A Lightweight Email–OTP Access Gate with Multi-Key Rate Limiting for Institutional LLM Chatbots in Low-Code Orchestration

John Cheung *

Posted Date: 9 January 2026

doi: 10.20944/preprints202601.0661.v1

Keywords: LLM chatbot; access control; authentication; email-OTP; rate limiting; low-code; cybersecurity; threat model; system design; transitional security

 doi:10.20944/preprints202601.0661.v1

Article

A Lightweight Email–OTP Access Gate with Multi-Key Rate Limiting for Institutional LLM Chatbots in Low-Code Orchestration

John Cheung

College of Professional and Continuing Education

Abstract

Institutional LLM chatbots are frequently deployed before enterprise single sign-on (SSO) is available, yet still require baseline access control, cost containment, and abuse prevention. This paper presents a pragmatic access-gating blueprint implementable in low-code orchestration platforms: (i) endpoint-specific, per-identity rate limiting composed across multiple keys (session/user id, email, and IP); (ii) eligibility enforcement via institutional email-domain allowlists with hardened normalization; and (iii) email-delivered one-time passcodes (OTP) to verify mailbox control before enabling chat functionality. Beyond describing control flow and state, we make security-critical choices explicit and standards-aligned (CSPRNG OTP generation; hashed-at-rest verifiers using HMAC with per-issuance nonce, explicit domain separation "context", and key identifiers; constant-time comparisons; TTL/attempt limits; single-active issuance with atomic rotation; session fixation defenses; CSRF-safe submit endpoints; key management and rotation). We quantify an online-guessing upper bound under stated limits and discuss why OTP spraying dominates risk, motivating anomaly-driven greylisting and multi-key limiter composition suitable for shared-IP campus environments. To reduce reliance on opaque low-code trust anchors, we provide a vendor-agnostic platform verification checklist and a reference external state-store pattern with *signed, replay-resistant, monotonic* state transitions and revocation semantics. Finally, we position email-OTP gates as a transitional control within broader enterprise LLM security posture (OWASP and participant-aware access control), and give a concrete migration roadmap toward stronger identity mechanisms (OIDC/OAuth, TOTP, and WebAuthn). The contribution is a systems design and operational blueprint rather than a novel algorithm or empirical study.

Keywords: LLM chatbot; access control; authentication; email-OTP; rate limiting; low-code; cybersecurity; threat model; system design; transitional security

---

1. Introduction

LLM chatbots are increasingly adopted for internal knowledge access, service support, and productivity tasks. Institutions typically require: (a) restricting access to eligible populations (e.g., staff and students), (b) curbing automated misuse (spam, probing, resource exhaustion), and (c) controlling cost and availability of downstream inference.

Enterprise deployments commonly rely on identity providers and SSO (OIDC/SAML) integrated with directory-backed authorization. In practice, early pilots are often deployed before SSO integration is available due to governance lead time, procurement, or engineering constraints. Teams may fall back to insecure or fragile measures (shared passwords, open access, manual intervention), increasing risk and operational burden.

This paper describes a lightweight, transitional access gate implemented within a low-code chatbot orchestration workflow: users authenticate by demonstrating control of an eligible institutional email address via an emailed OTP. Endpoint-specific rate limiting is applied to (i) requesting OTPs, (ii) submitting OTPs, and (iii) post-auth chat. The design is a targeted capability restriction and cost control mechanism [1].

 doi:10.20944/preprints202601.0661.v1

Scope and non-goals. The gate provides *mailbox-control* verification, not high-assurance identity proof. It should not be represented as equivalent to SSO or phishing-resistant MFA. It aims to reduce casual abuse and increase accountability during controlled pilots while the institution plans migration to stronger identity and authorization controls consistent with Digital Identity Guidelines [5,6].

Contributions.

1. An end-to-end specification of an email–OTP access gate with endpoint-specific, multi-key rate limiting in a low-code orchestration setting.
2. A platform-oriented threat model emphasizing low-code integrity risks and connector/webhook security.
3. Security-critical implementation guidance: CSPRNG; HMAC verifiers with explicit context; constant-time compare; TTL/attempt limits; atomic OTP rotation; session fixation and CSRF defenses; key management and rotation.
4. Standardized pseudo-code and configuration templates.
5. Vendor-agnostic platform validation checklists and a replay-resistant external state-store pattern with monotonic signed transitions and revocation semantics.
6. Governance guidance grounded in privacy risk frameworks [2,3] and a migration/deprecation roadmap toward OIDC/TOTP/WebAuthn [13–16,29].

2. Background, Standards, and Related Work

2.1. Lightweight Access Gates and Alternatives

Email-delivered OTP is widely used for moderate assurance access when SSO is unavailable. Related standardized OTP mechanisms include HOTP [7] and TOTP [8]. In web chat UI contexts, institutions often prefer short numeric codes for usability; for higher-stakes systems, longer/alphanumeric codes or phishing-resistant factors should be used.

Alternatives include:

- **Magic links**: signed, single-use URLs with short TTL (similar security envelope to email OTP but different UX and replay considerations).
- **TOTP apps**: stronger than mailbox-control if the shared secret is protected and phishing is mitigated; standardized in RFC 6238 [8].
- **WebAuthn/passkeys**: phishing-resistant public-key credentials for web applications [16].

