xChk: Bring Your Own Identity — Heterogeneous Assurance with Verifier-Determined Sufficiency

Sean MacGuire

Independent Researcher

sean@maclawran.ca

July 2026

Abstract

We present xChk, a reference identity provider for Bring Your Own Identity (BYOI): users enroll via heterogeneous proofs (government KYC, corporate SSO, WebAuthn/FIDO2, professional networks, live verification, longitudinal activity, behavioral signals) and disclose them as portfolio claims in standard OAuth 2.0 / OpenID Connect (OIDC) tokens, while each relying party applies its own sufficiency policy—the IdP transports claims and may evaluate an RP-supplied evidence policy for consent, but does not adjudicate access. Enrollment depth varies by modality (some paths are user-initiated; org KYB and officer binding are operator-assisted).

xChk also supports human-in-the-loop attestation for high-risk actions: humans can initiate attestations directly (browser UI / `POST /api/attestations`), and AI agents acting under those principals can trigger the same gateway via scope-gated authorize/attest—hash-chained human approvals on a shared verification graph (humans via OIDC; agents via API keys). A production deployment at `https://in.xchk.io` ships both initiation paths with bilateral RP evaluation at consent; one documented relying party (`https://crabbyed.com`, Appendix B) exercises Login with xChk.

Keywords Bring Your Own Identity · BYOI · Identity Assurance · OAuth/OIDC · OpenID Connect · Agent Attestation · Identity Provider · Delegation · Portfolio Claims · Human-in-the-Loop

1. Introduction

Identity providers (IdPs) such as Okta, Auth0, Microsoft Entra ID, and Google Identity Platform authenticate users and issue tokens enabling access to third-party applications via OAuth 2.0 [1] and OpenID Connect (OIDC) [2]. These systems share a fundamental limitation: they authenticate via a single modality (password, social login, or SAML federation) and issue a token asserting only that the user authenticated—not what level of identity assurance was established. A user authenticated via corporate SSO is indistinguishable from a user who has provided a government-issued ID and completed live-face verification, despite the latter representing a far higher degree of assurance.

Specialized identity verification platforms (Persona, Jumio, Onfido, Didit) perform KYC checks including government ID verification and liveness detection, but do not function as general-purpose OAuth/OIDC identity providers. Decentralized identity systems (W3C Verifiable Credentials [3], Microsoft Entra Verified ID, cheqd, Polygon ID) issue verifiable credentials but require relying parties to implement VC verification infrastructure and do not integrate with the OAuth/OIDC flow used by thousands of existing applications. Machine identity systems (SPIFFE/SPIRE [4], cloud IAM roles) operate in a separate domain from human identity federation and cannot convey the human operator behind a deployed agent.

Meanwhile, AI agents are increasingly performing consequential actions—financial transactions, file system mutations, external API calls—on behalf of human principals [5, 6]. These agents operate with increasing autonomy, and existing governance frameworks [7] focus on behavioral compliance but assume the identity and authorization layers are solved. AgentBound [8], a complementary runtime governance framework, evaluates whether an authorized action should execute under current behavioral context, but does not address how the agent's identity or its human principal's identity assurance is established.

The gap. To our knowledge, no production OAuth/OIDC identity provider combines BYOI multi-modality enrollment, portable portfolio claims with verifier-determined sufficiency, unified human/non-human governance on one verification graph, human-rooted delegation with cascading revocation, and human-in-the-loop attestation with optional blockchain anchoring. We detail this gap and xChk's shipped coverage in §2.8.

Contributions.

1. **Portfolio claims in OIDC with bilateral sufficiency**—heterogeneous enrollment modalities encoded as typed portfolio artifacts in standard OAuth/OIDC tokens; the IdP transports claims and may evaluate an RP-supplied evidence policy for consent, while the RP adjudicates access; production adds bilateral evaluation of the RP's institutional portfolio before subject portfolio release (§3.3).
2. **Human-rooted attestation on one verification graph**—humans and agents share org enrollment, portfolio assembly, and attestation records; humans federate via Login with xChk (surface A) and may initiate attestations directly; agents use API keys with authorize/attest, scope-monotonic delegation, cascading revocation, and hash-chained human approvals (surfaces B/C; §3.2, §4, §5.1).
3. **Production reference implementation** at `https://in.xchk.io` with one documented RP (`https://crabbyed.com`, Appendix B), ten portfolio artifact types, Hermes guardrails E2E integration, a public portfolio HMAC verifier (Appendix C), and a micro-benchmark of public endpoints (§5.6).

2. Related Work

2.1. Identity Federation and OAuth/OIDC Providers

Traditional IdPs (Okta, Auth0, Microsoft Entra ID, Google Identity Platform) authenticate users through a single modality (password, social login, or SAML federation) and assert only that authentication occurred. None encode heterogeneous identity assurance levels as standard OIDC claims. Step-up authentication and risk-based authentication [9] modulate the authentication challenge based on risk signals but do not produce a token carrying the user's full identity assurance profile for independent RP evaluation.

2.2. KYC and Identity Verification Platforms

Persona, Jumio, Onfido, and Didit perform government ID checks and liveness detection but issue verification results (boolean flags, reports, or W3C VCs)—not OAuth/OIDC tokens. A relying party using these platforms must implement separate integration, verification, and storage for each platform's output. xChk integrates these as enrollment modalities within a standard OAuth/OIDC flow.

2.3. Decentralized Identity (W3C VCs, DIDs)

Verifiable Credentials [3] and Decentralized Identifiers (DIDs) [10] enable cryptographically verifiable claims but require relying parties to implement VC verification infrastructure (DID resolution, credential validation, schema parsing) not present in standard OAuth/OIDC. Microsoft Entra Verified ID, cheqd, and Polygon ID issue VCs for specific claims but typically support one credential type per issuer rather than aggregating multiple heterogeneous identity proofs into a single token.

2.4. Machine Identity

SPIFFE/SPIRE [4] issues X.509-SVIDs for workload identity. Cloud IAM roles (AWS IAM, GCP IAM, Azure Managed Identities) provide identity documents for services. HashiCorp Vault issues short-lived tokens. These systems cannot convey the human operator who deployed a workload or the delegation chain authorizing an agent's actions, and they operate outside the OAuth/OIDC protocol family.

2.5. Agent Identity and Authorization

The OpenID Foundation has proposed OpenID Connect for Agents (OIDC-A) 1.0 [11], an extension to OIDC for representing, authenticating, and authorizing LLM-based agents. OIDC-A defines agent identity claims, delegation chain semantics, and attestation verification within the OAuth 2.0 ecosystem. xChk differs from OIDC-A in that xChk's identity provider is BYOI-native—it accepts enrollment via multiple independent human identity modalities and produces tokens carrying human assurance portfolios, while agent governance is practiced through a parallel authorize/attest API that can converge toward OIDC-A-compatible delegation claims. The two approaches are complementary: a relying party could accept xChk human portfolio tokens and OIDC-A agent claims in a single policy engine.

Forter's WO2026039510A1 [12] describes agent authentication protocols for payment/commerce domains using proprietary certificates—domain-specific, not standard OAuth/OIDC. Onesource's US20250373432A1 [13] describes compliance-token inheritance for healthcare with custom tokens—domain-specific, not general-purpose identity provider functionality.

Microsoft's Agent Governance Toolkit (AGT) [14] provides in-process policy middleware with `require_approval` hooks and local audit trails. AGT does not issue OAuth/OIDC portfolio tokens, perform multi-modality human enrollment, or hash-chained passkey attestation off the agent runtime; xChk composes with AGT via an adapter that routes approval obligations to the attestation gateway.

Context Lineage Assurance for Non-Human Identities [15] addresses cryptographic lineage for A2A interactions but focuses on cryptographic proof chains rather than multi-source identity enrollment. AgentRiskBOM [16] defines a security BOM for scoping agent risk (autonomy, tool permissions, approval gates) but is a metadata artifact, not an identity protocol. Agent registry surveys [17] catalog MCP, A2A, Entra Agent ID, and NANDA/AgentFacts approaches—none provide BYOI with heterogeneous modality aggregation.

