From audit to a production Android kiosk in five days — 168 tests written before the code

A customer-owned device that handles a One Talk customer's own login approvals locally, 24×7, with a tamper-evident audit trail — a first Android build, delivered via supervised AI engineering.

~5 days
From audit to production deploy

168
Unit tests, written before the code

100% local
No backend, no third-party clicker, no LLM

24×7×365
Unattended, with self-recovery

THE PROBLEM

Every employee login triggered a One Talk approval prompt to a designated admin phone: read the message, open the page, click through. After hours and on weekends, approvals piled up while users sat locked out. Third-party auto-clickers trade the availability problem for a security problem — the customer needed their own device handling their own approvals, with a record that would survive an audit.

WHAT WE BUILT

- A dedicated, customer-owned Android device that handles the customer's approval workflow entirely locally.
- Encrypted token storage (AES/GCM, hardware-backed keys) — and a single redactor that every log, UI, and sync path must pass through.
- A tamper-sealed, redacted audit chain recording every approval.
- A verification harness of 168 tests, written before the implementation, gating every build.

HOW IT WAS DELIVERED

Claude Code performed the initial audit and first patches; Codex executed the eight-phase hardening plan. The architecture, privacy boundaries, per-phase quality gates, and deploy runbook stayed human-owned. Audit to production: about five days.

> **The safeguard.** Everything runs locally on customer-owned hardware — no backend, no third-party service. Sensitive values pass through one redactor; no raw URL, SMS body, OTP code, or token ever leaves the device. Every action lands in a tamper-sealed audit chain.

THE RESULT

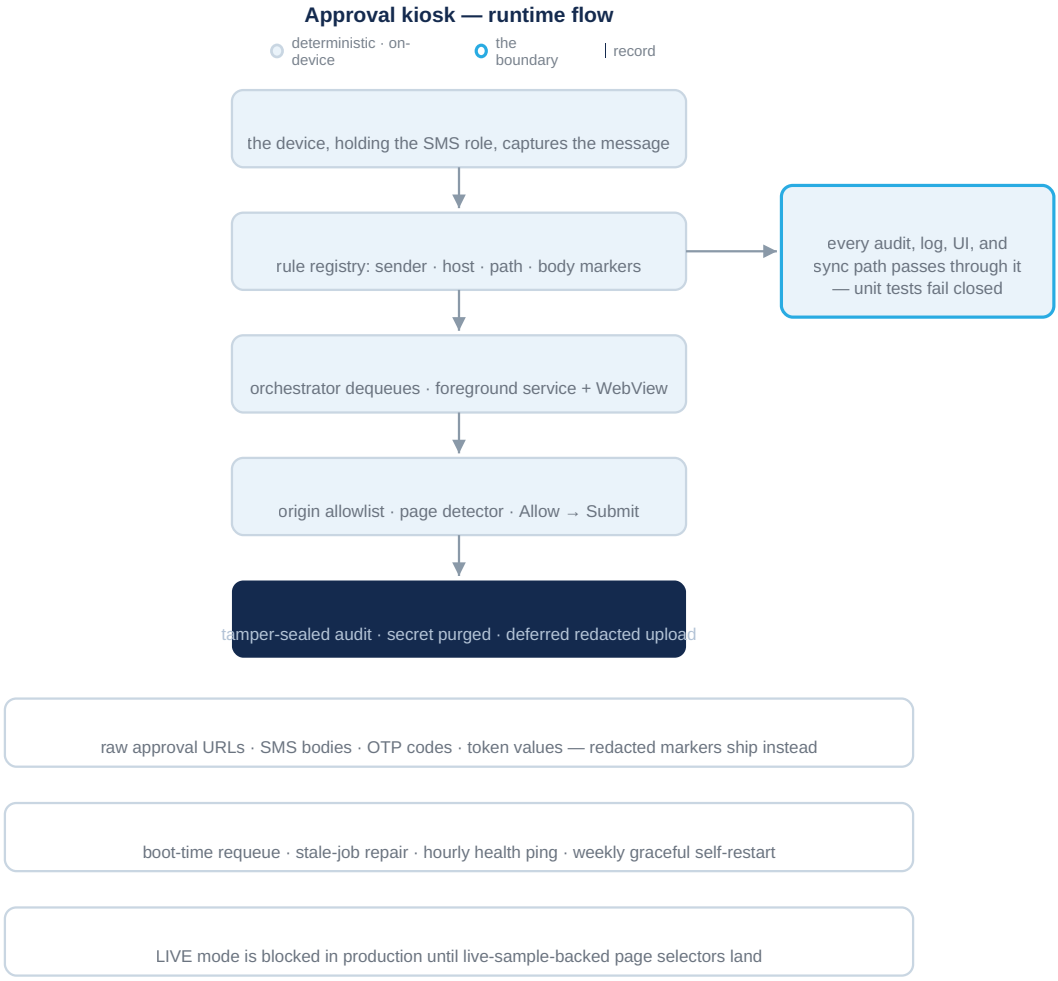
Deployed to one production device, in a 30-day soak: the customer's own approvals handled locally, around the clock, with a redacted audit record for every one — and nobody locked out at 9 p.m. on a Saturday.

Have an approval or operations workflow stuck on one person's manual clicks?

Start with an AI Workflow Teardown →

HOW IT WORKS — APPROVAL FLOW

All approval processing is fully local. Raw approval URLs live only in an encrypted pending-secrets table (AES/GCM, key held in the hardware keystore). The kiosk never sends raw URLs, SMS bodies, OTP codes, or token values off-device; there is no LLM, no cloud inference, and no third-party SDK call anywhere in the approval flow. The only outbound call is a deferred upload of the redacted audit chain.



