Cheating Detection in Anki Submission Forensics

This document describes detection methods for academic dishonesty in student Anki `.apkg` submissions for the ESL flashcard course. Detection relies on forensic analysis of SQLite database internals — note IDs, GUIDs, checksums, card IDs, deck IDs, review logs, and file-level hashes — to distinguish legitimate student work from copying, fabrication, and disengagement.

Related documents:

- ANKI_DB_SCHEMA.md — Full database schema reference
- STUDENT_PERFORMANCE_ANALYSIS.md — Performance metrics that double as cheating signals
- READING_MATERIAL_METHODOLOGY.md — Three-component exercise structure providing detection context

---

Table of Contents

---

1. Forensic Data Available in Anki Databases

Every `.apkg` submission contains a SQLite database with tables that carry rich fingerprinting data. The following fields are relevant for cheating detection.

1.1. `notes` Table

Field	Type	Forensic Value
`id`	INTEGER	Epoch millisecond timestamp of note creation. Unique per student for self-created cards. For shared decks, preserves the original author's timestamp.
`guid`	TEXT	10-character UUID fragment, randomly generated at creation time. Identical GUIDs across students for self-created content = definitive copying evidence.
`csum`	INTEGER	SHA1 checksum of the front field text, truncated to 32 bits: `int(sha1(front_text), 16) & 0xFFFFFFFF`. Two students independently creating the same front text will produce the same csum, so this field alone has limited value — the combination `(id, guid, csum)` is the meaningful fingerprint.
`mid`	INTEGER	Model (note type) ID. StudyAmigo uses a fixed value when creating cards through the app. Externally imported decks carry different model IDs.

Field	Type	Forensic Value
`mod`	INTEGER	Modification timestamp in seconds. Reflects when the note was last edited.
`flds`	TEXT	Raw field content (front and back), separated by U+001F. Direct content comparison.

Code reference: `server/app.py` — csum calculation uses `hashlib.sha1`, GUID generation uses `uuid.uuid4()`, note IDs use `int(time.time() * 1000)`.

1.2. `cards` Table

Field	Type	Forensic Value
`id`	INTEGER	Epoch millisecond timestamp. In StudyAmigo, `card_id = note_id + 1`.
`did`	INTEGER	Deck ID, generated as `int(time.time() * 1000)` at deck creation. For shared decks, this is assigned at import time and differs between students.
`ivl`	INTEGER	Current interval in days. Reflects individual review history.
`factor`	INTEGER	Ease factor × 1000. Reflects individual performance pattern.
`reps`	INTEGER	Total review count.
`lapses`	INTEGER	Number of times the card was forgotten.

1.3. `col` Table

Field	Type	Forensic Value
`crt`	INTEGER	Collection creation timestamp in seconds. Set when the user's Anki profile is created. Each student has a unique `crt` value — matching values between two students is conclusive evidence of database copying.
`mod`	INTEGER	Last modification timestamp.
`decks`	TEXT	JSON containing all deck definitions with their IDs, names, and modification times.
`models`	TEXT	JSON containing note type definitions.

1.4. `revlog` Table

The review log is the **ultimate behavioral fingerprint**. Even when two students import the same shared deck, their review logs are completely different.

Field	Type	Forensic Value
`id`	INTEGER	Millisecond-precision timestamp of the review event.
`cid`	INTEGER	Card ID that was reviewed.
`ease`	INTEGER	Student's response: 1=Again, 2=Hard, 3=Good, 4=Easy.
`time`	INTEGER	Time spent on the review in milliseconds.
`ivl`	INTEGER	New interval assigned after this review.
`lastIvl`	INTEGER	Previous interval before this review.

Field	Type	Forensic Value
`type`	INTEGER	0=learn, 1=review, 2=relearn, 3=cram.

Code reference: `server/app.py` — revlog insertion uses `int(time.time() * 1000)` for the review timestamp.

1.5. File-Level Hash

The MD5 or SHA256 hash of the entire `.anki2` database file. Because review logs, card scheduling state, and deck IDs always differ between students — even for the same shared deck — **identical file hashes between two students constitute 100% proof of raw file copying**.

---

2. The Critical Distinction: Shared Decks vs. Student-Created Decks

This is the most important concept in the entire detection system. Without understanding this distinction, any note-level comparison will produce massive false positives.