Separately, U.S. AI regulation is fragmenting along state lines [18]: California, Colorado, and dozens of other jurisdictions now impose distinct deployer and developer obligations (training-data transparency, automated decision-making, "AI acted autonomously" liability limits, content provenance, and sector rules). Existing agent registries and HITL products rarely treat *registration jurisdiction* as a first-class attribute of a non-human identity that humans can read at approval time and that auditors can recover from the attestation record.

2.6. Behavioral Governance

AgentBound [8] provides a runtime governance framework that evaluates each proposed agent action using three independent authorities: delegated authorization, owner-signed behavioral constitutions, and site action contracts. Their judgments are conservatively composed through a formal decision lattice (Deny $<$ Review $<$ Permit). AgentBound generates cryptographically verifiable governance receipts binding every action to the policy artifacts governing the decision. xChk is complementary to AgentBound: AgentBound assumes identity is established and governs *behavioral* correctness; xChk establishes identity assurance and enables *identity-layer* verification. A combined system would use xChk to establish the identity of the human principal and record human approvals, and AgentBound to govern whether individual actions comply with behavioral policy. AgentBound's review obligations (human approvals) can be satisfied through xChk's attestation gateway; completed approvals appear as portfolio artifacts and `xchk_attestations` claims in refreshed tokens.

2.7. Assurance Levels and Rich Authorization

NIST SP 800-63 [19] and eIDAS [20] define discrete identity assurance / authentication levels (IAL/AAL, LoA) that relying parties consume as ordinal steps. xChk instead exposes a typed **portfolio** of modality-specific artifacts so an RP can require combinations (e.g. KYC + passkey + helpdesk) rather than a single LoA integer. OAuth 2.0 Rich Authorization Requests (RAR) [21] and GNAP [22] address fine-grained authorization payloads; they are complementary prior art for expressing what an RP wants authorized, not substitutes for multi-source identity enrollment into OIDC claims.

2.8. Gap Analysis

To our knowledge, no production system combines all of: (1) BYOI enrollment across independent modalities, (2) heterogeneous assurance levels as standard OAuth/OIDC portfolio claims, (3) verifier-determined sufficiency with the IdP assisting but not adjudicating RP access, (4) unified human + non-human entity governance on one verification graph, (5) delegation chains terminating at a verified human operator with cascading revocation, (6) attestation history carried forward via surface B token embeds and `attestation_completed` portfolio artifacts, and (7) blockchain-anchored human-in-the-loop attestation (human-initiated or agent-triggered). xChk ships (1)–(6) in production and (7) when Ravencoin anchoring is enabled; item (4) covers agent governance on surface C while RP OIDC tokens remain human-only today (§5.1–§5.2).

3. Architectural Model

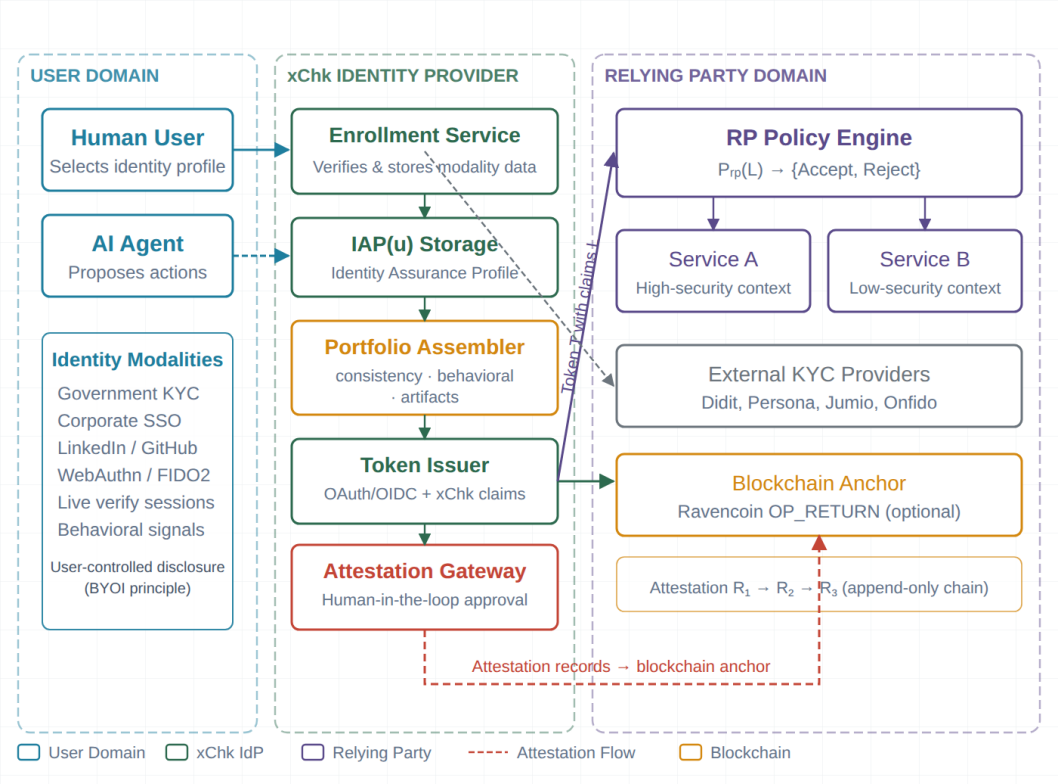

Figure 1: xChk System Architecture—*Three-domain architecture showing the User Domain (modalities, agent), xChk Identity Provider (enrollment, IAP storage, portfolio assembly, token issuer, attestation gateway), and Relying Party Domain (policy engine, services, blockchain anchor).*

This section and §4 give descriptive tuples and production mappings—an architectural model, not a security proof. The strongest structural properties we rely on are scope monotonicity down a delegation chain and cascading revocation of descendants (§4.4).

3.1. Entities and Modalities

Let $\mathcal{U}$ be the set of all users (human principals). Let $\mathcal{M} = \{m_1, \ldots, m_k\}$ be a finite set of identity verification modalities, where each modality $m \in \mathcal{M}$ is an enrollment channel that, upon successful completion, yields an assurance indicator $a(m)$ from a modality-specific assurance space A_m. Production implements these as typed portfolio artifacts (§5); examples include:

- $A_{kyc} = \{\text{none}, \text{basic}, \text{govt\_id}\}$ — government-issued identity verification (Didit)

- $A_{sso} = \{\text{none}, \text{tenant\_verified}\}$ — corporate SSO (Entra ID, Google Workspace)
- $A_{webAuthn} = \{\text{none}, \text{single\_device}, \text{multi\_device}\}$ — hardware-bound credentials (device biometric unlock at use time)
- $A_{professional} = \{\text{none}, \text{verified}\}$ — professional network verification (LinkedIn)
- $A_{live} = \{\text{none}, \text{pass}, \text{fail}\}$ — live helpdesk or interview verification
- $A_{longitudinal} = \{\text{none}, \text{present}\}$ — platform activity over time (GitHub, LinkedIn)
- $A_{behavioral} = \{\text{usual}, \text{new}, \text{unusual}\}$ — device, network, location, time-of-day consistency
- $A_{attestation} = \{\text{none}, \text{completed}\}$ — prior human-in-the-loop approvals

For each user $u \in \mathcal{U}$, the identity provider maintains an **identity assurance profile**:

$$\mathbf{IAP}(u) = \{(m, a(m), t(m), p(m)) \mid m \in \mathcal{M}_{enrolled}(u)\} \tag{1}$$

where $t(m)$ is the timestamp of modality m's most recent verification and $p(m)$ is an optional provider identifier (including `issuer` and `issuerTier` for institutional provenance: `unverified`, `domain_verified`, `entity_verified`, `officer_bound`). The set $\mathcal{M}_{enrolled}(u) \subseteq \mathcal{M}$ is the subset of modalities the user has chosen to complete—the BYOI principle.

