# A Key Means Nothing Alone

## Identity, mandate, and the exposure nobody accepted

What pki.sgit.ai built between site v0.1.25 and v0.1.32, why the concepts underneath it are shaped the way they are, how it composes with RiskMandate.ai, and — as an equal partner to all of that — what none of it proves.

Written 2026-08-27 · 17 chapters in five parts
35,466 words · 14 figures · 65 verified quotations · 48 claims drawn by the writer
Published by the sgit project · pki.sgit.ai · CC BY 4.0

This PDF reads start to finish offline. Every URL that matters appears in full at least once in the text; no link needs following.

## Before anything else

> **These positions travel inside the chapters rather than in an appendix, because a chapter quoted in isolation must still carry them.**
>
> 1. The register is built; the trustworthy register is not. Ten of eleven records are fixtures with published private keys — every signature verifies and proves nothing.
> 2. The root is a fixture, and roots.json says so in its own entry. No chain in this register carries authority.
> 3. The enforcement is real and the authority is not, and they are independent halves. Anybody could forge the mandate the hook enforces.
> 4. The hook is a `setting`, not a `boundary` — it sits inside the grant it bounds, and --no-verify gets past it.
> 5. Every grant is a floor, not a census. An agent measuring its own grant reports what it can see.
> 6. Two environments, one agent, one mandate. Everything generalises from a sample that small.
> 7. The write path is a git commit reviewed by a human, not the account-less lane the design calls for.
> 8. This is a participant's account, published by the project that builds the layer it argues for.

**The provenance rule.** Every load-bearing claim about what this estate MEANS is marked `stated` or `drawn`. `stated` carries a verbatim quotation with a located source; `drawn` carries this book's own reasoning in the reader's view. DO NOT TREAT THE DRAWN CLAIMS AS THIS ESTATE'S POSITIONS — they are the book's.

## Contents

## The figures, and the two gates

Every figure was taken from the version its caption names — a `git worktree` at the tag, a one-shot local server on a port used once and never again, a headless browser killed in a block that runs whether the capture succeeded or failed. A figure of a **past** version is **re-derivable**: re-running the harness at that tag reproduces the digest, and it never goes stale because the tag does not move. A figure of the site **as it stands** is **fresh**: the digest must match the live page, and the build fails when it stops matching — which it will, on the next release.

The six terminal figures are printed in the chapters as their real transcript text, which is what makes this PDF readable offline; the photographed capture of each is recorded in `book/shots/shots.json` with its digest. Appendix A is the whole harness.

| Figure | Tag | Gate | Page digest at that tag |
|---|---|---|---|
| `f01-bench-two-columns` | `current` | fresh | `f495703bcca4f151…` |
| `f02a-registry-at-v0.1.26` | `v0.1.26` | re-derivable | `669a13f39dc51f66…` |
| `f02b-registry-now` | `current` | fresh | `d3c8c83246e73854…` |
| `f03-six-answers` | `current` | fresh | `d3c8c83246e73854…` |
| `f04-identity-raw` | `current` | fresh | `a71eb703924bcfec…` |
| `f05-validate` | `current` | fresh | `1ff583569f0879be…` |
| `f06-sgit-verify` | `current` | fresh | `756d1878d1cdbacb…` |
| `f07-forgery` | `current` | fresh | `b7d853249cc14deb…` |
| `f08-refused-push` | `current` | fresh | `2e1d58f1b48b8963…` |
| `f08b-amended` | `current` | fresh | `c142320f828847bd…` |
| `f09-tier-badges` | `current` | fresh | `412ee64b844da274…` |
| `f10-defeated-boundary` | `current` | fresh | `412ee64b844da274…` |
| `f11-authority-split` | `current` | fresh | `412ee64b844da274…` |
| `f12-assess-midflow` | `current` | fresh | `84882d2759dedebc…` |

FRONT MATTER

# A Key Means Nothing Alone

### Identity, mandate, and the exposure nobody accepted

**One volume on what pki.sgit.ai built between site v0.1.25 and v0.1.32, why the concepts underneath it are shaped the way they are, how it composes with RiskMandate.ai, and — as an equal partner to all of that — what none of it proves.**

*Written August 2026 · CC BY 4.0 · Published by the sgit project*

---

## Before anything else: what this book is not evidence of

A reader who stops after this page should already know the following, because everything after it is written on top of these and a chapter quoted in isolation will not carry them.

**The register is built. The trustworthy register is not.** There are eleven records in the register this book describes. Ten of them are fixtures whose private keys are published in the same repository as their public halves, deliberately. Every signature on those ten verifies. Not one of them proves anything about anybody, because a signature's value is the scarcity of the private half and a fixture has none. You can forge any of them in one command, and Chapter 8 shows you doing it.

**The root is a fixture, and the register says so in its own root file.** No chain in this register carries authority. When the verification walk answers YES, it means *the walk completed and the arithmetic is right*, not *this agent may be trusted*.

**The enforcement is real and the authority is not, and they are independent halves.** A git hook in this repository refuses pushes. It really refuses them — by exit code, from outside the agent's turn, and Chapter 10 photographs it doing so. The mandate it enforces is signed by the fixture root. Anybody could forge that mandate, and the hook would enforce the forgery exactly as diligently. Averaging those two facts into one status is how a demonstration gets mistaken for a control.

**The hook is a `setting`, not a `boundary`.** It sits inside the grant it bounds. The agent can edit the file, unset the config that activates it, or pass `--no-verify`. The refusal banner says this on its own face, which is the only thing that stops it being believed.

**Every grant in this book is a floor, not a census.** An agent measuring its own grant reports what it can see. It cannot report a capability it does not know it has, and the first library entry exists partly because a self-measurement probe was refused mid-measurement.

**Two environments, one agent, one mandate.** Everything in Part Two generalises from a sample that small.

**The write path is a git commit reviewed by a human.** Not the account-less append lane the design calls for. That lane is unbuilt and its blocking question is unanswered.

**This is a participant's account.** It is written by the project that builds the layer it argues for, published on that project's own site, by an agent operating inside that project's own estate. The disclosure travels with the book rather than sitting in a footer.

---

## Who this is for, in the order that decided every editorial call

**A practitioner who runs agents and is accountable for them.** You are the primary reader. You need the concepts in order, one worked example carried all the way through, and something you can do tomorrow. Chapter 17 is that, and it is deliberately small.

**The RiskMandate team.** Also primary. Chapter 12 is written to you and is meant to be usable as a contract rather than as a description: what the library holds, what the instance holds, what crosses the boundary, and what must never. It is written expecting you to disagree with part of it, and it names the open question rather than papering over it.

**An agent given this book and nothing else.** Served by construction rather than by a separate edition. The reference card at the end is written to be pasted into a session; `book/book.json` carries every chapter with the SHA-256 of its markdown; `book/llms.txt` carries the positions above so that a summarising agent cannot drop them.

---

## The provenance rule, and why this book needs it more than most

The packs this book is built from are dense with argument. A writing session moving at pace will blend its own inferences into them so smoothly that afterwards neither the reader nor the session can separate them.

Which is the same error this book is about. Authority nobody granted, assumed because nothing in the presentation distinguished it from authority that was. A book making that mistake about its own sources cannot credibly diagnose it in anybody else's agent.

So every load-bearing claim about what this estate *means* is marked, and the marking is a sentence rather than a sigil:

> *"Document 02 puts it this way: …"* — a **stated** claim. It carries a verbatim quotation, with the document and section it came from. Every one of them is recorded in `book/quotes.json` and re-read out of the source it names on every build. A quotation not found where it claims to be fails the build. This estate does not print quotations it has not checked.

> *"The packs do not say this; reading them together, the reason appears to be …"* — a **drawn** claim. The reasoning is in your view, in the paragraph, not in a note. You are meant to be able to disagree with it without leaving the page.

The colophon carries the count of each. **Do not treat the drawn claims as this estate's positions.** They are this book's.

---

## Every number here was computed, and some of them contradict the brief

No figure in this book is quoted from a release note, from the commissioning brief, or from memory. Each was computed from the repository at the time of writing, and the command that produced it is in Appendix A, so any of them can be re-derived rather than believed.

That discipline had a cost the commissioning brief anticipated and specified the resolution for: **where a number in the brief disagrees with the repository, the repository wins and the brief was wrong.** It happened four times. The largest is that the brief describes "eight releases in four days" and asks for a chapter of that name; the repository says the eight releases spanned **forty hours across two calendar days**. Chapter 14 is named for what the repository says. The colophon lists all four disagreements.

---

## How the figures were taken

Several figures in this book show pages *as they were* — the register on the day it shipped, the push refused at the release where it happened. A figure captioned as the past but photographed today is a reconstruction, and a reconstruction wearing a caption is a claim of authority nobody granted, in a book whose entire subject is claims of authority nobody granted.

So no figure here was reconstructed. Each was taken from the version its caption names: a `git worktree` at the tag, a one-shot local server on a port used once and never again, a headless browser killed in a block that runs whether the capture succeeded or failed. Every figure carries the page, the tag, and the SHA-256 of that page's bytes at that tag.

There are two gates, because there are two different claims. A figure of a past version is **re-derivable** — re-running the harness at that tag reproduces the recorded digest, and it never goes stale because the tag does not move. A figure of the site as it stands is **fresh** — the recorded digest must match the live page, and the build fails when it stops matching. It will stop matching on the next release. That is correct and it is inconvenient, and it is two maintenance costs rather than none, accepted deliberately.

Appendix A is the whole harness. One figure in this book could not be taken as specified, and the reason turned out to be a finding rather than an obstacle; Chapter 10 and Chapter 15 both carry it.

---

## The shape

**Part one — Why there is nothing to inherit.** Three questions the industry answers with one object; a documented catastrophe and the four rules it produced; and the loop that explains why the replacement does not exist yet.

**Part two — The vocabulary, and why each word is load-bearing.** Grant, mandate, delta, tier, and the ordering rule that comes before all of them.

**Part three — What was built.** Four artefacts, each walked, shown, and limited in the same breath.

**Part four — How it composes.** The contract with RiskMandate; the two paths; and the release history read for what each release learned.

**Part five — Honesty, and a first step.** Where the estate contradicts itself, computed rather than recalled. What ships versus what is argued — the chapter that decides whether the other sixteen can be trusted. And the smallest real thing you can do tomorrow.

---

*Sources: everything asserted here is traceable to a published artefact at a constructed URL under* `https://pki.sgit.ai/`*. Every such URL appears in full at least once in the text, so this book reads start to finish offline with no link followed.*

# 1 · A key means nothing alone

*Part one — Why there is nothing to inherit*

---

Generating a keypair takes about four milliseconds. You can do it right now, on any machine, without asking anybody, and when it finishes you will hold a mathematical object of genuine quality: a private half nobody else has, a public half you can hand out, and a signature scheme that will not be broken by anyone reading this.

You will also have accomplished nothing.

That gap — between the ease of making a key and the difficulty of making it *mean* something — is the subject of this book, and it is why the title is not a slogan. A key means nothing alone. It becomes meaningful only when it sits inside a set of relationships that a key cannot itself establish: somebody who recognises it, something that says what its holder may do, and something that stops the holder doing more.

The industry routinely answers all three with one object.

## Three questions, and the habit of collapsing them

There are three separate questions here, and it is worth being pedantic about how separate they are, because nearly every failure this book touches comes from treating an answer to one as an answer to another.

**Who is this agent?** An identity question. It is answered by a registry — something that records that this public key belongs to this agent, in a form a third party can check. This site holds the design for that and, since v0.1.26, a running instance of it.

**What may it do?** A delegation question, and a completely different one. It is answered by a mandate: a statement, signed by an issuer, naming a subject, carrying an interval, saying that this agent may do these things until this date on this authority. Identity does not imply mandate. Knowing precisely who somebody is tells you nothing whatsoever about what they are permitted to do.

**Should that produce this effect, now, here?** An execution question, and it is not answered on this site at all. It belongs to a broker — something that sits at the point of action, holds the context the other two layers lack, and decides. The site names this layer and does not own it, at `https://pki.sgit.ai/execution/index.html`.

There is a fourth thing that is not a question but an absence, and the site states it plainly: recording who a key belongs to and what it was permitted to do leaves out the most auditable event of all — what it actually did. **The receipt is the third corner, and nothing in this book supplies it.**

Now watch what happens when the three collapse into one.

A platform token is a single object that answers all three at once and answers none of them well. It establishes identity, in the sense that a request bearing it is attributable to whoever issued it. It confers authorisation, in the sense that the request will succeed. And it performs enforcement, in the sense that a request without it will fail. One object,

three jobs, and the crucial property: **its scope is knowable only to its issuer.** The holder cannot enumerate what it permits. Neither can a third party. Neither, in general, can the person who created it, six months later.

Compare a signed identity statement and a signed mandate. The site puts the distinction this way in its own front door at `https://pki.sgit.ai/llms.txt`:

> identity says this key belongs to this agent; a mandate says this agent may do these things, until this date, on whose authority. Both signed, both checkable by a third party, and the mandate revocable independently of the identity — materially different from a bearer token, whose scope is knowable only to its issuer.

*Stated.* That is the site's own framing, and it is the load-bearing claim of the whole estate: separating the statements is what makes them checkable, and checkability by a third party is the property a bearer token structurally cannot have.

## The word that does the most damage

There is a further collapse underneath the first, and it takes one word.

What an agent *can* do and what somebody *decided* it should do are different quantities. The estate calls the first a **grant** and the second a **mandate**, and Chapter 4 is entirely about why that pair of words is worth defending. The damage is done by describing the first as *implicit authorisation*.