Secretless OTP constructions such as T/Key reduce server-side secret storage but change client requirements [26]. Time-dependent MAC and replay-resistant message authentication constructions are adjacent for designing time-bound tokens and magic-link style assertions [27].

2.2. Identity Assurance and Best-Practice Guidance

NIST SP 800-63-3 defines a componentized approach to identity assurance levels (IAL/AAL/FAL) [5]. NIST notes that this revision has been superseded by SP 800-63-4 as of August 1, 2025 [4]. We reference SP 800-63-3 for continuity and use it only for framing assurance limitations rather than claiming compliance.

For OAuth/OIDC end-states, we reference the OAuth 2.0 core framework [9] and bearer tokens [10], the OAuth threat model [11], PKCE [12], and the OAuth Security Best Current Practice [13]. OAuth 2.1 consolidates and updates these practices [14]. OpenID Connect Core defines authentication on top of OAuth 2.0 [15].

For web application security verification and threat framing, we reference OWASP ASVS [17] and OWASP Authentication/MFA guidance [18,19], and we connect gating to LLM-specific risk categories (e.g., prompt injection) per OWASP's Top 10 for LLM applications [20].

2.3. Rate Limiting

Token-bucket limiters are widely used due to simplicity and burst tolerance. Queueing analyses of token buckets for heterogeneous flows can motivate tuning under mixed workloads (backlog,

 doi:10.20944/preprints202601.0661.v1

delay, and loss tradeoffs) [25]. Other commonly deployed rate-limit algorithms include leaky bucket, sliding-window counters, and GCRA-style approaches; distributed enforcement can be implemented via centralized throttling services or edge proxies. This paper uses token buckets as a baseline because they are implementable in low-code contexts with minimal dependencies.

2.4. Continuous Human Verification and Greylisting

Greylisting commonly uses CAPTCHAs, proof-of-work, or step-up challenges when automation is suspected. Recent work proposes the *Human Challenge Oracle* (HCO) as a security primitive for *continuous*, rate-limited human verification, aiming to make sustaining many identities require ongoing human effort rather than a one-time check [32]. While our setting is an institutional access gate (not open consensus), the same intuition motivates our greylist design: anomaly triggers should shift the cost curve for large-scale OTP spraying by requiring per-request human work that is validated server-side and cannot be bypassed by API calls (Section 9).

2.5. Enterprise RAG Frameworks and Measurements in the Wild

Systems guidance for enterprise RAG chatbots emphasizes centralized gateways, access control, and security controls as part of an end-to-end stack. The FACTS framework (Freshness, Architectures, Cost, Testing, Security) systematizes control points and trade-offs for enterprise-grade RAG chatbots [30]. Complementary large-scale Internet measurements report widespread unauthenticated or weakly protected LLM endpoints in real deployments, reinforcing the need for secure-by-default gates even for early pilots [31].

2.6. Low-Code Orchestration and LLM Security Posture

Low-code orchestration accelerates deployment but concentrates risk in connector credentials, workflow webhooks, and conversation-state integrity. Participant-aware access control has been proposed as a deterministic safeguard for enterprise AI systems, complementing entry-point gating [28]. This paper treats email-OTP gating as an outer perimeter control that must coexist with retrieval-side ACL enforcement and participant-aware rules.

2.7. Privacy and Governance

Privacy risk in conversational AI spans the data lifecycle: collection, usage, and retention. Frameworks for privacy harms in conversational AI and user attitudes toward privacy/security support telemetry minimization and transparency practices [2,3].

3. Threat Model and Assumptions

3.1. Assets

Primary assets include: (i) service availability and inference budget; (ii) eligibility policy; (iii) confidentiality of prompts/responses; (iv) connector credentials (email, model gateway, retrieval connectors); and (v) integrity of authentication state and policy variables.

3.2. Adversaries

We consider adversaries who can:

- perform OTP spraying (high-volume OTP requests across many emails),
- attempt online guessing of OTPs under rate limits,
- enumerate allowlists via errors, timing, or deliverability side channels,
- exploit NAT/VPN/rotating IPs to evade simplistic limiters,
- attempt session fixation or replay across tabs/devices,
- target webhooks/connectors (credential theft, replay, SSRF-like egress abuse),
- inject adversarial content (prompt injection) to subvert downstream tools.

 doi:10.20944/preprints202601.0661.v1

3.3. Trust Anchors and Validation

The primary trust anchor is that the orchestration platform enforces session variables server-side and prevents client-side tampering of privileged state. Because platform guarantees vary widely, Section 9 provides (i) a vendor-agnostic validation checklist and (ii) an external state-store pattern that cryptographically binds state updates and rejects replay/stale transitions.

4. System Overview

4.1. Workflow Stages

1. **Rate limiting**: endpoint-specific token buckets composed across session/user id, email, and IP.
2. **Authentication check**: if session is authenticated and TTL-valid, route to chat.
3. **OTP issuance**: if an eligible email is provided, issue OTP and store only a verifier.
4. **OTP submission**: validate OTP under TTL/attempt caps; set authenticated state on success.
5. **Post-auth routing**: forward prompts to the LLM via a gateway and enforce downstream guardrails.