3.2. Identity Token Definition

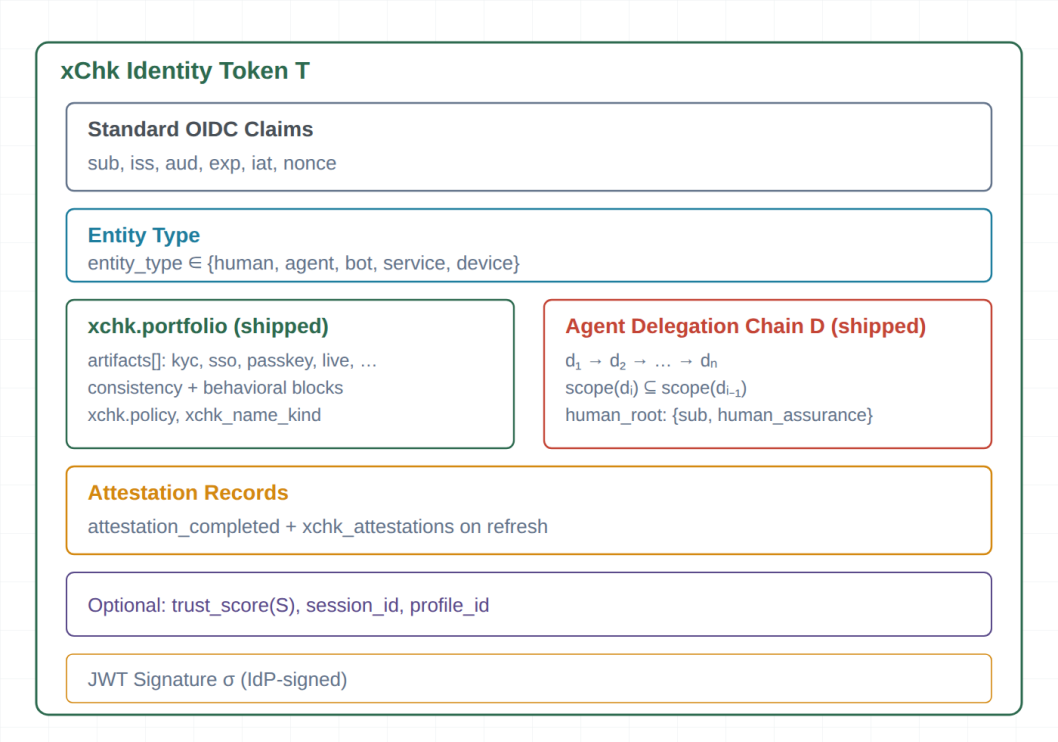

Figure 2: xChk Identity Token Structure—*Standard OIDC claims, entity type, human assurance claims, agent delegation chain, attestation history, and advisory score. Production tokens include nested* `xchk.portfolio` *claims;* `xchk_entity_type` *is top-level; the access-token path adds* `xchk_trust_score`*,* `xchk_portfolio_summary`*,* `xchk_delegation_chains`*, and* `xchk_attestations`*.*

An xChk identity token $\mathbf{T}$ for user u at time τ is a tuple:

$$\mathbf{T} = (sub, iss, aud, exp, iat, entity\_type, \mathbf{L}, \sigma) \tag{2}$$

where:

- $entity\_type \in \{\text{human}, \text{agent}, \text{bot}, \text{service}, \text{device}\}$ — entity-type taxonomy; production OIDC ID tokens issue `xchk_entity_type: human`; non-human entities are governed via the agent API (§5.1)
- $\mathbf{L} = \{(m, a(m)) \mid m \in \mathcal{M}_{enrolled}(u) \wedge a(m) \neq \text{none}\}$ is the set of identity assurance claims
- σ is the IdP's JWT signature over the token body

Production ID tokens (surface A) additionally carry a portfolio HMAC σ_p over the consent-time package $\{portfolio, policy, rpInstitutional, subjectRpPolicy\}$ (wire encoding below); RS256 validation of σ alone does not establish portfolio integrity.

For **agent** entity types, $\mathbf{L}$ additionally contains a **delegation chain** $\mathbf{D} = [d_1, \ldots, d_n]$ where each delegation link $d_i = (entity\_id, scope, parent\_id, attestation\_id)$ satisfies:

- $d_1.parent\_id$ is a human user $u \in \mathcal{U}$ (the root principal)
- For $i > 1$, $d_i.parent\_id = d_{i-1}.entity\_id$
- $scope(d_i) \subseteq scope(d_{i-1})$ (scope is monotonically non-expanding down the chain)
- Each d_i references an attestation record (§4.1) proving the parent authorized the delegation

In production, surface A OIDC ID tokens for human principals do not embed $\mathbf{D}$. When integrators supply `delegationChain` metadata at attestation create, completed attestations may surface $\mathbf{D}$ on surface B via `xchk_delegation_chains` assembled from portfolio artifacts (§4.4); the platform does not auto-walk the `parentAgentId` agent tree to mint chains.

The token also optionally includes:

- `xchk_trust_score`(S): advisory composite S on xChk access tokens
- `xchk_attestations`: ordered attestation records via `POST /oauth/refresh-token` and as `attestation_completed` portfolio artifacts
- `xchk_session`: session identifier for multi-entity workflows via `POST /api/session-tokens` and refresh-token embed
- `xchk_portfolio_summary`: time-bucketed artifact counts split by human and agent origin on access tokens and refresh
- Per-app artifact picker: user selects which artifact types each RP receives

Production OIDC encoding (v1). Production ID tokens (scope `xchk_portfolio`) expose:

- `xchk.portfolio`—`artifacts[]` with typed records, plus `consistency` (volume, longevity, pass rates) and `behavioral` (usual/new/unusual per dimension)
- `xchk.policy`—result of evaluating the RP's `xchk_evidence_policy` (or client-registered policy): `{ satisfied, missing, ...}`

- `xchk.signature` / `xchk_signature`—HMAC-SHA256 over the canonical consent-time package $\{portfolio, policy, rpInstitutional, subjectRpPolicy\}$ (signing key id `xchk-portfolio-v1`; RS256 JWT validation alone is insufficient)
- `xchk.rp_institutional` / `xchk.subject_rp_policy`—RP institutional portfolio snapshot and subject-policy evaluation at consent (bilateral evaluation, §3.3)
- `xchk_name_kind`, `xchk_legal_name_on_file`—name provenance (scope `profile`)
- `xchk_entity_type`—`human` on OIDC ID tokens when portfolio scope is present
- `sub`—pairwise per OAuth client (cross-RP correlation resistant)

Wire encoding (v1). Dotted names above (`xchk.portfolio`, etc.) are **documentation notation** for fields inside a nested JWT claim `xchk`, not literal dotted JWT keys. Surface A ID tokens and userinfo responses nest portfolio fields under `xchk` and duplicate `xchk_signature`, `xchk_signing_key_id`, and `xchk_entity_type` at the top level. Integrators read `idToken.xchk.portfolio` for the paper's `xchk.portfolio`. Surface B (`/api/login`, `POST /oauth/refresh-token`) uses flat top-level underscore claims (`xchk_trust_score`, `xchk_portfolio_summary`, `xchk_attestations`, `xchk_delegation_chains`) on the first-party `xchkAccessToken`. The HMAC canonical payload uses camelCase keys (`rpInstitutional`, `subjectRpPolicy`), distinct from JWT snake_case field names.

Doc notation	JWT access
`xchk.portfolio`	`payload.xchk.portfolio`
`xchk.policy`	`payload.xchk.policy`
`xchk.signature`	`payload.xchk.signature` or top-level `payload.xchk_signature`
`xchk.rp_institutional`	`payload.xchk.rp_institutional`
`xchk.subject_rp_policy`	`payload.xchk.subject_rp_policy`
`xchk_entity_type`	top-level `payload.xchk_entity_type` only

Optional scope `xchk_portfolio_proofs` adds expiring, audience-bound proof URLs and `sha256` / `rvnTxId` on attestation artifacts for verification independent of the IdP read path.

