PRE-GEN

Technical Specification

Valerii Egorov (Pastheroza)
Independent Researcher
ORCID: 0009-0000-2599-0547
Pastheroza@gmail.com

DOI: 10.5281/zenodo.20129901
Version 5 · 28 September 2026

Abstract

No standardized mechanism exists for determining whether AI-generated content is authorized before it is produced. Takedown requests, copyright suits, and content provenance standards all operate after the fact — after the image is generated, after the video is distributed, after the harm is done. Meanwhile, generative systems can depict or embody any entity to which rights attach: persons, estates, brands, synthetic characters, voices, virtual personas, and entity classes not yet anticipated.

I present PRE-GEN (PG), an open pre-generation authorization protocol for generative systems. PG introduces three mechanisms that close this gap: (1) a two-party cryptographically signed generative-rights license; (2) a pre-generation authorization query that AI providers execute before producing any output, answered with a signed decision — allow, not blocked, review, or deny; and (3) a pre-emptive opt-out with absolute priority — a subject's refusal terminates generation before any license is consulted, and no license can override it. After an allowed generation the provider files a receipt, and every decision is written to an externally anchored audit log. Version 5 fixes the identifier format, separates usage records from licenses, and lets several independent registries operate the same protocol without conflicting codes. The specification, byte-level test vectors, and conformance runner are published under Apache-2.0.

Keywords: pre-generation authorization, generative rights, two-party signed license, opt-out registry, signed decisions, PG codes, federated registries, pre-gen

1. Problem Statement and Scope

Generative AI systems can produce content depicting or embodying entities to which rights attach: natural persons, estates, brands and trademarks, synthetic characters, voices, and virtual personas. Every existing remedy operates post-hoc. Takedown requests address content already distributed. Copyright suits address harm already caused. Content provenance standards describe what was generated, but do not determine whether it should have been generated. There is no standardized mechanism for answering the authorization question *before* generation.

PRE-GEN (PG) is a pre-generation authorization protocol for generative systems. It introduces three key mechanisms:

Two-party signed generative-rights license. Every license is cryptographically signed by the rights-holder and countersigned by the registry operator. Neither party can act alone. The countersignature ensures that license issuance requires the explicit consent of the rights-holder, even if the registry operator is compromised.

Pre-generation authorization query. AI providers query the registry before generating any content and receive a signed decision. The decision precedes the first pixel. Where provenance standards describe what was generated, PG determines whether it should be generated.

Pre-emptive opt-out with absolute priority. A subject can refuse all generation before anyone attempts to obtain a license. The opt-out is checked before license resolution and cannot be overridden by any license.

Contribution

PG addresses a gap that existing standards were not designed for. Content provenance systems describe what was generated; PG determines whether it should be generated. Authentication protocols establish identity; PG establishes generative rights — what a specific party is authorized to generate depicting a specific subject, under what conditions, with what restrictions. The two-party signed license, the pre-generation query, the opt-out with absolute priority, the four-layer architecture, and a registry model in which any number of operators can issue compatible identifiers together constitute pre-generation authorization for generative AI systems.

Scope. PG standardizes compliant generation flows. It does not and cannot prevent unauthorized generation by uncooperative providers, local open-source inference, or offshore services. The protocol operates within the compliant ecosystem: providers who choose to integrate. A signed decision records what the registry answered to a declared request; it does not prove that the output matched the request, that every generation was reported, or that a use is lawful.

2. Protocol Layers

The protocol is organized in four independent layers. Each layer may be adopted separately. A provider that implements only Layer 1 is already useful; layers 2 through 4 add capability without requiring modification to Layer 1.

Layer 1 — Core Authorization Protocol. Two-party signed licenses, a verification API returning signed decisions, receipts filed after generation, and an append-only, hash-chained audit log with external anchoring. This is the minimum viable protocol.