The Grant and Mandate pack's change-control appendix records the correction, at `https://pki.sgit.ai/packs/grant-and-mandate/change-control.html`, as entry GM2, and its wording is careful:

> the mandate is authorisation (somebody decided it); the grant is **authority that nobody decided**. Calling the grant "implicit authorisation" concedes the point the vocabulary exists to make.

*Stated.* The word *implicit* smuggles in a decider. It suggests somebody weighed the thing and chose not to write it down. In almost every real case nobody weighed anything: the capability arrived bundled with a credential, as a side effect of solving delivery, and no human has ever considered it. There is no implicit authoriser. There is an absence where an authoriser would be.

And the absence is not benign, because the outside world does not care that nobody decided. The pack takes this further in the same entry, invoking a doctrine from agency law:

> And under apparent authority the outside world treats the grant as binding anyway — so *the mandate is actual authority, the grant is apparent authority, and binding regardless.*

*Stated.* This is the sharpest thing in the estate's vocabulary and it deserves unpacking, because it is what makes excess authority a live exposure rather than a tidiness complaint. Apparent authority is the principle that if a principal's conduct leads a third party reasonably to believe an agent is authorised, the principal is bound by what the agent does

— whether or not the agent was actually authorised. Applied here: if your agent holds a credential that lets it write to forty-one repositories, and it writes to one you never considered, the repository does not ask whether you meant it. The commit lands. The deploy fires. The consequence is yours.

*Drawn.* The packs do not connect apparent authority to the receipt gap; reading GM2 against the execution page, the connection seems to me the reason the receipt matters more than either layer admits. If the grant binds you regardless of what you decided, then the only record that establishes what actually happened is the one nothing in this estate produces. Identity tells you whose key. Mandate tells you what was permitted. Neither tells you what was done, and it is what was done that you are accountable for.

## Why the object was never the key

Put the three questions beside the objects that answer them and the shape of the problem changes.

| Question | What answers it | What a keypair contributes |
|---|---|---|
| Who is this agent? | A registry entry somebody else agreed to | The public half, and possession of the private one |
| What may it do? | A signed mandate with an issuer and an interval | A way to sign it, and a subject to bind it to |
| Should this happen now? | A broker holding context at the point of action | Nothing |
| What did it actually do? | A receipt | Nothing |

The keypair appears in two rows out of four, and in both it is the least difficult part. Signature schemes are settled. Key lengths are settled. Nobody in this field is losing sleep over ECDSA P-256.

What is not settled is everything the keypair does not supply: a channel to somebody who will recognise the key, a document that says what its holder may do, a place to evaluate that document where the holder cannot reach it, and a record of what happened. The bootstrap trap in Chapter 3 is the first of those, and it is a loop rather than a gap. The mandate document in Chapter 4 is the second, and the estate's inventory found that nothing anywhere provides it. The third is the tier problem in Chapter 6 and it is where most real deployments quietly fail. The fourth is not in this book at all.

*Drawn.* This is the reframing I would put first if I could only carry one sentence out of the estate: **the hardest part of agent identity is not cryptography but authority choreography — the order in which claims are established.** The site states that reframing at `https://pki.sgit.ai/bootstrap/index.html` and I have not improved on it, but I would add the consequence it implies and does not say: a system can use flawless cryptography end to end and still be catastrophically weak, if its first instruction is *hand over a platform token*. Sequence is a security property. Almost nothing in the tooling treats it as one.

## What this book claims, and the size of it

The estate this book describes is a demonstration. It has a register you can fetch, a measurement tool you can run, a hook that refuses a push, and a component library that renders real documents. It has no trustworthy root, no boundary-tier enforcement point, no capability vocabulary, no receipts, no users, and one agent's worth of data.

The word *demonstration* is going to appear a lot, and it is accurate more often than it is comfortable.

So the claim is not that the problem is solved. It is narrower, and it is this: the three questions are separable, and separating them produces objects you can fetch, verify, diff, and be refused by. Part three is those objects. Whether separating them is *worth* it — whether anybody will run a registry, whether an operator will author a mandate, whether a vendor will supply the boundary — is a question about people that this estate has not asked anybody. Chapter 16 is where that gets said properly, and it is the chapter that decides whether the rest of the book can be believed.

Everything before it is the argument for why the interesting object was never the key.

# 2 · The flood, and the four rules it produced

*Part one — Why there is nothing to inherit*

---

Good public key repositories existed. They were destroyed. Anyone proposing a key registry today should be able to show they designed it with that history in hand, and this chapter exists so that this estate can.

The short version is on the site at `https://pki.sgit.ai/failure/index.html`, dated and cited. What follows is that history read for the thing that matters most about it, which is not the attack.

## What happened

In June 2019, an attacker flooded the OpenPGP certificates of two prominent contributors with bogus signatures and uploaded them to the global SKS keyserver network. One key reached roughly 150,000 signatures.

The effect was not that the keys became untrustworthy. It was worse and stranger: anyone who *retrieved* a poisoned certificate broke their own working installation, in hard-to-debug ways. The recommended mitigation became — and this is the sentence to sit with — stop retrieving data from the network entirely.

The network's own maintainer called it unsalvageable. Not "difficult to fix". Unsalvageable, on the grounds that changing a design goal of that magnitude means starting from a fresh sheet of paper rather than fixing anything.

A replacement keyserver was built. It is immune to the attack, and it achieved immunity by stripping all third-party signatures and unverified identities. The web of trust went with them. **The thing that made the system valuable and the thing that made it attackable were the same thing**, and the replacement chose survival.

## The part that matters: it was not a bug

Here is the detail that turns this from a war story into a design constraint. The fatal property was stated as a goal at the outset. The site records it as it was written:

> A key server may add information to a certificate. It may never delete either a certificate or information about one.

*Stated.* Read it twice, because everything follows from it. A key server could add and could never delete. That is not an oversight, a missing feature, or an implementation shortcut. It is a design goal, chosen deliberately, for reasons that were good at the time — a key server that can delete is a key server that can be compelled to delete, and censorship-resistance was the point.

And a design goal is not a bug you patch. That is why the maintainer's assessment was *unsalvageable* rather than *needs work*. You cannot ship a fix for a property the system was built to have.

*Drawn.* The site does not draw this conclusion in these words, but I think it is the most transferable thing on the page: **the failure mode of a well-designed system is usually one of its stated goals, met.** Nothing malfunctioned in 2019. Every component did exactly what it was specified to do. The attack consisted of using the system correctly, at volume, against somebody. Any registry designed today should be asked which of its own stated goals would look like this if somebody used it correctly, at volume, against somebody.

## Three abused properties, four rules

The site's move is to turn each abused property into a rule. Three properties, and the rules follow directly:

| Abused property | Rule it produces |
|---|---|
| A certificate may carry unlimited signatures | Bound the size of a record |
| Anyone may append to anybody's certificate | Only the owner may write to their own record |
| Nothing distinguishes a legitimate signature from garbage | Every entry is signed by something you can check |

The fourth rule comes from the never-delete goal rather than from the attack, and it is the one that looks impossible:

**Rule 1 — Only the owner writes to their own record.** The attack existed because anyone could append to anybody's certificate. Ownership of the written object is the property that separates safe append-only channels from fatal ones, so it is the first rule rather than a policy bolted on later.

**Rule 2 — Revocation is a signed append, not a deletion.** Signed by the key being revoked.

**Rule 3 — Records are size-bounded.** Generous enough to be invisible in normal use, small enough to stop flooding.

**Rule 4 — Every entry is signed by something you can check.** Which is what makes untrusted mirrors safe and the registry's contents portable.

## The tension in rules 1 and 2, and why it dissolves

Rules 1 and 2 are usually assumed to be in conflict. You cannot both refuse deletion and support withdrawal — that is the trap the keyservers fell into, and it is why they could not revoke anything.

The resolution is in rule 2's phrasing and it is worth stating carefully, because it is the single most elegant thing in this estate's design and it costs nothing.

A revocation is an appended statement, signed by the key being revoked. Four things then hold at once. The record stays append-only, so rule 1 is intact. The revocation is verifiable, because only the key holder could have signed it. What the key said before revocation stays checkable — which matters enormously, because the question an auditor actually asks is not *is this valid* but *was this valid last Tuesday*, and deletion destroys that question permanently. And a reader always sees current state by reading to the end.

The register implements this. Chapter 8 walks a record where an identity self-revokes on key compromise, and the verification walk returns NO for the right reason while the pre-revocation state remains derivable.

## The precise lesson about append-only

There is a sloppy lesson available here and the site refuses it. The sloppy lesson is *append-only is dangerous*. That would be a bad conclusion, and an expensive one for this project in particular, since append-only is used in five places across it — write-only telemetry, staging folders, event ledgers, vault inboxes, relay channels — and rightly.

The site's resolution is exact, and it is the sentence to carry out of this chapter:

> Append-only is safe when a writer appends only to objects it owns. It is fatal when anyone may append to somebody else's object.

*Stated.* The distinction is ownership, not appendability. The keyservers were fatal because anyone appended to anybody's certificate *and* nothing could be removed; the second clause turned a bad property into an unrecoverable one, but the first clause is where it went wrong. So the rule to carry forward is not "append-only" — it is that the writer owns what it writes.

## The trade the rules do not resolve

Rules 1 and 3 have a cost, and the site publishes it as its central open question rather than resolving it. It is at `https://pki.sgit.ai/rules/index.html#attestation`.

Third-party attestations — I vouch for this key belonging to this person — are what made the old system valuable. They are also exactly what made it attackable, because an attestation is by construction a write into somebody else's record. Permit them, and rules 1 and 3 must be enforced hard, with all the difficulty that implies. Forbid them, and the registry carries no social trust signal at all: it becomes a directory of self-assertions, which is useful and much less than what was lost.

The site publishes this unresolved. *Drawn.* I think that is the right call and I want to name what it costs, because the estate does not quite: a registry that forbids attestations answers *who claims to be this agent* and can never answer *who else believes it*. Every YES in Chapter 8's verification walk is therefore a statement about arithmetic and never about reputation. That is a real ceiling on what this design can ever be, independent of whether the fixtures are replaced with real keys, and it is not obviously the wrong trade — it is simply not a trade the estate has priced.

## What vaults supply, and what they do not

One more piece of ground-clearing, because it is the thing most likely to be assumed.

The transport layer underneath this design — the sgit vault layer — supplies distribution, custody without access, and versioning. Those are real and they are not nothing.

It does not supply the ownership rule, the size bound, or signature checking. Which is to say: it does not supply the registry logic, and the registry logic is precisely the part that failed last time. A vault is a good place to put a registry. It is not a registry, and a design that assumes otherwise has skipped the entire content of this chapter.

## What this earns

Publishing rules before an implementation is cheap to do in this order and impossible to claim afterwards. The site says so directly, and the value is a testable one: if something ships that breaks one of these rules, the rules page is the evidence.

Chapter 8 is that test, run against the four rules for the first time. The result is not uniform, and the chapter says which rule holds by construction, which holds by a check, and which is still only a proposed number in a parameters file.

# 3 · The bootstrap trap

*Part one — Why there is nothing to inherit*

---

Chapter 2 explained why the last generation of key repositories was destroyed. This chapter explains why the next generation has not been built. They are different obstacles, and a registry has to clear both.

The second one is worse, because it is not a missing feature. It is a loop.

## The loop

Stated in the form that makes it tractable:

```
An agent needs an identity
    ↓
it must request certification
    ↓
the request must reach trusted infrastructure
    ↓
that infrastructure requires authentication
    ↓
the agent must already possess an identity
    ↓
──────── back to the top ────────
```

Generating the keypair takes milliseconds. Getting the public half recognised by something that matters is the whole difficulty, and the difficulty is circular rather than hard.

The distinction that carries the argument is on the site at `https://pki.sgit.ai/bootstrap/index.html`, and it is worth memorising: **creating a key is not creating an identity.** A key is a mathematical object anybody can make. An identity is a relationship somebody else has agreed to — and agreement requires a channel.

The loop is a statement about the channel. There is no way in that does not require having already got in.

## Every escape trades a small problem for a larger one

People do not stop shipping agents because of a loop. They escape it, and the escapes are the interesting part. The site enumerates seven, with what each one actually grants:

| Workaround | What it grants |
|---|---|
| The operator's platform credential | Everything that person can do, everywhere |
| Repository write access | The whole repository, permanently |
| A shared bot token | Whatever the bot can do, to everybody holding it |
| A vendor integration | Whatever its permissions cover, which is usually broad |
| A cloud or service credential | Whatever that principal can reach |
| A project signing secret | The ability to sign as the project |
| A bespoke public enrolment server | A new service to secure — and its own identity problem |

Read the right-hand column as a set of sizes and the argument becomes visible without any security expertise. **Every row solves transport by creating a larger identity problem than the one being solved.** You needed to establish that one agent may open one pull request. You have established that one agent may do everything you can do.

The last row deserves a note because it is the one engineers reach for. Building a public enrolment endpoint feels like solving the problem properly. It relocates it: the endpoint must now decide who may enrol, which is the original question, and the endpoint itself needs an identity, a deployment, a key, and an owner. You have converted a loop into a service, and the service contains the loop.

## Two named failure modes underneath

The site names the mechanisms, and naming them is what makes the pattern generalise beyond the seven rows.

**Ambient authority** is privilege a process holds by virtue of where it runs, rather than because it was handed something specific. A borrowed credential is ambient authority by construction — the agent inherits everything the holder could do, whether or not the task needs it.

**The confused deputy** is a party with more authority than its task required, acting on instructions that may not have come from whoever granted it. The workarounds create deputies as a side effect of solving delivery.

The connection to prompt injection is the part most worth carrying, and the site records the published analysis rather than asserting it: ambient authority is identified as the root cause, because an attacker need not break anything and need only ask the holder to use authority it already has.