Access-token path (surface B). `/api/login` and `POST /oauth/refresh-token` mint an `xchkAccessToken`—a first-party JWT distinct from the OIDC access token returned by `/oauth/token`—carrying `xchk_trust_score`, `xchk_portfolio_summary`, `xchk_delegation_chains`, `xchk_attestations`, and optional `xchk_session`. Integrators call refresh after attestation approval to pick up new records without a full OIDC re-login.

3.3. BYOI Decision Problem

Let $\mathbf{T}$ be an xChk identity token presented to a relying party (RP). The RP maintains a policy function:

$$P_{rp} : 2^{\mathcal{M} \times A} \to \{\text{Accept}, \text{Reject}\} \tag{3}$$

where the input is a subset of $(m, a(m))$ pairs from $\mathbf{T}$ (in production: the artifact types the user consented to disclose). Critically:

- The IdP does **not adjudicate access**—it verifies enrollment, assembles claims, and may evaluate an RP-*supplied* policy for display (`xchk.policy` on the consent screen)
- RP does not see the full $\mathbf{IAP}(u)$—only the subset $\mathbf{L} \subseteq \mathbf{IAP}(u)$ the user chooses to present (per-app artifact picker)
- The user may withhold artifact types; `xchk.policy.satisfied` reflects the disclosed subset

This defines a **three-way separation of concerns**:

1. **User** controls which modalities to enroll and which artifacts to disclose per RP
2. **IdP** verifies modality enrollment truthfully and transports claims (plus optional policy evaluation assist)
3. **RP** evaluates sufficiency and grants or denies access

This separation emphasizes **portable multi-modality assurance portfolios** over conventional federation, where tokens typically assert only that authentication occurred (often via a single modality) rather than a user-selected set of typed assurance artifacts for independent RP evaluation.

Bilateral evaluation (production). When scope `xchk_portfolio` is requested, evaluation is concurrent in both directions: the RP's evidence policy is evaluated against the subject's disclosed portfolio (`xchk.policy`), and a subject-defined policy is evaluated against the RP's **institutional verification portfolio** assembled from the OAuth client's owning organization (DNS domain control, KYB, officer binding, participation history). The platform default subject policy requires `domain_control`; unsatisfied evaluation returns `403` (`rp_institutional_policy_failed`) and disables portfolio release on the consent screen. The IdP still does not grant or deny RP access—it gates what the subject chooses to share. Token claims `xchk.rp_institutional` and `xchk.subject_rp_policy` are included in the portfolio HMAC payload.

3.4. Trust Score

An optional trust score S may be computed as a weighted function of enrolled modalities:

$$S = \phi(a(m_1), \ldots, a(m_k), t(m_1), \ldots, t(m_k), N_a) \tag{4}$$

where N_a is the number of attested actions and ϕ is a monotonic non-decreasing function. Production ships `xchk_trust_score` on surface B access tokens only; RPs on surface A more commonly evaluate raw portfolio artifacts and `xchk.policy`. No production RP documented here gates access on S alone.

4. Attestation Protocol

4.1. Attestation Record

An attestation record R is a tuple:

$$R = (att\_id, action, entity\_id, approver\_id, outcome, \tau, sig, chain\_hash) \tag{5}$$

where:

- att_id — globally unique identifier
- $action = (type, target, args)$ — the action being attested
- $entity\_id$ — the entity that proposed the action (human principal or agent)
- $approver\_id$ — the human approver's user identifier
- $outcome \in \{\text{approved}, \text{declined}, \text{escalated}\}$
- τ — UTC timestamp
- sig — HMAC signature over the canonical record hash

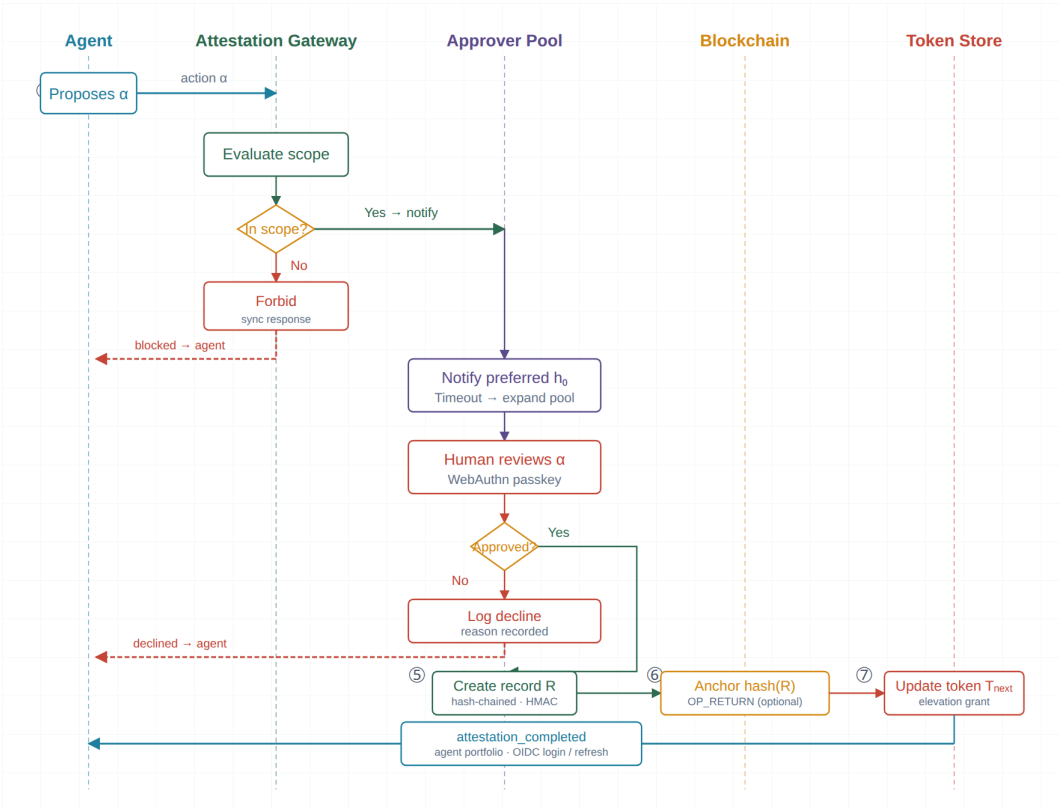

Figure 3: Attestation Protocol Sequence—*Production attestation sequence (agent-triggered variant shown; humans may initiate via browser UI / `POST /api/attestations`): agent proposes action α; gateway evaluates scope via `POST /api/agents/authorize`; progressive notification of approver pool; human approves/declines via WebAuthn passkey (required); attestation record R created (hash-chained, HMAC-signed); optional Ravencoin OP_RETURN anchor; short-lived elevation grant for retry; R appears as `attestation_completed` portfolio artifact on the approver's next OIDC login and via `xchk_attestations` on token refresh.*

- $chain\_hash$ — SHA-256 hash of the previous attestation record in the chain (or null for the first)

Attestation records form an append-only chain. The hash of the latest attestation record may be written to a blockchain for decentralized timestamp anchoring when Ravencoin RPC is configured.