4.2. State Variables

Minimum recommended state (in-platform or externalized):

- **Session**: `session_id`, `auth_state`, `session_issued_at`, `session_expires_at`, `session_version`, `session_kid`
- **OTP**: `otp_email`, `otp_nonce`, `otp_verifier`, `otp_issued_at`, `otp_attempts`, `otp_kid`
- **Operational**: `reject_reason` (safe user-facing), structured audit logs (server-side), and optional `greylist_until`.

4.3. Control Flow Protocol

1. Determine endpoint type: *request-OTP*, *submit-OTP*, or *post-auth chat*.
2. Apply endpoint-specific multi-key rate limiting (Section 7). If blocked, return a uniform message.
3. If `auth_state=1` and session TTL valid, route message to LLM (with downstream guardrails).
4. Else if message is an eligible email, issue OTP (store verifier, send email) subject to greylisting checks.
5. Else if message is an OTP submission, validate (TTL + attempts + constant-time verifier compare + binding).
6. Else return a uniform guidance message (do not reveal allowlist membership or email existence).

5. Security-Critical Implementation Details

5.1. OTP Generation and Parameters

A 6-digit numeric OTP yields $N = 10^6$ possibilities. With strict TTL and attempt limits, this may be acceptable for low-stakes pilots; higher-stakes contexts should increase N (8 digits or alphanumeric) or transition to phishing-resistant MFA (WebAuthn) [16]. Note that standards-based OTP schemes exist (HOTP/TOTP) [7,8]; the email-OTP described here is *server-issued* and primarily intended to prove mailbox control, not to implement HOTP/TOTP.

5.2. CSPRNG Requirement

OTPs and nonces MUST be generated using a cryptographically secure RNG. Avoid custom PRNGs or time-based generation.

5.3. Hashed-at-Rest Verifier with Explicit Context (Domain Separation)

Do not store OTP plaintext. Store:

$$\texttt{otp\_verifier} = \mathrm{HMAC}_{K_v[\texttt{kid}]}(\texttt{otp\_code} \,\|\, \texttt{email\_canon} \,\|\, \texttt{otp\_nonce} \,\|\, \texttt{context}),$$

 doi:10.20944/preprints202601.0661.v1

where `kid` identifies the key version and `context` is an explicit domain-separation string.
Defining `context`. To avoid ambiguity, define:

$$\texttt{context} = \texttt{DS} \,\|\, \texttt{app\_id} \,\|\, \texttt{env} \,\|\, \texttt{purpose} \,\|\, \texttt{endpoint},$$

e.g., `DS="cpce-otp-gate:v1"`; `app_id` a stable identifier ("llm-chatbot"); `env` in {prod,staging}; `purpose="otp_verifier"`; `endpoint="submit_otp"`. This prevents cross-application reuse and clarifies verifier semantics under refactoring or multi-tenant deployments.

5.4. Constant-Time Comparison

Use constant-time equality (e.g., `hmac.compare_digest`) to avoid timing leaks [17].

5.5. Key Management, Rotation, and Binding to `kid`

Store K_v in a secret manager (never embedded in workflow code). Use per-environment keys and (optionally) per-application scoping. Rotate keys on a schedule with an overlap window (accept current and previous) to avoid outages during rotation.
Binding verifiers to a key version. Store `otp_kid` alongside the verifier. On validation, compute the candidate verifier using `otp_kid`. During overlap rotation, optionally also attempt verification with the immediately previous key only if `otp_kid` is missing (backward compatibility). This avoids ambiguity and limits the blast radius of a compromised key.

5.6. Key-Compromise Resilience and Offline-Verification Risk

Storing only a keyed verifier prevents accidental disclosure of OTPs in logs or database dumps. However, if an attacker obtains *both* (i) the OTP issuance record (`email_canon`, `otp_nonce`, `otp_verifier`) and (ii) the verifier key K_v (or an oracle that computes HMAC under K_v), they can test all N candidate codes offline and recover the OTP within the TTL window. This is not the dominant threat model for most pilots—an adversary with K_v access is typically a high-privilege compromise—but the blast radius should be explicit and reduced.
Defense-in-depth options.

- **Prefer non-exportable keys**: keep K_v inside a KMS/HSM/TEE and invoke HMAC via an authenticated API (rate-limited and audited). Non-exportability reduces the feasibility of large offline guessing even if the OTP state store is exposed.
- **Minimize verifier exposure**: store OTP state in a dedicated backend store with least privilege; never log `otp_nonce` or `otp_verifier`; delete records on success and after TTL+grace. Treat access to OTP state as sensitive as access to session cookies.
- **Increase brute-force cost when needed**: enlarge N (e.g., 8 digits or short alphanumeric), shorten TTL, and reduce attempt caps for higher-stakes contexts.
- **Optional memory-hard wrapping**: if offline compromise risk is a concern, store a memory-hard verifier such as $\text{Argon2id}(\text{HMAC}_{K_v}(\cdot),\ \text{salt} = \texttt{otp\_nonce})$, implemented in an external authentication service to avoid unpredictable low-code runtimes [35,36]. This does not prevent compromise, but can slow offline checking.
- **Key separation**: derive purpose-specific keys (OTP verifiers vs. session assertions vs. rate-limit keyed hashes) using HKDF to avoid cross-protocol key reuse [33].