2.1. Student-Created Decks: Every Fingerprint is Unique

When a student creates flashcards through the StudyAmigo application or Anki:

- `notes.id` is the current epoch millisecond — reflects when the student actually typed the card.
- `notes.guid` is a fresh UUID fragment.
- `notes.csum` is the SHA1 of whatever front text the student typed.
- `cards.id` is unique (typically `note_id + 1`).
- `cards.did` is the deck ID the student was working in.
- All `revlog` entries reflect the student's actual study sessions.

Therefore: If two students' databases contain notes with the same `notes.id` or `notes.guid` for student-created content, this is strong evidence of copying.

Verified from actual submissions: Comparing Rayssa's and Joao Vitor's GENETIC_ENGINEERING decks (both from E05), every note ID, GUID, csum, card ID, model ID, and deck ID was completely different. Zero overlap on any fingerprint.

2.2. Shared/Public Decks: Note Fingerprints are Identical by Design

When students import the same public deck (such as "1000 Basic English Words"):

- `notes.id`, `notes.guid`, `notes.csum`, and `cards.id` are **preserved from the original deck author**. For "1000 Basic English Words," the note IDs date to April 2019. These values are identical across every student who imports the deck.
- `cards.did` **differs** — assigned at import time, unique to each student.
- All `revlog` entries are **completely different** — each student's review timestamps, ease ratings, and time-spent are unique.

Verified from actual submissions: Comparing Rayssa's and Edina's "1000 Basic English Words" submissions for E05 — 960 notes with identical IDs, GUIDs, and csums. But deck IDs differed, and their 1,170 vs. 1,817 review log entries were entirely different.

2.3. Decision Tree: Classifying Notes as Shared vs. Student-Created

Before performing any cross-student comparison, classify every note in a database:

```
For each note in the database:
│
├─ Is note.id in the Known Shared Deck Registry?
```

```
│   └─ YES → Label as SHARED. Skip in cross-student comparison.
│
├─ Does note.mid differ from the StudyAmigo default model ID?
│   └─ YES → Label as IMPORTED (external source).
│            Check against registry of known public decks.
│
├─ Is the note.id timestamp before the course start date?
│   └─ YES → Label as IMPORTED (pre-dates course).
│
└─ note.id is within course dates AND note.mid matches StudyAmigo
    └─ Label as STUDENT-CREATED. Include in cross-student comparis
```

Only notes labeled STUDENT-CREATED should be compared across students for cheating detection.

---

3. Detection Methods by Cheating Type

3.1. Type 1 — Raw Database Copying

Description: A student submits another student's entire `.apkg` file as their own, either unmodified or with trivial changes (e.g., renaming the file).

Detection Signals:

Signal	Detection Method	Confidence
Identical file hash	MD5/SHA256 of the `.anki2` database file	**100%** — absolute proof
Identical `col.crt`	Compare collection creation timestamps across all students	**100%** — two students cannot register at the exact same second
Identical deck ID set	Compare the set of all `cards.did` values	**Very high** — deck IDs are epoch-millisecond

Signal	Detection Method	Confidence
		timestamps
Identical `revlog` entries	Compare first 10 revlog records (timestamps, ease, time)	**100%** — millisecond timestamps cannot coincide by chance

Implementation: This is the first and simplest check. Compare file hashes pairwise across all submissions for an exercise. Any match is conclusive. Then compare `col.crt` values as a secondary confirmation.

3.2. Type 2 — Content Copying (Student-Created Cards)

Description: A student copies another student's flashcard content into their own database — perhaps by importing the other student's exported deck, or by re-entering their content.

Detection Signals:

Signal	Detection Method	Confidence
Matching `notes.id` on student-created notes	Compare note IDs after filtering shared deck notes	**Very high** — note IDs are epoch-millisecond timestamps; two students cannot create notes at the same millisecond
Matching `notes.guid` on student-created notes	Compare GUIDs after filtering	**Very high** — UUIDs are randomly generated
Matching `notes.csum`	Same front text on multiple student-created cards	**Medium** — coincidence possible

Signal	Detection Method	Confidence
+ identical `notes.flds`		for common vocabulary
Identical card creation timestamp clusters	Two students' note creation timestamps within seconds of each other	**High**

Important: A single matching `csum` is not sufficient evidence — two students independently creating a card with front text "Hello" will produce the same csum. The combination of `(csum, id, guid)` all matching, or identical `flds` content across 5+ cards, increases confidence.