4.2. Attestation Lifecycle (production)

The attestation gateway accepts human-initiated requests (`POST /api/attestations` / browser UI) and agent-triggered requests. When an agent entity a proposes an action α that triggers a governance gate, the gateway executes:

1. **Capture**: The agent runtime emits action $\alpha = (type, target, args, context)$
2. **Authorize**: `POST /api/agents/authorize` returns `allow`, `attest`, or `forbid`; `attest` mints a short-lived `authorizeSessionId` bound to the action hash
3. **Approver resolution**: Policy rules select approver(s) $H \subseteq \mathcal{U}$ based on action type and severity; creation is rejected when `requiredApprovals` exceeds the available approver pool
4. **Progressive notification**: Preferred approver h_0 is notified (Slack, Telegram, SMS, WhatsApp); on timeout t, the approver pool expands
5. **Decision**: Approver $h \in H$ reviews α, verifies via WebAuthn passkey, and produces $outcome$

6. **Record creation**: Attestation record R is created with hash chaining and HMAC signing; optional `delegationChain` metadata may be supplied by the integrator
7. **Blockchain anchoring** (optional): $hash(R)$ written to Ravencoin OP_RETURN when `RVN_RPC_PASS` is configured
8. **Elevation grant**: On approval, an HMAC-signed grant (≈15 min TTL, action-bound) allows the agent runtime to retry execution
9. **Portfolio update**: R surfaces as an `attestation_completed` artifact on the human principal's next OIDC authorization and via `xchk_attestations` on token refresh (§3.2)

Enterprise systems can consume completion webhooks—no production ERP/procurement adapter is shipped.

4.3. Runtime Scope Enforcement (production)

Agents authenticate with per-agent API keys, not OIDC tokens. Scope compliance is evaluated at authorize time: `POST /api/agents/authorize` classifies the action area, matches against the agent's registered grants, and returns `forbid` when out of scope. Out-of-scope actions may trigger attestation (escalation) rather than silent execution.

For delegation chain $\mathbf{D}$ embedded in token $\mathbf{T}$ or recorded on attestation create, $scope\_compliant(\alpha, \mathbf{D}) \iff \exists d_i \in \mathbf{D} : (op{:}r) \in scope(d_i)$. Expanded scope must satisfy $scope'(d_i) \subseteq scope(d_{i-1})$. Production evaluates scope at authorize time and embeds `xchk_delegation_chains` on xChk access tokens (surface B) from portfolio artifacts.

4.4. Delegation Chains and Cascading Revocation

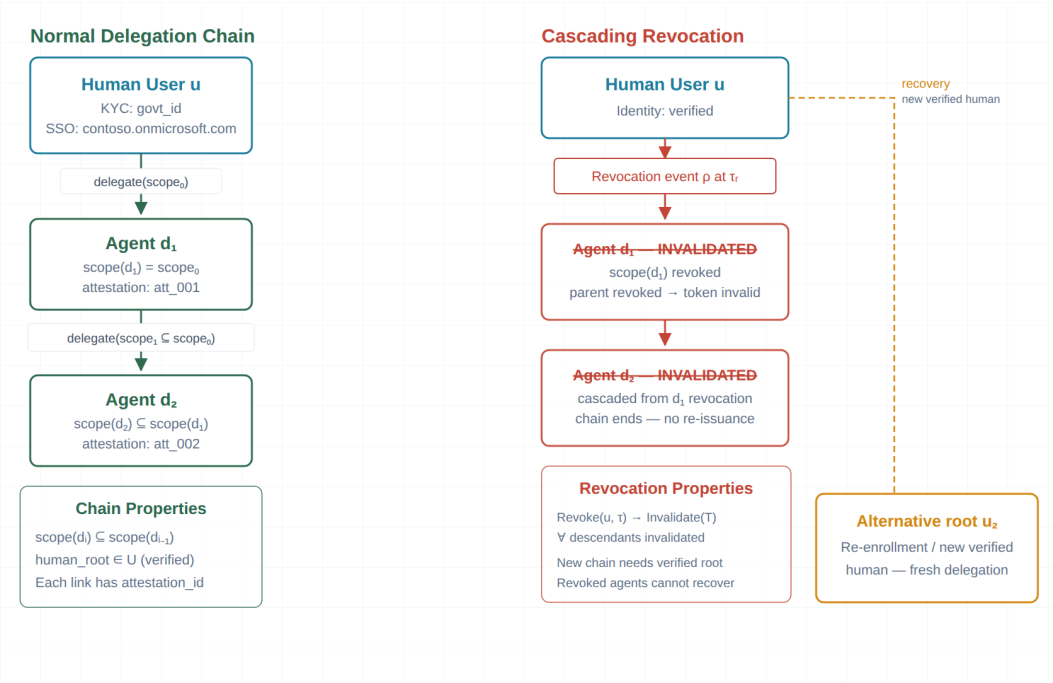

Figure 4: Delegation Chains and Cascading Revocation—*Left: Normal delegation chain from human root u through agents d_1, d_2 with monotonically non-expanding scope. Right: Revocation event ρ at the human root invalidates descendants; the chain ends at revoked agents. Recovery requires a separately verified alternative root u_2 (re-enrollment or new human)—not re-issuance from invalidated agents.*

xChk access tokens (surface B) carry $\mathbf{D}$ via `xchk_delegation_chains` assembled from `attestation_completed` portfolio artifacts. The chain payload is **integrator-supplied** metadata on attestation create

(`delegationChain` in the attestation request); the platform does not auto-walk the `parentAgentId` tree to mint token chains. By contrast, `parentAgentId` drives **cascading revocation**—a BFS tree walk on agent delete revokes descendant agents and their API keys (§5.2).

Define a revocation event ρ for entity e at time τ_ρ. The revocation propagates as:

$$\text{Revoke}(e, \tau_\rho) \Rightarrow \{\text{Invalidate}(\mathbf{T}) \mid \mathbf{T}.\mathit{entity\_id} \in \text{Descendants}(e)\} \tag{6}$$

where $\text{Descendants}(e)$ is the transitive closure of entities whose delegation chain contains e as an ancestor. Production revokes descendant agents and their API keys when a parent agent is deleted.

4.5. Multi-Entity Session Token

For workflows involving multiple agents operating under related delegation chains, a session token $\mathbf{T}_s$ is defined:

$$\mathbf{T}_s = (\mathit{session\_id}, \{\mathbf{D}_e \mid e \in \mathbf{E}_{\mathit{session}}\}, \mathit{scope}_s, S_s) \tag{7}$$

where $\mathit{scope}_s = \bigcap_e \mathit{scope}(\mathbf{D}_e)$ and $S_s = \min(\{S_e\})$. `POST /api/session-tokens` creates a session from up to 20 entities, computes intersection scope and composite trust score, and returns a `sessionId` embeddable in the `xchk_session` claim via `POST /oauth/refresh-token`.

5. Implementation

A reference implementation of the xChk identity provider and attestation gateway is deployed at `https://in.xchk.io`. One documented relying party—`https://crabbyed.com` (Crabby Editor, Appendix B)—exercises surface A Login with xChk in production; broader third-party adoption is not yet demonstrated. The deployment covers OAuth 2.0 / OpenID Connect authorization code flow with PKCE, RS256-signed ID tokens and access tokens, a JWKS endpoint, `GET /oauth/userinfo`, and an OpenID Connect discovery document.

5.1. Production Delivery Surfaces

Throughout §5, **shipped** means the construct is available in production on at least one of the surfaces below. The tuples in §3–§4 describe the intended token layout; the table in §5.2 states where each construct is emitted today.

Surface	Who uses it	What ships
A — RP OIDC ID token	Third-party apps via Login with xChk (`/oauth/authorize` → `/oauth/token`)	`xchk.portfolio`, `xchk.policy`, portfolio HMAC (`xchk.signature`), bilateral RP institutional claims, profile claims, `xchk_entity_type: human`, optional `xchk_portfolio_proofs`
B — xChk access token	First-party app via `/api/login`; refresh via `POST /oauth/refresh-token`	`xchk_trust_score`, `xchk_portfolio_summary`, `xchk_attestations`, `xchk_delegation_chains`, optional `xchk_session`
C — Agent API	Registered agents (API keys)	`POST /api/agents/authorize`, attestation create/respond, External Approval Policies, cascading revoke on agent delete, elevation grant verify

Table 1: Production delivery surfaces.

