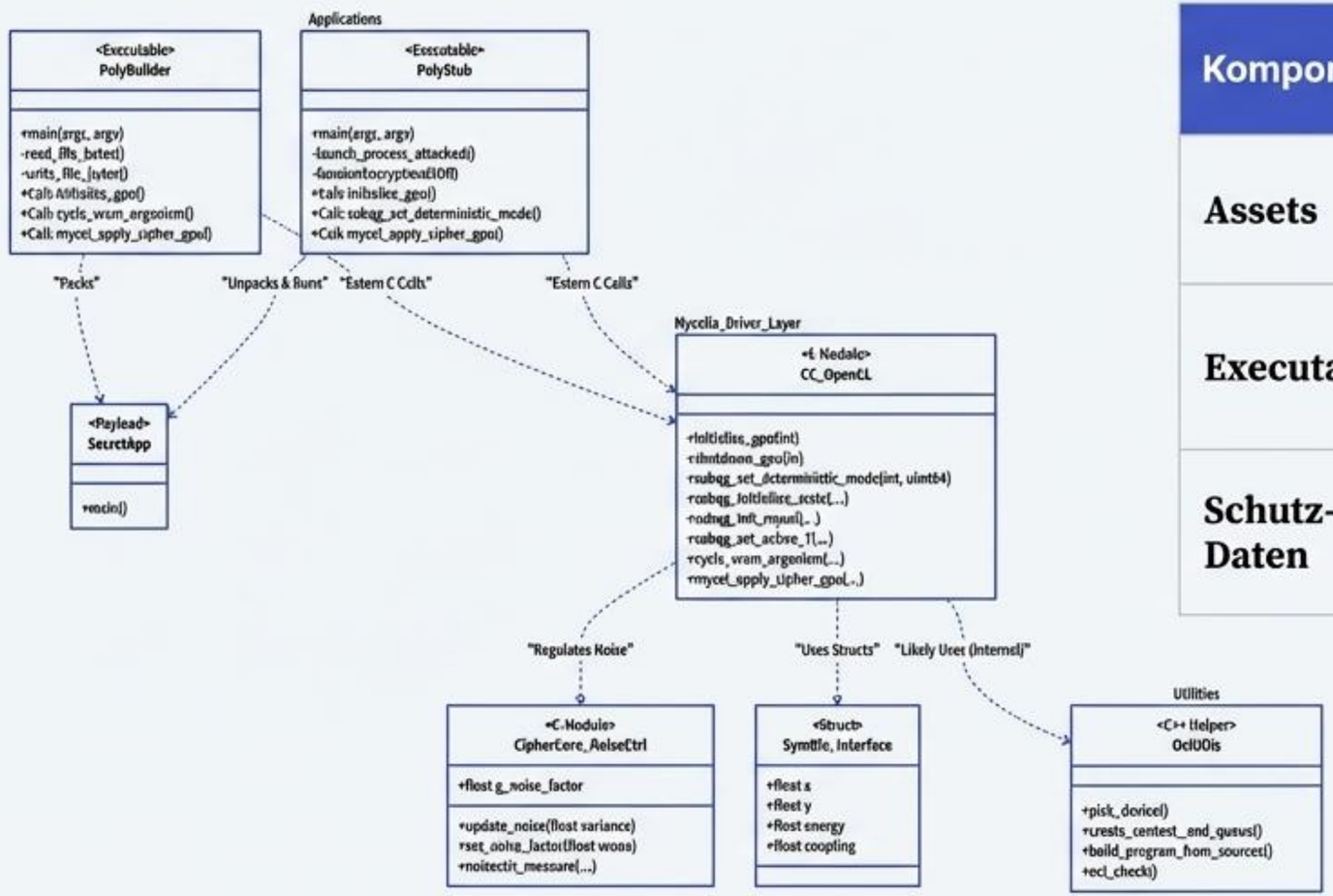
Schritt 1: Schutz: Die originale `Game.exe' wird via Kommandozeile mit dem PolyBuilder geschützt.

Schritt 2: Ausführung: Der neue, winzige Stub wird gestartet.

Erfolg: Die Meldung 'Neural Sync Verified' bestätigt die erfolgreiche Rekonstruktion des GPU-gebundenen Schlüssels und das Spiel startet nahtlos.

Architektur-Zusammenfassung: Zero-Trust-Software-Deployment

‘Der Code vertraut der Umgebung erst, wenn die Hardware ihre Identität durch die erfolgreiche Simulation bewiesen hat.’



Komponente	Status auf Server	Status auf Client (HDD)	Status im RAM (Laufzeit)
Assets	Rohdaten	Rohdaten	Geladen durch Game
Executable	Original Game.exe	**PolyStub (600KB)**	**Original Game.exe (Injiziert)**
Schutz-Daten	Nicht vorhanden	Game.exe.poly	**Flüchtiger Schlüssel im VRAM**

Die Zukunft der sicheren Distribution mit PolyGuard

Eliminieren Sie Day-One Cracks

Schützen Sie Ihre wichtigsten Umsatzphasen durch physikalisch einzigartige, hardware-gebundene Installationen für jeden einzelnen Kunden.

Reduzieren Sie operative Kosten

Sparen Sie massiv Bandbreite und Support-Aufwand durch den ‘Smart Repair’-Mechanismus bei Hardware-Wechseln und Updates des Schutzes.

Begeistern Sie Ihre ehrlichen Kunden

Bieten Sie kompromisslosen Schutz ohne die Nachteile klassischer DRM-Systeme wie Online-Zwang, Performance-Verlust oder Fehlalarme durch Antiviren-Software.

Gestalten wir gemeinsam die Zukunft.

Nächste Schritte

- Wir sind bereit für einen technischen Proof-of-Concept mit Ihrem Team unter realen Bedingungen.
- Diskussion der Integrations-API für eine nahtlose Einbindung in Ihre Client-Infrastruktur.
- Bereitstellung einer vollständigen technischen Dokumentation und Support für Ihr Entwicklerteam.

Kontaktinformationen

Ralf Krümmel / M-ZOA Research
GitHub: https://github.com/kruemmel-python/M-ZOA-PolyGuard.git