Implementation: For each exercise, extract all student-created notes (using the decision tree from Section 2.3). Build `(note_id, guid)` tuple sets per student. Compute pairwise intersections. Any intersection > 0 for student-created notes is suspicious. An intersection of 5+ notes is strong evidence.

3.3. Type 3 — Review Fabrication / Not Actually Studying

Description: A student creates cards (or imports them) but does not genuinely review them. They click through reviews as fast as possible to generate review counts without actual learning.

Detection Signals:

Signal	Threshold	Confidence
All ease = 4 ("Easy")	> 90% of reviews rated "Easy"	**High** — genuine ESL learners show a mix of ratings

Signal	Threshold	Confidence
Very short review times	Average `revlog.time` < 2,000 ms (2 seconds)	**High** — cannot meaningfully read and recall in under 2 seconds
All reviews in a single session	All `revlog.id` timestamps within a 5-minute window	**Medium-High** — cramming is not cheating per se, but reviewing 50+ cards in under 2 minutes is
Zero lapses	`cards.lapses = 0` on 100% of reviewed cards	**Medium** — possible for very easy material, suspicious for ESL students on English vocabulary
Sustained review rate > 1 card per 3 seconds	Total reviews / total time span	**High** — physically impossible to study genuinely
Review time has no variance	Standard deviation of `revlog.time` < 500 ms	**High** — genuine review shows variable engagement depending on card difficulty

Real example from E05 submissions: One student reviewed 10 GENETIC_ENGINEERING cards with 100% ease=4 ratings, all within a 23-second window (average 2.3 seconds per card). Compare this to another student who reviewed 21 cards with 115 total reviews over multiple days, with ease ratings of 1, 2, 3, and 4, showing genuine spaced repetition engagement. The `revlog` table makes this distinction trivially detectable.

3.4. Type 4 — AI-Generated Cards Submitted as Original Work

Description: A student uses ChatGPT, Claude, or similar to generate flashcard content from the reading passage instead of creating cards through their own engagement with the text.

Detection Signals:

Signal	Detection Method	Confidence
Card creation speed anomaly	Cards created faster than physically possible (e.g., 20 complex cards in 2 minutes) — measured from gaps between consecutive `notes.id` timestamps	**Medium-High**
Stylistic uniformity	All cards follow the same syntactic template (e.g., all start with “What is…”, all backs follow identical format with emoji headers)	**Medium**
Vocabulary above student’s level	Card content uses vocabulary or grammar beyond what the student demonstrates in their review performance	**Medium**
Suspiciously comprehensive coverage	Cards systematically cover every paragraph of the passage, unlike typical student selective extraction	**Low-Medium**
Absence of L1 influence	No Portuguese-influenced phrasing, spelling, or word choice that would be expected from ESL students at this level	**Medium**

Implementation: This is the hardest cheating type to detect automatically and may require:

1. **Card creation speed analysis**: Compute the time gap between consecutive `notes.id` values. A student typing cards manually will show gaps of 30 seconds to several minutes. A student pasting AI-generated content will show gaps of 3–5 seconds (just enough to paste and confirm).
2. **Template detection**: NLP analysis of card content structure. If 15 out of 15 cards start with an emoji header, include a "Resumo:" section, list "Vocabulário importante:" with bullet points, and end with "Ponto gramatical relevante:" — that is a generated template, not spontaneous student work.
3. **Cross-referencing quality with performance**: A student who creates graduate-level cards with perfect grammar but fails at basic vocabulary review has a credibility gap.

Connection to exercise structure: Under the three-component model from READING_MATERIAL_METHODOLOGY.md, Component B cards (tiered passage) have a known source text. Card content can be compared against the source passage — suspiciously comprehensive coverage (every sentence of the passage turned into a card) suggests automated generation.

3.5. Type 5 — Shared Deck Review Without Engagement

Description: A student imports a shared deck and clicks through reviews mindlessly to generate review counts without actual learning effort.

Detection Signals:

Signal	Threshold	Confidence
Very short review times	Average `revlog.time` < 2,000 ms on shared deck cards	**High**

Signal	Threshold	Confidence
Uniform ease rating	All ease = 4, or suspiciously random distribution (25% each of 1, 2, 3, 4)	**Medium-High**
No relearning events	`revlog.type` never equals 2 (relearn) despite hundreds of reviews	**Medium**
No failed reviews	Zero `ease = 1` across hundreds of reviews of English vocabulary	**Medium** — suspicious for ESL students
Review time has no variance	Standard deviation of `revlog.time` < 500 ms across 50+ reviews	**High** — genuine review produces variable engagement

This type overlaps with Type 3 but is specific to the shared deck component. It is less severe than content copying but still undermines the learning objective.