5.7. Binding, Atomic Rotation, and Concurrency

Bind OTP issuance to canonicalized email and to a server-issued `session_id`. On OTP re-issue, atomically overwrite prior nonce/verifier (single active issuance) and accept only the most recent version. For multi-device or multi-tab scenarios, store OTP state keyed by (`email_canon`, `session_id`) in a backend store when supported, and update using compare-and-swap (CAS) semantics (Section 9).

 doi:10.20944/preprints202601.0661.v1

5.8. TTL, Attempt Caps, Cooldown, One-Time Use

Enforce TTL (e.g., 10 minutes), attempt caps (e.g., 5), and cooldown after repeated failures (e.g., 15 minutes). Invalidate OTP on first successful use. Use uniform failures and add small jitter for repeated failures to reduce enumeration and automation.

5.9. Email Parsing, Canonicalization, and Allowlist Correctness

Email parsing and canonicalization are error-prone. Follow a conservative approach:

- Parse `local-part@domain` per Internet email conventions [21,22].
- Casefold the domain to lower-case. For local-part, default to casefolding for compatibility, but do not assume semantic meaning across domains [21,22].
- Normalize internationalized domains using IDNA2008 guidance [23]; consider rejecting Unicode confusables [24].
- Enforce allowlists via exact domain match, unless policy explicitly allows subdomains.

Aliases and plus-addressing. Because alias semantics are domain-specific, the blueprint separates: (i) `email_delivery` used for sending, and (ii) `email_rate_key` used for rate limiting and spraying detection. Institutions can configure a per-domain rule (e.g., strip "+tag" for domains where plus-addressing is known) to prevent alias-based evasion, without over-normalizing unknown domains.

5.10. Shared Mailboxes, Role Accounts, and Forwarding Aliases

Domain allowlists are a coarse eligibility control: they may admit shared helpdesk inboxes, distribution lists, role accounts (e.g., `it-support@`), and forwarding aliases that undermine individual accountability. Because mailbox semantics are institution-specific, we recommend making the policy explicit and testable:

- **Define eligible mailbox types**: personal staff/student accounts where a single person is accountable for mailbox access.
- **Maintain a denylist for non-person entities**: (i) known role accounts, (ii) shared mailboxes, and (iii) distribution lists. A conservative default is to deny addresses whose local-part matches common role patterns (e.g., `admin`, `support`, `help`, `noreply`) unless explicitly excepted.
- **Prefer directory-backed validation when available**: if the institution has an identity directory export (LDAP/AD/SCIM), validate that the address corresponds to an active person record and (optionally) that it is not marked as a group/distribution mailbox. This avoids relying on string heuristics alone.
- **Treat forwarding as reduced assurance**: if policy permits forwarded addresses, communicate that accountability is weaker and consider requiring step-up (TOTP/WebAuthn) for sensitive actions.

6. Quantifying Online-Guessing Risk

Under uniform random OTP and independent guesses, success probability in one TTL window is bounded by:

$$P_{\text{win}} \leq \frac{m}{N},$$

where m is allowed guesses per bound identity (e.g., per email) and N is code space size. For $m = 5$ and $N = 10^6$, $P_{\text{win}} \leq 5 \times 10^{-6}$.

Across W independent windows, a union bound yields $P_{\text{total}} \leq Wm/N$. Attackers can parallelize across many emails; thus the dominant risk driver is large-scale OTP spraying and enumeration. This motivates (i) strict request-OTP throttling, (ii) global distinct-email anomaly metrics, and (iii) greylisting (CAPTCHA/proof-of-work) before OTP issuance.

 doi:10.20944/preprints202601.0661.v1

7. Multi-Key Rate Limiting and Composition

A recurring operational question is how to compose rate limits across email/session/IP, especially behind shared-IP campus NAT.

7.1. Limiter Model

We recommend token buckets per (`endpoint`, `keytype`, `keyvalue`). Token buckets provide burst tolerance, and queueing analyses help reason about backlog/latency under mixed workloads [25].

7.2. Staged Composition Policy: Hard Gates, Soft Backstops, Greylisting

To limit false positives under NAT, we recommend a staged policy:

- **Hard gates**: enforce per-email and per-session buckets for OTP request and OTP submit.
- **Soft backstops**: treat IP buckets primarily as anomaly signals rather than immediate blocks (except for extreme abuse).
- **Escalation**: when IP anomalies occur (e.g., many distinct emails), require greylisting friction *before* OTP issuance; do not send OTP until greylist is satisfied.

7.3. Concrete Policy Template

Table 1. Example multi-key rate-limit template. Tune using pilot telemetry and shared-IP realities.

Endpoint	Primary keys	Example policy (token bucket)
Request OTP	email + session IP (soft)	3/10min per email; 10/10min per session. Greylist if >20 distinct emails/10min per IP; hard block if >200 requests/10min.
Submit OTP	email + session	5 attempts/10min per email; 8/10min per session; 15min cooldown after cap.
Post-auth chat	session/user id	60 msgs/hour; daily token budget; separate cap for tool calls.

7.4. Composing Conflicting Signals

When signals conflict (e.g., email/session allow, IP anomalous), we recommend:

1. For OTP endpoints: require *email AND session* to pass; if IP is anomalous, return a greylist challenge (not a hard block) unless extreme.
2. For chat: enforce session/user limits and optionally apply IP limits only at extreme thresholds to avoid penalizing campus NAT.