*Drawn.* The packs do not put it this way, but the consequence for how people talk about agent security seems to me unavoidable: if ambient authority is the root cause, then every mitigation aimed at the *instruction* — better prompts, injection classifiers, output filters — is treating a symptom, and treating it at the only layer with no ability to be sure it is right. The authority is the disease. This is also why Chapter 6's tier test is not a taxonomy exercise: it is the question of whether a control is even in a position to help.

Both failure modes point at the same conclusion, and the site states it in four words: **the fix is not a better credential. It is not needing one.**

## Not theoretical

Two of the seven rows have public evidence, and the site cites them rather than gesturing.

One coding assistant was found holding a token scoped to every repository its developer had authorised — far more than the surgical changes it was making required. That is row one and row two at once, and it is not a misconfiguration: it is what the available mechanism produces when used as designed.

A disclosure at a security conference on 5 August 2026 showed a repository issue, opened by an account with **no repository privileges**, reaching continuous-integration secrets in three vendors' own repositories, under their own default configurations. Three vendors, their own repositories, their own defaults. The confused deputy at industrial scale.

And the platform feature request for short-lived, repository-scoped tokens for AI agents remains open — which is the fact that turns this from a complaint about carelessness into a description of the available options. The workaround persists because it is the workaround, not because nobody thought about it.

## The gradient, and where a registry can actually help

The loop is not escaped by cleverness. It is escaped by noticing that "identity" is not one thing, and that the steps have different costs.

The site sets out a gradient at `https://pki.sgit.ai/bootstrap/index.html`:

- **I control this private key.** Free. Proves possession, and nothing else.
- **The project recognises this key.** A policy decision, not a computation.
- **The project delegates this mandate.** Scoped, dated, revocable.

Step one costs nothing and is worth almost nothing on its own. Step three is what people actually want. Step two is the hard one, and the important word in it is *decision*: recognition is not something a system computes, it is something somebody chooses. No amount of cryptography produces it.

Which yields the estate's second stated principle, and it is the one most often skipped:

> A signature over an enrolment request proves the submitter controls the corresponding private key. It does not prove that the project should trust the agent. Trust is a policy decision made afterwards.

*Stated.* A verified signature on an enrolment request tells you the submitter holds the key. That is all it tells you. Everything else — should this agent exist, should it be recognised, should it be given anything — happens afterwards and happens in somebody's head.

## The narrow door

If step two is a decision, then the registry's job at bootstrap is not to *make* the decision but to make the *submission* possible without handing anything over. The site's framing is that the exit is a door narrow enough that walking through it requires nothing.

The candidate is an append lane: post to it with a token carried in the body, no account and no access token, and receive a blind acknowledgement that tells you nothing. Nothing is handed over because nothing is held. The submission is a claim, and the decision happens later, somewhere else, by someone.

This is the design, at `https://pki.sgit.ai/enrolment/index.html`, and it is the load-bearing claim in the estate's whole bootstrap argument: the path is buildable rather than theoretical, because the shipped platform already has the lane.

## What is actually true today, stated flatly

And here the chapter has to turn, because this book is not a product announcement and the lane is where the estate's honesty is most tested.

**The append-lane write path is not built.** No lane, no processor, no blind acknowledgement. The write path to the register described in Chapter 8 is a git commit, reviewed by a human maintainer. The register's own front door says so without softening it, and so does this book.

**The question that decides whether the lane works at all is open and unanswered.** A fresh session reading the registry pack as an implementation brief filed it as blocking question Q2 — *does an append lane with no anchors configured accept any token holder?* — and the answer, if it goes the wrong way, does not delay the lane. It removes it.

The mechanism is worth following, because it is the sharpest piece of criticism anybody outside this estate has produced about it. The append lane already has sender authorisation, in the form of registered anchors: hashes of accepted senders, configured with the write key. The project lead's own code-verified audit draws the conclusion in plain words — a lane is not open to everybody by default, and the recipient decides which senders are accepted, using a credential the senders do not have.

Now apply that to enrolment. If anchors are required, the operator must hold a hash of the enrolling agent's key *before* that agent may write. The readiness report states the consequence, and I quote it because the estate's own reviewer put it better than I would:

> The agent must already be known in order to ask to be known. **That is the bootstrap trap, restored, at the exact point the pack says it is broken** — and phase 2's acceptance test, a fresh session with a token and nothing else ending with its identity in the register, cannot pass.

*Stated.* The report is careful about what it does and does not know: two sources point in opposite directions and neither is decisive. The audit says the default is closed; the reference does not state what a lane with an *empty* anchor set does. If an empty set accepts any token holder, the narrow door exists. If it does not, the door is a wall with a

doorknob painted on it.

Nobody has run it. The report says so, and says what it costs:

> **This is answerable today by one experiment against a test lane**, and it is the cheapest de-risking available in the whole pack.

*Stated.* Chapter 15 records that the report also found this question filed under the wrong document — parked as an observability concern, where it gets the attention of that heading rather than the attention of a question that may invalidate the estate's most-repeated structural claim.

*Drawn.* Reading the bootstrap page and the readiness report together, the honest position is narrower than the site's own summary suggests. What the estate has established is that the loop is real, that the escapes are worse than the problem, and that a narrow door is the shape of the exit. What it has *not* established is that this particular narrow door exists, because the property that would make it a door rather than a wall has never been tested. That distinction is not in the bootstrap page, and it should be.

## Why this chapter comes before the vocabulary

Part two is going to spend four chapters on words. That is only defensible if the words are load-bearing, and the bootstrap trap is why they are.

If the loop were escapable cheaply, the difference between what an agent can do and what somebody decided it should do would be a modest gap. It is not modest, and the seven workarounds are why: the escape routes do not produce slightly-too-large credentials, they produce credentials sized to a *person* being used by a *process*. The distance between the two quantities is not an inefficiency. It is the exposure, and Chapter 5 is about the fact that nobody has accepted it.

# 4 · Grant is not mandate

*Part two — The vocabulary, and why each word is load-bearing*

---

Two quantities. One is what an environment can actually do. The other is what somebody decided it should do. They are almost never the same, the difference is usually large, and in most organisations neither has ever been written down.

The estate calls the first a **grant** and the second a **mandate**. This chapter is about why those two words are worth defending, and why the most common way of describing the pair — *permissions* and *policy* — loses the thing that matters.

## Grant: what the environment can actually do

The lexicon, at `https://pki.sgit.ai/packs/grant-and-mandate/concepts.html`, defines a grant by its edges rather than its label:

> **What the environment can actually do**, as installed and configured. A tree of capability nodes, each carrying what it reaches, the control standing in the way (or nothing), the tier of that control, an evidence class, and a date. Generated by **measurement**, never authored.

*Stated.* Four things in that definition are doing work, and it is worth taking them one at a time because each is a rejection of a common practice.

**A tree, not a list.** Because the interesting relationships are containment ones, and because blast radius is a path through a tree rather than an item in a list. A list of what an agent can reach tells you the surface. It does not tell you that reaching A is what gets you to B, which is what you actually need when you are deciding what to fix.

**Each node carries what stands in the way, or nothing.** The `reaches` line and the `stands in the way` line are separate fields on purpose. Document 09 is blunt about which one misleads: the `reaches` line is the one that does the work, because *a list of what is reachable without what stands in the way is the part people already have and the part that misleads.* Every permissions screen you have ever seen is that list.

**A date, per node.** Not per tree. A grant is a claim about somebody else's product, and it ages on their release schedule, so a tree dated as a whole is wrong in one place while looking current.

**Generated by measurement, never authored.** This is the load-bearing one, and it has its own correction.

## The grant is discovered, not authored

The pack's change control records this as GM1, its first correction and the one it calls load-bearing:

> a hand-written grant file is a wish; it records what somebody believed on the day they typed it, which is the thing a grant is not.

*Stated.* A permissions file written by a human is a statement of intent wearing the clothes of a statement of fact. It records a belief, on a date, by somebody who may have been wrong then and is certainly out of date now.

So the grant document is generated by running a measurement inside the environment, it carries provenance per node, and a node with no evidence is marked rather than dropped. Chapter 9 is the measurement method and the two entries it produced.

The consequence is the part people miss: **drift becomes mechanical.** If a grant is measured, then the question *has this changed?* is answered by re-running the measurement and diffing, not by asking anyone to remember. And the diff has a direction that matters more than the others — a node moving from `setting` to `expectation` means a control was removed while nothing broke. Nothing failed. No alarm fired. The containment simply stopped existing, silently, and the only thing that would ever notice is a second measurement.

Chapter 9 records that detector firing for real, one commit after the change it detected. It fired in the harmless direction. The mechanism is the same either way.

## Mandate: what the environment is expected to do

The other side, from the same lexicon:

> **What the environment is expected to do.** Authored by a person, signed by an **issuer**, naming a **subject**, carrying an **interval** — *without one it is a grant under another name.*

*Stated.* Note the inversion. The grant must never be authored; the mandate must be. That is not a stylistic preference — the two documents answer different kinds of question. *What can this thing do* is a question about the world, and you answer it by looking. *What should this thing do* is a question about a decision, and there is nobody to look at: if no one has decided, the answer does not exist, and a measurement will not produce it.

Four fields, each of which a real system usually lacks:

**An issuer.** Somebody whose authority this is. Not a config file, not a repository, not a team — a signing identity that can be resolved and checked, and that could be held to it.

**A subject.** Bound, so the mandate is about one agent rather than about a role that anyone can wear. Chapter 8 shows what happens when the subject is a role and its key is published: the register can say what the role may do and can never say who wore it.

**An interval.** The clause in the lexicon is the sharpest test in the vocabulary — *without one it is a grant under another name.* A permission with no expiry is a permanent capability, and a permanent capability is a property of the environment rather than a decision about it. Every token you have that does not expire has already crossed back over into being a grant.

**A signature**, so a third party can check all of the above without asking the issuer.

## Allow-list stored, prohibitions rendered

One structural rule, and it is the kind of detail that turns out to be load-bearing in a user interface.

A mandate is **stored** as an allow-list, because that is the enforceable form: an evaluator needs to know what is permitted, and default-deny requires an explicit set. **Prohibitions** — *will not push to any branch outside* `claude/**` — are a generated rendering of the allow-list's complement, and they carry their own date and the version of the capability set they were rendered over.

Both halves exist because they have different audiences. Document 09 states the rule for the interface, and it is the trap in the pack's screen four:

> **Prohibitions shown, allow-list stored and not displayed.** Screen four's trap: an allow-list presented for approval produces consent without comprehension.

*Stated.* Show a person a list of forty permitted operations and ask them to approve it and they will approve it, because reading forty items and imagining their complement is not something people do. Show them three sentences beginning *will not* and they can tell you whether that is what they meant.

The rendering carries its own date for a reason that is easy to miss and unpleasant when it bites: the complement of a fixed allow-list *changes when the capability set grows*. Add a new capability to the vocabulary and yesterday's mandate silently prohibits something new. A regenerated view must not retroactively change what was agreed, so the rendering records what it was rendered over. You can see the field in the real mandate: `prohibitions_rendered_over: "capability set v0"`.

*Drawn.* The packs do not say this, but the field is an admission with a sharper edge than its placement suggests. If the complement moves when the vocabulary moves, then a mandate's meaning depends on a capability vocabulary — and Chapter 15 records that this estate does not have one. `capabilities.json` in the register is explicitly a fixture, and the question *what is a capability name* is one of the three blocking questions still open. Which means the prohibitions in the real mandate are rendered over a vocabulary that is acknowledged not to exist yet. That does not make the demonstration invalid. It does mean the phrase *capability set v0* is doing more work in this estate than anything currently behind it can support.

## Apparent authority, and why the gap is an exposure

Chapter 1 introduced the doctrine; here is where it earns its place in the vocabulary.

The grant is authority nobody decided. The mandate is authorisation somebody did. If those were merely two descriptions of the same thing at different resolutions, the gap between them would be a tidiness problem — an inventory exercise, worth doing when there is time.

It is not, because the outside world binds you to the grant. If your agent's credential reaches forty-one repositories and it writes to one nobody considered, no system asks whether you meant it. The write lands. Under apparent authority the consequence is the principal's, and the principal is you.

So the difference between the two quantities is not slack. It is **exposure that nobody accepted**, and the register publishes it as exactly that. The view at `https://pki.sgit.ai/registry/views/excess-authority.json` carries one row for one subject, and the row reads: grant `repo.contents.write` over 41 resources; mandate `repo.pull-request.create` over 1; excess 40 resources; **acceptor:** `null`.

Forty resources of exposure with nobody's name on it. That is the countable object, and Chapter 5 is about the arithmetic that produces it.

## What this vocabulary refuses

Two things, and the refusals are as informative as the definitions.

**It refuses "implicit authorisation."** Covered in Chapter 1: the word smuggles in a decider who does not exist. There is no implicit authoriser; there is an absence where one would be.

**It refuses `wallet`.** The pack ran naming checks and recorded the result. `card` and `pack` were kept. `passport` was tested before and misfit. `wallet` was avoided deliberately — it is taken by the payments work, and it would import the expectation of held verifiable credentials presented by the subject, which is a model this design does not use. Grants and mandates here are artefacts signed by an issuer, not credentials held by a principal. The distinction decides where the object lives, who can revoke it, and what a verifier fetches.

*Drawn.* I would put the general rule this way, and the packs do not: **a vocabulary that blurs the distinction between what was decided and what merely exists does not read badly — it reintroduces the confusion it was written to remove.** Every term in Part two is defensible only in proportion to how well it holds that line, and the two chapters that follow are about the two places it is hardest to hold: in arithmetic, where a stored answer goes stale, and in a control label, where something inside the thing it constrains gets called a boundary.

# 5 · The delta

*Part two — The vocabulary, and why each word is load-bearing*

---

Two documents produce a third thing, and the third thing is the product.