3.6. Type 6 — Cross-Exercise Inconsistency

Description: Different people complete different exercises for the same student account, or a student outsources specific exercises to someone else.

Detection Signals:

Signal	Detection Method	Confidence
Wildly different card creation styles	NLP comparison of card content across exercise windows (e.g., E03 cards are simple word pairs, E05 cards are elaborate formatted entries)	**Medium**
Study time pattern shifts	Student studies at night for E01–E03 but mornings for E04–E06	**Low-Medium**
Competency jumps	Zero activity in E04, then suddenly high-quality activity in E05 with a different card style	**Medium**
Different model IDs across exercises	Cards from different exercises use different note types without explanation	**Medium-High**
Deck naming convention changes	Consistent naming in early exercises, completely different style later	**Low**

Implementation: This requires the temporal/behavioral analysis infrastructure from STUDENT_PERFORMANCE_ANALYSIS.md Section 6. Compute per-exercise behavioral profiles (study time distribution, card creation speed, review time patterns, ease distributions) and measure pairwise distance between exercises for each student. Flag students whose inter-exercise variability exceeds a threshold.

---

4. Confidence Levels and Evidence Hierarchy

4.1. Evidence Tiers

Tier	Confidence	Signals
Tier 1 — Conclusive	> 95%	Identical file hash; identical `col.crt`; identical `revlog` timestamps
Tier 2 — Strong	70–95%	Matching note IDs/GUIDs on student-created notes; physically impossible review speed (> 1 card per 2 seconds for 50+ cards); physically impossible card creation speed
Tier 3 — Moderate	40–70%	> 90% ease=4 ratings; zero lapses on all cards; multiple csum matches on student-created content; stylistic uniformity suggesting AI generation
Tier 4 — Weak	< 40%	Single csum match; behavioral changes between exercises; study time pattern shifts; high cramming ratio

4.2. Composite Scoring

Multiple weak signals together can elevate confidence. A simple scoring model:

Signal Tier	Points per Signal
Tier 1	+100 (auto-flag)
Tier 2	+30
Tier 3	+10
Tier 4	+3

Total Score	Action
≥ 100	Conclusive evidence. Confront the student with specific data.
70–99	Strong suspicion. Review the evidence manually before confrontation.
40–69	Worth investigating. Pull the student's full timeline and compare with flagged peer.
< 40	Insufficient evidence. Note for monitoring in future exercises.

4.3. One Tier 1 Signal is Enough

Any single Tier 1 signal — identical file hash, identical `col.crt`, or identical revlog timestamps — is conclusive on its own. No composite scoring is needed. The composite model is designed for cases where only Tier 2–4 signals are present.

---

5. Implementation Recommendations

5.1. Phase 1 — File-Level Checks (Immediate, Low Effort)

Build a script that:

1. Accepts a directory of `.apkg` files for an exercise.
2. Extracts the `.anki2` (or `.anki21`) SQLite database from each ZIP archive.
3. Computes MD5 and SHA256 hashes of each database file.
4. Compares all pairs. Flags exact matches.
5. Extracts `col.crt` from each database. Flags matches.
6. Outputs a collision report.

This alone catches the most blatant cheating (raw database copying) with zero false positives.

5.2. Phase 2 — Note-Level Comparison (High Priority, Medium Effort)

Extend the Phase 1 script to:

1. For each student database, classify notes as "student-created" vs. "shared deck" using the decision tree from Section 2.3.
2. For student-created notes, extract `(notes.id, notes.guid, notes.csum, notes.flds)`.
3. Compute pairwise intersections between all student pairs on `(id, guid)`.
4. Flag pairs with any intersection for student-created notes.
5. Report the matching notes with their content for instructor review.