This "hard AND + soft anomaly" policy is implementable in low-code stacks and prioritizes low false positives while still detecting spraying.

7.5. Distributed Spraying Beyond Per-IP Throttling

Distributed attackers can rotate IPs; incorporate:

- per-domain quotas (cap OTP sends per domain/time),
- global anomaly metrics (distinct emails per time, failure ratios),
- reputation signals (known bad ASNs, prior failure ratios),
- greylisting friction (CAPTCHA/proof-of-work) that cannot be bypassed by API calls (Section 9).

7.6. Tuning for Bursty Legitimate Usage Behind Shared IPs

Institutional deployments frequently face "bursty" but legitimate usage (e.g., a computer-lab class where many students share a single NATed IP). To reduce false positives without materially weakening anti-spraying protections:

- **Keep hard gates on email/session**: these scale with people rather than with network topology.

 doi:10.20944/preprints202601.0661.v1

- **Raise IP anomaly thresholds in campus networks**: treat IP primarily as a soft signal and only hard-block at extreme levels; otherwise escalate to greylisting.
- **Prefer greylisting to blanket IP blocks**: when an IP looks anomalous (many distinct emails), require a human challenge *before* sending OTPs. This aligns with continuous-human-effort principles (HCO-style) where sustained scale incurs sustained effort [32].
- **Use time-of-day and cohort signals**: if the institution knows scheduled lab times or cohorts, temporarily widen buckets for known windows while maintaining per-email limits.

Table 2. Illustrative tuning profiles. Tune using pilot telemetry and adversarial tests.

Profile	Environment	Suggested adjustments
Campus lab NAT	Many users share one IP	Increase IP anomaly thresholds; keep per-email request caps; prefer greylist over IP hard-blocking.
Remote workforce	Low IP sharing	Treat IP as a stronger anomaly signal; lower greylist thresholds; shorten greylist windows.

8. Deliverability, Anti-Enumeration, and Assurance Communication

Email OTP is vulnerable to phishing and mailbox compromise. To avoid overconfidence, we recommend explicit messaging grounded in OWASP guidance [19].

8.1. Assurance Statement Template

> **Assurance level.** This service uses *email code verification* to confirm control of an eligible mailbox. It is intended for controlled pilots and does **not** provide the same assurance as SSO (OIDC) or phishing-resistant MFA (WebAuthn). Do not share codes. If you suspect mailbox compromise, stop using the service and contact support.

8.2. Deliverability and Anti-Phishing Measures

Align sender identity with SPF/DKIM/DMARC; use stable sender identity and templates; add clear anti-phishing cues (never request passwords; code expires; official sender). These steps materially impact completion rates in practice.

8.3. Anti-Enumeration and Bounce Side Channels

Uniform user-facing responses should not reveal whether an email address exists, whether it is allowlisted, or whether delivery succeeded. Avoid surfacing SMTP bounce status to end users; bounces can be handled asynchronously for sender reputation management, but the UI should not become an oracle.

A practical error and latency equalization recipe.

1. **Uniform envelope**: return the same HTTP status and response shape for *all* outcomes of request-OTP and submit-OTP (success, ineligible, rate-limited, wrong code).
2. **No early exits**: run a constant "baseline" amount of server work on every path (e.g., compute a dummy HMAC and perform a constant-time comparison) before responding.
3. **Timing equalization**: enforce a minimum response time (e.g., a fixed response "envelope") and add bounded jitter on failures to reduce oracle precision for automated probing.
4. **Side-channel hygiene**: do not expose SMTP/DMARC failures, allowlist membership, or rate-limit key selection in UI text; log these server-side only.
5. **Measure leakage**: instrument and periodically compare response-time distributions for eligible vs. ineligible paths; treat a consistent gap beyond network noise as a defect to remediate.

8.4. Data Minimization, Retention, and Institutional Compliance

Access gates process personal data (email identifiers, IP addresses, session identifiers) and may sit in front of chat transcripts that contain sensitive content. Institutions should adopt a "minimum

 doi:10.20944/preprints202601.0661.v1

necessary" approach aligned with internal governance and applicable regulations (e.g., GDPR/FERPA/CCPA) and document it in the pilot plan.

Practical defaults.

- **Do not log OTP codes**. Avoid logging OTP nonces or verifiers unless strictly required for debugging, and never in plaintext.
- **Short retention for OTP state**: keep OTP issuance state for at most TTL+grace (minutes), then delete.
- **Bounded retention for security events**: keep authentication/rate-limit audit events for a limited period (e.g., 30–90 days) consistent with incident response needs and institutional policy.
- **Minimize IP storage**: where feasible, store truncated or hashed IPs for anomaly detection, and restrict access to raw IPs.
- **Separate auth telemetry from chat content**: store access-gate telemetry separately from conversation transcripts, with independent retention and access controls.

This paper does not provide legal advice; deployments should be reviewed by the institution's privacy/compliance office.

9. Low-Code Platform Hardening and Reference External State Pattern

Because platform integrity varies, operators should validate and/or externalize critical state.