Layer 2 — Identity Resolution. Determines whether a prompt references a registered subject. Each registry publishes a public subject index; detection runs on the provider's side and produces *candidates*, never proof. Providers may use name matching, embedding classifiers, voice-print or face matchers, or third-party entity linking without modifying the authorization layer. Authorization is always asked for a named subject.

Layer 3 — Policy Profiles. Jurisdiction- and business-dependent rules: registration trust, review of commercial requests, territorial restrictions, legal overrides, and dispute flows are expressed as registry policy, not as core protocol logic.

Layer 4 — Provenance Bridge. The registry issues a signed assertion (`pg.assertion.v1`) that a provider embeds in its own C2PA manifest, pointing to the decision and receipt behind a file. Layer 4 is useful but not required for the authorization guarantee.

3. Trust Model

Every subject is bound to an Ed25519 signing key. The key may be held by the subject (self custody) or held encrypted by the registry and used only on the subject's instruction (managed custody). In both cases licenses carry a real subject signature; managed custody moves the trust from the subject's device to the registry's key vault, and a registry offering it must say so. The registry operator publishes its own keys, with their history, as a signed key set at `/v1/keys`. AI providers authenticate with a credential issued to them by the registry; end users connect their registry account to a provider once. All parties may be mutually distrustful.

The principal threat model consists of six scenarios: (1) a licensee fabricating authorization it does not hold; (2) the registry operator fabricating authorization on behalf of a subject; (3) an AI provider generating without consulting the registry; (4) a third party tampering with stored records or the audit log; (5) a bad actor registering as a subject they do not represent; (6) a valid decision replayed for another user, product, or project.

Threats (1) and (2) are addressed by two-party signatures. Threat (3) cannot be prevented by any protocol; receipts, assertions, and the absence of them provide evidence, not enforcement. Threat (4) is addressed by the audit log: append-only hash chain, per-event signatures, and external anchoring. Threat (5) is addressed by registration trust states and the dispute flow (§4.2). Threat (6) is addressed by binding the decision to the declared intended use and end user, and by accepting exactly one receipt per decision.

4. Entities, Registration, and Cryptographic Material

4.1 Subjects

Every subject is identified by a `subject_id` and a permanent PG code (§5), and is bound to a public signing key registered at issuance. A subject is any entity to which generative rights attach: natural persons (likeness, voice, mannerisms); estates of deceased persons; brands and trademarks; synthetic characters; voice profiles; typefaces; artistic styles; product designs; and creative works whose generative reproduction requires authorization. The protocol does not define what can be a subject — it defines how subjects are registered, how rights are expressed, and how authorization is verified.

4.2 Registration trust

Registering a subject does not by itself prove ownership. A registry records two independent trust axes, and both are visible to verifiers:

Lifecycle — `self_attested`, `pending_verification`, `verified`, `commercial_enabled`, and the restricted states `suspended`, `disputed`, `revoked`, `archived`. Commercial licensing requires both the subject and the account that owns it to be verified.

Authority — `self`, `agency_asserted`, `consented`, `verified`: how well the registrant's right to act for the subject is proven. Every decision carries this value as `subject_authority`, so a provider can see how strong the backing of an allow is.

Which evidence moves a subject between states is registry policy (Layer 3). Subjects flagged as high-risk — minors above all — carry fixed prohibitions that no license can lift (`PG_PROHIBITED_USE`). A registry may add a protection flag on the owner's request; only the registry may remove one.

4.3 Registry operator, providers, and licensees

The registry operator holds an issuer keypair that countersigns licenses and signs decisions, audit events, evidence bundles, and assertions. Key rotation keeps every previous public key in the signed key set, so past signatures stay verifiable.

An AI provider obtains a credential from the registry after proving control of its company: a sandbox credential, which reaches synthetic test subjects only, and a production credential after a DNS record on the company's domain. The credential is a server-side secret and never appears in a browser, a URL, or a prompt.

A licensee — typically an end user of the provider — does not hold cryptographic material. The user connects their registry account to the provider once (§9); afterwards every query is server-to-server. No protocol material appears in user-facing prompts. This eliminates prompt injection as a vector against the protocol and keeps inference APIs unchanged.