5.3. Phase 3 — Review Pattern Analysis (Medium Priority, Medium Effort)

Extend the analysis to review behavior:

1. For each student, compute:

    1. Average review time (`AVG(revlog.time)`)
    2. Ease distribution (percentage of each rating 1–4)
    3. Review time variance (`STDEV(revlog.time)`)
    4. Lapse rate
    5. Session duration (time span from first to last review)
    6. Reviews per second rate

2. Flag students whose profiles exceed the thresholds defined in Section 3.3 and 3.5.

3. Generate a per-student review behavior summary.

5.4. Phase 4 — AI-Generated Content Detection (Lower Priority, Higher Effort)

This requires more sophisticated analysis:

1. Card creation speed analysis from `notes.id` timestamp gaps.
2. Content template detection (structural uniformity across cards).
3. Cross-referencing card creation quality with review performance.
4. Comparison of card content against known source passages (for Component B exercises).

5.5. Script Architecture

Build a single Python script `server/cheating_detection.py`, following the pattern established by `server/generate_user_timeline.py`:

- Takes a directory of `.apkg` files and an exercise identifier as input.
- Produces a structured report (Markdown or JSON) with:
    - File-level collision matrix
    - Note-level collision pairs (after shared deck filtering)
    - Per-student review behavior anomaly flags

    - Composite risk scores
- Has configurable thresholds for all detection signals.
- Reuses the database access patterns from `generate_user_timeline.py`.

---

6. The Shared Deck False Positive Problem

This section addresses the single most common error an automated detection system — or an instructor doing manual checks — would make.

6.1. The Problem

The "1000 Basic English Words" deck (and the planned `SHARED_CURATED_vX` deck from READING_MATERIAL_METHODOLOGY.md) produces identical `notes.id`, `notes.guid`, and `notes.csum` values across **every student** who imports it. A naive note-comparison algorithm will flag every pair of students as cheaters — producing `n * (n-1) / 2` false positives for a class of `n` students.

For a class of 48 students, that is **1,128 false positive pairs**.

6.2. Known Shared Deck Registry

Maintain a registry of known shared decks with their identifying characteristics:

Deck Name	Identifying Signature	Origin
1000 Basic English Words	Note IDs dating to ~April 2019 (`notes.id` ≈ `1556033745xxx`); model ID `1555424123593`	Public Anki deck
Verbal Tenses	StudyAmigo sample deck; deck ID = 2; known note ID range	Shipped with StudyAmigo

Deck Name	Identifying Signature	Origin
`SHARED_CURATED_vX`	Instructor-distributed; note IDs pre-recorded at deck creation time	Course material

When creating and distributing the curated shared deck (Component A), **record all note IDs at creation time** and add them to this registry. This eliminates false positives from the start.

6.3. Filtering Algorithm

```
STUDYAMIGO_MODEL_ID = 1700000000001  # Default model ID in StudyAm
KNOWN_SHARED_NOTE_IDS = set()        # Populated from registry

def classify_note(note, course_start_epoch_ms):
    """Classify a note as 'shared' or 'student_created'."""

    # Check against known shared deck registry
    if note.id in KNOWN_SHARED_NOTE_IDS:
        return 'shared'

    # Check model ID — external imports use different models
    if note.mid != STUDYAMIGO_MODEL_ID:
        return 'shared'  # Imported from external source

    # Check timestamp — pre-course notes are imported
    if note.id < course_start_epoch_ms:
        return 'shared'

    # Note was created during the course using StudyAmigo
    return 'student_created'


def compare_students(student_a_notes, student_b_notes, course_star
    """Compare two students' notes for copying, excluding shared c
```

```
    a_created = {n for n in student_a_notes if classify_note(n, co
    b_created = {n for n in student_b_notes if classify_note(n, co

    # Compare on (id, guid) tuples
    a_fingerprints = {(n.id, n.guid) for n in a_created}
    b_fingerprints = {(n.id, n.guid) for n in b_created}

    collisions = a_fingerprints & b_fingerprints

    if len(collisions) > 0:
        return {
            'flagged': True,
            'collision_count': len(collisions),
            'colliding_notes': collisions
        }

    return {'flagged': False}
```

6.4. Edge Cases

Edge Case	What Happens	How to Handle
Student creates a card with the same front text as a shared deck card	Same `csum` but different `id` and `guid`	Use `(id, guid)` as primary keys, not `csum` alone
Student exports their own deck and another student imports it	Matching `notes.id` and `notes.guid` with the source student's timestamps	Detectable: note timestamps will be from the source student's activity window, not the importer's
Instructor distributes a deck	All students have identical note IDs for Component A cards	Pre-record note IDs when creating the deck; add to exclusion registry

