



Secret Garden: A Cryptographic Gateway for AI Processing of Regulated Health Data

VRG LABS

October 2026

Status: in active development. Open core released under the Apache-2.0 license.

1. The problem: PHI leaves your control long before it reaches a model

Every organization that wants to use an external AI model, an analytics service, or a specialized processing vendor for work involving patient data runs into the same structural problem. The useful work happens outside the boundary where protected health information (PHI) is allowed to exist. The moment a record is sent to a processor, copies of that record begin to multiply: in the processor's request logs, in its intermediate storage, in its subprocessors' systems, and in whatever observability tooling sits along the path. Each copy is PHI. Each copy carries breach exposure, retention questions, and audit obligations.

The conventional controls do not change this shape. Transport encryption protects the wire, not the endpoint. Contractual language allocates liability after the fact. De-identification, done to the standard HIPAA actually requires, is a destructive, expert-driven process that is difficult to apply to free-text clinical material at request time, and it destroys the ability to connect a result back to the patient it belongs to, which most clinical and operational workflows require.

What is needed is a control that sits at the boundary itself: one that decides what may leave, transforms it before it leaves, keeps the keys that make the transformation reversible, and restores the result on return, so that the outside processor never holds identifiable data in the first place. That is the role Secret Garden is built to fill.

2. The gateway approach

Secret Guardian is a cryptographic gateway proxy for HIPAA-regulated data. It deploys as software inside the customer's own network boundary, in the VPC, VNet, or on-premises network where PHI is permitted to exist. Tenant systems send records to the gateway. The gateway inspects each record, replaces identifiers with tokens under keys that never leave the customer's control, seals the tokenized record to a specific registered processor, and, when the processor's response returns, verifies it and re-links the tokens to the original values before delivering the result to the tenant. Processors only ever see tokens.

The flow, step by step:

```
tenant system
  | record (cleartext, inside the boundary)
  v
Secret Guardian gateway
  1. detect:    structured rules + statistical named-entity detection
  2. judge:     policy per tenant/processor pair; default-deny;
                anything below the confidence gate is quarantined
  3. tokenize:  identifiers replaced with deterministic tokens
                under the tenant's key
  4. seal:      tokenized record wrapped in a signed hybrid
                post-quantum envelope for one processor
  |
  v
registered processor (sees tokens only)
  |
  v
Secret Guardian gateway
  5. verify:    signature, replay, and schema checks on the response;
                return-path scan for cleartext identifiers
  6. re-link:   tokens resolved back to original values
  |
  v
tenant system (re-linked record)
```

Three design decisions shape everything downstream.

First, processors are reachable only through a registry. A processor exists to the gateway only as a complete registration: identity, endpoint, public keys, an expected record schema, and a recorded Business Associate Agreement (BAA) status. Only registrations whose BAA status is active admit traffic. An unregistered processor is not merely discouraged; the gateway will not dial it. Suspended or BAA-inactive registrations refuse all traffic while remaining on record.

Second, the gateway fails closed. Detection unavailable, a malformed registration, a broken audit chain, an undeclared field in a response: each of these stops traffic or quarantines the record rather than letting it pass. Deliberately degraded operation is the one exception, and it is

operator-controlled: off by default, announced in banners and written to the audit log while it is active, and never silent.

Third, records are handled in structured and in FHIR modes. For processors registered for FHIR payloads, every resource in a request must be of a registered resource type or the whole request is refused before any scanning occurs; identifier elements are tokenized per policy; dates of birth are generalized to year only, one way, and are never restored on return; and binary attachment content is blocked unless policy explicitly rules on it, because no text detection can judge a blob.

3. Detection: finding identifiers before anything leaves

Detection runs in two layers, because identifiers appear in two forms.

Structured and patterned content is judged by deterministic rules. The detection target is the set of identifier types enumerated in HIPAA's Safe Harbor method: names, addresses and ZIP codes, dates, telephone and fax numbers, email addresses, Social Security numbers, medical record numbers, health plan and account numbers, certificate and license numbers, vehicle and device identifiers, URLs, IP addresses, biometric identifiers, full-face photographs, and any other unique identifying number or code. Matching is shape-based and applies normalization before judgment, so identifiers presented in altered or obfuscated forms are judged on a canonical form, while the spans reported always refer to the original text.

Free text, such as clinical notes, is judged by a named-entity-recognition (NER) sidecar that runs inside the tenant's boundary alongside the gateway and combines a statistical named-entity model with the deterministic pattern rules of the first layer. The sidecar holds no keys and returns findings only, never the matched text, so the detection channel cannot become a second copy of the PHI it exists to protect. The gateway independently validates every finding against the original text before acting on it.

A confidence policy then decides what happens. Findings carry a confidence value; any finding below the configured threshold quarantines the request instead of tokenizing it, and the same gate applies to processor responses on the return path. Policy is expressed per tenant and processor pair over a closed set of actions (tokenize, redact, block, pass), and it is default-deny: an identifier class with no explicit rule blocks the request. Policy also treats dates and ZIP codes as quasi-identifier classes, distinct from direct identifiers.

4. Tokenization and key custody

Detected identifiers are replaced with deterministic tokens: the same value, for the same tenant, produces the same token, and a different tenant's key produces an unrelated one. Tokens are computed as keyed HMAC-SHA-256 values under a per-tenant key. Determinism is deliberate. It is what allows a processor to return results that the gateway can re-link, and it allows the same patient to map to the same token across encounters, which longitudinal workflows require. It is also, as Section 6 states plainly, one of the reasons tokenized output remains PHI.