9.1. Vendor-Agnostic Platform Validation Checklist

Before relying on in-platform variables for authentication, verify:

1. **Server-side enforcement**: can clients modify conversation variables or system messages?
2. **Isolation**: are variables isolated per user/session and protected from cross-tenant leakage?
3. **Provenance**: is message history treated as untrusted input (no privilege escalation via injected roles), consistent with LLM risk guidance [20]?
4. **Connector scope**: are secrets stored in a vault with least privilege and rotation [17]?
5. **Webhook security**: are webhook endpoints authenticated and replay-protected (timestamp + nonce)?

9.2. Replay-Resistant External State-Store with Monotonic Signed Transitions

To reduce reliance on platform guarantees, store canonical session state externally (e.g., Redis/-PostgreSQL) keyed by `session_id`:

$$\texttt{rec} = (\texttt{auth\_state}, \texttt{expires\_at}, \texttt{version}, \texttt{revoked\_at}, \texttt{kid}).$$

Return to the low-code layer a signed assertion:

$$\sigma = \mathrm{HMAC}_{K_s[\texttt{kid}]}(\texttt{session\_id} \,\|\, \texttt{auth\_state} \,\|\, \texttt{expires\_at} \,\|\, \texttt{version} \,\|\, \texttt{kid} \,\|\, \texttt{context}_s),$$

with `context_s` domain-separated similarly to Section 5.

Why sign if we also read canonical state? If the gateway can securely fetch the canonical record `rec` on every request, a plain state-store lookup (with server-side ACLs) is sufficient and the signature can be omitted. The signature is most valuable when some workflow steps run in components whose integrity is difficult to verify (opaque low-code steps, client-mediated transitions, or webhook callbacks): it binds *what the low-code layer claims* about `auth_state/expires_at/version` to a server-held secret, preventing privilege escalation via state tampering. Replay resistance and revocation still require a monotonic check against canonical state (or an equivalent revocation oracle).

Stateless token variants. Some deployments may prefer short-lived stateless bearer tokens (e.g., JWTs) for the authenticated session [34]. Stateless tokens reduce state-store reads but weaken revocation and replay control unless paired with an epoch/version oracle; thus we treat them as an optimization for low-sensitivity pilots with short session TTLs, not as a default for institutional systems.

 doi:10.20944/preprints202601.0661.v1

Replay resistance and monotonicity. On each state update, increment `version` and write using CAS (compare-and-swap) semantics: accept an update only if `expected_version` matches the current stored version. On each request, validate σ and *also* fetch `rec` and reject assertions whose `version` is stale (less than the stored version). This prevents replay of earlier signed transitions even if `expires_at` has not elapsed.

Revocation semantics. Support immediate revocation by setting `revoked_at` and incrementing `version`. All future requests must check `revoked_at` and deny if set. For incident response, optionally maintain a per-user or per-domain revocation list or epoch counter.

9.3. Concrete Web Primitives for Session Fixation and CSRF Defenses

When the OTP submit endpoint is reachable from browsers, implement concrete primitives consistent with OWASP guidance [18]:

- **Session cookies**: `Secure`, `HttpOnly`, and `SameSite=Lax` (or `Strict` where feasible).
- **Rotate session identifiers**: issue a fresh `session_id` after successful OTP verification (mitigates fixation).
- **CSRF**: require POST for submit-OTP; validate `Origin`/`Referer` headers; and use either synchronizer tokens or double-submit cookies.
- **CORS**: do not allow cross-origin credentials except the intended origin.

If the low-code platform cannot enforce these primitives, prefer the external-state pattern and require server-validated signed request tokens for state transitions.

9.4. Greylisting Mechanism That Cannot Be Bypassed

Greylisting should be checked server-side in the OTP issuance step. A concrete pattern is: (i) require a CAPTCHA token or proof-of-work solution for anomalous IPs, (ii) validate it server-side, and (iii) store `greylist_until` in the external state-store. OTP issuance checks `greylist_until` regardless of client presentation; thus API callers cannot bypass the requirement.

10. Standardized Pseudo-Code Snippets

10.1. OTP Issuance and Verification

```
# Kv and Ks are stored in vault/KMS; keys are selected by kid
# context fields domain-separate app/env/purpose/endpoint

make_context(app_id, env, purpose, endpoint):
  return "cpce-otp-gate:v1" || app_id || env || purpose || endpoint

issue_otp(email_delivery, session_id, now):
  email_canon = canonicalize(email_delivery) # domain IDNA, confusables policy
  require(eligible_domain(email_canon))
  require(!greylisted(session_id, ip, now))
  otp_code = CSPRNG_numeric(6)
  otp_nonce = CSPRNG_bytes(16)
  otp_kid = current_kid("Kv")
  ctx = make_context(app_id, env, "otp_verifier", "submit_otp")
  verifier = HMAC(Kv[otp_kid], otp_code || email_canon || otp_nonce || ctx)
  store_otp_CAS(session_id, email_canon, otp_nonce, verifier, otp_kid,
              issued_at=now, attempts=0, version+=1) # single-active issuance
  send_email(email_delivery, otp_code, ttl_minutes=10)
  return uniform_ok()

verify_otp(email_input, session_id, user_code, now):
  s = load_otp(session_id)
```

 doi:10.20944/preprints202601.0661.v1


```
  if canonicalize(email_input) != s.email_canon: return fail_uniform()
  if now - s.issued_at > TTL: return fail_uniform()
  if s.attempts >= MAX_ATTEMPTS: return fail_uniform_cooldown()
  ctx = make_context(app_id, env, "otp_verifier", "submit_otp")
  cand = HMAC(Kv[s.otp_kid], user_code || s.email_canon || s.otp_nonce || ctx)
  if const_time_equal(cand, s.verifier):
      invalidate_otp_CAS(session_id) # one-time use
      session_id2 = rotate_session_id()
      set_auth_state_external_CAS(session_id2, auth_state=1, expires=now+8h)
      return success()
  else:
      increment_attempts_CAS(session_id)
      sleep(jitter())
      return fail_uniform()
```