5. Identifiers

Every subject and license has a permanent, human-readable PG code. The format is frozen: a code is cited in licenses, disputes, and court filings, so changing how it is computed would invalidate every code already issued.

Shape	Example	Meaning
`PG-<serial><check>`	`PG-000042*`	A subject
`PG-<CLASS>-<serial>-<tail><check>`	`PG-RND-000042-ZZZZZZ1`	A license
`PG-<ISSUER>-<serial><check>`	`PG-NWRD-000042=`	A subject in another registry

The alphabet is Crockford base32 — digits and uppercase letters without I, L, O, U — so a code survives being read off a screenshot. The serial is zero-padded to at least six digits and widens freely. The license tail is six random characters, so the number of licenses on a subject cannot be inferred. The check character is a position-weighted sum modulo 37 over the body, with five extra symbols (`* ~ $ = !`) valid only in the check position; it catches every single-character substitution and every adjacent transposition. A mistyped code is reported as mistyped, distinctly from not found.

Every registry other than the origin one prefixes its codes with exactly four letters from the same alphabet. The prefix is part of the checked body, and bare codes belong to the origin registry forever (§11). Two legacy subject shapes from before the check character (`PG-STD-000123`, `PG-SUB-000123`) remain valid. An identifier is resolved by lookup, never parsed for meaning; the issuer prefix is the one thing software may read from it, to know which registry to ask. Anyone can resolve a code without an account through the registry's public resolver.

6. License Schema and Signing

A license body (`pg.license.v2`) contains the subject and licensee identifiers, license class, scope (modalities), default-deny categories, immutable denials, required obligations, contract start and expiry, product and project, purpose, rights-holder rules text, and an extensions array for typed constraints. Optional fields — allowed channels, territories and providers, blocked providers, maximum uses, concurrency limit, granted rights, transferability — are included only when set, so a

license signed before a field existed keeps verifying byte for byte. A license is usually bound to one named project; another project is refused (`PG_PROJECT_MISMATCH`).

Classes. `RND` — research, education, and editorial use; `PRM` — commercial use; `STD` — personal use, kept for licenses already issued. Personal use is now a usage record rather than a license (§7), so no new `STD` licenses are issued. The classes `EDT`, `EST`, `ENT`, `PER`, and `OWN` are retired: their identifiers keep resolving, and a retired class is never issued again or given a new meaning.

Extensions express restrictions evaluated at verification time, keyed by versioned type — `pg.temporal_window.v1`, `pg.geo_fence.v1`, `pg.volume_cap.v1`. Unknown extension types fail closed: the engine refuses rather than silently allowing.

Canonical form. Every signed object is serialized as JSON with keys sorted by code point, no whitespace, non-ASCII characters as raw UTF-8, and integers only. Unicode is not normalized: visually identical NFC and NFD strings sign differently, by design. Hashes are SHA-256 in lowercase hex; signatures are Ed25519 over the canonical bytes with the signature field excluded; a key is named by `pg-ed25519:` and the first 32 hex characters of the SHA-256 of the public key.

Issuance is a two-pass operation. The subject signs the canonical body. The registry verifies that signature, allocates the license identifier, and countersigns. Verification reverses the construction; if either signature fails, the license is treated as forged and the verifier returns `PG_INVALID_SIGNATURE`, distinct from `PG_NO_LICENSE` in the audit log.

7. Verification Pipeline

The verification pipeline runs before any output is produced. Once a depiction exists, it can be copied and distributed regardless of any later refusal. The pipeline enforces contextual authorization: not only whether a license exists, but whether the specific intended use falls within its scope.

A provider sends a server-to-server request to `/v1/verify` with its credential, the licensee identifier (omitted for personal use), the subject, a SHA-256 `prompt_hash` — never the prompt text — the model, the modality, the end user's identifier at the provider, and an `intended_use` object: use case (`personal`, `educational`, `research`, `editorial`, `commercial`), channel, product, project, territory, and categories. The engine evaluates the request in a fixed order; each stage is terminal:

Stage 1 — Subject. The subject is resolved; an unknown one returns `PG_NO_SUBJECT`.