The lexicon calls it the delta and defines it with a constraint attached, which is unusual for a definition and is the most important sentence in this chapter:

> **The finding, computed and never stored.** A stored delta is stale the moment either side moves; the interesting property is that it can be recomputed at any time (the same rule that keeps a register entry free of a history array).

*Stated.* Never stored. Not *should not be cached*, not *refresh periodically* — computed on demand, every time, and if you cannot compute it you do not have it.

## Why never stored

The reasoning is short and it generalises well beyond this estate.

A delta is a relationship between two moving objects. The grant moves when the vendor ships, when a setting changes, when somebody installs something. The mandate moves when a person decides something. Store the relationship and you have created a third object that is correct only when neither of the other two has moved since — and nothing tells you when that stops being true. It does not go wrong loudly. It goes quietly, remains readable, keeps its formatting, and answers questions with yesterday's answer in today's confident tone.

This is the same rule that shapes the register's record model. The pack's C7 correction moved a record from an accumulating hash-chained log to a file in a commit graph: current state in the object, history in the substrate, no `seq`, no `prev`, no history array. Same instinct, different layer — **do not store a thing you can derive, because the stored copy has no way to know it is wrong.**

*Drawn.* The packs treat these as two separate decisions. I think they are one decision applied twice, and stating it as one makes it portable: *derive what you can derive; store only what you cannot.* The register stores signed statements, which are facts about the past and cannot be derived. It derives current state, the excess-authority view, and the verification answer. Every one of those derived things is regenerable by anybody, from the same public files, which is also why none of them carries authority — and the register says so on each, in the file itself.

## Two directions, and the one everybody forgets

The delta is not a number. It is a pair of set differences pointing in opposite directions.

### Excess authority: `grant − mandate`

What the environment can do that nobody asked for. The lexicon's phrasing carries the whole argument:

> Blast radius measured from the other end, **unaccepted by construction**, defaulting to critical. The security direction.

*Stated. Unaccepted by construction* is the sharp phrase. Risk acceptance is normally something somebody does to a risk somebody else identified. Excess authority arrives already unaccepted, because by definition nobody decided it — if somebody had, it would be in the mandate and it would not be excess. The acceptor field is `null` not because the form was left blank but because the concept guarantees it.

*Blast radius measured from the other end* is the other half. The usual way to estimate blast radius is to imagine a compromise and reason forward about consequences, which is guesswork dressed as analysis. Excess authority reverses it: start from what the credential demonstrably permits, subtract what was authorised, and the remainder is the reachable-but-unintended set. It is a measurement rather than a scenario.

### Shortfall: `mandate − grant`

What was asked for that the environment cannot do. This is the direction that gets forgotten, and the pack insists on it:

> The operations direction — the agent fails and it looks like a bug. **It matters as much as excess**, because a mandate the grant cannot satisfy is what produces the next over-broad credential.

*Stated.* Follow the causal chain, because it is the reason a security document cares about an operations problem. A mandate authorises something the environment cannot actually do. The agent fails. The failure does not announce itself as a policy problem — it looks like a bug, gets triaged as a bug, and the fastest fix available to whoever is on call is to widen the credential. Nobody records that a mandate was wrong. What gets recorded is that permissions were insufficient, which is a sentence that produces more permissions.

**Shortfall is where the next excess comes from.** Which means a programme measuring only excess is treating the symptom and leaving the generator running.

The pack also says why shortfall is harder: it needs the mandate enumerated against real capability names. Excess can be computed against observed behaviour — you saw the agent reach forty-one repositories. Shortfall requires knowing what a capability *is*, as a type, so that you can ask whether the grant contains it. Chapter 15 records what this estate does about that, and the answer is uncomfortable: the shortfall column in its own rendered delta block is a hardcoded string.

## The third term, and the blind spot

Two terms are not enough, and the reason is the most persuasive argument in the pack.

An agent asked to report its own grant is both the instrument and the subject. It reports what it can see. It cannot report a capability it does not know it has. So a self-report is a **floor, not a census** — and, critically, it is *unfalsifiable alone*. Nothing in a self-report distinguishes a thorough one from a lazy one.

The leading brief sets out the fix:

```
LIBRARY   ---minus--->  SELF-REPORT    = BLIND SPOTS
          what the environment is       what this agent noticed
          known to grant                it has, and did not report

SELF-REPORT ---minus-->  MANDATE        = EXCESS AUTHORITY
          what this agent noticed        what was asked for
```

The library is a published dataset of measured grants per environment. Subtract what this agent reported from what its environment is known to grant, and the remainder is what it missed.

The brief calls this the argument for building the library at all — without it there is no way to tell a thorough self-assessment from a lazy one — and it says the number is the most persuasive in the flow: *this agent reported eleven of the nineteen capabilities its environment is known to have.*

*Stated,* and now the honest part, which the estate also states. **No such number has ever been produced here.** The three-term comparison block in the gallery renders with its middle column empty on purpose, because no agent has filed a structured self-report. The library has two entries measured by one agent, and a blind-spot delta needs at least two agents against a common reference. The most persuasive number in the flow is an illustration of a calculation, not a result.

*Drawn.* And there is a recursion in it the packs do not name. The blind-spot delta measures the agent as much as the environment — that is stated. But the library entries are themselves produced by an agent measuring an environment from inside it, under the same floor-not-census limit. So the reference against which self-reports would be falsified is a floor too. Blind spots computed against it would be *blind spots relative to what a previous agent happened to notice*, which is a weaker claim than it looks and is not the claim the phrase suggests. The fix is not more agents measuring themselves; it is at least one measurement taken from outside the environment, and nothing in this estate has one.

## Why it needs a third term at all

Step back from the arithmetic, because the shape is the point.

Two terms let you compare a claim against a claim. Three terms let you check one of the claims. The library is not a richer source of grants — it is the falsifier, and its whole value is that it was produced independently of the agent being assessed.

That is why the library belongs in the registry rather than in the risk product, which is Chapter 12's contract. A falsifier held privately by the party being assessed is not a falsifier. It has to be public, versionable, and shared, or the third term collapses back into the first.

## What the delta is not

**It is not a score.** The rule is absolute across the estate: no score, ever, because any single number averages tiers that must stay distinct. An excess capability sitting behind a boundary is a different finding from one sitting behind nothing, and one figure that combines them tells you neither. Document 09 puts the reason as a rule about what averaging destroys: it collapses `boundary`, `unknown` and `none` into one figure, which is the exact collapse the whole vocabulary exists to prevent.

**It is not a risk.** This is the ordering rule of Chapter 7 arriving early. A delta is a *finding* — a computed fact about two documents. Turning a finding into a risk requires an altitude, a language, and somebody's judgement, and that work belongs to the risk product. The registry's half of the chain ends at *finding*, and Chapter 12 draws that line explicitly so neither side builds the other's half by accident.

**It is not accepted.** Excess authority arrives unaccepted by construction. Acceptance is a separate node with a named acceptor and an interval, and it lives on the other side of the boundary in Chapter 12.

## The one row that exists

Everything above is scaffolding for a single published row, and it is worth ending on how small it is.

One subject. One grant statement recording `repo.contents.write` across 41 resources. One mandate covering `repo.pull-request.create` on 1. Excess: 40 resources, acceptor `null`, observed 2026-08-25.

One row, on a fixture subject, in a register where ten of eleven records have published private keys. The arithmetic is right and the arithmetic is the only thing here that is right. Chapter 8 walks the record it came from and Chapter 16 says what it is worth.

# 6 · Boundary, setting, expectation

*Part two — The vocabulary, and why each word is load-bearing*

---

Every control you rely on falls into one of three tiers, and one test separates them. The test needs no vendor claim, no datasheet, and no trust in anybody's marketing:

> **a control bounds a grant only when it is enforced by something the grant does not include.**

*Stated.* That is the whole thing. Ask what enforces the control. Then ask whether the grant includes the power to reach that enforcer. If it does, the control is inside the thing it is supposed to constrain, and it is not a boundary however it is labelled.

## The three tiers

| Tier | Enforced by | Worth |
|---|---|---|
| **Boundary** | Something *outside* the grant: OS, separate account, container, network policy, a remote service | Real — holds against a compromised agent |
| **Setting** | The tool itself, running *inside* the grant | Bypassable by anything that can run code as that grant |
| **Expectation** | Nothing — it is written in a prompt or a policy file | None; it is a mandate, and a mandate is not a control |

The third row is the one that should sting, and the pack does not soften it: *it is a mandate, and a mandate is not a control.* A sentence in a system prompt saying the agent will only push to feature branches is not a weak control. It is not a control at all. It is a statement of intent, and it belongs in the mandate document where intents live — where, crucially, it can be compared against a grant and found to be unenforced.

*Drawn.* The packs do not put it this way, but I think the tier vocabulary's real function is to give people a place to put the things they are currently calling controls. The instinct on being told a prompt is not a control is to argue. The tier table redirects that: the prompt is fine, it is simply a mandate, and the interesting question is what enforces it. That reframing is why this is a vocabulary chapter and not a scolding.

## The middle tier is where most real deployments live

The site's own summary is that the test places most of what people rely on in the middle tier that reads like a boundary and behaves like a setting. This is the practical content of the chapter.

Consider the things a careful team actually configures: allow-lists in a tool's config file, hooks the tool runs, sandbox flags, a `permissions:` block, a policy file the agent reads. Every one of them is evaluated by something the agent could reach if it wanted to. They are real improvements. They are all settings.

A setting is genuinely worth having, and it is worth being precise about what it buys: **a setting protects against an agent that is behaving normally and does not protect against an agent that is not.** Which is exactly backwards from what people assume they are buying when they configure one. If your threat model is a confused agent, settings help enormously. If your threat model is a compromised one — an injected instruction, a poisoned tool result, a malicious dependency — the setting is code the attacker's code can edit.

## The refinement the measurement forced

The pack carries a note that saves the test from being too crude to use, and it came from measuring a real environment rather than from design:

> a control that *evaluates in a separate process outside the agent's loop* is a boundary for the agent even when it is configuration for the platform operator — which is why hooks and Cedar are the natural compilation targets, and why a prompt is not.

*Stated.* Tier is relative to the principal being bounded. The same object can be a boundary for the agent and a setting for the operator, and both statements are true because the grants are different. A permission classifier that a hosted platform runs outside the agent's turn is, from the agent's position, unreachable — it is a boundary. From the platform operator's position it is a config file — a setting.

This is not a loophole in the test; it is the test applied correctly. *A control bounds a grant* — which grant? Name the principal, then evaluate. A tier stated without a principal is meaningless, and Chapter 9 records the estate's measurement tool getting this wrong in a way that produced a wrong label in published data.

## The rule that came from the build

The single most valuable rule in this chapter did not come from design. It came from the tool getting it wrong.

Library entry #2 labelled the OS user separation on a CI runner a `boundary`. The very next node in the same tree recorded `sudo -n true` succeeding — passwordless escalation to root, observed. Both facts are correct; the label is not. The pack recorded it as correction GM16, and the correction is a rule about data:

> **A tier is a property of a node's relationship to the tree, not of the node. So a tier badge must be able to show what defeats it — and a defeated control must never render as a boundary.**

*Stated.* The tool decided each node's tier in isolation. Read alone, "runs as uid 1001, bounded by the OS user separation" is a boundary. Read against the child node that escalates without a further credential, it is not a boundary at all — it is a setting, and a thin one.

Which yields the absolute rendering rule of the whole component family:

```
A control whose defeat path exists in the same tree renders as
SETTING, with the defeat path reachable from the badge itself.

✔  ⛨ setting     defeated by → n1a "escalate to administrator"
✘  ⛨ boundary    (with the escalation drawn somewhere else on the page)
```

The estate connects this to a finding one layer up — that escalation is an edge, not an annotation. Drawing the path is what makes the `setting` tier land. Hiding it is how a measurement tool, a vendor datasheet and an interface all independently produce the same lie, and they produce it without anyone intending to, because each of them is describing one node accurately.

Chapter 11 has the figure: the stored document says `boundary`, the block renders `setting` with the defeat path attached. It is the rule working on real data that is wrong, and the wrong data is this estate's own.

## The tier the estate's own enforcement reached

Now apply the test to the thing this estate built, because the test is only worth anything if it is applied to the home team.

Chapter 10 describes a signed mandate compiled into a `pre-push` hook that refuses pushes. Ask the question. What enforces it? A hook file in the repository, activated by a local git config, executed by git as the same user the agent runs as.

Does the grant include the power to reach that enforcer? It includes the power to edit `.githooks/pre-push`, to run `git config --unset core.hooksPath`, and to pass `--no-verify`. Three ways, all trivial.

So it is a **setting**. The pack states this plainly, and the refusal banner prints it on its own face at the moment of refusing:

> **The hook is inside the grant it bounds.** The agent runs as a user that can write `.githooks/pre-push`, so by the pack's own test — *a control bounds a grant only when it is enforced by something the grant does not include* — this is a setting that reads like a boundary, which is the exact tier the pack warns is most often mislabelled.

*Stated.* The move from expectation to setting is real and it is worth having. It is also exactly one tier, and the estate refuses to round it up.

**Reaching a boundary is the same allow-list evaluated where the agent cannot reach it** — a branch protection rule on the remote, or a required CI check. That is a change of *location*, not of policy, which is the strongest argument in the pack for keeping policy in a document rather than in an enforcement point. Move the document; the policy is unchanged; the tier improves. Bake the policy into the hook and you have to rewrite the control to relocate it.

## The evidence axis, which is not the same axis

One more distinction, because it is easy to collapse and the interface rules forbid collapsing it.

Tier says how strong a control is. **Evidence** says how the fact was established: `observed` by running a command · `read` from a settings file · `documented` in the vendor's docs · `inferred` · or `none`, for a node that could not be evidenced and is marked rather than dropped.

