# Project Simurgh: Privacy-Preserving Device Integrity Proofs for Capture-Resistant High-Stakes Sessions

Mohammad Raouf Abedini
Department of Computing
Macquarie University
Sydney, Australia
mohammadraouf.abedini@students.mq.edu.au
https://raoufabedini.dev

***Abstract*—Project Simurgh is a privacy-preserving integrity architecture for high-stakes AI-mediated and remote assessment sessions. Existing browser-based proctoring systems commonly treat screen-capture output as a faithful representation of the physical display. However, documented operating-system display-affinity behaviours on Windows and macOS show that visible windows can be omitted from capture surfaces, invalidating capture-based trust assumptions [1].**

**Simurgh responds by replacing surveillance-first visual monitoring with signed, metadata-only integrity proofs. The system combines browser behavioural telemetry, local device integrity daemons, cryptographic challenge-response proofs (Ed25519 for the browser-paired integrity-proof envelope; ECDSA P-256 for the cross-platform device-shield daemons), native OS metadata scanners, server-side verification, and HMAC-SHA256-linked tamper-evident audit trails. It does not collect screen pixels, webcam frames, microphone audio, typed content, pasted content, raw process names, raw window titles, window handles, process identifiers, or personal device identifiers. Every anomaly is treated as a signal for human review; no automatic misconduct finding is ever produced.**

**We implement Simurgh as a cross-platform research prototype spanning a Node.js verifier, a browser SDK, and localhost daemons for macOS (Swift/CryptoKit), Windows (.NET), and Linux (Rust). The Windows Device Shield validates detection of `WDA_MONITOR` and `WDA_EXCLUDEFROMCAPTURE` on Windows 10 Pro build 19045. The Linux research path adds signed daemon proofs, X11 window-state metadata coverage, Wayland portal property probing, XWayland partial coverage, display-server mismatch enforcement, and development systemd user lifecycle support. Automated validation includes 331 Node.js unit tests, 33 Rust tests, 11 Windows .NET tests, 8 macOS Swift tests, end-to-end smoke tests, cybersecurity audits, privacy audits, and CI gates.**

**This paper presents Simurgh's threat model, architecture, proof protocol, privacy contract, platform implementations, evaluation results, limitations, and ethical deployment boundaries. We argue that capture-resistant integrity systems should verify signed environment metadata rather than collect more invasive surveillance data.**

***Index Terms*—academic integrity, device attestation, display affinity, Ed25519, ECDSA P-256, integrity proofs, online proctoring, privacy-preserving security, screen capture evasion**

LLM-assisted development methodology disclosed in Section VIII-F. Preprint v1.0; project implementation baseline: Project Simurgh v0.4.18. DOI: 10.5281/zenodo.20374849.

## I. Introduction

Remote proctoring systems that monitor examinations through the WebRTC `getDisplayMedia()` API [2] rest on a single implicit assumption: the captured frame faithfully represents the physical display. Prior work has shown this assumption is false [1]. Both Windows and macOS expose documented, user-level OS APIs that exclude application windows from all screen-capture pipelines while leaving them fully visible on the physical monitor. Commercial AI-overlay tools already exploit these APIs to provide live AI assistance during examinations and technical interviews, invisible to any capture-based monitoring system [3]–[5].

The defensive response cannot simply be "capture more." Existing proctoring approaches already collect webcam feeds, gaze streams, keystroke content, audio, and full-screen video, imposing well-documented psychological costs [6] and raising serious privacy concerns [7], [8]. Adding hardware capture cards or GPU-level frame hooks compounds the privacy problem while still being defeatable by sufficiently motivated adversaries.

*Project Simurgh* takes a different path. Rather than attempting to close the gap between the physical display and the capture surface, Simurgh abandons that surface entirely as a trust anchor. Instead, it establishes session integrity through signed, metadata-only device proofs: the local integrity daemon attests to its own identity and environment state using ECDSA P-256 over SHA-256 [9], while the browser-paired integrity-proof envelope uses Ed25519 [10]; the server verifies each proof against the session's registered key; and every accept and reject event is written to an HMAC-SHA256-linked audit chain [11] that any inspector can verify offline. No screen pixels, webcam frames, audio, typed content, pasted content, raw process names, raw window titles, or personal device identifiers are collected at any layer.

This paper makes the following contributions:

- **Privacy-preserving integrity architecture**: We design and implement a system that replaces visual surveillance with signed metadata-only proofs, demonstrating that session integrity can be assessed without collecting any content-level data.

- **Cross-platform research prototype**: We implement Simurgh as a working system spanning a browser telemetry SDK, a Node.js verifier, and localhost integrity daemons for macOS (Swift/CryptoKit), Windows (.NET), and Linux (Rust), unified under a common proof contract.
- **OS-level metadata detection without screen capture**: We show that the display-affinity risk class documented in [1] can be detected through native OS metadata scanners — capture-exclusion counts on Windows and macOS, window-manager state metadata on Linux — without ever accessing window content, titles, or identifiers.
- **Evaluated prototype with manual-review-only decision model**: We evaluate the system through 383 automated tests across four language runtimes (Node.js, Rust, .NET, and Swift), end-to-end smoke tests, cybersecurity and privacy audits, and real-device Windows validation, while preserving the invariant that every anomaly signal requires a human reviewer to reach any integrity conclusion.

The remainder of this paper is organised as follows. Section II motivates the capture-fidelity failure and privacy problems with existing proctoring. Section III presents the threat model and design goals. Section IV describes the system architecture. Section V details the proof protocol. Section VI covers platform implementations. Section VII specifies the privacy model. Section VIII reports the evaluation. Section IX analyses security properties and remaining attacks. Section X addresses ethics and deployment limits. Section XI surveys related work. Section XII discusses future directions. Section XIII concludes.

## II. Motivation: The Capture-Fidelity Failure

### A. The Invisible Window Attack

The motivating vulnerability for this work is the Invisible Window attack class, formally documented in [1]. Both Windows 10 v2004+ and macOS provide documented, user-level APIs that exclude application windows from all screen-capture pipelines while maintaining their visibility on the physical display. On Windows, `SetWindowDisplayAffinity` with the `WDA_EXCLUDEFROMCAPTURE` flag [12] causes a window to produce no pixels in any capture output. On macOS, `NSWindow.SharingType.none` [13] achieves an equivalent effect through the CoreGraphics capture pipeline. Both Apple Product Security and Microsoft MSRC classified these behaviours as consistent with documented functionality and by-design, respectively [1]. Empirical evaluation in [1] confirmed 100% evasion across Windows 10/11, macOS 14, and macOS 26.3.1, with zero visual artefacts.

Three distinct subclasses exist: (1) capture-invisible overlays using `WDA_EXCLUDEFROMCAPTURE` or `SharingType.none`, undetectable from any DOM event or `getDisplayMedia()` stream; (2) click-through overlays using `WS_EX_TRANSPARENT` or `ignoresMouseEvents`, which do not fire focus/blur events in the exam browser; and (3) GPU-layer overlays that hook DirectX or Metal directly, bypassing both DOM events and the OS capture pipeline. Commercial tools such as Cluely [3] and Interview Coder [4] combine these techniques, and a 2026 industry survey estimated that 35% of candidates in technical assessments showed signs of AI-assisted cheating [5].

### B. Why Screen Capture Is the Wrong Trust Boundary

The WebRTC Screen Capture API [2] delegates pixel composition entirely to the OS compositing pipeline. The browser has no mechanism to detect omissions from that pipeline; it receives a valid-looking frame that simply does not include excluded windows. As [1] formalises:

> **Display Fidelity Property.** A capture system satisfies display fidelity if and only if the captured frame $F$ is pixel-identical to the physical display output $D$ for all visible screen regions.