Edge Case	What Happens	How to Handle
to all students (Component A)		
Two students independently create a card "What is DNA?"	Same `csum`, same `flds`, but different `id` and `guid`	Not flagged by `(id, guid)` comparison. If content comparison is used, require 5+ matching cards to flag
Student creates cards in Anki desktop then imports to StudyAmigo	`notes.mid` differs from StudyAmigo default	Classified as "imported" by the model ID check. May need instructor judgment

---

7. Integration with the Data Pipeline

7.1. Performance Metrics as Cheating Signals

Several metrics already defined in STUDENT_PERFORMANCE_ANALYSIS.md serve double duty as cheating indicators:

Performance Metric	Normal Use	Cheating Detection Use
Retention rate (§5.1)	Learning effectiveness	Suspiciously perfect retention (100%) on difficult material
Average review time (§5.2)	Engagement depth	Very short times indicate mindless clicking

Performance Metric	Normal Use	Cheating Detection Use
Lapse rate (§5.3)	Long-term retention	Zero lapse rate as supporting evidence for fabrication
Cramming ratio (§6.3)	SRS compliance	Context for review fabrication detection
Session distribution (§6.1)	Study habits	Cross-exercise consistency checks
Ease factor distribution (§5.5)	Card difficulty	Uniform ease=4 detection

7.2. Exercise Structure Provides Context

The three-component exercise structure from READING_MATERIAL_METHODOLOGY.md provides critical context for detection:

Component	What the Instructor Knows	Detection Advantage
A (Shared Deck)	Exact card set — note IDs are pre-recorded	Review patterns are the only distinguishing data. Focus on Type 3 and Type 5 detection.
B (Tiered Passage)	Exact source text the student read	Card content can be evaluated against the passage. Enables Type 4 (AI generation) detection by checking for suspiciously

Component	What the Instructor Knows	Detection Advantage
		comprehensive coverage.
C (Free Choice)	Nothing — student selects their own material	Hardest to detect AI generation. Note-level and review-level checks still apply. Source URL submission requirement provides some accountability.

7.3. Per-Exercise Reporting

For each exercise submission window, the cheating detection system should produce:

1. **File-level collision report** — any matching file hashes or `col.crt` values.
2. **Note-level collision report** — student-created note matches after shared deck filtering.
3. **Review behavior anomaly report** — students exceeding Type 3 or Type 5 thresholds.
4. **Per-student composite risk score** — combining all signals using the scoring model from Section 4.2.

7.4. Semester-Level Pattern Tracking

Across exercises, track whether a student is repeatedly flagged:

- A single anomalous exercise may be explainable (bad day, time pressure, technical issues).
- Repeated flags across E01–E09 are much harder to dismiss.
- A student flagged for Type 3 (review fabrication) in 3+ exercises warrants a conversation regardless of the individual confidence level.

---

8. Worked Examples

8.1. Scenario: Two Students with Identical Files

Situation: Students Alice and Bob submit `.apkg` files for E05.

Detection:

```
Step 1: File hash comparison
  Alice.anki2  MD5 = 9b6913c366c50aa54668ef62200eab41
  Bob.anki2    MD5 = 9b6913c366c50aa54668ef62200eab41
  → MATCH. Identical file. Tier 1 signal. Score: +100.

Step 2: Confirm with col.crt
  Alice col.crt = 1739602800  (Feb 15, 2025)
  Bob   col.crt = 1739602800  (Feb 15, 2025)
  → MATCH. Same collection creation time. Tier 1 signal. Score: +1

Step 3: Revlog comparison (first 3 entries)
  Alice: (1746569602579, ease=3, time=7460), (1746569616303, ease=
  Bob:   (1746569602579, ease=3, time=7460), (1746569616303, ease=
  → MATCH. Identical review history. Tier 1 signal. Score: +100.

Conclusion: Bob submitted Alice's database file. Total score: 300.
            Conclusive evidence. No ambiguity.
```

8.2. Scenario: Student Copied Another's Self-Created Cards

Situation: Students Carol and Dave both submit GENETIC_ENGINEERING decks for E05. Carol created hers during the exercise window. Dave imported Carol's export.