There is no AI *in* the product — the AI was in the delivery.

Two design choices stand out. **The single redactor:** not a new redaction algorithm — the architectural decision to centralize the boundary, then prove every leak path is bounded by it. **Fail-closed encryption:** raw approval URLs exist only AES/GCM-encrypted with a hardware-keystore key; decryption fails closed on corruption.

DEFENSIVE DESIGN — THE REDACTOR IS THE BOUNDARY

Every value crossing an audit, log, UI, or sync boundary passes through a single sensitive-data redactor. Allowlisted hosts keep host and path with query and fragment redacted; a non-allowlisted host — a hostile redirect — collapses to a fixed marker; OTP-shaped digit runs are stripped from rejected SMS bodies. The contribution wasn't a new redaction algorithm. It was the architectural decision to centralize the boundary — and then proving every leak path was bounded by it.

RELIABILITY AND GUARDRAILS

MECHANISM	WHAT IT DOES
Encrypted pending secrets	AES/GCM with a hardware-keystore-backed key; the raw URL never touches disk in plaintext; decryption fails closed on corruption
Tamper-sealed audit chain	Each audit event hashes with the previous — tampering becomes detectable after the fact
Boot receiver + stale-job repair	On reboot, workers are re-scheduled and stuck jobs requeued; a paged scan tolerates a long backlog
Crash breadcrumbs	A default uncaught-exception handler writes a redacted fatal-event audit row before the OS kills the process — forensics survive process death
Foreground-service wrapper	The approval WebView runs inside a foreground service so Doze and modern background restrictions can't silently kill mid-approval
Weekly graceful self-restart	An idle-constrained process recycle defeats slow accumulators in heap, native memory, and OEM service drift
Health ping + payload caps	An hourly health tick flags a revoked SMS role; bridge payloads are capped and audited on overflow; an app-wide exception handler catches anything unhandled

INTEGRATION STACK

LAYER	TOOL	ROLE
OS	Android 11+ (minSdk 30, targetSdk 35)	Host; SMS role; hardware keystore; alarm + work scheduling
Persistence	AndroidX Room	Six entities: approval jobs, audit events, encrypted pending secrets, runtime flags, inbound messages, selector packs
Scheduling	AndroidX WorkManager	Stale-job repair, audit upload, hourly health ping, weekly self-restart, selector-pack refresh
WebView	AndroidX WebKit	Hardened in-app WebView with message listener and document-start scripts
Encryption	Hardware keystore + AES/GCM	Approval-URL encryption; the key never leaves the secure element
Testing	JUnit + Truth + Robolectric	168 unit tests across staging and prod variants, plus instrumented tests
Deploy & observability	adb + macOS launchd	Manual install; a soak collector and daily checkpoint record memory, role state, crashes, worker firings, restarts

WHAT I BUILT — VS. WHAT THE AGENTS AND TOOLS DID

- **Engineering process** — the audit brief, the multi-phase hardening plan, and the deploy runbook with rollback. Claude Code ran the audit and first patches; Codex executed the eight-phase hardening; the boundaries stayed mine.
- **Carrier rule registry** — hand-curated allowlist of senders, hosts, path patterns, body markers, and OTP shapes, with a release gate blocking LIVE in production until live-sample selectors land.
- **Selector pack + signature** — a versioned, signed bundle of page selectors, updateable without shipping a new APK.
- **Redaction discipline** — one sensitive-data redactor that every audit, log, UI, and sync path routes through, with unit tests that fail closed.
- **State machine + runtime guardrails** — deterministic job transitions, boot-time requeue, duplicate guard, retry policy, crash breadcrumbs, weekly self-restart.
- **Verification harness** — per-phase test / lint / assemble gates before each commit, a device-side smoke test before deploy, and a daily soak checkpoint.

EARLY RESULTS — 30-DAY SOAK IN PROGRESS

METRIC	OBSERVATION
Build gates	168/168 unit tests passing (staging + prod); 0 lint errors; both build variants green
Crashes attributed to the app, first ~18 h	0
Memory at +18 h	86 MB PSS / 7 MB native heap — no observable leak so far
Health pings	Firing hourly, all successful; SMS role held throughout
Live carrier SMS round-trip	Not yet measured — pending a carrier test sender

Shipped to a single production device on 2026-05-03; the numbers above describe the first ~24 hours of unattended operation, not a long-run measurement. A measurement window will replace these with harder data.

TRADEOFFS AND LIMITS

Selector pack ships with placeholder carrier markers. LIVE mode is release-gate-blocked in production until real page samples land; v2 adds a signed remote pack-distribution pipeline.

Audit upload is stubbed; detection has no formal benchmark yet. The local tamper-sealed chain is canonical and operator-exportable; v2 adds a real backend with mutual TLS and a labeled-corpus regression suite.

Single device, single SIM, single user — matching the customer's footprint. Remaining provisioning-stage hardening items are tracked in the private deploy runbook and gated before any second device ships.

CONSULTING TAKEAWAY

This was a first Android project. The approach: treat Claude Code and Codex as supervised implementation labor — hand them the audit, the hardening plan, and the deploy runbook — while owning the architecture, the privacy posture, and the verification gates. Inside the app, the discipline that mattered was a single redactor every path had to pass through. Around the agents, it was a per-phase gate that had to go green before the next phase started.

The portfolio claim isn't "I'm an Android engineer now." It's that systems judgment scales across stacks when implementation is delegated to LLM coding agents — and the human stays accountable for what crosses every boundary.