This property does not hold on any tested platform when any visible window has a capture-exclusion flag set. The trust boundary proctoring systems implicitly rely on — that the OS faithfully reports what the user sees — is structurally absent from the current platform landscape. Adding more resolution, frame rate, or webcam channels does not address the structural gap; it increases the surveillance cost without closing the attack surface.

### C. Privacy Problems with Surveillance-First Proctoring

Current proctoring platforms collect webcam video, microphone audio, gaze streams, keystrokes, screen recordings, browser histories, and system process lists [14], [15]. Research consistently documents the privacy, equity, and psychological harms of this model: elevated student anxiety [6], algorithmic bias in facial recognition [8], and structural privacy violations in e-learning systems [7]. Vulnerability in student digital well-being under AI-powered educational decision systems is a specific emerging concern [16]. Even setting aside the attack surface, the surveillance model imposes costs that may not be justified by its integrity benefits [17], [18].

Simurgh's design premise is that these costs are unnecessary. Session integrity can be assessed through metadata that does not include any user-generated content, and anomaly signals can be surfaced for human review without ever building a pixel-level record of the student's screen.

## III. Threat Model and Design Goals

### A. Adversary Model

The adversary is a test-taker with standard (non-administrator) access to their own computer who wishes to consult unauthorised AI assistance during an assessment. The adversary can run compiled native applications, control all software on their machine, and install browser extensions. The adversary cannot modify the proctoring server, the exam platform, or the OS kernel. The adversary is specifically capable of deploying capture-invisible overlays of all three subclasses described in Section II-A.

### B. Attacker Capabilities Against Capture-Based Systems

Against a browser-only proctoring system that relies solely on `getDisplayMedia()`, the adversary can achieve complete evasion by deploying a `WDA_EXCLUDEFROMCAPTURE`-flagged or `SharingType.none` window. No DOM event, no `getDisplayMedia()` artefact, and no browser-observable state change reveals the presence of the overlay. This is the baseline attack that motivates the Simurgh architecture.

### C. What Native OS Metadata Can Provide

Unlike the browser capture pipeline, the OS window-management layer retains records of all top-level windows regardless of their capture-exclusion state. On Windows, `GetWindowDisplayAffinity` returns the affinity flag for any enumerable window. On macOS, `CGWindowListCopyWindowInfo` returns the `kCGWindowSharingState` property for all windows. These metadata APIs are accessible to any native process with standard user privileges. Crucially, they report exclusion *presence* without reading window content, titles, or process names. Simurgh's native daemons exploit this gap: they run at the OS level where the capture-exclusion flag is visible, sign a summary of the metadata, and transmit the signed proof to the verifier.

### D. Design Goals

1) **Capture independence**: Session integrity must be assessable without any screen, webcam, audio, or content capture.
2) **Privacy preservation**: No typed content, pasted content, raw window titles, raw process names, window handles, process identifiers, or personal device identifiers may be collected at any layer.
3) **Cryptographic verifiability**: Every accepted or rejected integrity signal must be signed by the device daemon and written to a tamper-evident server-side audit chain, making post-hoc modification detectable in exported audit records.
4) **Replay resistance**: A captured proof from session $S$ must not be replayable in session $S'$.
5) **Manual-review-only decision model**: No component of the system may produce an automatic misconduct finding. Every anomaly signal is an input to a human reviewer.
6) **Honest non-claims**: The system must not claim capabilities it does not have. GPU-layer overlays, kernel-level compromises, read-only workflows, and hardware-rooted attestation are explicitly out of scope.

## IV. System Overview

Figure 1 shows the high-level system architecture. Simurgh operates as four coordinated layers: a browser telemetry SDK, a local integrity daemon, a Node.js verifier/server, and an instructor reporting and audit layer.

### A. Browser Telemetry Layer

The browser SDK, embedded in the exam page, collects a behavioural metadata stream covering approximately 18 browser-side behavioural event types; the full event registry (`src/academic/academicEvents.js`) additionally defines pairing, daemon-proof, scanner, and session-lifecycle events emitted server-side. The collected signals include: keystroke count (not content), character-typed count (not content), paste event count and character length (not paste content), focus-loss count and duration, idle-gap maxima, effective words-per-minute rate, and keydown timing intervals (up to 200 samples, no content). The raw student identifier is hashed with SHA-256 at server ingress; the raw value is never written to disk, logged, or transmitted further.

The telemetry stream is deliberately narrow. No DOM content, no clipboard text, no window titles, no `getDisplayMedia()` stream, and no webcam feed are collected.

### B. Local Integrity Daemon

The integrity daemon is a native process running on the student's machine at `127.0.0.1`. Its responsibilities are:

1) Maintain an ECDSA P-256 key pair: persistent on macOS (stored in the macOS Keychain under account `p256-signing-key`) and on Linux (an identity file at mode `0600` within a `0700` state directory under `$XDG_STATE_HOME`); the current Windows research daemon generates the key ephemerally for the lifetime of the process, with persistent storage backed by DPAPI deferred to a future stage.
2) On request, query OS metadata APIs to obtain the current platform-specific display-integrity summary.
3) Construct a signed proof envelope containing the version, platform, session identifier, node identity hash, timestamp, challenge (a server-issued one-time token used for replay protection in the device-shield daemon proof), `capabilities` sub-object (four boolean scanner-capability flags), `signals` sub-object (window counts and helper status), privacy mode, and ECDSA P-256 signature over the canonical serialisation of all non-signature fields.
4) Respond to pairing challenges issued by the server.

The daemon explicitly does not enumerate window titles, process names, process identifiers, raw window handles, or any content-level data. The OS scanner returns platform-specific metadata summaries — capture-exclusion counts on Windows and macOS, and window-state or portal metadata on Linux — not a list of which windows exist or what they display.

### C. Server Verifier

The Node.js server receives telemetry events and proof envelopes from the browser SDK. For each proof, it performs the following verification steps, detailed further in Section V:

1) Schema validation: all required fields for the relevant proof envelope must be present and correctly typed.

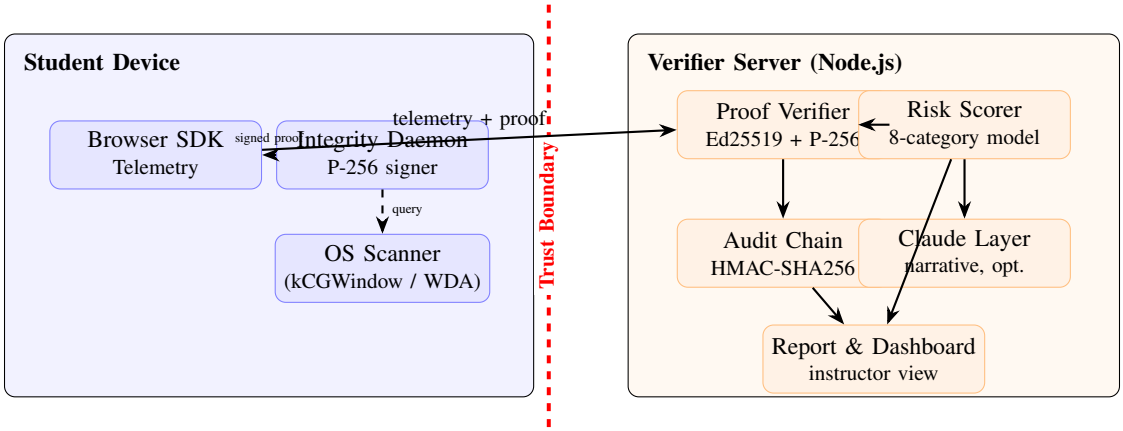


Fig. 1. Simurgh system architecture. The student device runs a browser SDK (telemetry collection) and a local integrity daemon (ECDSA P-256-signed proofs of OS metadata; the browser-paired integrity-proof envelope uses Ed25519). The daemon queries OS-level window-state metadata without accessing content. All proof material crosses the trust boundary to the Node.js verifier, which checks signatures, scores risk, appends to the audit chain, optionally invokes the Claude narrative layer, and exposes a read-only instructor dashboard and report.