Token keys are held in a key vault under the customer's control. In the production posture, tenant keys are generated and held in hardware-backed storage, an HSM reached through the standard PKCS#11 interface, and are non-extractable: tokenization is performed against the key inside that boundary, so key material is never present in the gateway's ordinary process memory.

Above the tenant keys sits a ceremony master key, a key-encryption key that wraps the per-tenant keys at rest. Custody of the master is split using Shamir secret sharing with a threshold of two of three shares, held by separate custodians. A single share is, on its own, information-theoretically useless, and shares are verified at reconstruction, so a tampered share is detected rather than silently used. The processor never holds a share and never runs the ceremony tooling.

The operational rules around the split are enforced by the ceremony tooling. Reconstruction and rekeying require two distinct recorded operators, the two-person rule, and every custody event is written to the same tamper-evident audit record described in Section 7. Any use of the master outside routine custody is a break-glass event, and any break-glass event forces a complete rekey before the key hierarchy returns to service.

5. Cryptography: standardized primitives, hybrid post-quantum transport, no home-rolled crypto

Secret Guardian composes its cryptography from standardized primitives and established implementations. No cryptographic primitive in the system is home-rolled.

Each message to a processor is sealed in an envelope with its own per-message data key. Key establishment is a hybrid key encapsulation mechanism combining classical X25519 with ML-KEM-768, the post-quantum KEM standardized as FIPS 203. The data key is derived with HKDF-SHA-256 from both shared secrets, so neither leg of the hybrid alone determines the key. The payload is encrypted with AES-256-GCM under a fresh random nonce, and the envelope's header fields are bound into the authenticated data, so re-labeling an envelope, changing its recipient, tenant, or processor, breaks decryption and signature verification. Any KEM suite other than the hybrid is rejected, and there is no downgrade negotiation.

In the production posture, envelopes are signed with ML-DSA-65, the post-quantum signature scheme standardized as FIPS 204, with signing keys held separately from encryption keys for every party, and endpoints reject unsigned envelopes. On receipt, the gateway verifies before it decrypts: signature, replay, and freshness checks all pass before the envelope is opened, and responses are held to the same standard on the return path.

Transport follows the same posture. The TLS floor is version 1.3 with mutual TLS and exact certificate identities as the authorization model, and gateway endpoints admit only the hybrid X25519MLKEM768 key-exchange group. Peers that cannot offer the hybrid group are refused rather than accommodated with a weaker one.

6. The honest boundary: tokenized is not de-identified

This section is the one to quote back at us, because it is where gateway vendors most often overstate.

Re-linkable tokenized data is still PHI. A Business Associate Agreement is still required with every processor, and Secret Guardian does not replace that legal instrument, verify it, or make tokenized output "de-identified" in the sense of HIPAA Safe Harbor or expert certification. The gateway's registry enforces the recorded BAA status because the record is operationally useful; the legal obligation exists independently of the software. Longitudinal linking exists precisely because customers need the same patient to resolve to the same token, and that linkability is a feature. It is also why the output remains PHI. Any reading of tokenization as anonymization is wrong, and this project's documentation is written to refuse that reading.

What the gateway actually provides is narrower, and real:

- Breach-impact reduction. A processor-side breach exposes tokens and record structure, not names, numbers, and dates. Outside the customer's boundary the cleartext exists nowhere; inside it, it lives in the tenant's own systems and passes through the gateway - it never rests at the processor.
- Key segregation. The means of reversal, the per-tenant keys and the split-custody master above them, never travel with the data and are never held by the processor. Compromise of a processor does not yield the keys; compromise of tokenized data without the keys does not yield patients.
- Continuous compliance evidence. Every decision the gateway makes (what was detected, what was tokenized, what was refused, which registry state was in force) is written to a tamper-evident audit log at the time it happens, rather than reconstructed for an audit after the fact. Section 7 describes the mechanism.
- Enforcement at the boundary. Policy stops being a document and becomes a gate: unregistered processors cannot be reached, BAA-inactive processors cannot receive traffic, and sub-threshold detections are quarantined instead of shipped.

Evaluators should hold the product to exactly this claim set, no more.

7. Continuous assurance: an adversarial loop and a tamper-evident record

Two mechanisms keep the claims in this paper honest after it is written.

The first is a continuous adversarial testing loop: a red-team/blue-team harness that attacks the detector with mutated and obfuscated identifiers on isolated test infrastructure that holds synthetic fixtures only and has no network path to any production system. Attacks are admitted to the regression corpus only by demonstration, and every attack that ever evades becomes a

permanent test case, so a fixed weakness cannot silently regress. The loop's own stated limit applies to everything it reports: measured coverage describes known attack classes, and a higher coverage number does not prove the absence of unknown ones.

The second mechanism is the audit record. The gateway writes a tamper-evident audit log whose entries are cryptographically hash-chained; the chain is verified before the gateway extends it, and a broken chain prevents startup. Key custody operations write to the same record, so one export covers both the data path and the custody path. Event payloads carry identifiers, hashes, counts, and field paths, never values, so the log itself is not a PHI copy. Auditors receive the chain together with a programmatic verification interface.

8. Status and availability

Secret Guardian is in active development. Its open core is available under the Apache-2.0 license. The architecture described in this paper - boundary deployment, policy-driven detection and tokenization, split-custody key management, hybrid post-quantum transport, and a tamper-evident audit record - is the architecture of the product under development. Detailed design documentation is available to evaluators during evaluation, under discussion with VRG Labs.

Contact

VRG Labs admin@variantresearchgroup.com variantresearchgroup.com