10.2. Token Bucket and Staged Multi-Key Composition

```
allow(endpoint, keys):
  # keys: {session_id, email_rate_key, ip}
  if endpoint in {REQ_OTP, SUBMIT_OTP}:
     if !bucket_allow(endpoint, "email", keys.email_rate_key): return BLOCK
     if !bucket_allow(endpoint, "session", keys.session_id): return BLOCK
     if ip_anomaly(endpoint, keys.ip): return GREYLIST
     return ALLOW
  else:
     if !bucket_allow(endpoint, "session", keys.session_id): return BLOCK
     return ALLOW
```

10.3. Signed Transition Verification (External State-Store)

```
verify_session_assertion(assertion):
  rec = db.get(assertion.session_id)
  if rec.revoked_at != null: return FAIL
  if assertion.version != rec.version: return FAIL # rejects stale replay
  if now > rec.expires_at: return FAIL
  ctxs = make_context(app_id, env, "session_assertion", "any")
  sig2 = HMAC(Ks[rec.kid], assertion.sid || rec.state || rec.expires ||
              rec.version || rec.kid || ctxs)
  if !const_time_equal(sig2, assertion.sig): return FAIL
  return OK
```

11. Operational Evaluation Protocol

We abstain from claims of measured improvement without data. For evidence-based tuning, a minimal 4–8 week pilot should measure:

- authentication completion rate and P50/P95 time-to-auth,
- OTP deliverability latency distribution, bounce/spam-folder incidence, and support tickets,
- limiter trigger rate and false-positive analysis for shared IPs (campus NAT),
- model usage and cost before/after gating,
- sensitivity analyses for code length/TTL/attempt caps and anomaly thresholds,
- adversarial load tests (spraying across rotating IPs and emails) and greylist effectiveness.

 doi:10.20944/preprints202601.0661.v1

12. Migration and End-to-End Security Posture

Email-OTP gates are outer-perimeter controls. For confidentiality, retrieval-side ACL enforcement and participant-aware rules are complementary and often necessary [28]. A concrete migration pathway is to integrate OIDC/OAuth tokens into AI tooling and tool protocols, improving identity assurance and auditability [13–15]. Recent work provides a concrete example for emerging AI tool stacks: integrating OAuth 2.0 and OIDC into Model Context Protocol (MCP)-enabled developer environments, including an implementation blueprint and case study measurements [29]. For phishing resistance, deploy WebAuthn where possible [16]; for intermediate steps, consider TOTP [8]. Institutions should define triggers for migration: sustained attack pressure, expansion to sensitive workflows, unacceptable phishing risk, or high support burden.

13. Limitations

Email OTP remains vulnerable to phishing and mailbox compromise. Multi-key limiter composition and anomaly thresholds are heuristic without empirical validation; shared-IP environments can still experience friction. Platform state integrity may not be verifiable; this motivates externalization patterns but does not eliminate the need for careful operational assurance. The blueprint provides informal bounds (e.g., m/N) and best-practice guidance, not formal proofs under all adversaries nor deployment measurements.

14. Conclusion

We presented a security-aware, low-code friendly blueprint for gating institutional LLM chatbots when SSO is unavailable. The design makes explicit the security-critical implementation details that determine practical robustness, provides multi-key limiter composition suitable for shared-IP campuses, and adds replay-resistant, monotonic signed-transition patterns to mitigate low-code trust gaps. We contextualize the gate within established authentication standards and best-practice guidance and provide a clear migration roadmap toward stronger identity and access control.

References

1. Markus Anderljung and Julian Hazell. Protecting society from AI misuse: when are restrictions on capabilities warranted? *arXiv:2303.09377*, 2023. https://arxiv.org/abs/2303.09377
2. Ece Gumusel, Kyrie Zhixuan Zhou, and Madelyn Rose Sanfilippo. User privacy harms and risks in conversational AI: a proposed framework. *arXiv:2402.09716*, 2024. https://arxiv.org/abs/2402.09716
3. Mutahar Ali, Arjun Arunasalam, and Habiba Farrukh. Understanding users' security and privacy concerns and attitudes towards conversational AI platforms. *arXiv:2504.06552*, 2025. https://arxiv.org/abs/2504.06552
4. NIST. Digital Identity Guidelines (SP 800-63 suite). https://pages.nist.gov/800-63-3/
5. Paul A. Grassi, James L. Fenton, et al. NIST Special Publication 800-63-3: Digital Identity Guidelines. National Institute of Standards and Technology, 2017 (updates posted online). https://pages.nist.gov/800-63-3/sp800-63-3.html
6. Paul A. Grassi, James L. Fenton, et al. NIST SP 800-63B: Authentication and Lifecycle Management. NIST Computer Security Resource Center. https://csrc.nist.gov/pubs/sp/800/63/b/upd2/final
7. D. M'Raihi, M. Bellare, F. Hoornaert, and D. Naccache. RFC 4226: HOTP: An HMAC-Based One-Time Password Algorithm. IETF, 2005. https://www.ietf.org/rfc/rfc4226.txt
8. D. M'Raihi, J. Rydell, M. Pei, S. Machani, and J. Erlandson. RFC 6238: TOTP: Time-Based One-Time Password Algorithm. IETF, 2011. https://datatracker.ietf.org/doc/html/rfc6238
9. D. Hardt. RFC 6749: The OAuth 2.0 Authorization Framework. IETF, 2012. https://datatracker.ietf.org/doc/html/rfc6749
10. M. Jones and D. Hardt. RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage. IETF, 2012. https://datatracker.ietf.org/doc/html/rfc6750
11. E. Hammer-Lahav, D. Recordon, and D. Hardt. RFC 6819: OAuth 2.0 Threat Model and Security Considerations. IETF, 2013. https://datatracker.ietf.org/doc/html/rfc6819