2) Nonce check: the nonce must not have been seen in this session (replay guard).
3) Signature verification: Ed25519 `verify(canonical, pubkey, sig)` for the browser-paired proof envelope; ECDSA P-256 over SHA-256 for the device-shield daemon envelope.
4) Node continuity: the submitting node's hash must match the bound node for this session.
5) Audit append: the result (accept or reject with reason) is written to the HMAC chain.

Risk scoring aggregates telemetry events and proof results into a session risk score across eight weighted categories: paste behaviour, focus events, typing patterns, idle periods, display-affinity signals, helper connection state, daemon proof signals, and session lifecycle. The risk scorer produces three fixed verdict strings, hard-coded and unsuppressable through configuration:

- **Critical**: "Manual review required. No automatic misconduct finding."
- **Warning**: "Manual review recommended. No automatic misconduct finding."
- **Safe**: "No anomalies detected."

### D. Instructor and Reporting Layer

The instructor dashboard displays risk category breakdowns, event timelines, per-session integrity state (paired/unpaired, node hash, proof count), and audit chain status. The report export includes the full audit chain as a JSON object that can be verified offline using the included `verify-audit.mjs` tool.

### E. Tamper-Evident Audit Chain

Integrity systems must preserve not only device-state evidence but also the history of verifier decisions. Without a tamper-evident log, a proof verifier could accept, reject, delete, or reorder events after the fact without detection. The HMAC chain therefore protects *review integrity* — the reliability of the audit record presented to a human reviewer — rather than display integrity.

Every state-changing event — telemetry accept, proof accept, proof reject, pairing accept, pairing reject, session state transition, and Claude narrative emission — is written to an HMAC-SHA256-linked audit chain [11]. Each entry's `prev` field contains the HMAC-SHA256 signature of the previous entry, computed over the full JSON serialisation of that entry, making any retroactive modification detectable as a chain-break. The audit chain HMAC key is server-held; it is not shared with the daemon, the browser, or any AI analysis layer.

## V. Proof Protocol

Figure 2 illustrates the full proof protocol sequence. Section V-A through V-F describe each step.

### A. Pairing

Before a proof can be accepted, the server and daemon must complete a pairing handshake. The server issues a one-time challenge token via `/api/pairing/challenge`. The challenge has a 60-second time-to-live and is consumed on first use; any unconsumed pending challenge for the session is cleared when a new one is issued. The daemon signs the challenge using its private key (Ed25519 in the browser-paired proof path, ECDSA P-256 in the device-shield daemons) and submits the signed pairing proof to `/api/pairing/complete`. The server verifies the signature, extracts the raw public key from the proof, computes `node_id_hash = SHA-256(pubkey)` (64-character lowercase hex; the daemon path carries an explicit `sha256:` algorithm prefix), and binds this hash to the session. Subsequent pairing attempts from a different node hash are rejected.

### B. Challenge Issuance and Proof Envelopes

Simurgh defines two related proof envelopes. The **browser-paired integrity-proof envelope**

(simurgh-integrity-proof-v1), produced by the browser-paired Node-side signer, contains eleven required fields: version, platform, session_id, node_id_hash (64-char hex), node_public_key (32 bytes Ed25519, base64), nonce (12–64 bytes, base64), timestamp, capabilities (four boolean scanner-capability flags), signals (window counts and helper status), privacy_mode, and signature (64 bytes, base64). The **device-shield daemon proof envelope**, produced by the macOS, Windows, and Linux daemons, carries instead twelve fields: type, session_id, exam_id, sequence, timestamp, node_id_hash (with sha256: prefix), daemon_version, platform, capture_excluded_window_count, helper_state, challenge, and signature, with an SPKI-DER ECDSA P-256 public key delivered alongside the proof. Two distinct canonical serialisations are used. The browser-paired proof uses a recursive key-sorted JSON serialisation produced by proofCanonicalise.js, mirrored byte-for-byte in the Swift node implementation (simurgh-node-macos). The device-shield daemon proof uses a top-level key-sorted serialisation produced by canonicaliseDaemonPayload, mirrored in the Rust daemon's canonical_json.rs. Golden-fixture interop tests lock both implementations independently.

### C. Signed Daemon Proof and Verification

Browser-paired integrity proofs are verified via Node.js crypto.verify(null, canonical, ed25519SpkiPubkey, sigBytes) over a raw 32-byte Ed25519 public key wrapped in a DER SubjectPublicKeyInfo envelope. Device-shield daemon proofs are verified via crypto.verify("sha256", canonical, p256SpkiPubkey, sigBytes) over an SPKI-wrapped ECDSA P-256 public key. The server performs a strict triple check (E1): (a) the claimed node_id_hash must equal SHA-256(decoded pubkey); (b) the raw public key must match the session's registered key; (c) the signature must verify against the canonical payload. Any failure on any check produces a uniform invalid_signature reason code to prevent oracle attacks.

### D. Replay Rejection

A global nonce guard maintains a set of all nonces seen in accepted browser-paired proofs, expiring entries after five minutes. The proof's own session_id field and ECDSA signature provide session binding; the nonce guard prevents any proof nonce from being submitted twice. When a proof arrives, its nonce is checked against this global set before signature verification. If the nonce has been seen, the proof is rejected with nonce_replayed and the event is appended to the audit chain. Nonces are 12–64 bytes of random data generated by the daemon (16 bytes in the macOS implementation); collision probability under a birthday attack is negligible.

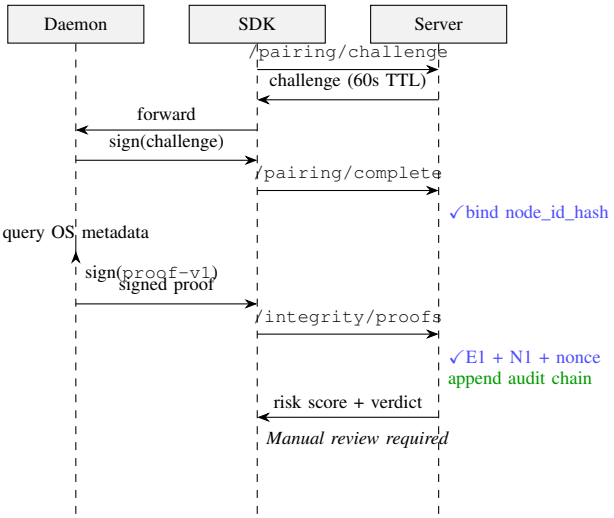


Fig. 2. Simurgh proof protocol. Pairing binds the session to one node. Each proof uses E1 (node_id_hash ∧ pubkey ∧ signature) and N1 (paired-node continuity). All results append to the HMAC audit chain.