Integrator surfaces. Use **surface A alone** when a third-party RP only needs Login with xChk portfolio claims and bilateral consent (standard OIDC + nested `xchk`). **Add surface B** when the integrator must read post-approval `xchk_attestations`, delegation chains, trust score, or session embeds without a full OIDC re-login (call refresh after approve). Use **surface C** for agent runtime governance (API keys, authorize/attest, elevation grants)—not as an OIDC bearer path for RPs.

5.2. Implementation Status

Table 2: Implementation status of all constructs.

Construct	§	Surf.	Status
OAuth/OIDC + PKCE, RS256, JWKS, pairwise `sub`	3.2	A	Shipped
`xchk.portfolio` + `consistency` / `behavioral`	3.2	A	Shipped
`xchk.policy` + `xchk_evidence_policy`	3.3	A	Shipped
Per-app artifact picker / disclosure	3.2	A	Shipped
Fresh verification (WebAuthn / PIN) at consent	3.3	A	Shipped
`xchk_entity_type` on OIDC ID token	3.2	A	Shipped — `human`
`issuer` + `issuerTier` provenance	3.1	A	Shipped
Org trust ladder (DNS → KYB → officer binding)	3.1	A	Shipped
`xchk_portfolio_proofs`	3.2	A	Shipped
`xchk_trust_score`, `xchk_portfolio_summary`	3.4	B	Shipped
`xchk_attestations` embed	4.2	B	Shipped — refresh after approve
`xchk_delegation_chains`	4.4	B	Shipped
Multi-entity `xchk_session`	4.5	B	Shipped — session-tokens + refresh
`attestation_completed` portfolio artifact	4.2	A	Shipped — on next OIDC login
`POST /api/agents/authorize`	4.3	C	Shipped
Attestation hash chain + HMAC + elevation grant	4.1–4.2	C	Shipped
Insufficient-approver validation	4.2	C	Shipped
External Approval Policies (EAP)	4.2	C	Shipped
Cascading agent revocation	4.4	C	Shipped
Hermes guardrails E2E	4.2	C	Shipped — integration; ∼50 patterns when Hermes runtime present (not vendored in xchk)
Ravencoin OP_RETURN anchor	4.2	C	Partial — requires Ravencoin node
Portfolio HMAC on ID token	3.2	A	Shipped — consent snapshot
Bilateral RP institutional portfolio evaluation	3.3	A	Shipped — hard gate on release
Production OAuth enforce (client tier + redirect URI)	3.1	A	Shipped — DNS-verified redirect
`GET /oauth/userinfo`	3.2	A	Shipped — same nested `xchk` shape; live-evaluated (ID token is consent snapshot)
`legally_binding` attestation completion gate	4.2	C	Shipped — requires `officer_bound`
Fleet pack batch import (`POST /api/agents/import`)	4.3	C	Shipped
Agent plate id + registration venue (`plateState`)	5.4	C	Shipped — cascade: explicit → owner KYC → `deployRegion` → FL
Preregister invite domain gate	3.1	C/API	Shipped — requires `entity_verified` org
Full non-human `entity_type` on RP OIDC token	3.2	A	Future — agents use surface C

5.3. Identity Provider (OIDC)

- Standard OAuth 2.0 authorization code flow with PKCE (S256)
- RS256-signed ID tokens with pairwise subjects; `GET /oauth/userinfo` returns the same nested `xchk` claims when `xchk_portfolio` scope is granted, but **live-evaluates** portfolio and policy (ID token carries the consent-time snapshot)
- Extended claims: `xchk.portfolio`, `xchk.policy`, `xchk.signature`, `xchk.rp_institutional`, `xchk.subject_rp_policy`, `xchk_name_kind`, `xchk_legal_name_on_file`, `xchk_entity_type`
- Evidence policy: RPs transmit base64url JSON via `xchk_evidence_policy`; IdP evaluates against the user's portfolio and returns `xchk.policy` before consent—the RP still decides access
- Bilateral consent: subject-defined policy evaluated against the RP's institutional portfolio (domain control, KYB, officer binding); default requires DNS-verified `domain_control`; portfolio release blocked when unsatisfied
- Portfolio integrity: HMAC-SHA256 over the canonical $\{portfolio, policy, rpInstitutional, subjectRpPolicy\}$ snapshot (`xchk-portfolio-v1` key id); verification procedure in Appendix A
- Production OAuth gates: OAuth client verification tier synced from org ladder; redirect URIs must match DNS-verified domains in production (localhost exempt in development)

- Fresh-verification enforcement: WebAuthn passkey or PIN re-verification during consent when the policy requires it
- Per-app disclosure: users choose which artifact types to share with each RP

Verification portfolio. Assembled at token issuance from ten artifact types:

Artifact Type	Source	Assurance Signal
`kyc_verified`	Didit (govt ID + liveness)	authLevel, provider, timestamp
`corporate_sso_linked`	Entra ID, Google Workspace	Tenant ID, domain, tenant age
`platform_identity_verified`	LinkedIn Identity Verification API	Verified name, ID category
`workplace_verified`	LinkedIn Workplace Verification	Organization name, date
`longitudinal_activity`	GitHub OAuth, LinkedIn	Account age, contributions, followers
`passkey_registered`	WebAuthn/FIDO2 authenticator	Credential ID, registration date
`helpdesk_pass`	xChk helpdesk sessions	Session ID, proof URL
`interview_pass`	xChk identity interview	Session ID, proof URL
`attestation_completed`	xChk attestation records	Attestation ID, optional `rvnTxId`, `sha256`
`behavioral_signals`	Device, network, location, time	Usual/new/unusual per dimension

Table 3: Verification portfolio artifact types.

Every artifact names its `issuer` and `issuerTier` (`unverified`, `domain_verified`, `entity_verified`, `officer_bound`). Organization enrollment adds DNS domain verification, KYB (Didit), and officer binding for institutional attestations.

5.4. Attestation Gateway

- Hermes guardrails E2E: xChk ships the integration (`examples/hermes-attestation-guardrails/`); dangerous-command classification delegates to Hermes `detect_dangerous_command` ($\sim$50 patterns when that runtime is installed—pattern catalog not vendored in xchk repos; demo fallback uses a small regex set)
- Hash-chained attestation records with HMAC signing
- Optional Ravencoin OP_RETURN anchoring (v1/v2, 80-byte payload, `xCk` magic prefix)
- Progressive notification with approver pool expansion on timeout
- Notification channels: Slack, Telegram, SMS, WhatsApp
- Multi-approver escalation chains with policy-based routing; rejects create when approver pool is smaller than `requiredApprovals`
- Server-computed `bindingClass` (organizational / `legally_binding`) from purpose templates and policy names; terminal `legally_binding` completion requires org tier `officer_bound`
- External Approval Policies (`POST /api/external-approval-policies`) for cross-domain attestation authorization
- Fleet pack batch import (`POST /api/agents/import`) with role/intent registry for multi-agent deployment
- WebAuthn passkey approver verification on attest-reply (enrollment PIN is not accepted for attestation unlock)
- `POST /api/agents/authorize` $\rightarrow$ `allow` | `attest` | `forbid`; signed elevation grant on approve

Jurisdiction of registration (production embodiment). Each registered agent carries a stable plate identifier (`TYP-BODY`, e.g. `OPS-YDDS`) and a U.S. registration venue (`plateState`: currently `FL`, `CA`, or `CO`)—chosen the way contracts pick a forum, not a claim about where model weights sit. Venue resolution order: (1) explicit operator-set `plateState`; (2) human owner location from KYC document state; (3) agent declared deploy / physical location (`deployRegion`, including common cloud-region hints); (4) platform default `FL`. Provenance is stored as `plateStateSource` (`explicit` | `owner_kyc` | `agent_deploy` | `default`). Deploying or acting in another jurisdiction (a "foreign land") does not extinguish home registration: place-of-operation rules may *add* obligations; they do not erase the home venue recorded on the agent and its attestations. Attestation records retain plate and venue so an auditor can map a human approval to the regulatory surface under which the agent was licensed—without reading soft display names or API-key owner email. The production UI presents venue as state license-plate graphics (Florida, California, Colorado) so approvers see *who* the agent is and *where it is registered* in one glance; the cryptographic record stores the jurisdiction code, not the graphic.