Stage 2 — Opt-out. An active opt-out covering the request returns `PG_SUBJECT_OPTED_OUT` immediately, before any caller or license state is consulted. No license can override it.

Stage 3 — Caller. The provider credential and, where a licensee is named, its connection are checked (`PG_NO_PAIR`). A sandbox credential reaches synthetic subjects only (`PG_SANDBOX_SUBJECT_ONLY`).

Stage 4 — Subject status and trust. Pending, paused, disputed, and withdrawn subjects return their own codes. A provider the rights-holder has blocked is refused (`PG_PROVIDER_VETOED`). Fixed prohibitions for high-risk subjects apply (`PG_PROHIBITED_USE`). The trust lifecycle and, for commercial use, the owner's verification are enforced (`PG_SUBJECT_NOT_LICENSABLE`, `PG_OWNER_NOT_VERIFIED`).

Stage 5 — Personal use. A personal request with no license that passes a fixed floor — images only; no commercial, advertising, endorsement, political, or adult use — is answered `not_blocked` with `PG_STD_TRACKING_ONLY`: nothing prohibits it and nothing is granted. The provider may generate under its own policy and report the use; there is no license and no receipt.

Stage 6 — License resolution. Every license for the subject and licensee is checked for status, expiry, contract window, revocation, and both signatures. A license bound to one end user is refused for another (`PG_IDENTITY_MISMATCH`); a license bound to a product or project is refused for a different one. If a license restricts a dimension that the request omits, the request is refused rather than allowed by omission (`PG_MISSING_CONTEXT`).

Stage 7 — Scope. Immutable denials are evaluated first (`PG_IMMUTABLE_DENIAL`); then default-deny categories, the use-case rank of the license class (`PG_TIER_VIOLATION`), channel, territory, modality, and granted rights (`PG_SCOPE_VIOLATION`, `PG_RIGHTS_NOT_GRANTED`).

Stage 8 — Extensions and limits. Typed extensions are evaluated; purchased quantities are reserved atomically (`PG_USAGE_LIMIT`) and concurrency is capped (`PG_CONCURRENCY_LIMIT`). A commercial generation of a high-risk subject may be held for a person to decide (`PG_HELD_FOR_REVIEW`).

Every request returns a signed decision (`pg.decision.v1`) with a `disposition` — `allow`, `not_blocked`, `review`, or `deny` — and a `PG_*` reason code classified as hard (no retry, reshaped request, or new license changes the answer) or soft. An unknown code is treated as hard. The decision also carries the obligations to apply, the `policy_version` of the rules that produced it, the subject's revocation epoch, a cache scope (only a plain allow is cacheable, briefly), the echoed intended use, and, on `PG_NO_LICENSE`, a remediation link where the user can request a license. All decisions are appended to the audit log.

8. Opt-Out Registry and Policy Profiles

8.1 Total opt-out

The opt-out registry is structurally distinct from the license registry: it expresses the refusal of permission rather than its grant. An opt-out names a subject, a reason class (personal, estate, safety, legal), a scope (all modalities or a specific one), and an authority proof. The reason text is stored privately and is never returned in any API response or audit log. Registering an opt-out is free.

A registry should make an opt-out harder to register than a license, and publish it as pending during a cooling period: a fraudulent license costs money and leaves a trail, while a fraudulent opt-out would let bad actors lock subjects out of legitimate use.

8.2 Jurisdiction profiles and legal overrides

The opt-out is absolute within the protocol. However, courts and legislatures operate outside it. A court may order generation despite an active opt-out. This is not a contradiction: the protocol is a lock; a court order is a warrant. The lock does not become weaker because warrants exist. A legal override is recorded as a parallel, time-limited and scope-limited record, never as a modification of the opt-out, and the jurisdiction profile defines its format and validation.