### E. Raw-Field Rejection

The proof schema validator rejects any proof containing explicitly forbidden fields, including screen_pixels, screenshot, webcam_frame, audio, typed_answer, paste_content, window_title, process_name, raw_process_names, raw_window_titles, hardware_serial, and biometric. The capabilities sub-object is validated against exactly four boolean keys: screencapturekit_available, window_enumeration, sharing_state_scan, and helper_bridge. The signals sub-object is validated against exactly four defined keys: node_uptime_ms, window_count, capture_excluded_window_count, and helper_status.

### F. Risk Mapping

Accepted proofs are scored against the display-affinity category of the eight-category risk model. A non-zero capture_excluded_window_count in an accepted proof elevates the affinity risk score proportionally. Rejected proofs (nonce replay, signature failure, node mismatch) also contribute to the helper-connection risk category, as repeated rejections may indicate tampering. All risk scores aggregate into a session-level verdict that is presented to the instructor with mandatory manual-review language.

## VI. Platform Implementations

### A. macOS Device Shield

The macOS daemon is implemented in Swift using the CryptoKit framework. It generates and stores an ECDSA P-256 signing key (P256.Signing.PrivateKey) in the macOS Keychain at first launch under the account p256-signing-key. The daemon exposes a 127.0.0.1-bound HTTP listener for proof

and pairing endpoints. The OS scanner queries `CGWindowListCopyWindowInfo` with the combined `kCGWindowListOptionOnScreenOnly | kCGWindowListExcludeDesktopElements` mask and reads the `kCGWindowSharingState` property for each returned window. Its `signals` sub-object reports `node_uptime_ms`, `window_count`, `capture_excluded_window_count` (windows with sharing state `kCGWindowSharingNone`), and `helper_status`. ScreenCaptureKit [19] availability is exposed to the verifier as the `screencapturekit_available` capability flag for forward compatibility; the current scanner remains single-path CoreGraphics, and an adaptive ScreenCaptureKit query path is reserved as a future-stage upgrade while preserving the metadata-only contract.

### B. Windows Device Shield

The Windows daemon is implemented in .NET, using `ECDsa.Create(ECCurve.NamedCurves.nistP256)` for proof signing. The Windows scanner enumerates all top-level windows using `EnumWindows` and calls `GetWindowDisplayAffinity` on each handle. It counts windows with `WDA_MONITOR` and `WDA_EXCLUDEFROMCAPTURE` flags and reports them in the signed proof as `capture_excluded_window_count` (umbrella: `capture_restricted_window_count`) and `monitor_only_window_count`. No window title, class name, process name, or PID is included in the summary; the raw window handle is used only transiently during enumeration and is never written to the proof. Real-device validation on Windows 10 Pro build 19045 confirmed correct detection of both flags, signed proof submission, server-side acceptance, and correct propagation of the affinity count into the risk score, dashboard, and audit chain.

### C. Linux Display Integrity Research

The Linux daemon is implemented in Rust, using the `p256::ecdsa` crate for proof signing. Due to the diversity of the Linux desktop landscape, the Linux implementation is positioned as a display integrity research contribution rather than a complete deployment target. Three scanner paths are implemented:

**X11**: The Rust daemon connects to the X server via `x11rb` and enumerates top-level windows. It reads the `_NET_WM_STATE` property and window manager hints to identify windows exhibiting overlay-relevant window-manager states. The Linux X11 path reports window-manager metadata counts — including managed, override-redirect, above, fullscreen, and skip-taskbar windows — and does not claim Windows/macOS-style capture-exclusion detection. No window title or class is read.

**Wayland**: Direct window enumeration is not available to unprivileged clients in Wayland. The daemon probes the `org.freedesktop.portal.Desktop` service via D-Bus using `NameHasOwner` (to detect portal presence) and reads the `ScreenCast.AvailableSourceTypes` property (to detect active screencast capability). It never calls `CreateSession`, `SelectSources`, `Start`, or other portal methods that would trigger a user consent prompt. A banned-method grep test in CI enforces this invariant.

**XWayland**: The X11 scanner path runs against the `$DISPLAY` provided by XWayland, providing partial coverage of applications running in XWayland compatibility mode. Coverage is reported as `coverage=xwayland_partial` in the proof.

A display-server lock prevents proof acceptance from sessions where the detected display server does not match the expected type, providing defence against display-server spoofing.

### D. Cross-Platform Verification Contract

The macOS, Windows, and Linux daemons do not emit byte-identical envelopes across all implementation stages. Instead, they satisfy a shared verification contract: each proof is session-bound, challenge-bound, timestamped, signed by a registered node key, privacy-normalised, and processed by the server through the same signature check (E1), replay guard, node-continuity rule (N1), raw-field rejection, risk mapping, and audit-chain append pipeline. The canonical serialisation (`proofCanonicalise.js`) is mirrored in Swift (`ProofSigner.swift`) and Rust (`canonical_json.rs`), with byte-identical golden fixture tests enforcing cross-implementation consistency for the fields shared between envelope types.

## VII. Privacy Model

### A. Metadata-Only Collection

Simurgh's privacy model distinguishes between *behavioural metadata* (what the student did) and *content* (what the student wrote, said, or displayed). Only the former is collected; Table II specifies the boundary for each signal class.

### B. Forbidden Fields

The following data types are explicitly forbidden at every layer of the system: screen pixels; webcam frames; microphone audio; typed content; pasted content; raw window titles; raw window class names; raw process names; process identifiers; window handles; raw public keys in audit payloads (only hashes are stored); hardware serial numbers; MAC addresses; hostnames; and personal device identifiers. The `node_id_hash` field is a pseudonymous verifier identifier derived from the daemon public key; it is used only for session and node continuity and does not expose any hardware or user identity. The proof schema validator and privacy audit scripts enforce these constraints as regression gates.

### C. No Screen, Audio, Webcam, or Content Collection

Simurgh does not implement and does not call `getDisplayMedia()`, `getUserMedia()` with video or audio, any screenshot API, any audio capture API, or any clipboard-read API. The browser SDK does not

TABLE I
PLATFORM CAPABILITY MATRIX

| Platform | Signal source | Impl. | Validation | Known limitation |
|---|---|---|---|---|
| macOS | `kCGWindowSharingState` (CoreGraphics) | Swift | Swift tests (8) | SCKit adaptive path deferred to future stage |
| Windows | `WDA_MONITOR`, `WDA_EXCLUDEFROMCAPTURE` | .NET/Win32 | Real device (Win10 Pro 19045) | No kernel/GPU overlay visibility |
| Linux X11 | Window-manager states (`_NET_WM_STATE`) | Rust/x11rb | CI + Xvfb | Real desktop validation pending |
| Linux Wayland | Portal property metadata only | Rust/D-Bus | Mock D-Bus + CI | No surface/window enumeration |
| XWayland | X11 bridge, XWayland `$DISPLAY` | Rust/x11rb | CI | Partial coverage (`coverage=xwayland_partial`) |

TABLE II
PRIVACY COLLECTION BOUNDARY

| Signal | Collected | Never collected |
|---|---|---|
| Keystrokes | Count per window | Keystroke content |
| Characters typed | Count only | Typed text |
| Paste events | Count + char length | Paste content |
| Focus loss | Count + duration (ms) | Window title/target |
| Idle gaps | Maximum gap (ms) | Content during idle |
| WPM | Effective WPM | Text being typed |
| Timing intervals | Up to 200 samples | Key content |
| Device integrity metadata | Platform-specific counts/booleans | Titles, PIDs, names, handles |
| Student identity | SHA-256 hash | Raw identifier |

TABLE III
AUTOMATED VALIDATION SUMMARY

| Suite | Runtime | Result |
|---|---|---|
| Node.js unit tests | Node.js 22 | 331 / 331 pass |
| Rust daemon tests | Rust stable | 33 / 33 pass |
| Windows .NET tests | .NET 8 | 11 / 11 pass |
| macOS Swift daemon tests | Swift/Xcode | 8 / 8 pass |
| Stage 2.7 cross-platform smoke | Node.js + Rust | All scenarios pass |
| Stage 2.8 Linux smoke | Node.js + Rust | 16 scenarios pass |
| Security audit assertions | Node.js | 30 / 30 pass |
| Privacy audit gates | Node.js | All pass |
| npm advisory audit | npm | 0 high/critical |

attach any event listener that could capture content. These non-implementations are verified in the privacy audit that runs as part of the CI check suite.