Detection:

```
Step 1: File hash comparison
  Carol.anki2  MD5 = 4c145174799ceb827792b8cb516f34af
  Dave.anki2   MD5 = e6d327a73e9852523f35419f34054e0d
  → No match (Dave has different revlog entries). File-level check

Step 2: Note classification
  Carol: 21 notes. Model ID = StudyAmigo default. Timestamps in E0
         → All 21 classified as STUDENT-CREATED.
  Dave:  21 notes. Model ID = StudyAmigo default. Timestamps in E0
         → All 21 classified as STUDENT-CREATED.

Step 3: Pairwise (id, guid) comparison on student-created notes
  Carol notes: {(1746567923374, 'jTr=Z0$F^2'), (1746567985772, 'qL
  Dave notes:  {(1746567923374, 'jTr=Z0$F^2'), (1746567985772, 'qL
  → 21 out of 21 notes match. Tier 2 signal. Score: +30.
  → Intersection size 21 is far above the threshold of 5.
     Confidence elevated to near-Tier-1.

Step 4: Timestamp analysis
  Carol's notes were created May 6, 2025 18:45–19:10 (25-minute sp
  Dave's notes have IDENTICAL creation timestamps (same May 6 windo
  Dave registered his Anki profile on a different date (col.crt di
  → Dave imported Carol's deck. The note timestamps are Carol's, n

Conclusion: Dave imported Carol's exported deck as his own work.
            Score: 30+ but collision count (21/21) makes this effe
```

8.3. Scenario: False Positive from Shared Deck

Situation: Students Eve and Frank both submit “1000 Basic English Words” decks. A naive comparison flags them.

Naive detection (WRONG):

```
Step 1: Note comparison without filtering
  Eve:   960 notes
  Frank: 960 notes
  Intersection on (id, guid): 960 matches!
  → ALERT: 960 matching notes! Cheating detected!

This is WRONG. Both students imported the same public deck.
```

Correct detection:

```
Step 1: Note classification
  Eve:   960 notes. Model ID = 1555424123593 (NOT StudyAmigo defau
         Note IDs date to April 2019 (before the course).
         → All 960 classified as SHARED. Zero student-created note
  Frank: 960 notes. Same model ID and timestamp range.
         → All 960 classified as SHARED. Zero student-created note

Step 2: Pairwise comparison on student-created notes
  Eve student-created: {} (empty set)
  Frank student-created: {} (empty set)
  → Intersection: 0. No flag raised.

Step 3: Review behavior comparison (separate check)
  Eve:   1,170 reviews over 14 days. Average time: 8.3 seconds. Ea
         12% Again, 18% Hard, 45% Good, 25% Easy.
  Frank: 890 reviews over 11 days. Average time: 6.1 seconds. Ease
         8% Again, 22% Hard, 50% Good, 20% Easy.
  → Both show legitimate review patterns. No anomaly flags.

Conclusion: No cheating detected. Both students legitimately impor
            studied the same public deck. The shared deck filter e
            all 960 false positive matches.
```

8.4. Scenario: Student Clicking Through Reviews

Situation: Student Greg submits an E05 database with 10 GENETIC_ENGINEERING cards and 10 reviews.

Detection:

```
Step 1: Review behavior analysis

  SELECT
      COUNT(*) as total_reviews,
      AVG(time) as avg_time_ms,
      MIN(id) as first_review,
      MAX(id) as last_review,
      (MAX(id) - MIN(id)) / 1000.0 as span_seconds
  FROM revlog;

  Result: 10 reviews, avg time 2300ms, span 23 seconds.

  Ease distribution:
  SELECT ease, COUNT(*) FROM revlog GROUP BY ease;
  → ease=4: 10 (100%)

  Relearning check:
  SELECT COUNT(*) FROM revlog WHERE type = 2;
  → 0 relearning events.

Step 2: Flag evaluation
  ✓ Average review time 2.3 seconds (< 3 second threshold)     → T
  ✓ 100% ease=4 (> 90% threshold)                              → T
  ✓ All reviews in 23-second window (single session)            →
  ✓ Zero lapses                                                 →
  ✓ Zero relearning events                                      →

  Composite score: 56. Action: "Worth investigating."

Step 3: Compare with a legitimate student for context
  Student Hannah, same deck, same exercise:
  21 cards, 115 reviews, over 5 days, avg time 11.2 seconds,
```

```
  ease distribution: 5% Again, 15% Hard, 55% Good, 25% Easy.

  The contrast is stark. Greg did not study. Hannah did.

Conclusion: Greg clicked through all reviews in under 30 seconds w
            engaging with the material. Score 56 — investigate, ar
            behavioral evidence is clear upon inspection.
```

Appendix A: SQL Query Reference

A.1. Extract Collection Fingerprint

```
-- Collection creation time and modification time
SELECT crt, mod, scm FROM col;
```