Agents use API keys and the authorize/attest API—not OIDC tokens—for runtime governance. Human principals use Login with xChk for RP federation.

5.5. API Authentication and Deployment

API middleware accepts Firebase ID tokens, xChk OAuth access tokens, Zendesk launch tokens, and SSO launch tokens. `/api/login` mints an `xchkAccessToken` for non-anonymous users; `POST /oauth/refresh-token` refreshes it with updated portfolio summary, attestation embeds, delegation chains, and optional session reference. `POST /api/session-tokens` creates multi-entity session records surfaced in the `xchk_session` claim (§4.5).

Deployment: Node.js (Express), MongoDB, PM2 on Ubuntu 24.04, Nginx TLS termination; static frontend at `/var/www/xchk-app`.

5.6. Micro-benchmark (public endpoints)

We measured end-to-end HTTPS latency from a client on the public Internet to production `in.xchk.io` (2026-07-11). Samples are wall-clock `curl` transfer times (TLS + server). Portfolio HMAC verify is a local microbench matching Appendix A (not network-bound).

Operation	n	p50	p95	Notes
`GET /.well-known/openid-configuration`	50	159 ms	200 ms	Public discovery
`GET /oauth/jwks`	50	161 ms	187 ms	Public JWKS
`POST /oauth/token` (invalid client)	50	162 ms	191 ms	Reject path; not a successful code exchange
`POST /api/agents/authorize` (no API key)	50	165 ms	213 ms	Auth-fail path; not a scoped allow/attest decision
Portfolio HMAC-SHA256 verify (local)	5000	0.014 ms	0.031 ms	Synthetic 10-artifact portfolio

Table 4: Public-endpoint and local HMAC micro-benchmark (2026-07-11).

Successful authorize/attest and full OIDC code exchange latencies depend on credentials and human approval and are not reported here. Human-in-the-loop attestation remains dominated by approver response time (§6.4).

6. Discussion

6.1. Security Considerations

Threat model (summary). We consider: (i) an honest-but-curious RP that receives only consented claims and may attempt cross-RP correlation; (ii) a compromised modality issuer or stolen SSO session for a single enrollment channel; (iii) a stolen agent API key used to propose actions; (iv) IdP database compromise exposing stored portfolios and attestation records; (v) leakage of `ATTESTATION_SIGNING_SECRET` used for portfolio HMAC and attestation integrity. Pairwise `sub`, per-app disclosure, and multi-modality RP policies mitigate (i)–(ii). Surface C authorize/attest plus human approval and elevation grants mitigate (iii) for gated actions (stolen keys can still call APIs until revoked). (iv) and (v) are out of scope for RP-side JWT verification alone—self-hosted deployments must

protect the signing secret; hosted deployments coordinate secret distribution out of band. RS256 JWT validation without portfolio HMAC does not detect portfolio substitution inside an otherwise valid ID token.

Token forgery: The IdP signs tokens with RS256 JWT signatures. RPs verify against the published JWKS endpoint. Portfolio packages additionally carry an HMAC (`xchk.signature`) over the disclosed snapshot (Appendix A; verifier in Appendix C). Self-hosted deployments share the attestation signing secret with integrators who verify portfolio integrity independently of ongoing IdP read access; hosted deployments at in.xchk.io coordinate secret distribution out of band.

Replay attacks: OIDC `nonce` and `exp` prevent token replay. The attestation gateway binds `authorizeSessionId` to action hashes.

Modality compromise: If a single modality is compromised (e.g., a leaked corporate SSO session), remaining modalities continue to provide assurance. RPs may require multiple artifact types for sensitive operations.

Session integrity: The attestation chain enforces temporal ordering via `chain_hash`. Ordering anomalies may indicate session takeover or replay.

Blockchain anchoring: When configured, Ravencoin provides a timestamped commitment to the attestation hash. The blockchain is not an execution layer; records remain in the IdP database and are verifiable against the anchor.

6.2. Privacy Considerations

BYOI limits exposure: users choose enrollment and per-RP disclosure; tokens carry assurance indicators, not raw KYC documents. Pairwise `sub` prevents cross-RP correlation. The per-app artifact picker lets users withhold artifact types—`xchk.policy` is evaluated on the disclosed subset only. `xchk_portfolio_proofs` issues expiring, audience-bound proof URLs. Users may invoke **Forget Me** erasure, which invalidates proof links early. Name provenance (`xchk_name_kind`) lets RPs accept pseudonymous-but-accountable identities (`registered_alias` with KYC on file) without receiving the legal name. Session photos and likeness data are acknowledged PII-lite exceptions stored for verification, not exported as raw biometrics in tokens.

6.3. Relationship to Behavioral Governance

As in §2.6, AgentBound [8] governs behavioral correctness while xChk governs identity assurance and human-in-the-loop attestation; AgentBound review obligations can flow through xChk's attestation gateway, and governance receipts plus attestation records together bind behavioral and identity-layer decisions.

6.4. Limitations

- **RP adoption:** RPs must parse custom claims and evaluate policies; standard OIDC libraries do not do this automatically.
- **Delivery surfaces:** Extended claims (`xchk_attestations`, delegation chains, session) require the xChk access-token refresh path (§5.1), not standard RP OIDC ID tokens alone.
- **External validation:** One documented relying party (`https://crabbyed.com`, Appendix B) exercises surface A in production; broader integrator adoption remains future work.
- **Enrollment UX:** Modalities differ in self-service depth—passkey registration, corporate SSO linking, and helpdesk/interview flows are user-initiated; org KYB, officer binding, and some professional-network paths require operator configuration.
- **Commitment-class classifier:** Attestation `bindingClass` uses server heuristics and policy templates; optional org-configured rules or LLM classifiers are not yet deployed platform-wide.
- **Trust score advisory role:** `xchk_trust_score` is advisory; RPs define thresholds over portfolio signals.
- **Blockchain anchoring:** Requires a connected Ravencoin node with funded wallet when enabled.
- **Latency:** Human-in-the-loop attestation introduces variable delay—unsuitable for hard real-time loops.

- **Evaluation:** No formal user study; §5.6 reports public-endpoint and local HMAC micro-benchmarks only. Successful authorize/attest timings remain future work.

7. Conclusion

We presented xChk, a BYOI architecture for OAuth/OIDC that encodes heterogeneous identity assurance as machine-readable portfolio claims and leaves access sufficiency to each relying party. We described an architectural model with a three-way separation—user controls disclosure, IdP transports claims (and optionally evaluates an RP-supplied policy for consent), RP adjudicates access—extended with bilateral evaluation of the RP's institutional portfolio before subject portfolio release. A reference implementation at in.xchk.io ships on three surfaces (§5.1); surface A is documented with one production RP (Appendix B). Agent governance, optional Ravencoin anchoring, and full non-human OIDC entity types follow §5.2 Partial/Future rows.

8. Data and Code Availability

The reference implementation is deployed at `https://in.xchk.io`. Source code is maintained in private repositories and is **not publicly available** at the time of writing. Third parties can interact with the live system without a local checkout: OpenID Connect discovery (`/.well-known/openid-configuration`), the OAuth playground (`https://in.xchk.io/oauth/playground.html`), the Login with xChk integrator guide (`https://in.xchk.io/login-with-xchk.html`), and the public portfolio HMAC verifier (`https://in.xchk.io/verify-portfolio-hmac.cjs`, Appendix C). The paper PDF with embedded architecture diagrams is at `https://in.xchk.io/xchk-paper-with-diagrams.pdf`. No proprietary dataset is required to exercise the OAuth portfolio flow against production; integrators may register OAuth clients on the hosted service.