8.3 Rights-holder controls

Besides the opt-out, a rights-holder may deny specific categories, block specific providers, require review of each request, and attach display-only rules text. These controls are enforced by the pipeline; the rules text itself is shown, not enforced.

9. Provider Credentials and User Connection

Provider onboarding. A provider applies with its identifier, legal name, domain, a security contact, and a contact where rights-holders can reach it. A verified account email on the company's domain yields a sandbox credential; a DNS TXT record on that domain yields a production credential. Credentials are shown once and can be rotated; the old secret stops working immediately.

User connection. Connecting an end user replaces the pair handshake of version 4. The provider sends the user to the registry with a return address registered in advance. The user confirms, and the registry sends back a single-use code valid for five minutes. The provider's server exchanges it, with its credential, for the user's licensee identifier and an identity link. On every later query the provider names that user; a license the user obtained is bound to that link and answers only for them. Personal-use queries need no connected user.

A provider becomes available to users other than its own team after it completes one production allow followed by its matching receipt. Connections may be revoked from either side; revocation is delivered by signed webhooks with replay protection and event deduplication.

10. Receipts, Audit Log, Anchoring, and Evidence

Receipts. After an allowed generation the provider files a receipt (`pg.receipt.v2`) naming the decision, prompt hash, output hash, model, and the obligations it applied. The opt-in `pg.receipt.v3` additionally signs the lifecycle event — `preview`, `output_accepted`, `output_delivered`, or `output_published`. A provider with a registered key signs its receipts; the registry stores a hash. Exactly one receipt is accepted per decision, so a replayed decision cannot yield a second one. A receipt attests the provider's claim; it does not prove that every generation was reported.

Audit log. Every state-changing operation produces an audit event with an identifier, action, entity, actor, timestamp, metadata, `prev_hash`, event hash, and a signature by the issuer key. The chain gives ordering integrity; the per-event signature gives authenticity even against an attacker with database write access. A Merkle root over event hashes is exposed with inclusion proofs, and anchoring publishes the signed root to public timestamp calendars outside the operator's control.

Evidence. For any decision the registry returns a signed evidence bundle (`pg.evidence.v1`): the decision, the license it resolved against, every related audit event, an inclusion proof where one exists, and the key set — everything needed to verify it offline, years later.

11. Multiple Registries

PG is an open standard: any company may operate a registry, and compatible registries are listed publicly at pregen.org. To keep their codes from ever colliding, every registry other than the origin one prefixes each code it issues with a four-letter issuer prefix assigned through that public list, and a prefix is never reassigned. A registry that receives a code with a prefix it does not hold answers that the code belongs to another registry and points to the list, instead of reporting it as not found.

Specified for later versions: a common discovery document for each registry's endpoints and keys; one client that reaches every listed registry; opt-outs and protections for minors honoured across registries holding the same person; and a freeze on new licenses while two registries dispute the same subject. The cross-signing, trust delegation, and transparency gossip described in version 4 remain future work.

12. Versioning

A PRE-GEN version is one published release of this specification, numbered as on Zenodo under the concept DOI 10.5281/zenodo.20129901: versions 1 to 3 were published as Prompt Protocol in May 2026, version 4 as PRE-GEN on 1 June 2026, and this document is version 5. Nothing issued ever changes meaning between versions: a code, signed object, or test vector valid under one version stays valid under every later one; a version may stop issuing something, never stop recognizing it.

Each signed object carries its own schema marker inside the signed bytes — `pg.decision.v1`, `pg.license.v2`, `pg.receipt.v2` and `v3`, `pg.evidence.v1`, `pg.assertion.v1` — and these move independently of the PRE-GEN version: adding an optional field never bumps a marker; removing a field or changing its meaning does. Decisions also carry a `policy_version` naming the rules that produced them. Where this text and the published test vectors disagree, the vectors win; an implementation conforms when it reproduces every vector.