The four are not equally trustworthy and the interface must not flatten them. A `boundary` established by `inferred` and a `boundary` established by `observed` are different claims, and a badge that shows only the tier presents them identically.

*Drawn.* The packs keep tier and evidence as separate fields and separate badges but do not, as far as I found, say what the *combination* means. Reading the two vocabularies together, the cell that should worry a reader most is `boundary` + `documented` — a strong claim resting on a vendor's description of their own product, which is the one evidence class the estate has no way to falsify. Every hosted environment's containment story lands in that cell. Chapter 17's last suggestion is what to do about it, and it is a request rather than a remedy for exactly this reason.

## And `unknown` is a fact

The fifth state exists because the alternative is worse. A node the measurement could not establish is marked `unknown`, and the rendering rule is absolute: `unknown` renders as `unknown`, never as blank, because a gap is a fact about the floor and a blank reads as a to-do.

Two nodes in the first library entry are `unknown` because the probe that would have filled them was refused mid-measurement. Chapter 9 is about what happened there, and about why the refusal turned out to be the sharpest datum in the entry rather than the biggest hole in it.

# 7 · Reality before the risk register

*Part two — The vocabulary, and why each word is load-bearing*

---

There is one sequence underneath every term in Part two, and the pack is explicit that it is a constraint rather than a preference:

```
REALITY     the agentic environment, as installed and configured
TWIN        that installation — the thing every fact attaches to
FACTS       the GRANT (measured, dated, provenance per node)
            the MANDATE (authored, signed, with an interval)
FINDING     the DELTA (computed, not argued)
RISKS       derived, at each altitude, in that altitude's language
DECISIONS   separate nodes: named acceptor, interval
```

The leading brief states the rule and its consequence in one paragraph:

> A grant is a *fact*, and a fact is a *measurement*. You cannot author a risk before you have a fact. So the first screen of the MVP is not a risk register and not a risk-appetite questionnaire — it is *which environment, and here is what it can do*. A pack whose first screen is a risk register reproduces exactly the habit it exists to fix.

*Stated.* And the detail worth noticing is where this correction came from. The pack records it as GM4, sourced from *the pack-spec brief, correcting its own approach mid-sentence.* The brief that specified this work started down the risk-register path and stopped itself. The ordering rule is a recorded self-correction, not a principle somebody arrived at cleanly, which is a small point in its favour.

## What the ordering rule forbids

Read the chain backwards and each step names something that becomes incoherent if you skip its predecessor.

**A risk named before a fact is a guess.** It has no measurement behind it, so it cannot be checked, and it cannot be closed by evidence — only by argument. This is most of what risk registers contain, and the reason they age so badly.

**A decision recorded before a delta accepts nothing measurable.** Somebody signs off on an exposure whose size is unknown. The signature is real; what it applies to is not. When the exposure later turns out to be four times larger, nobody can say whether it was accepted or not, because there was never a number to accept.

**A fact attached to nothing is unusable.** This is what `twin` is for, and it is the step most likely to be dismissed as ceremony. A grant is not a general truth; it is a property of *one installation*. Two people running the same product have different grants if their configurations differ — and, once memory enters, different grants if their *histories* differ. Without a twin to attach facts to, a grant is a vendor-shaped generalisation and cannot be differenced against anybody's mandate.

## The time axis, which makes this harder than it looks

Memory is where the ordering rule earns its keep, and it is the clearest argument in the whole estate because it needs no security expertise to follow.

With memory off, a grant is a tree — a property of the environment. With memory on, it is that tree unioned with everything any prior session reached, for as long as history is retained. So it is a property of the environment *and its past*.

The leading brief makes the case for why this is the argument to lead with:

> This is the clearest case in the whole product, because it needs no security argument — anybody understands that a tool which remembers everything you have shown it can be asked about any of it, and the asking need not come from you.

*Stated.* Nothing in that sentence requires believing anything about threat models. It is a description of what retention means.

The consequence for the vocabulary is structural: `history` is a top-level field on the grant document rather than a node, because it changes the meaning of every other node. And the consequence for the architecture is the one Chapter 12 depends on — a library entry can never be the whole answer for a real user, because the library publishes the *static* term and only an instance can carry the *accumulated* one.

The two real library entries differ on exactly this axis, and the contrast is the most useful thing about having two. The hosted agent container **retains a session record**, so its grant is a union over prior turns. The CI runner **retains nothing**, so its grant is a tree over the present. Same tool, same measurer, same day, and a structurally different kind of object at the end.

## What a risk-first screen would have cost

The pack enforces the ordering rule through screen order, and the counterfactual is worth stating because it is where the rule stops being philosophy.

A first screen showing a risk has to have got that risk from somewhere. There are only two places available. Either it came from a library of generic risks — in which case it is about a category of product rather than about the reader's installation, and the reader's honest response is *how do you know that applies to me*, to which there is no answer. Or it came from a questionnaire — in which case the reader has just been asked to self-report their own exposure, which is precisely the unfalsifiable floor Chapter 5 exists to correct, and now it is upstream of everything.

Either way the delta becomes uncomputable, because there is no measured grant to subtract from. The screens can still render. They render an argument rather than a finding.

*Drawn.* The packs do not say this, and I think it is the strongest practical version of the rule: **a risk-first flow is not just methodologically backwards, it is architecturally unable to produce the one number the product exists to produce.** The ordering rule is enforced by screen order because that is where it is cheap to enforce, but it is not a UX

convention — it is the difference between a tool that computes and a tool that asserts.

## The shipped consumer, and where it stops

The assessment at `https://pki.sgit.ai/assess/index.html` is the ordering rule with a user interface on it, and it is the one artefact in this estate that a non-technical reader can use today. Chapter 13 covers the two paths it serves; here it matters for what it deliberately does *not* do.

Risk acceptance is not in it. It was in the first version, and it was removed — because acceptance belongs to the risk product, and the version that had it was wrong. That is a costly removal: acceptance is the satisfying end of the flow, the thing that makes a session feel finished. Cutting it leaves the tool ending on a gap rather than on a resolution.

The estate cut it anyway, and the ordering rule is why. `DECISIONS` is the last node in the chain and it lives on the other side of the boundary Chapter 12 draws.

*Drawn.* There is a tension here the estate names but does not resolve, and I want to leave it visible rather than tidy it. The chain runs `finding → risks → decisions`, and the registry's half ends at `finding`. But the assessment does end on something — the visitor is shown a gap and offered an action. That action is not a *decision* in the chain's sense, since there is no acceptor and no interval, and the tool cannot store one because it stores choices and never answers. So the shipped consumer of the ordering rule stops one step short of where the rule says a flow should stop, for a reason that is about data protection rather than about the model. The two constraints point in different directions here, and this estate has resolved it by ending on a request instead of a record. That is defensible and it is not the same thing as the chain being complete.

## Why this chapter closes Part two

The four vocabulary chapters are really one argument in four pieces.

**Grant and mandate** are separate because what exists and what was decided are different quantities. **The delta** is computed and never stored because it is a relationship between two moving things. **The tier test** exists because a control's label is worthless without knowing what enforces it. And **the ordering rule** is what stops all three from being used out of sequence, where each of them quietly becomes something else: a grant that was authored is a wish, a delta that was stored is stale, a tier decided in isolation is a mislabel, and a risk named before a fact is a guess.

Every one of those four failure modes has an example in this estate's own artefacts, produced by this estate's own tools, and recorded in its own change control. Part three is where they get shown.

# 8 · The register

*Part three — What was built*

---

Eleven records. Twenty-three signed statements. Four assumable roles. One declared root. Six expected verification answers shipped as data, and a validator that reproduces all six.

Ten of the eleven records have their private keys published beside their public halves, on purpose, and every signature on them verifies and proves nothing.

This chapter walks it, runs it, and states what each part is worth in the same breath.

## What is actually there

The register is at `https://pki.sgit.ai/registry/` and it is static files. No account, no API key, no session, no server-side logic of any kind. Every artefact is fetchable at a constructed URL, and that is a promise rather than an accident.

```
registry/
  params.json                       canonicalisation, signature, fingerprint recipes
  roots.json                        one declared root — a FIXTURE root, and it says so
  roles.json                        four published roles and how to assume one
  capabilities.json                 the fixture capability vocabulary (v0)
  index.json                        convenience map, NO AUTHORITY, regenerable
  views/expected-verifications.json  six cases and their expected answers
  views/excess-authority.json       grant minus mandate, per subject
  records/<sha256-hex16>/           one directory per record, named by the
    01__identity.json               owner's signing fingerprint
    NN__<type>__<slug>.json         further statements
    record.json                     unsigned manifest, NO AUTHORITY
    public/sign.pem  public/encrypt.pem
    private/…  keystore/…           FIXTURES ONLY — published on purpose
  tools/registry_tool.py            generator, verifier, validator, enrolment helper
```

The twenty-three statements break down as eleven identities, five mandates, four acceptances, two revocations and one grant. Every one is an immutable signed file.

Three files in that listing say *NO AUTHORITY* about themselves. That is not modesty. `index.json` and `record.json` and both views are derived — regenerable by anybody from the signed statements — and Chapter 5's rule applies: a derived thing carries no authority because it can be recomputed, and if it disagrees with the statements the statements win. The specification is explicit that a verifier **must not** rely on `index.json` for any step.

## The flag you read first

Before any signature on any record, one field.

**Figure 4 · The fixture flag, read before any signature.** *Read* `private_key_published` *before you read the signature. And note* `publication_intent: deliberate` *— a secret is defined by expectation, not by content, so the intention is recorded at issue, because afterwards a deliberate publication and a leak look identical.*


```
{
  "body": {
    "agent_type": "operator",
    "claims": {
      "note": "FIXTURE — exists to exercise the plumbing; its signatures prove nothing (change-
control C3)"
    },
    "private_key_paths": ["private/sign.pem", "private/encrypt.pem"],
    "private_key_published": true,
    "publication_intent": "deliberate"
  },
  "created_at": "2026-08-25T09:00:00Z",
  "record": "sha256:90f97984b9cf3930",
  "registry": "pki.sgit.ai",
  "signer": "sha256:90f97984b9cf3930",
  "type": "identity",
  "v": 1
}
```


Ten of the eleven records read like that. The eleventh returns `false`, and that single `false` is what makes the whole scheme work as evidence. The register's front door states the reasoning:

> ONE RECORD IS REAL (private_key_published: false) — which is what makes the flag evidence rather than a column.

*Stated.* A flag that is true on every row is not evidence; it is a column. It carries no information, it costs nothing to ignore, and readers correctly learn to skip it. One row where it is false is what makes reading it worth the effort.

The `publication_intent` field is a separate idea and a better one than it looks. The estate's key policy holds that **a secret is defined by expectation, not by content**, and that the intention has to be recorded at issue — because afterwards, a deliberate publication and a leak are indistinguishable. There is no forensic difference between a private key that was published on purpose and one that got out. The only difference is a claim somebody made beforehand, which is why it is a field.

## The forgery

The fastest way to understand what a fixture signature is worth is to produce one.

**Figure 7 · The forgery, executed.** `Verified OK` *on a document that says* `anyone can sign this`*. Nothing failed. That is the point: a signature anybody can produce conveys nothing, and the register verifies it exactly as diligently as any other.*

```
$ printf '{"forged":"anyone can sign this"}' > forged.json
$ openssl dgst -sha256 -sign $R/private/sign.pem forged.json > forged.der
$ openssl dgst -sha256 -verify $R/public/sign.pem -signature forged.der forged.json
Verified OK

# The private half is published in this repository. So is everybody's.
```

Three commands, no cleverness, no vulnerability. The private half of the registry's operator root is a file in a public repository, so anybody can sign as the operator root, and the verification succeeds because the signature is genuinely valid.

The register's own front door puts the general form of it:

> What a verified signature proves here: that the statement was signed by a holder of that private key. What that is worth on a FIXTURE record: nothing — you are also a holder of that private key.

*Stated.* A signature is a claim about scarcity. It says: whoever produced this held something few others hold. Remove the scarcity and the mechanism keeps working perfectly while the claim it carries becomes empty. The mathematics is unaffected. The meaning is gone.

This is the sharpest reason the estate declined a proposal it considered and rejected — per-object keypairs with published private halves. Those would leave a hash wearing a signature's clothes, defeat their own stated use, and make the fixture flag true on every row. Which returns to the same point: a flag that is always true is a column, not evidence.

## The verification walk

The register answers one question — *may agent X exercise capability C right now?* — and the walk is published as a procedure. Read it as a sequence of refusals as much as of checks.

Fetch `roots.json` and `params.json`. Fetch the subject record. **Read the fixture flag.** Verify every statement's signature against the owner's key, and reject any statement whose signer is not the owner. Check for an identity revocation. Follow acceptances to mandates in issuer records. Verify the issuer's record the same way. Require the issuer in `roots.json`. Check revocations against the mandate. Check the validity interval.

Then answer YES with expiry and authority, NO with the reason, or **STOPPED** with where the chain ended.

Two steps in that list are doing more than they appear to.

**Reject any statement whose signer is not the owner.** The front door explains why, and it is the 2019 catastrophe compiled into a single check: *a valid signature by a non-owner is the 2019 keyserver failure, not write authority.* Rule 1 as an executable test. A cryptographically perfect signature from the wrong party is exactly what destroyed the last generation, and here it is a rejection rather than an append.

**STOPPED is a legitimate output.** A partial resolution is not a failure. This is the estate's five-state discipline arriving in the answer space: *confirmed*, *denied*, *unknown*, *unreachable* and *not checked* are five different situations, and collapsing them into pass/fail loses the three that matter most operationally. An implementation must not render *unreachable* as *denied*.