A.2. Extract All Note Fingerprints

```
-- All notes with fingerprinting fields
SELECT id, guid, mid, mod, csum, sfld FROM notes ORDER BY id;
```

A.3. Extract Student-Created Notes (Filtering Shared Decks)

```
-- Notes created during the course window with StudyAmigo model ID
-- Replace COURSE_START_MS and STUDYAMIGO_MID with actual values
SELECT id, guid, csum, flds, sfld
FROM notes
WHERE id > :course_start_ms
  AND mid = :studyamigo_model_id
ORDER BY id;
```

A.4. Extract Deck Structure

```
-- All unique deck IDs used by cards
SELECT DISTINCT did FROM cards;

-- Deck definitions from collection metadata
SELECT decks FROM col;
```

A.5. Review Behavior Summary

```
-- Per-student review behavior profile
SELECT
    COUNT(*) as total_reviews,
    AVG(time) as avg_time_ms,
    MIN(time) as min_time_ms,
    MAX(time) as max_time_ms,
    SUM(CASE WHEN ease = 1 THEN 1 ELSE 0 END) as again_count,
    SUM(CASE WHEN ease = 2 THEN 1 ELSE 0 END) as hard_count,
    SUM(CASE WHEN ease = 3 THEN 1 ELSE 0 END) as good_count,
    SUM(CASE WHEN ease = 4 THEN 1 ELSE 0 END) as easy_count,
    SUM(CASE WHEN ease = 4 THEN 1.0 ELSE 0 END) / COUNT(*) * 100 a
    SUM(CASE WHEN type = 2 THEN 1 ELSE 0 END) as relearn_count,
    MIN(id) as first_review_ms,
    MAX(id) as last_review_ms,
    (MAX(id) - MIN(id)) / 1000.0 as total_span_seconds,
    COUNT(*) * 1000.0 / NULLIF(MAX(id) - MIN(id), 0) as reviews_pe
FROM revlog;
```

A.6. Review Behavior Per Deck

```
-- Breakdown by deck (shared vs. student-created)
SELECT
    c.did,
    COUNT(*) as total_reviews,
    AVG(r.time) as avg_time_ms,
    SUM(CASE WHEN r.ease = 4 THEN 1.0 ELSE 0 END) / COUNT(*) * 100
```

```
    SUM(CASE WHEN r.ease = 1 THEN 1 ELSE 0 END) as again_count,
    SUM(CASE WHEN r.type = 2 THEN 1 ELSE 0 END) as relearn_count
FROM revlog r
JOIN cards c ON r.cid = c.id
GROUP BY c.did;
```

A.7. Card Creation Speed Analysis

```
-- Time gaps between consecutive card creations (for AI-generatior
SELECT
    n1.id as note_id,
    n1.sfld as front_text,
    n1.id - LAG(n1.id) OVER (ORDER BY n1.id) as gap_ms
FROM notes n1
WHERE n1.id > :course_start_ms
ORDER BY n1.id;
```

A.8. Revlog Signature (First N Entries)

```
-- Extract review sequence signature for cross-student comparison
SELECT id, cid, ease, ivl, time, type
FROM revlog
ORDER BY id
LIMIT 20;
```

Appendix B: `.apkg` File Format

An `.apkg` file is a standard ZIP archive containing:

File	Description
`collection.anki21`	Main SQLite database (Anki 2.1+ format). This is the primary database to analyze.

File	Description
`collection.anki2`	Legacy SQLite database (older Anki format). May or may not be present.
`media`	JSON file mapping numeric keys to media filenames. Often `{}` (empty) for text-only decks.
`0`, `1`, `2`, …	Optional media files (images, audio) referenced by the `media` manifest.
`meta`	Optional metadata file.

Extraction:

```
import zipfile
import tempfile
import os

def extract_anki_db(apkg_path):
    """Extract the SQLite database from an .apkg file."""
    with tempfile.TemporaryDirectory() as tmpdir:
        with zipfile.ZipFile(apkg_path, 'r') as z:
            z.extractall(tmpdir)

        # Prefer .anki21 (newer format), fall back to .anki2
        db_path = os.path.join(tmpdir, 'collection.anki21')
        if not os.path.exists(db_path):
            db_path = os.path.join(tmpdir, 'collection.anki2')

        return db_path  # SQLite database ready for analysis
```