### D. Manual Review Only

Critical verdicts are accompanied by the hard-coded string "Manual review required. No automatic misconduct finding." Warning verdicts produce "Manual review recommended. No automatic misconduct finding." Neither string can be suppressed through configuration. The system produces no binary misconduct verdict, no suspension action, no notification to any authority, and no automated disciplinary record or enforcement action outside the human-review workflow. All conclusions require a human reviewer.

## VIII. EVALUATION

### A. Functional Validation

Table III summarises the automated validation results across the prototype.

The Node.js unit tests cover the proof validator, pairing registry, nonce guard, HMAC chain, risk scorer, session state machine, telemetry normaliser, and report builder. Rust tests cover the Linux daemon scanner paths (X11, Wayland, XWayland), the cross-platform canonicalisation implementation, and the Linux daemon proof flow; macOS daemon correctness is covered by the Swift test suite (`AffinityScannerTests.swift`, `ScannerProofTests.swift`, `PrivacyNormaliserTests.swift`, `DaemonDoctorTests.swift`; 8 tests). The Windows .NET tests exercise the display-affinity scanner and signed proof contract.

### B. Security Audit

A ten-question internal security audit evaluated the proof protocol against the following threat scenarios: pairing replay, browser-forged `signature_status`, macOS key protection, challenge single-use and expiry, node continuity per session, audit hint spoofing, rate-limit sizing, browser trust boundary, audit emission on rejection, and demo failure state reproducibility. All ten questions received positive answers with code-level evidence and regression gate coverage (10/10).

An additional 30-assertion cybersecurity audit across 16 security dimensions was conducted for the Stage 2.8 Linux path, covering: nonce handling, signature verification, replay protection, display-server lock, Wayland consent safety, XWayland partial coverage boundary, audit completeness, rate limiting, privacy contract, anti-overclaiming wording, and CI enforcement. All 30 assertions pass.

### C. Privacy Audit

The privacy audit verifies the following properties as regression-gated checks: (1) no `window.getDisplayMedia()` call in the browser SDK or server; (2) no clipboard-read API call; (3) no `getUserMedia()` call with video or audio; (4) no raw window title or process name in any proof schema field; (5) no raw student identifier written past the ingress hasher; (6) no raw private key or raw public key in any audit chain entry; (7) forbidden Wayland portal methods absent from the Rust scanner. These checks run as part of the `scripts/check.sh` gate suite on every CI run.

### D. Real-Device Windows Validation

The Windows Device Shield was validated on a physical Windows 10 Pro machine (build 19045). The validation confirmed: `WDA_MONITOR`-flagged windows increment `monitor_only_window_count` (and the umbrella `capture_restricted_window_count`); `WDA_EXCLUDEFROMCAPTURE`-flagged windows increment

TABLE IV
BROWSER-ONLY VS. DAEMON-AUGMENTED DETECTION

| Scenario | Browser-only | Simurgh (daemon) |
|---|---|---|
| Normal window (no flag) | Visible in capture | No anomaly signal |
| `WDA_MONITOR` window | Black/restricted in capture | `monitor_only_window_count` $> 0 \Rightarrow$ Warning |
| `WDA_EXCLUDEFROMCAPTURE` | Invisible in capture | `capture_excluded_window_count` $> 0 \Rightarrow$ Critical |
| macOS `SharingType.none` | Invisible in capture | `capture_excluded_window_count` $> 0 \Rightarrow$ Critical |
| GPU-layer overlay | Invisible in capture | Not detectable (NC01) |

`capture_excluded_window_count`; unflagged windows contribute to neither counter. Signed proofs from the .NET daemon were accepted by the Node.js verifier; tampered proofs, replayed nonces, and wrong-node proofs were rejected with correct reason codes and audit entries. The affinity counts propagated correctly into the risk score, the instructor report, and the audit chain export.

### E. Linux CI and Display-Server Validation

Ubuntu CI validation runs the full Linux Rust test suite under mandatory `Xvfb` with `SIMURGH_REQUIRE_XVFB_TESTS=1`. Shellcheck linting enforces all lifecycle scripts. The Wayland probe path is validated via a mock D-Bus environment. Display-server mismatch enforcement is tested via the 16-scenario smoke suite, which includes headless, X11, XWayland, and Wayland-advertised configurations. The CI pipeline rejects any run where Xvfb is missing when the flag is set, preventing false-pass results on headless runners.

### F. Browser-Only vs. Daemon-Augmented Detection

Table IV contrasts the detection capability of a browser-only proctoring approach (relying solely on `getDisplayMedia()` and DOM events) against Simurgh's daemon-augmented approach across the four window states that motivate this work.

The browser-only column illustrates the capture-fidelity failure: excluded windows produce no detectable artefact in `getDisplayMedia()` output. The daemon column shows that the same windows are detectable through OS metadata APIs accessible to any native process.

### G. False-Positive Calibration

To bound the risk of unwarranted manual-review triggers, we evaluated the risk scorer against 20 synthetic clean sessions representing normal exam behaviour: steady typing (20–95 WPM), occasional focus losses (0–3 blurs), legitimate short pastes (0–78 characters), network reconnects, daemon minor warnings, and helper-slow-to-connect scenarios. No session used an attack overlay, no session had paste $\geq 80$ characters with fewer than 20 typed characters, and no session had `capture_excluded_window_count` $> 0$ except one deliberate boundary case.

Results: 19/20 sessions scored Safe (risk score $\leq 19$); 0/20 scored Warning; 0/20 scored Critical from behavioural signals alone. The one Critical case (S11, score = 85) was a deliberate boundary scenario: a macOS system-tray utility window with `SharingType.none` set by the OS itself (e.g., a screen-sharing or accessibility helper). This is a known limitation of the affinity override (H02): the override fires on *any* excluded window, including legitimate system utilities. The false-positive rate for behavioural signals alone is **0/19 = 0%** across these clean sessions; the affinity channel has a non-zero false-positive rate dependent on the mix of system utilities on the student's machine. Institutions deploying Simurgh should conduct platform-specific baselining before production use.

### H. Risk Sensitivity Analysis

To assess whether the risk verdicts are sensitive to the choice of threshold values, we varied the Critical and Warning floors by $\pm 10\%$, $\pm 20\%$, and $\pm 30\%$ and re-scored five representative attack sessions and five clean sessions. Attack sessions covered: large paste with minimal own typing; capture-excluded window; superhuman WPM; long idle followed by large paste; and combined excluded-window plus large paste. Clean sessions are the representative subset from Section VIII-D.