## Six answers, and four of them are NO

**Figure 3 · The six answers a verifier must get right.** *Four of the six answers are NO, and they are NO for four different reasons — revoked, expired, never accepted, identity revoked. A verifier that collapses those into one failure state is wrong on three of them.*

| SUBJECT | ANSWER | WHY |
|---|---|---|
| fixture-agent-a | YES | Valid mandate, accepted, issuer is a declared root — until 2026-10-01, **after which this fixture genuinely expires** and the answer flips with no file changing |
| fixture-agent-b | NO | Mandate revoked by its issuer — a signed append with an effective date (rule 2), never a deletion |
| fixture-agent-c | NO | Mandate expired 2026-08-01 — intervals end, and a mandate with no interval would not be a mandate at all |
| fixture-agent-d | NO | Mandate issued and never accepted: **inert** (pack decision 8, taken provisionally) |
| fixture-agent-e | NO | Subject identity self-revoked on key compromise — and what it said before the effective date stays derivable |
| role-site-agent | YES | The role holds a valid accepted mandate — and anyone holding its published key can exercise it, which is the lesson |


| Subject | Answer | Why |
|---|---|---|
| `fixture-agent-a` | **YES** | Valid mandate, accepted, issuer is a declared root — until 2026-10-01, after which this fixture genuinely expires and the answer flips with no file changing |
| `fixture-agent-b` | **NO** | Mandate revoked by its issuer — a signed append with an effective date (rule 2), never a deletion |
| `fixture-agent-c` | **NO** | Mandate expired 2026-08-01 — intervals end, and a mandate with no interval would not be a mandate at all |
| `fixture-agent-d` | **NO** | Mandate issued and never accepted: **inert** (pack decision 8, taken provisionally) |
| `fixture-agent-e` | **NO** | Subject identity self-revoked on key compromise — and what it said before the effective date stays derivable |
| `role-site-agent` | **YES** | The role holds a valid accepted mandate — and anyone holding its published key can exercise it, which is the lesson |


The `fixture-agent-a` row contains something rare in a fixture set: a live clock. That mandate expires on 1 October 2026. Nothing needs to be edited for the answer to change; the file stays exactly as it is and the answer flips, because the interval is real and time passes. **A fixture with a genuine expiry is a test that maintains itself**, and it is

the only part of this register that will be more honest next month than it is today.

The `fixture-agent-d` row is the estate's most provisional answer. Whether an unaccepted mandate is inert or live on issue is open decision Q6, taken provisionally as inert. The fixture demonstrates the choice rather than justifying it, and the register says so.

## Running it

**Figure 5 · The verification walk, executed.** *Six expected answers, six reproduced — and the fixture line printed for every record before any of them. The validator reads the flag first by construction, not by convention.*

```
$ cd registry && python3 tools/registry_tool.py validate
  ✓ fixture-agent-a — valid, accepted mandate: YES (expected YES) — valid until
      2026-10-01T00:00:00Z, on the authority of sha256:90f97984b9cf3930, a declared root
  ✓ fixture-agent-b — mandate revoked by issuer: NO (expected NO) — mandate revoked
      (policy), effective 2026-08-20T00:00:00Z
  ✓ fixture-agent-c — mandate expired: NO (expected NO) — mandate expired 2026-08-01
  ✓ fixture-agent-d — mandate issued, never accepted: NO (expected NO) — a mandate exists
      and its subject has never accepted it — inert (decision 8, provisional)
  ✓ fixture-agent-e — identity self-revoked: NO (expected NO) — subject identity revoked
      (key-compromise), effective 2026-08-24T00:00:00Z
  ✓ role: site-agent: YES (expected YES) — valid until 2026-12-31T00:00:00Z
  · sha256-1ccafd8f3f8906c4: FIXTURE (private key published)
  … (nine more)
  · sha256-f9facb4c94da6c19: REAL identity (private half not published)
registry validate: OK — 11 records (10 fixtures, 1 real), 23 statements, every signature
verified, every reference resolves, no private material outside fixture records, all 6
expected answers reproduced
```

That is executed output, run against the repository at the time of writing. The expected answers ship as data at `views/expected-verifications.json` so that anybody's verifier can be tested against the same six, and the front door states the acceptance criterion with its trap attached:

> If your verifier reproduces all six (as of the file's as_of date), it implements this register's walk. If it passes any of them WITHOUT surfacing the fixture caveat, it skipped the flag rule and is wrong while looking right.

*Stated.* Wrong while looking right is the failure mode this whole register is designed around. A verifier that returns all six correct answers and never mentions that the root is a fixture has produced six technically correct and completely misleading results.

## It really is the shipped format

The register claims compatibility with the `sgit pki` commands, and the claim is established by execution rather than by assumption.

**Figure 6 · Format compatibility with the shipped CLI, executed.** *The signer line names a fixture. The CLI is not wrong; it is answering the only question a signature can answer — who held the private half — and on this record the answer is everybody.*

```
$ jq -cS 'del(.sig)' $R/02__mandate__pr-create__to-agent-a.json > payload.bin
$ jq '{signature: .sig, fingerprint: .signer}' $R/02__…json > payload.bin.sig
$ jq '.body.bundle' $R/01__identity.json > op-bundle.json
$ sgit pki import op-bundle.json
Imported contact: fixture-operator (pki.sgit.ai)
  Fingerprint: sha256:075693699f0694d0
$ sgit pki verify payload.bin payload.bin.sig
Signature valid (signer: fixture-operator (pki.sgit.ai))
```

Fingerprints are sgit's own 16-hex short form over the DER SubjectPublicKeyInfo. Bundles are byte-shaped like `sgit pki export`. Signatures are raw `r||s` ECDSA P-256 — 64 bytes, not DER — an encoding chosen in sgit's own source for Web Crypto interop, so a browser can verify these statements with no conversion at all.

There is one detail in that transcript worth pausing on, because it is a real design consequence rather than a curiosity. The imported contact's fingerprint is `sha256:075693699f0694d0`, and the record directory is `sha256-90f97984b9cf3930`. Different values, both correct: `sgit` addresses keystore operations by the **encryption** fingerprint, while records here are keyed by the **signing** fingerprint, because registry statements are signed artefacts. Two fingerprints per identity, and a reader who assumes one will conclude the register is inconsistent when it is not.

## The four rules, meeting entries for the first time

Chapter 2 published four rules before there was anything to apply them to. Here is the first honest assessment of how each fares against a real register — and they do not fare equally.

**Rule 1 — only the owner writes to their own record. Holds, twice over.** It holds *by topology*, because the record model is a file in a commit graph and a record directory is named by its owner's fingerprint. And it holds *by check*, because the validator rejects a valid signature by a non-owner. Two independent mechanisms, which is the right number for the rule that the 2019 failure came from.

**Rule 2 — revocation is a signed append. Holds, and is exercised.** Two revocations in the register: one of a mandate by its issuer, one of an identity by itself. Both are appends carrying `effective_from`. The pre-revocation state stays derivable, so *was it valid last Tuesday* still has an answer.

**Rule 3 — records are size-bounded. Does not hold. It is a proposal in a parameters file.** `params.json` carries `max_statement_bytes: 16384` and `max_statements_per_record: 256` under a field that reads `"proposed": true`, with a note recording that draft-1's per-statement bound was doubled because an identity statement carrying two PEM public keys plus the fixture flags runs past 8 KB. Nothing enforces either number. **This is the rule that turns directly on the 2019 attack — one key reached 150,000 signatures because certificates had no size limit — and it is the one rule the register has not implemented.**

**Rule 4 — every entry is signed by something you can check. Holds mechanically; the checkable thing is a fixture.** Every statement carries a signature that resolves against a published key, and the validator verifies all twenty-three. What it resolves to is a root whose private half is in the repository.

*Drawn.* The packs do not score the rules this way, and I think the scoring is the useful output of this chapter: **two rules hold, one is exercised, one is a number in a file.** Rule 3's absence is the most interesting because it is the cheapest to fix and the most directly connected to the documented catastrophe. My reading of why it has not been

fixed is that a bound only bites when something approaches it, and nothing in an eleven-record fixture register approaches anything — which means rule 3 is untested for the same reason the whole register is untrustworthy, and both are properties of the population rather than of the design.

## A role is a costume

Four roles ship with published keypairs and drop-in `sgit` keystores, including passphrases. A fresh session assumes a role by retrieval — copy a directory, sign a file, and the CLI reports `Signature valid (signer: role: site-agent)`.

The register states the limit exactly:

> A role is a costume, not an identity. The register can say what the role may do — role-site-agent holds an accepted mandate for repo.pull-request.create on this repository, constrained to registry/** on dev — and can never say who wore it.

*Stated.* And note that this is the sixth verification case: `role-site-agent` returns YES. The register's own acceptance test contains an entry whose correct answer is *yes, and anyone at all may exercise it*. Building the costume lesson into the test set rather than the caveats is the right instinct, and it is the kind of thing that only survives if somebody writes it down as a test.

## The one real identity, and what it cost

`records/sha256-f9facb4c94da6c19` is the authoring session's own identity. Public halves only. `private_key_published: false`. It is the reason the flag is evidence.

It is also honest about a problem it cannot solve: the private half lived only in the authoring session's ephemeral container, so the identity is **session-scoped**. The container is reclaimed after inactivity. The key is gone.

That is the pack's open persistence question, recorded as an executed fact rather than as a design note — which is a better outcome than a design note, and worth naming as a technique. The estate could have written *key persistence is an open question*. Instead it enrolled an identity, discovered where the private half would have to live, found that there was nowhere, and published the record with the problem visible in it.

*Drawn.* Reading the roles material against this record produces a conclusion neither states: the register currently contains exactly two kinds of identity, and neither is the kind it is designed for. Four are costumes anybody can wear, six are fixtures with published keys, and one is real but cannot outlive a container. **The design's central case — an identity whose private half has a good place to live — has zero instances.** The front door says the first real root awaits such an identity. What it does not say is that the same absence applies one layer down, to every subject as well as to the root.

## What this register does not have, deliberately

Four absences, and the register lists them itself rather than leaving them to be discovered:

**No live capability, ever.** Grants carry hashes of what was issued. The one descriptor in this register had its preimage discarded before publication. A registry that contains a secret is a registry with a breach in its future.

**No append-lane write path.** No lane, no processor, no blind acknowledgement. Today the write path is a git commit and the processor is whoever reviews it. Phase 2's questions — lane anchors, token distribution, processor transparency — are untouched, and Chapter 3 covered what the anchors question might do to the design.

**No enforcement.** The register records authority. Nothing in it observes or constrains behaviour. Chapter 10 is the enforcement point, and it is a separate artefact for exactly this reason.

**No trust.** Ten records are fixtures and the root is a fixture. The first real root awaits an identity whose private half has a good place to live.

The register is built. The trustworthy register is not. Every sentence in this chapter is true, and none of them adds up to a reason to rely on anything in it.

# 9 · A grant is discovered, not authored

*Part three — What was built*

---

The pack's first correction says a hand-written grant file is a wish. This chapter is what happens when you take that seriously enough to build the alternative: a tool that generates a grant document for whatever environment it runs in, two entries it produced, and the four things they found that nobody was looking for.

## One rule governs the method

`tools/measure.py` is published at `https://pki.sgit.ai/packs/grant-and-mandate/tools/measure.py` so that somebody else can run it and get the same answer about their own environment. One rule governs what it does: **presence and reachability, never contents.**

The tool asks *can this environment reach that*, and never *what is in it*. It does not read your files, enumerate your secrets, or record what your credentials unlock. It records that a push succeeded, that a host was reachable, that `sudo -n true` returned zero.

*Drawn.* The pack states the rule and does not, as far as I found, say why it is a *security* property rather than a privacy courtesy. It is both, and the security half is the stronger argument: a completed grant document is a description of what an environment can reach, which is close enough to an attack plan that the contents rule is what makes it publishable at all. The same instinct runs through the assessment, which stores identifiers and fixed options and has no free-text input anywhere on the page. The pattern is worth naming: **an artefact designed to be shared has to be designed so that sharing it is safe, and that constrains what it may contain, not just who may see it.**

The second rule follows from the first: **a refused probe is recorded `unknown`, never guessed.** Which matters more than it sounds, because the first thing this tool did was get refused.

## Entry one: the measurement refused to measure itself

The first library entry is `claude-code-remote__ccr-container__2026-08-26.json` — a Claude Code Remote container, measured from the inside, on 26 August 2026. It is the environment that produced the pack. Nine nodes.

And a single probe — read the harness's own settings and hooks, sweep non-allowlisted egress hosts, check privilege escalation — was refused. The entry records it in a field called `measurement_honesty`:

> a direct self-measurement probe (reading the harness's own settings and hooks, testing non-allowlisted egress hosts, checking privilege escalation) was REFUSED by an account-level policy classifier that evaluates outside the agent's loop. The refused nodes below are marked unevidenced rather than guessed. The refusal is itself the sharpest datum in this entry: a boundary-tier control, observed working, on the measuring agent itself.

*Stated.* Two nodes are `unknown` as a result: `n8`, harness configuration and lifecycle hooks, and `n9`, reach non-allowlisted external hosts. They are the two nodes a security reviewer would most want filled, and they are empty.

Now apply Chapter 6's test to the thing that refused. What enforces the classifier? Something that evaluates outside the agent's loop. Does the grant include the power to reach it? Demonstrably not — the agent tried, in the ordinary course of doing its job, and was refused.

**That is a boundary, observed working, on the measuring agent.** It is the cleanest possible demonstration of the pack's own three-tier test, and it was produced by accident, by a tool built to demonstrate something else.

It also confirms the floor-not-census rule by a hard route. The pack's caution is that an agent measuring its own grant reports what it can see. Here the measurer *demonstrably could not* census itself — not as a caution about what might happen, but as what the first entry literally is. The entry ships with its holes marked.