A. Portfolio HMAC Verification

RPs verifying surface A ID tokens should recompute the portfolio HMAC after RS256 JWT validation:

1. Extract portfolio fields from the nested `xchk` claim: `payload.xchk.portfolio`, `payload.xchk.policy`, `payload.xchk.rp_institutional`, and `payload.xchk.subject_rp_policy` (paper notation `xchk.portfolio`, etc.). Use top-level `payload.xchk_signature` or `payload.xchk.signature` for the HMAC digest.
2. Build canonical UTF-8 JSON with recursively sorted object keys:

```
{
  "policy": { "satisfied": true, "missing": [] },
  "portfolio": { "artifacts": [], "consistency": {}, "behavioral": {} },
  "rpInstitutional": { "artifacts": [] },
  "subjectRpPolicy": { "satisfied": true, "missing": [] }
}
```

3. Compute HMAC-SHA256 using the first 32 hex characters of `SHA256(ATTESTATION_SIGNING_SECRET)` as the key.
4. Compare the hex digest to `xchk.signature` / `xchk_signature`. Mismatch indicates tampering or token/portfolio substitution.

Key id `xchk_signing_key_id: xchk-portfolio-v1` identifies the signing scheme. Full integrator steps: `https://in.xchk.io/login-with-xchk.html`. A runnable verifier matching this procedure is Appendix C.

B. Documented RP Integration (crabbyed.com)

Crabby Editor (`https://crabbyed.com`) is a verification-first web application that uses Login with xChk for continuity after identity verification—not as a generic social-login replacement. It is the only RP integration documented in this paper at production scale; it shares operational ownership with the xChk deployment and should be read as a **reference integration**, not independent ecosystem adoption.

OAuth client. Registered OAuth client `client_id=crabbyed.com` with production redirect URI on the Crabby Editor origin. Login entry point: `https://crabbyed.com/login.html`.

Authorization flow (surface A).

1. Browser initiates OAuth 2.0 authorization code flow with PKCE (S256) against the xChk authorize endpoint (`/oauth/authorize` on `https://in.xchk.io`).
2. Scopes: `openid`, `profile`, `email`, `xchk_portfolio`; optional `xchk_portfolio_proofs`.
3. User authenticates on xChk, selects per-app artifact disclosure, and completes bilateral consent (subject policy vs RP institutional portfolio).
4. Crabby Editor backend exchanges the code at `POST /oauth/token`, verifies the RS256 ID token (discovery + JWKS), and optionally verifies the portfolio HMAC (Appendix A).
5. Backend mints a Crabby session from `sub` / `email` and stores disclosed portfolio claims for policy evaluation.

RP policy evaluation. Crabby Editor reads nested `xchk.portfolio` artifacts and `xchk.policy` from the consent-time ID token snapshot. Access decisions are RP-local: the IdP displays policy satisfaction on the consent screen but does not grant or deny Crabby access. Typical gates inspect artifact types (e.g. helpdesk pass, interview pass, KYC linkage) via the RP's own sufficiency function—the BYOI pattern from §3.3.

Integration surface area. Beyond standard OIDC (discovery, JWKS, PKCE, token exchange), the RP adds: parsing nested `xchk` claims, optional portfolio HMAC verification, and application policy over artifact types. Integrators can reproduce the flow using the hosted OAuth playground (`https://in.xchk.io/oauth/playground.html`) before deploying a callback. Step-by-step guide: `https://in.xchk.io/login-with-xchk.html`.

What this appendix does not claim. No controlled A/B measurements are reported beyond the public-endpoint micro-benchmark in §5.6; no second independent RP is documented; agent surfaces B/C are exercised on in.xchk.io but not through Crabby Editor's OIDC login path.

C. Portfolio HMAC Verifier (Node.js)

Runnable verifier matching Appendix A and production `portfolioSignatureUtils.js`. Also published at `https://in.xchk.io/verify-portfolio-hmac.cjs`.

```
// verify-portfolio-hmac.cjs --- xchk-portfolio-v1
const crypto = require('crypto');

function portfolioSigningKey(secret) {
  return crypto.createHash('sha256').update(secret)
    .digest('hex').slice(0, 32);
}

function sortKeysDeep(value) {
  if (Array.isArray(value)) return value.map(sortKeysDeep);
  if (value && typeof value === 'object') {
```

```
    return Object.keys(value).sort().reduce((acc, key) => {
      acc[key] = sortKeysDeep(value[key]);
      return acc;
    }, {});
  }
  return value;
}

function verifyPortfolioSignature({
  portfolio, policy, rpInstitutional, subjectRpPolicy,
  signature, secret,
}) {
  const key = portfolioSigningKey(secret);
  const canonical = JSON.stringify(sortKeysDeep({
    portfolio,
    policy: policy ?? null,
    rpInstitutional: rpInstitutional ?? null,
    subjectRpPolicy: subjectRpPolicy ?? null,
  }));
  const expected = crypto.createHmac('sha256', key)
    .update(canonical).digest('hex');
  if (!signature || expected.length !== signature.length) return false;
  return crypto.timingSafeEqual(
    Buffer.from(signature, 'hex'),
    Buffer.from(expected, 'hex'));
}

module.exports = {
  verifyPortfolioSignature, portfolioSigningKey, sortKeysDeep
};
```

Full file (with usage comments): `https://in.xchk.io/verify-portfolio-hmac.cjs`.

References

[1] D. Hardt. The oauth 2.0 authorization framework. RFC 6749, IETF, 2012.

[2] N. Sakimura et al. Openid connect core 1.0. OpenID Foundation, 2014.

[3] W3C. Verifiable credentials data model v1.1. W3C Recommendation, 2022.

[4] SPIFFE. Spiffe standard. CNCF, `https://github.com/spiffe/spiffe`, 2020.

[5] T. Kampik et al. Governance of autonomous agents on the web: Challenges and opportunities. *ACM Trans. Internet Technol.*, 22(4):1–31, 2022.

[6] S. Schultze et al. Building the web for agents: A declarative framework for agent-web interaction. arXiv:2511.11287, 2025.

[7] H. Wang, C. M. Poskitt, and J. Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents, 2026.

[8] A. Kaul, Q. Lan, and P. Gupta. Behavioral governance for autonomous AI agents: The AgentBound framework. arXiv:2606.30970, 2026.

[9] A. O. Osho et al. Risk-based authentication: A survey. arXiv:2108.07890, 2021.

[10] W3C. Decentralized identifiers (dids) v1.0. W3C Recommendation, 2022.

[11] OpenID Foundation. OpenID Connect for Agents (OIDC-A) 1.0. arXiv:2509.25974, 2025.

[12] Forter Ltd. Systems and methods for implementing automated agent authentication protocols. WO2026039510A1, 2024.

[13] Onesource Solutions International Inc. System for federated compliance-token inheritance. US20250373432A1, 2024.

[14] Microsoft Corporation. Agent governance toolkit (AGT). `https://github.com/microsoft/agent-governance-toolkit`, 2026.

[15] A. Singh et al. Context lineage assurance for non-human identities in critical multi-agent systems. arXiv:2509.18415, 2025.

[16] S. Dutta and A. K. Moharir. AgentRiskBOM: A risk-scoping security bill of materials for agentic AI systems. arXiv:2606.21877, 2026.

[17] A. Singh et al. Evolution of AI agent registry solutions. arXiv:2508.03095, 2025.

[18] Orrick AI Law Center. US AI law tracker. `https://ai-law-center.orrick.com/us-ai-law-tracker-see-all-states/`, 2026. Accessed 2026.

[19] NIST. Digital identity guidelines. Technical Report NIST SP 800-63-3, National Institute of Standards and Technology, 2017.

[20] European Parliament and Council. eIDAS regulation (EU) no 910/2014, 2014.

[21] T. Lodderstedt et al. OAuth 2.0 rich authorization requests. RFC 9396, IETF, 2023.

[22] J. Richer and F. Imbault. Grant negotiation and authorization protocol. RFC 9635, IETF, 2024.