Results are stable for clean sessions across all threshold variants: 0/5 false positives at every setting. For attack sessions, the baseline detects 4/5 sessions (the undetected session is a 280 WPM burst without paste or affinity signals, which falls below the threshold). At $\pm 10\%$ and $\pm 20\%$ variants, 4/5 attack sessions are still detected. At $+30\%$ (thresholds raised to Critical $\geq 91$, Warning $\geq 52$), 3/5 are detected; at $-30\%$ (thresholds lowered to Critical $\geq 49$, Warning $\geq 28$), 4/5 are detected with Warning/Critical severity shifting for one session. The affinity override (H02) is threshold-independent: any capture-excluded window forces Critical regardless of the weighted sum. These results suggest the scoring is not threshold-tuned to win: clean sessions are Safe across all variants and attack-session detection degrades gracefully rather than cliffing at a single tuned value.

### I. Nonce TTL and Replay Safety

The nonce guard TTL of five minutes might appear to create a replay window: once a nonce expires from the guard, a captured proof could theoretically be re-submitted. This concern is addressed by the 30-second timestamp tolerance (`TIMESTAMP_PAST_MS = 30 000`): any proof older than 30 seconds is rejected for timestamp staleness before the nonce check is reached. The nonce guard's five-minute TTL is therefore redundant protection beyond the timestamp window — a defense-in-depth design, not a security gap. The device-shield daemon proof path uses server-issued challenges that are consumed on first use, providing independent replay resistance without relying on either nonces or timestamps.

### J. Performance Characterisation

Table V reports server-side verification latency (500-run average, Node.js 22, macOS M-series) and proof payload sizes.

The per-proof verification budget is under 0.2 ms server-side, well within the rate-limited ceiling of 30 proof submissions per minute per session. Payload size is modest; even at maximum submission rate, proof traffic is under 40 KB/min per session. Daemon-side proof generation latency (cryptographic signing) depends on platform; typical Ed25519 and ECDSA P-256

TABLE V
SERVER-SIDE LATENCY AND PAYLOAD SIZES

| Operation | Measured |
|---|---|
| Ed25519 verify (browser-paired proof) | 0.083 ms |
| ECDSA P-256 verify (daemon proof) | 0.054 ms |
| Risk scorer (8 categories) | 0.003 ms |
| Proof canonicalisation (browser) | 0.005 ms |
| Audit chain append (HMAC) | 0.006 ms |
| Browser-paired proof payload (JSON) | 692 B |
| Device-shield daemon proof (JSON + pubkey) | 648 B |
| Combined per submission | ≈1.3 KB |

signing operations on modern hardware complete in under 1 ms, making Simurgh's proof overhead negligible relative to exam session duration.

### K. LLM-Assisted Development Methodology

The proof protocol, risk scorer, platform daemon implementations, test suites, and audit gate framework were developed using Claude Code powered by Anthropic's Claude Sonnet 4.6. The LLM was used as an implementation assistant under explicit anti-overclaiming constraints derived from the RESEARCH_PROGRAMME.md document. All non-claims were manually verified. This disclosure follows the methodology transparency standard established in [1].

## IX. SECURITY ANALYSIS

### A. What Simurgh Detects

Simurgh produces integrity signals for the following anomalies at the metadata level:

- Presence of capture-excluded windows (`WDA_EXCLUDEFROMCAPTURE`, `WDA_MONITOR`, `SharingType.none`) during an active session.
- Abnormal paste behaviour (high paste count, large paste character volumes).
- Unusual focus-loss patterns (high loss count, long accumulated duration).
- Anomalous typing patterns (WPM far outside baseline, irregular keydown timing distribution).
- Extended idle periods inconsistent with active exam behaviour.
- Daemon proof failure events (replay, signature failure, node mismatch) indicating potential daemon tampering.
- Session-level anomalies (premature disconnect, irregular state transitions).

### B. What Simurgh Cannot Detect

The following attack classes are explicitly outside the current system boundary:

**GPU-layer overlays**: Tools that hook DirectX or Metal compositor layers and render directly to the GPU display output bypass both the OS window-management APIs and any capture pipeline. No OS-level window enumeration will see these overlays. This limitation is documented in the Stage 1 limitations document and the RESEARCH_PROGRAMME.md.

**Read-only workflows**: A student who silently reads from a second device, printed notes, or a physical reference without interacting with a capture-invisible overlay will produce no anomalous signals in the Simurgh data stream. Gaze-tracking approaches exist [20], [21] but are explicitly out of scope for this system.

**Kernel-level compromise**: A compromised operating system or kernel-level rootkit can modify the return values of any API the daemon queries. No user-space tool can detect this. Hardware-rooted attestation (Section XII-B) is the long-term direction.

**Click-through overlays without display-affinity flag**: A click-through overlay that does not set a capture-exclusion flag is not detectable through the affinity scanner. Browser focus-loss events may trigger if the user clicks, but a purely read-only click-through overlay with no keyboard interaction may produce no signal.

### C. Tamper and Replay Resistance

The HMAC-SHA256 audit chain makes retroactive modification of any event record detectable by any verifier holding the HMAC key. The nonce guard prevents replaying a valid proof from one session epoch into another. The E1 triple check (hash + key + signature) prevents forging a proof that appears to come from a registered node. The N1 node-continuity invariant prevents an adversary from switching daemons mid-session after an initial pairing. Rate limits (30 proof submissions per minute, 10 pairing challenges per minute) bound amplification attacks.

### D. Design Alternatives Considered

Table VI summarises the main alternative design directions considered and the rationale for their rejection in favour of signed OS metadata proofs.

### E. Remaining Attack Surface

An adversary who can run the Simurgh daemon client unmodified but controls the proof's `signals` sub-object (or, in the daemon envelope, the affinity-count fields) through OS-level manipulation can submit a dishonest but validly signed proof. The server cannot distinguish a lying daemon from an honest one if the signing key is genuine and the proof format is correct. Signed proofs attest identity (*this key submitted this data*) but not truthfulness (*this data faithfully reflects OS state*). Hardware-rooted attestation, discussed in Section XII-B, addresses this gap. Until then, the proof architecture provides meaningful replay and forgery resistance while being honest about the remaining trust assumption.

## X. ETHICS AND DEPLOYMENT LIMITS

### A. No Automatic Misconduct Finding

Simurgh is designed explicitly as an evidence-gathering tool for human reviewers, not a disciplinary decision engine. The

TABLE VI
DESIGN ALTERNATIVES AND REJECTION RATIONALE

| Alternative | Why rejected |
|---|---|
| More screen capture | Still trusts the capture pipeline — excluded windows remain invisible regardless of resolution or frame rate |
| Expanded webcam/gaze | More invasive and biased; still bypassable by GPU-layer overlays; adds cost without closing the structural gap |
| Browser-only detection (DOM) | Browser has no access to OS capture-exclusion metadata; focus/blur events fire only on click-through, not on capture-invisible overlays |
| Kernel-level agent | Requires elevated privileges; unacceptable for research prototype; introduces significant privacy risk |
| Hardware capture card | Privacy-heavy; impractical at scale; does not detect GPU-layer overlays or second-device read-only workflows |
| Full Wayland enumeration | Requires `CreateSession`/`SelectSources` portal calls that trigger a user consent prompt, violating the metadata-only privacy contract |
| Configurable verdict strings | Any configurability risks inadvertent suppression of manual-review language; hard-coded strings are the only safe design |
| Per-session nonce guard | Added session-lifecycle coordination complexity without security benefit; session binding via signed `session_id` is cryptographically stronger |

hard-coded "Manual review required" wording in the risk scorer is not a UI choice; it reflects an architectural commitment. The system does not produce a verdict, flag a student to any authority, or trigger any automated action. A high risk score is a recommendation for a human to look at the session's audit chain, not a finding of misconduct.