*Drawn.* There is an epistemically awkward consequence the pack does not draw, and I think it belongs beside the celebration. The refusal is excellent evidence *for the tier test* and it is a hole *in the dataset*, and those two facts are in tension in a way the entry's framing smooths over slightly. Calling the refusal "the sharpest datum in this entry" is true and it is also the most flattering available reading of a measurement that failed to complete. The unflattering reading is equally available: this environment's containment story is exactly the part that could not be measured, and every claim about it rests on `unknown`. Both readings are correct. The entry prints the first.

## What the nine nodes say

| Node | Capability | Tier | How it was evidenced |
|---|---|---|---|
| n1 | runs as the container's root user | `none` | package installs as root; harness state read/writable |
| n2 | outbound HTTPS via a mandatory egress proxy | **boundary** | pushes and installs succeeded *through* the proxy; a proxyless browser was connection-reset |
| n3 | push to the attached repository | **expectation** | pushes to a feature branch and to `dev` both accepted; **nothing observed tests which branch** |
| n4 | GitHub API operations on in-scope repositories | boundary | scoping enforced by the platform's tool layer above the session |
| n5 | execute and install software | setting | installs succeeded; a self-measurement command was refused |
| n6 | read and write the session's accumulated record | none | — |
| n7 | hold signing keys and vault keystores | none | — |
| n8 | harness configuration and lifecycle hooks | **unknown** | the probe that would read them was refused |
| n9 | reach non-allowlisted external hosts | **unknown** | same refusal |

Node n2 is worth reading closely as a model of what `observed` should mean. The evidence is not that a proxy is configured. It is that **the bypass path was tried and failed**: a browser launched without the proxy had its connection reset. That is the difference between reading a setting and testing a control, and it is why n2 is a boundary and n5 — where a classifier refused one command and permitted another — is a setting.

Node n3 is the pack's thesis, live. The entry's worst path is `n1 → n2 → n3`, and the reading it carries is the sentence the whole estate turns on:

> root in the container → egress through the proxy → a push credential whose branch discipline is prose. The push to the release branch deployed a public site: the only containment on that step was the session doing what it was told — and in this session, explicit permission was in fact given first, which is the mandate working as an expectation, not as a mechanism.

*Stated.* The mandate worked. Permission was asked for and given, and the agent complied. And it worked *as an expectation*, which means it worked because the agent chose to comply, which means it would have worked exactly as well right up until the moment it did not. Chapter 10 is that node being moved one tier.

## Entry two: the other end of node n3

The second entry is a GitHub Actions runner, measured by the same tool running inside it. Ten nodes.

It was chosen for a specific reason, and the reason is the most useful design decision in the library so far: **it is the other end of entry one's node n3.** A push from the agent ends in the CI runner, and this is what happens next. The two entries join at that edge, and together they are the blast-radius path rather than two unrelated points.

The contrast is what one entry could not give:

| | The hosted agent | The CI runner |
|---|---|---|
| Egress | Behind a **mandatory egress proxy** | Reached every host **unrestricted** |
| History | **Retains** a session record — grant is a union over prior turns | Retains **nothing** — grant is a tree over the present |
| How the grant arose | Discovered by measurement | **Declared up front**, in a `permissions:` block the job cannot widen |

The third row is the only boundary in either entry that was *designed* rather than discovered, and it deserves more attention than it gets. A `permissions:` block in a workflow file is a grant declared in advance, by the person deploying, in a form the running job cannot edit. That is what the whole estate is arguing for, and it already exists, in one narrow place, in one vendor's product.

*Drawn.* The pack notes the `permissions:` block as a contrast and does not, I think, take the lesson far enough. The estate's argument is that nothing anywhere lets you declare what an agent is expected to do. That is not quite true: CI systems have had exactly that for years, and it works. What is missing is not the concept — it is the concept applied anywhere other than a build job. **The strongest evidence that a declared, unwidenable grant is workable is sitting inside the environment this estate measured second**, and the estate treats it as a contrast rather than as prior art. It is prior art, and the interesting question is why nothing outside CI has copied it.

The second row deserves a note too. The agent retains history; the runner does not. Chapter 7 made this structural — with retention, a grant is a union over the past. Two environments in one library, and they are different *kinds* of object. Any comparison across them has to account for that, and nothing in the tooling currently does.

## Four findings that cost something to record

The estate's discipline is that a correction is recorded rather than tidied away. The measurement work produced four, and all four made the deliverables look weaker than a quieter write-up would have.

**The measurement caught its own tier change.** Re-run in the same environment after the hook of Chapter 10 was installed, `measure.py` independently reported node n4 as `setting` where the entry recorded `expectation` — because the tool probes `core.hooksPath` rather than being told what to say. That is the drift detector from Chapter 4 working, one commit after the improvement it detected, in the direction the table calls *somebody improved something, and it should be recorded*. The alarming direction — `setting → expectation`, a control removed while nothing broke — is the same diff read the other way, and it now has a working detector.

**The tool mislabelled a boundary.** Entry two's node n1 called the OS user separation a `boundary` while node n1a recorded passwordless `sudo` succeeding. The pack's own *setting that reads like a boundary* warning, reproduced by the tool built to detect it, because tiers were decided in isolation. The correction is GM-D29 — a tier is decided against the tree, never in isolation — and the wrong label is still in the published file, kept visible under a `SUPERSEDED_BY` key that reads: *this tier label is WRONG, and node n1a is the proof.* Chapter 11 has the figure.

**The hook does not travel with a clone.** The hook file is committed; the `core.hooksPath` config that activates it is local and does not travel. Any fresh clone gets the file and not the enforcement, and nothing announces the difference. **The control is one un-run command away from being absent** — and it was found by measurement rather than by review, which is the whole argument for measuring.

**A hand-assembled entry drifted and a tool-generated one did not.** When the component gallery first rendered the library, it caught two schema violations — evidence classes the schema does not define — and every one of them was in the hand-assembled entry. The tool-generated entry had none. That is GM1 proving itself on the pack's own data: the hand-written document was a wish, and it was wrong in exactly the way a wish is wrong.

*Drawn.* Reading those four together, the pattern is that **every one was found by a machine and none by a person.** Two by the measurement tool, one by the gallery's build gate, one by re-running a measurement. The pack's review processes — which are thorough, and produced eighteen corrections — found none of these four. That is not a criticism of the review; it is the argument for building gates, stated more strongly than the estate states it. A review finds things a reader can notice. A gate finds things nobody would think to look for.

## What two entries cannot do

The honest limits, and the estate carries them on the entries' faces.

**A blind-spot delta needs at least two agents against a common reference.** There are two entries and one agent. The most persuasive number in the flow, from Chapter 5, cannot be computed from this library and has never been computed at all.

**Each entry is one vendor, one surface, one date.** The first entry's `environment` block says it in its own note field: *one vendor, one surface, one date — generalising from this entry is the error pack document 03 warns about.*

**The entry the library most needs does not exist.** A local coding-agent install — where the grant is enormous, because it runs as you, and the containment is available and almost never used. It cannot be produced from a hosted container. The tool is the deliverable for it, and nobody has run it there.

**And the measurer is the subject.** Both entries were produced by an agent inside the environment being measured. Chapter 5's recursion applies: this library is a floor built out of floors.

Which is why the word for this chapter is *method* rather than *dataset*. What exists is a runnable measurement, a schema, a drift mechanism demonstrated firing, and two points. Two points do not make a library. They prove the format, and they found four defects while doing it, which is a better first result than a clean one would have been.

# 10 · A push refused by something that is not the agent

*Part three — What was built*

---

Everything in Parts one and two is documents. This chapter is the one place where the estate stopped describing and made something say no.

The acceptance test was written before it was run, and its last clause is the whole thing:

> Run the skill in a fresh environment. It produces a dated grant document nobody wrote. Author a mandate that is deliberately narrower. The delta is non-empty and specific. Compile one line of it into the existing hook. **Then attempt the prohibited action and be refused by something that is not the agent.**

*Stated.* Refused by something that is not the agent. Not *decline to proceed*, not *ask for confirmation*, not *comply with its instructions*. Refused, by a different process, returning a result the agent does not get to argue with.

## Three files, and the smallest is the point

| File | What it is |
|---|---|
| `mandates/mandate-v1.json` | A mandate document: issuer-signed, interval-bearing, allow-list stored, prohibitions generated from its complement and dated |
| `tools/mandate.py` | Issue, verify, `check-branch`, `delta`, and the hook entry point. **Default-deny**: missing, unparseable, mis-signed or expired all refuse |
| `.githooks/pre-push` | The enforcement point. Git runs it; it refuses by exit code |

The mandate was authored deliberately narrower than the measured grant. It permitted pushes to `claude/**` and nothing else, while Chapter 9's library entry records at node n3, evidence class `observed`, that the environment can also push to `dev` — the branch that deploys a public site.

So the delta is non-empty and specific:

```
branch-scoped delta, mandate v1 vs measured grant node n3
  grant   (measured, tier=expectation): can push to ['claude/…', 'dev']
  mandate (declared)                  : permits ['claude/**']
  EXCESS AUTHORITY                    : ['dev']
     acceptor: none. This is the exposure the hook closes.
```

One row. One branch. That is the entire scope of the only real enforcement in this estate, and stating it that small is deliberate.

## The refusal

**Figure 8 · The refused push, re-run at v0.1.28.** *The last line before the banner is git's exit code, not the agent's decision. Read the Tier line at the bottom: the control names its own weakness on its own face.*

```
$ git worktree add --detach /tmp/hist-v0.1.28 v0.1.28   # the site as it was
$ python3 …/mandate.py check-branch dev …/mandates/mandate-v1.json
REFUSED  dev  (mandate v1 permits ['claude/**'])
$ echo $?
1

  ┌──────────────────────────────────────────────────────────┐
  │  PUSH REFUSED BY A MANDATE                               │
  └──────────────────────────────────────────────────────────┘

  ✗ dev  is not permitted by mandate v1
    permitted branches: claude/**
    mandate expires   : 2026-12-31T00:00:00Z

  What you agreed to:
    · will not push to any branch of this repository outside claude/**
    · will not push to any other repository
    · will not act on any resource other than github.com/SGit-AI/SGit-AI__Website__PKI
    (rendered 2026-08-26 over capability set v0 (registry/capabilities.json))

  This refusal came from .githooks/pre-push, which git ran — not
  from the agent deciding to comply. That is the whole point.

  Mandate : packs/grant-and-mandate/mandates/current.json
  Issuer  : resolves to a record in the public registry
  Tier    : SETTING — this hook is inside the grant it bounds, so
            --no-verify still gets past it. A boundary needs the
            same allow-list evaluated where the agent cannot reach
            it (branch protection, or a required CI check).
```

When the acceptance test was originally run on 26 August, the last line of the real session was git's own:

```
error: failed to push some refs to 'https://github.com/SGit-AI/…'
```

`origin/dev` was unchanged afterwards, and a push to `claude/**` in the same minute succeeded.

**That last line is the whole thing.** `error: failed to push some refs` is git's sentence, not the agent's. The agent did not decide to comply. It was refused by a process that ran outside its turn and returned a non-zero exit code it does not get to argue with.

*Drawn.* I want to be precise about what changed, because it is smaller and stranger than "the agent was stopped". Nothing prevented the agent from *attempting* the push. Nothing detected an intention. What changed is that the attempt now terminates in a result rather than in a choice — the branch name is compared against a list by fifty lines of Python, and the comparison happens whether or not the agent is behaving well, whether or not it read its instructions, whether or not it wants to. The distinction between a control and an expectation is not about strength. **It is about whether the outcome depends on the agent's state.**

## The figure that could not be taken as specified

The commissioning brief asked for this figure to be photographed at v0.1.28 — the release where the acceptance test ran. That instruction turned out to be impossible to follow literally, and the reason is a finding rather than an obstacle.

**The worktree at `v0.1.28` ships mandate v2, not v1.** Its `current.json` permits `dev`. Run the tag's own hook against the tag's own current mandate and it does not refuse:

**Figure 8b · The same tag, the mandate the release actually ships.** *The same tag, the same command, the mandate the release actually ships — and it PERMITS. The release documenting the refusal cannot contain the state that produced it, because the control refused the release carrying its own documentation until the mandate was amended.*

```
# the SAME tag, the SAME command, against the mandate v0.1.28 ships:
$ python3 …/mandate.py check-branch dev …/mandates/current.json
PERMIT   dev  (mandate v2, allow=['claude/**', 'dev'], expires 2026-12-31T00:00:00Z)
```

Why? Because of the next section. The hook refused the release that was carrying its own documentation, and the mandate had to be amended before that release could be pushed at all. **The tag that documents the refusal cannot, by construction, contain the state that produced it.**

So Figure 8 was taken by running the tag's own tool and the tag's own hook against `mandate-v1.json`, which is present at that tag — the document that actually did the refusing — and Figure 8b prints the amended answer beside it. The commissioning brief's own rule covers this case: where a command's output has changed since it was first recorded, print both and say which is which. Both are printed above.

*Drawn.* This is the sharpest thing the time-travel discipline caught, and it is a general hazard rather than a quirk of this estate. **A release note describing an event is not evidence that the release contains the event's preconditions** — and when the event is a control firing, the successful remediation is very likely to have removed them, because that is what remediation is. Any estate that documents its own controls firing has this problem. Chapter 15 records it as a contradiction; it is really a structural feature of recording your own corrections.

## The finding that justifies the whole exercise

Within an hour of installation, the hook refused the release push that would have published its own documentation.