 doi:10.20944/preprints202601.0661.v1

12. N. Sakimura, J. Bradley, and N. Agarwal. RFC 7636: Proof Key for Code Exchange by OAuth Public Clients. IETF, 2015. https://datatracker.ietf.org/doc/html/rfc7636
13. T. Lodderstedt, M. Jones, and D. Hardt. RFC 9700: OAuth 2.0 Security Best Current Practice. IETF, 2025. https://datatracker.ietf.org/doc/rfc9700/
14. OAuth Working Group. The OAuth 2.1 Authorization Framework (Internet-Draft). IETF Datatracker, accessed 2026-01. https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/
15. OpenID Foundation. OpenID Connect Core 1.0 incorporating errata set 2. https://openid.net/specs/openid-connect-core-1_0.html
16. W3C. Web Authentication: An API for accessing Public Key Credentials. https://www.w3.org/TR/webauthn-3/
17. OWASP. OWASP Application Security Verification Standard (ASVS). https://owasp.org/www-project-application-security-verification-standard/
18. OWASP. OWASP Authentication Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
19. OWASP. OWASP Multifactor Authentication Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html
20. OWASP. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/
21. J. Klensin. RFC 5321: Simple Mail Transfer Protocol. IETF, 2008. https://datatracker.ietf.org/doc/html/rfc5321
22. P. Resnick. RFC 5322: Internet Message Format. IETF, 2008. https://datatracker.ietf.org/doc/html/rfc5322
23. J. Klensin. RFC 5890: Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework. IETF, 2010. https://datatracker.ietf.org/doc/html/rfc5890
24. Unicode Consortium. UTS #39: Unicode Security Mechanisms. https://www.unicode.org/reports/tr39/
25. Henrik Schiøler, John Leth, and Shibarchi Majumder. Markovian performance model for token bucket filter with fixed and varying packet sizes. *arXiv:2009.11085*, 2020. https://arxiv.org/abs/2009.11085
26. Dmitry Kogan, Nathan Manohar, and Dan Boneh. T/Key: second-factor authentication from secure hash chains. *arXiv:1708.08424*, 2017. https://arxiv.org/abs/1708.08424
27. Boudhayan Gupta. A replay-attack resistant message authentication scheme using time-based keying hash functions and unique message identifiers. *arXiv:1602.02148*, 2016. https://arxiv.org/abs/1602.02148
28. Shashank Shreedhar Bhatt, Tanmay Rajore, Khushboo Aggarwal, et al. Enterprise AI must enforce participant-aware access control. *arXiv:2509.14608*, 2025. https://arxiv.org/abs/2509.14608
29. Manideep Reddy Chinthareddy. Enterprise Identity Integration for AI-Assisted Developer Services: Architecture, Implementation, and Case Study. *arXiv:2601.02698*, 2026. https://arxiv.org/abs/2601.02698
30. Rama Akkiraju et al. FACTS About Building Retrieval Augmented Generation-based Chatbots. *arXiv:2407.07858*, 2024. https://arxiv.org/abs/2407.07858
31. Xinyi Hou, Jiahao Han, Yanjie Zhao, and Haoyu Wang. Unveiling the Landscape of LLM Deployment in the Wild: An Empirical Study. *arXiv:2505.02502*, 2025. https://arxiv.org/abs/2505.02502
32. Homayoun Maleki, Nekane Sainz, and Jon Legarda. Human Challenge Oracle: Designing AI-Resistant, Identity-Bound, Time-Limited Tasks for Sybil-Resistant Consensus. *arXiv:2601.03923*, 2026. https://arxiv.org/abs/2601.03923
33. H. Krawczyk and P. Eronen. RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). IETF, 2010. https://datatracker.ietf.org/doc/html/rfc5869
34. M. Jones, J. Bradley, and N. Sakimura. RFC 7519: JSON Web Token (JWT). IETF, 2015. https://datatracker.ietf.org/doc/html/rfc7519
35. A. Biryukov, D. Dinu, and D. Khovratovich. RFC 9106: The Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. IETF, 2021. https://datatracker.ietf.org/doc/html/rfc9106
36. OWASP. OWASP Password Storage Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html