### B. Research Prototype Status

Project Simurgh is a research prototype at version 0.4.18. It has not been reviewed by any data-protection authority, accessibility auditor, or institutional ethics board beyond the development team's own review process. It has not been piloted with actual students in any live examination context. Institutional deployment would require, at minimum: a formal data-protection impact assessment (FERPA, GDPR, or equivalent); an accessibility review against WCAG 2.2 AA; an external red-team engagement; signed and notarised macOS/Windows distribution packages; and an institutional pilot under ethics oversight.

The system should not be described as production-ready, compliant with any specific framework without naming the assessment, or ready for deployment by universities without the above-listed review processes.

The motivating threat claim (Section II-A) rests on the companion Invisible Window disclosure [1], which is a Zenodo preprint that has not yet undergone external peer review. Independent replication of the `WDA_EXCLUDEFROMCAPTURE` evasion result is pending. Simurgh's defensive architecture is valid conditional on C01; reviewers should assess the companion paper independently.

### C. Deployment Governance Checklist

Any institution considering Simurgh for a regulated assessment context should complete the following gates before any live use. None of these gates are currently passed by the Simurgh research prototype.

1) **Data-protection impact assessment** (FERPA, GDPR, or equivalent jurisdiction) with explicit documentation of data types collected, retention periods, and student access/deletion rights.
2) **Accessibility review** against WCAG 2.2 AA, including the daemon installation flow and the instructor dashboard.
3) **External red-team engagement** covering at minimum: pairing replay, helper spoofing, audit chain manipulation, and false-positive rate on representative student machines.
4) **Student notice and appeal process** — students must be informed that a native daemon is running and have a documented process to flag incorrect verdicts.
5) **Data retention policy** — session audit exports must have a defined retention window and deletion procedure.
6) **Human review policy** — the institution must define who reviews Critical/Warning sessions, under what timeline, and what the escalation path is. No automated action may substitute.
7) **Signed and notarised distribution packages** for macOS and Windows, or equivalent institutional MDM deployment.
8) **Institutional ethics board sign-off** before any pilot involving actual students.

### D. Institutional Review Requirements

The staged deployment roadmap in the Simurgh RESEARCH_PROGRAMME identifies the following gates before any regulated-environment use: privacy/legal review (including data retention and student-access policies), accessibility review, external red-team engagement (pairing replay, helper spoofing, audit chain manipulation), signed distribution pipeline, and LMS integration design. None of these gates are currently passed. We include this explicit statement in accordance with the ACM and IEEE codes of ethics [22], [23].

## XI. RELATED WORK

### A. Proctoring System Security and Criticism

Simko et al. [24] catalogue community-developed proctoring evasion techniques, documenting the adversarial dynamic between proctoring systems and test-takers. Their taxonomy includes the display-affinity subclass that Simurgh directly addresses. Balash et al. [14], [15] examine educator and student perspectives, documenting the privacy and usability costs of existing systems. Luijben et al. [25] formalise five security requirements; Simurgh addresses Requirement 2 (work authenticity) through the proof architecture while preserving Requirement 4 (data protection) through the metadata-only model.

### B. Behavioural Detection

A substantial body of work applies gaze tracking [20], [21], mouse dynamics [26], keystroke analysis, and multimodal fusion [27], [28] to exam integrity. Simurgh collects behavioural signals in the same spirit but differs in two important ways: (1) it collects count-level metadata rather than biometric streams; and (2) it pairs behavioural signals with cryptographic device proofs rather than treating browser-observable signals as the sole evidence source.

### C. Hardware Attestation and Zero Trust

The NIST Zero Trust Architecture [29] framework motivates treating the client device as untrusted until it provides verifiable proof of identity and state. Simurgh's signed proof envelope (Ed25519 for the browser-paired proof path, ECDSA P-256 for the device-shield daemons) is a software instantiation of this principle, pending hardware-rooted attestation. Trusted Platform Module [30] and Intel TXT provide hardware-backed attestation primitives; WebAuthn [31] provides a browser-accessible version using platform authenticators. Simurgh's architecture is designed to accept hardware-rooted proofs in a future stage (Section XII-B) without breaking the existing proof contract.

### D. Privacy-First Assessment

Paris et al. [7] and Mukherjee et al. [8] document the privacy limitations of current proctoring systems. Duncan and Joyner [17] argue for alternative assessment designs that are structurally resistant to cheating. Prinsloo et al. [16] identify AI-powered educational decision systems as a vulnerability for student well-being. Simurgh's metadata-only model is a direct response to these concerns: by design, it cannot produce the surveillance record that existing systems create.

## XII. Discussion and Future Work

### A. Agent Shield

Stage 3 of the Simurgh roadmap introduces an Agent Shield: a proof-based integrity layer for AI computer-use sessions, where an LLM agent performs tasks on behalf of the session holder. Agent-mediated sessions present a distinct integrity surface, as the agent itself is a new attack vector (tool poisoning, prompt injection, system-prompt leakage). The Agent Shield will extend the existing proof envelope to include an agent attestation field: a signed summary of the agent's visible tool invocations and reasoning traces, bounded by the same metadata-only privacy contract.

### B. Hardware-Rooted Attestation

The current Ed25519 + ECDSA P-256 proof architecture provides software-level identity binding: the proof attests that *this key signed this data at this time*. It does not attest that the data faithfully reflects the OS state, because the key itself resides in user space. Hardware-rooted attestation via TPM [30] or Secure Enclave would move the signing key into hardware, making it non-extractable and binding the attestation to physical device identity. Apple's Secure Enclave and Windows' TPM 2.0 both provide the necessary primitives; the challenge is integrating them into the existing proof contract without breaking the metadata-only privacy model. Attestation keys must not leak device identity across sessions.

### C. Privacy-Preserving Visual Verification

Stage 4 explores the possibility of recovering some visual integrity signal without collecting screen pixels. Approaches under consideration include on-device inference of scene-level properties ("student is seated at a desk") using a private enclave, rather than cloud video analysis; homomorphic comparisons of frame hashes without transmitting content; and hardware display-output attestation using HDCP link integrity signals. All of these approaches face significant engineering challenges and are research directions rather than near-term plans.

## XIII. Conclusion

This paper has presented Project Simurgh, a privacy-preserving integrity architecture for high-stakes AI-mediated and remote assessment sessions. We have shown that the display-fidelity failure documented in the Invisible Window attack [1] — the gap between the physical display and the screen-capture surface — can be addressed without resorting to more invasive surveillance. Instead, Simurgh establishes session integrity through signed, metadata-only device proofs, native OS metadata scanners, and a tamper-evident audit chain.

The core architectural argument is straightforward: if the OS compositing pipeline cannot be trusted to faithfully expose all visible content to the browser capture API, then the appropriate response is to move the trust anchor to the OS level itself, where the capture-exclusion state is visible to any native process. Simurgh does exactly this, at the price of requiring a native daemon on the student's device — a cost that is explicit and reasonable, compared to the alternative of collecting full screen video.

Simurgh does not claim to detect all cheating or all overlays. It shows that a capture-resistant integrity layer can be built using signed, metadata-only device proofs, and that this layer detects a documented class of capture-excluded OS windows without collecting visual or content data. The four contributions — the privacy-preserving architecture, the cross-platform prototype, the display-affinity detection without content capture, and the evaluated manual-review-only decision model — are a concrete existence proof that capture-resistant integrity assessment is technically feasible while preserving meaningful privacy guarantees. We hope this work encourages further research into proof-based, metadata-only integrity systems and provides a documented, auditable foundation for institutional engagement.