13. Limitations

The protocol does not enforce that AI providers consult the registry. PG is a governance and compliance protocol, not a universal enforcement protocol. Its value is proportional to the size of the compliant ecosystem.

Identity resolution remains an open problem. Name matching catches direct mentions, misspellings, and many obfuscations, but not indirect references, style transfer, or visual resemblance; recognizing a face or a voice is left to the provider's own detectors. A decision covers one subject: an output depicting several subjects cannot yet be authorized as a group. An output hash identifies exact bytes and does not survive re-encoding.

The protocol does not adjudicate fair use, parody, or other legal exceptions. It refuses by default and provides cryptographic evidence; legal exceptions are resolved through jurisdiction profiles (§8.2).

14. Implementation Notes

The protocol is defined independently of specific libraries. The published test vectors pin canonical JSON, SHA-256, Ed25519 (RFC 8032), and the PG code format byte for byte, and a conformance runner replays them against any implementation. User connection uses a one-time authorization code in the manner of OAuth 2.0 (RFC 6749); webhooks are signed with HMAC-SHA256; the provenance bridge targets C2PA; audit roots are timestamped through OpenTimestamps. The reference registry is PRAMPTA, the first registry operating PG in production; a TypeScript client is published as `@prampta/sdk`.

15. Ongoing and Future Work

Work on PRE-GEN is ongoing; the specification will be revised as the protocol matures and adopters provide feedback. Planned work includes the multi-registry interfaces of §11, group authorization for outputs with several subjects, a training opt-out register for dataset collection, generation guidelines passed from rights-holders to providers, and signed commercial terms and settlement records. Research also continues on complementary technologies under the codename PRE-LEARN and other supplementary ideas that may strengthen the protocol or develop into independent projects. Details will be published separately when ready.

16. Changes Since Version 4

Identifiers. The `PG-CLASS-SEQUENCE` form is replaced by check-summed subject codes and license identifiers (§5), with four-letter issuer prefixes for other registries.

License classes. `STD`, `RND`, `PRM`; `EDT`, `EST`, `ENT`, `PER`, `OWN` retired. Personal use is a usage record answered `not_blocked`; no new `STD` licenses are issued.

Decisions carry a four-value disposition, policy version, cache scope, revocation epoch, subject authority, and remediation link; the prompt is sent as a hash, never as text.

Callers. Providers authenticate with their own credential and connect end users once; licenses can be bound to an end user, a product, and a project.

Added: receipts, evidence bundles, assertions, the refusal-code registry with hard and soft codes, registration trust states, published test vectors and a conformance runner, and the public list of registries.

References

[1] S. Josefsson, I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, IETF, 2017.

[2] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, IETF, 2012.

[3] Coalition for Content Provenance and Authenticity. Content Credentials Specification v2.1 (ISO 22144). https://c2pa.org/specifications/, 2025.

[4] B. Laurie, A. Langley, E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[5] D. Crockford. Base 32. https://www.crockford.com/base32.html.

[6] P. Todd. OpenTimestamps. https://opentimestamps.org.

[7] D. Crocker, P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234, IETF, 2008.

[8] European Parliament. Regulation (EU) 2024/1689 (AI Act). Official Journal of the European Union, 2024.

[9] S. Dathathri et al. Scalable watermarking for identifying large language model outputs (SynthID). DeepMind, 2024.

[10] V. Egorov. PRE-GEN — specification, test vectors, and conformance runner. https://www.pregen.org, github.com/Pastheroza/PRE-GEN-site, 2026.

Authorship and translation note. This specification was written by Valerii Egorov (Pastheroza), an independent researcher (Pastheroza@gmail.com, ORCID: 0009-0000-2599-0547). The author acknowledges that his English is not strong; the original manuscript was drafted in his native language and adapted into English with the assistance of AI translation tools. AI-assisted drafting may introduce errors, misinterpretations, or inaccurate representations of the author's intended ideas. The author bears sole responsibility for the technical content and welcomes corrections through the repository's issue tracker.