## Acknowledgements

This research was conducted using Claude Code powered by Anthropic's Claude Sonnet 4.6. The author thanks Anthropic for providing the research tooling. This paper is a companion to the Invisible Window disclosure paper [1], which documented the motivating attack class. The author thanks the reviewers of the Simurgh RESEARCH_PROGRAMME and Stage 2 closeout

documentation for their feedback on the non-claims posture adopted throughout this work.

## References


[1] M. R. Abedini, "The Invisible Window: Exploiting OS-Level Display Affinity to Bypass WebRTC Proctoring Systems," https://doi.org/10.5281/zenodo.20319832, 2026, zenodo preprint. Accessed: 2026.

[2] J.-I. Bruaroey and E. Alon, "Screen Capture," https://www.w3.org/TR/screen-capture/, 2025, w3C Working Draft, World Wide Web Consortium.

[3] Cluely, "Cluely — AI-Powered Real-Time Interview Assistant," https://cluely.com/, 2025, accessed: 2026.

[4] Interview Coder, "Interview Coder — AI Interview Assistant for Technical Interviews," https://www.interviewcoder.co/, 2026, accessed: 2026.

[5] FabricHQ, "Interview Cheating in 2026: The Rise of AI Tools Like Cluely and Interview Coder," https://www.fabrichq.ai/blogs/interview-cheating-in-2026-the-rise-of-ai-tools-like-cluely-and-interview-coder, 2026, accessed: 2026.

[6] R. Conijn, A. Kleingeld, U. Matzat, and C. Snijders, "The fear of big brother: The potential negative side-effects of proctored exams," *J. Computer Assisted Learning*, vol. 38, no. 6, pp. 1521–1534, 2022.

[7] B. Paris, R. Reynolds, and C. McGowan, "Sins of omission: Critical informatics perspectives on privacy in e-learning systems in higher education," *J. Assoc. Inf. Sci. Technol.*, vol. 73, no. 5, pp. 708–725, 2021.

[8] S. Mukherjee, V. Distler, G. Lenzini, and P. Cardoso-Leite, "Balancing the perception of cheating detection, privacy and fairness: A mixed-methods study of visual data obfuscation in remote proctoring," in *Proc. 2024 European Symp. Usable Security (EuroUSEC '24)*. Karlstad, Sweden: ACM, 2024.

[9] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," FIPS PUB 186-5, 2023, u.S. Department of Commerce. https://doi.org/10.6028/NIST.FIPS.186-5.

[10] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, 2017, internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc8032.

[11] H. Krawczyk, M. Belláre, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, 1997, internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc2104.

[12] Microsoft, "SetWindowDisplayAffinity function (winuser.h)," https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowdisplayaffinity, 2025, microsoft Learn. Accessed: 2026.

[13] Apple Inc., "NSWindow.SharingType," https://developer.apple.com/documentation/appkit/nswindow/sharingtype, 2025, apple Developer Documentation. Accessed: 2026.

[14] D. G. Balash, E. Korkes, M. Grant, A. J. Aviv, R. A. Fainchtein, and M. Sherr, "Educators' perspectives of using (or not using) online exam proctoring," in *Proc. 32nd USENIX Security Symp.*, Anaheim, CA, 2023, pp. 5091–5108.

[15] D. G. Balash, D. Kim, D. Shaibekova, R. A. Fainchtein, M. Sherr, and A. J. Aviv, "Examining the examiners: Students' privacy and security perceptions of online proctoring services," in *Proc. 17th Symp. Usable Privacy and Security (SOUPS)*, 2021, pp. 633–652.

[16] P. Prinsloo, M. Khalil, and S. Slade, "Vulnerable student digital well-being in AI-powered educational decision support systems (AI-EDSS) in higher education," *British J. Educ. Technol.*, vol. 55, no. 5, pp. 2075–2092, 2024.

[17] A. Duncan and D. Joyner, "On the necessity (or lack thereof) of digital proctoring: Drawbacks, perceptions, and alternatives," *J. Computer Assisted Learning*, vol. 38, no. 5, pp. 1482–1496, 2022.

[18] M. Khalil, P. Prinsloo, and S. Slade, "In the nexus of integrity and surveillance: Proctoring (re)considered," *J. Computer Assisted Learning*, vol. 38, no. 6, pp. 1589–1602, 2022.

[19] Apple Inc., "ScreenCaptureKit," https://developer.apple.com/documentation/screencapturekit/, 2025, apple Developer Documentation. Accessed: 2026.

[20] S. Kaddoura, S. Vincent, D. J. Hemanth, and I. Ashraf, "Computational intelligence and soft computing paradigm for cheating detection in online examinations," *Appl. Comput. Intell. Soft Comput.*, vol. 2023, p. 3739975, 2023.

[21] B. J. Ferdosi, M. Rahman, A. M. Sakib, T. Helaly, and P. Chakraborty, "Modeling and classification of the behavioral patterns of students participating in online examination," *Human Behavior and Emerging Technol.*, vol. 2023, p. 2613802, 2023.

[22] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," https://www.acm.org/code-of-ethics, 2018, accessed: 2026.

[23] Institute of Electrical and Electronics Engineers, "IEEE Code of Ethics," https://www.ieee.org/about/corporate/governance/p7-8, 2024, accessed: 2026.

[24] L. Simko, A. Hutchinson, A. Isaac, E. Fries, M. Sherr, and A. J. Aviv, "'Modern problems require modern solutions': Community-developed techniques for online exam proctoring evasion," in *Proc. 2024 ACM SIGSAC Conf. Computer and Communications Security (CCS '24)*, Salt Lake City, UT, 2024.

[25] R. Luijben, F. van den Broek, and G. Alpár, "Security requirements for proctoring in higher education," in *2024 IEEE Global Eng. Educ. Conf. (EDUCON)*, Kos Island, Greece, 2024, pp. 1–5.

[26] X. Wang, Q. Zheng, K. Zheng, T. Wu, and G. M. Perez, "User authentication method based on MKL for keystroke and mouse behavioral feature fusion," *Security and Communication Networks*, vol. 2020, p. 9282380, 2020.

[27] J. Guan, X. Li, Y. Zhang, and K. Andersson, "Design and implementation of continuous authentication mechanism based on multimodal fusion mechanism," vol. 2021, 2021, p. 6669429.

[28] Y. Atoum, L. Chen, A. X. Liu, S. D. H. Hsu, and X. Liu, "Automated online exam proctoring," *IEEE Trans. Multimedia*, vol. 19, no. 7, pp. 1609–1624, 2017.

[29] National Institute of Standards and Technology, "Zero Trust Architecture," NIST Special Publication 800-207, 2020, https://doi.org/10.6028/NIST.SP.800-207.

[30] Trusted Computing Group, "Trusted Platform Module Library Specification, Family 2.0," https://trustedcomputinggroup.org/resource/tpm-library-specification/, 2019, accessed: 2026.

[31] D. Balfanz, A. Czeskis, J. Hodges, J. Jones, M. B. Jones, A. Kumar, A. Liao, R. Lindemann, and E. Lundberg, "Web Authentication: An API for accessing Public Key Credentials Level 2," https://www.w3.org/TR/webauthn-2/, 2021, w3C Recommendation. Accessed: 2026.