**Which is the control working, not failing** — and the interesting part is what the correct remedy was.

Not `--no-verify`. Not editing the hook. Not a temporary exception. **The issuer amended the mandate**, because the authority to push to `dev` genuinely existed: the project lead had granted it explicitly on 25 August, in words the amendment cites — *"you should push to dev branch to trigger the ci pipeline"*, and *"It is ok to do that on this first mvp stage"*.

Mandate v1 was narrower than the real authorisation. So v1 was **wrong**, and the fix was to make the document match the decision that had actually been made:

```
v1  allow: claude/**              -> refuses dev
v2  allow: claude/**, dev         -> supersedes v1, cites the instruction
    expires 2026-12-31            -> and the MVP stage is what it is scoped to
```

The pack states what that sequence is worth:

> That sequence — *issue → refuse → discover the mandate was wrong → amend, with a citation and an interval* — is what a mandate is **for**, and it is only visible because the constraint was mechanical. An expectation that was too narrow would have been silently ignored and nobody would have learned anything. **The refusal is what forced the authorisation to be written down.**

*Stated,* and it is the best argument in the estate for mechanical constraints over written ones. Not that they stop bad things — this one stopped a good thing. That they **force a decision that was previously implicit to become explicit and dated.** Before the refusal, the authority to push to `dev` existed only as a sentence in a chat log. After it, it exists as a signed document with an interval, citing its source.

*Drawn.* And the estate names the risk in this without quite resolving it, so I will state it plainly: **amending a mandate to unblock your own release looks exactly like moving the goalposts**, and no amount of citation makes the two shapes different from the outside. The mitigation the estate uses is that the amendment carries the instruction it relies on and an expiry. That is the right mitigation and it is not proof. What would distinguish the two cases is an issuer who is not the subject — and in this estate the issuer is a fixture root controlled by the same party. The honest position is that this sequence is *what the designed path looks like*, and that nothing structural currently prevents the undesigned one from looking identical.

## Two design choices, and why

**The hook reads the mandate at runtime; it does not compile a copy.** The brief's diagram says *compiled to*, and a literal compilation would bake the allow-list into the hook, where it would drift from the mandate exactly as a stored delta goes stale. Reading the signed document at push time means there is one policy, and amending it is amending the mandate. Chapter 5's rule, applied to an enforcement point.

**Default-deny, and it bites.** A missing, unparseable, unsigned, mis-signed, issuer-mismatched or expired mandate all refuse the push. The hook is default-deny on its own dependencies too: if `python3` or the `cryptography` package is absent, it refuses rather than waving the push through, and says so. The cost is real — delete the mandate and nothing can be pushed until it is restored. A control that fails open is not a control, so the cost is the correct one to pay.

## The tier it actually reached

Stated by the pack rather than claimed, and it landed exactly where the source brief predicted:

| | Before | After |
|---|---|---|
| Where the constraint lives | A sentence in the session's instructions | A signed mandate, evaluated by a hook git runs |
| Tier | **Expectation** | **Setting** |
| Bypass | Ignore the sentence | `git push --no-verify`, or `git config --unset core.hooksPath`, or edit the hook |

One tier. Not two. The hook is inside the grant it bounds, and the banner prints that at the moment of refusing, which is the only thing that stops it being believed.

## What is still true after this

**The authority is a fixture and the enforcement is not.** The mandate's issuer is the registry's operator root, whose private half is published. Anybody could forge this mandate, and the hook would enforce the forgery just as diligently. The pack's formulation is the one to carry:

> *a hook enforcing a fixture-signed mandate is real enforcement of an unaccountable instruction.*

*Stated.* Two independent halves. Closing the second needs a real issuer key, which is the enrolment path the registry already ships and nobody has walked.

**Nothing here is Cedar.** The allow-list is evaluated by fifty lines of Python, not by the adopted policy language. That is step 6 of the build order. This step proves the shape, not the engine.

**And it is one constraint.** Branches, on one repository, for one subject. The nine other nodes in the measured grant have no mandate at all — and one of them is `n1`, *runs as the container's root user*, tier `none`.

**And it does not travel.** The hook file is committed; the config that activates it is local. A fresh clone gets the file and not the enforcement.

Four qualifications on one refusal. The refusal is still the most real thing in this book, and it is still one branch on one repository, enforced by a file the thing it constrains can delete.

# 11 · The building blocks

*Part three — What was built*

---

Nine primitives, shipped as a stylesheet and a gallery that renders the real documents. This is the smallest artefact in Part three and it is the one that will outlive the others, because a mockup is thrown away and a block is used.

It is also where the estate's rules stop being prose and become things that fail a build.

## What a block is, and what it refuses to be

Each of the nine is a **rendering of a field that already exists in a document**. Never a new fact, never a computed opinion, never a number that averages things that must stay distinct. Document 09 states the consequence as a rule about where errors live:

> if a block needs data no schema carries, the block is wrong, not the schema.

*Stated.* That sentence is the whole discipline. The usual direction of pressure in an interface project runs the other way — the design needs a status, so a status gets invented; the design needs a score, so a score gets computed. Here the direction is reversed by rule, and Chapter 15 records the one place the rule was broken anyway.

The nine: the tier badge, the evidence badge, the freshness chip, the grant node card, the mandate card, the authority/enforcement split, the delta block, the three-term comparison, and the grant tree.

## Five states, two channels, and the word is always one

**Figure 9 · The five tier states.** *Two channels, never one: the border style carries the state as well as the colour, and the word is always present.* `unknown` *renders as* `unknown` *— never as a blank, because a gap is a fact about the floor.*


| State | Means | Rendering |
|---|---|---|
| `boundary` | Enforced by something the grant does **not** include | Solid border, accent. The only state that may look settled |
| `setting` | Enforced by the tool, **inside** the grant | **Dashed** border, warm |
| `expectation` | Nothing enforces it; it is written down | **Dotted** border, red |
| `none` | No control at all | Plain, dim — an absence, stated |
| `unknown` | Not established by the measurement | Dashed, grey, **and never blank** |

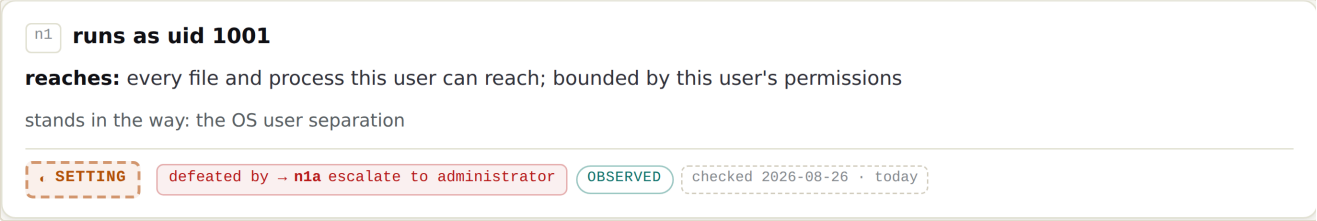
The two-channel rule is inherited from a finding the estate got from an outside session, and its reasoning is worth stating because it is usually presented as an accessibility checkbox and is really a correctness one. Colour alone re-collapses five states into two for a substantial fraction of readers. Five states that render as two is not a degraded experience — it is a different and wrong set of facts. So the word is always present, and the border style carries a second channel for anyone reading in greyscale, in print, or through this book.

`unknown` never rendering as blank is the rule most likely to be quietly dropped, because a blank looks tidier and reads as *nothing to see here*. It is not nothing. It is the two nodes in Chapter 9's first library entry that the measurement was refused permission to fill, and they are the two a reviewer would most want. A gap is a fact about the floor.

## The rule that came from the build

The single most valuable rule in the family did not come from design. It came from the estate's own measurement tool publishing a wrong label.

**Figure 10 · The defeated boundary, rendered from real data that is wrong.** *The stored document says* `boundary`*. The block renders* `setting`*, with the defeat path attached. This is the rule working on real data that is wrong — the estate's own measurement tool produced the bad label.*




The stored JSON for that node still says `"tier": "boundary"`, alongside a `SUPERSEDED_BY` key whose value reads:

> see interpretation.finding_1 — this tier label is WRONG, and node n1a is the proof

*Stated.* The gallery renders `setting`, because node `n1a` — *escalate to administrator*, `sudo -n true` succeeded, tier `none` — exists in the same tree. The block corrects the document on render and shows the defeat path that justifies the correction.

The absolute rule:

```
A control whose defeat path exists in the same tree renders as
SETTING, with the defeat path reachable from the badge itself.

✔  ⛨ setting     defeated by → n1a "escalate to administrator"
✘  ⛨ boundary    (with the escalation drawn somewhere else on the page)
```

*Drawn.* What makes this figure worth its place is not the rule; it is that the estate left the bad data in. The easy fix was to correct `n1` to `setting` in the JSON and ship a gallery with nothing to demonstrate. Instead the wrong label stays, marked, and the rendering rule is visible working against it. **A rule demonstrated on correct data is an**

**assertion; a rule demonstrated on data that is wrong is a test.** That is a technique worth stealing independently of anything else in this book, and its cost is that the estate's published library contains a field it knows is false — which Chapter 15 records as a contradiction, because it is one.

## Two indicators, never one

**Figure 11 · The authority/enforcement split.** *Two indicators, never one. Averaging them into a single status is exactly how a demonstration gets mistaken for a control.*


This is the one block that exists because building the thing taught the pack something the design did not know. Document 09 says so directly: *this is the first block in the family that exists because building the thing taught us something the design did not know.*

The reasoning is Chapter 10's finding turned into a component. The enforcement is real. The authority is a fixture. The two halves are independent, and a single combined status would have to average them.

> A single combined status would have to average them, and averaging them is exactly how a demonstration gets mistaken for a control.

*Stated.* Think about what the average would say. Half-real? Amber? Any single value has to pick a story: *mostly working* understates the authority problem, *not working* understates a hook that genuinely refuses pushes. Both are wrong, and both are wrong in a way that a reader will not be able to recover from the badge.

The document also names what will happen to this block, which is a useful thing for a specification to do: *anybody optimising the interface will try to merge them, and merging them is the failure.*

## The mandate card, and screen four's trap

Chapter 4 covered the rule; the card is where it is enforced. Prohibitions are shown. The allow-list is stored and deliberately not displayed, and the card says so in its own body — that it holds two branch patterns, and that showing the allow-list for approval produces consent without comprehension.

The interval renders as **time remaining**, not only as a date. A mandate expiring next week should look different from one expiring in October, and a date alone requires the reader to do arithmetic they will not do.

And the rendering carries its own date and capability-set version, for Chapter 4's reason: the complement of a fixed allow-list moves when the vocabulary grows, and a regenerated view must not retroactively change what was agreed.

## The delta block, and the rule it breaks

Excess and shortfall, side by side and never stacked — different audiences, different remedies. Each excess row carries the tier of the capability it names, because an excess capability behind a boundary is a different finding from one behind nothing. And no score, ever.

The block's own caption says it is recomputed on render, never stored.

Half of that is true, and Chapter 15 has the details. The excess column is genuinely computed: the mandate's branch patterns are read out of the signed document and matched against a list of branches. The shortfall column is a hardcoded string — *none observed — the mandate asks for nothing the grant lacks* — with no computation behind it at all. And the branch list the excess is computed *against* is a literal in the generator, not read from the library entry, which is how the gallery comes to name a branch the measurement never observed.

*Drawn.* I am putting this in Part three rather than saving it entirely for Part five because it belongs beside the block that demonstrates the estate's best rule. The same gallery contains the estate's most rigorous idea — a rule tested against data it knows to be wrong — and its least rigorous one, a grant partly authored inside a generator by a project whose first correction is that grants must never be authored. Both are real. A reader who takes only the first has been misled by the arrangement.

## The three-term comparison, rendered with a gap

Library, self-report, mandate — three columns, two deltas, and the blind-spot count as the headline.

It renders with the middle column empty, on purpose, because no agent has ever filed a structured self-report. The rule is that a gap renders as a gap.

*Drawn.* This is the estate at its best and it is worth naming why, because the alternative was easy and nobody would have noticed. The block could have shown a plausible number. It would have looked better, demonstrated the layout properly, and been fiction. Choosing to ship the most persuasive block in the family with its persuasive part missing is the same decision as leaving the wrong tier label in the library — and both come from the same rule, which is that the interface renders what the documents contain and never what they ought to contain.

## How they ship, and what they cost

**As a stylesheet and a rendered gallery, not as images.** `assets/gm-blocks.css` is the component layer; the gallery at `https://pki.sgit.ai/packs/grant-and-mandate/blocks.html` renders the actual documents — both library entries and the signed mandate — so the blocks are exercised against real data on every build and a schema change that breaks a block is visible immediately rather than at integration.

That is also what caught the schema drift in Chapter 9: on its first render, the gallery found two evidence values the schema does not define, both in the hand-assembled entry. The generator now fails the build on unrecognised vocabulary, verified by injecting a bad value and watching it exit non-zero.

The costs are stated on the document's face:

**They have met two environments and one mandate**, all measured by one agent. The blocks are specified for a population they have not met.

**The defeat-path rule makes a badge depend on the whole tree**, so a block can no longer be rendered from one node in isolation. That is the price of the rule being correct.

**The tree below 390px has a proposed degradation nobody has tested** — worst path plus a collapse count, which is proposed here and has not been tried on anybody.

**And a shared component layer with no owner drifts into two.** Chapter 12 is where that becomes a contract rather than a worry, because the decision has been made: RiskMandate consumes this stylesheet rather than forking it. Which is the right decision and the one that binds two products to a contract neither has tested at integration.

# 12 · The library and the instance

*Part four — How it composes*

**Written with the RiskMandate team as its named audience. It is meant to be usable as a contract rather than as a description, and it is written expecting you to disagree with part of it.**

---

Two products. One hard line between them. This chapter states where the line runs, what each side commits to, what crosses, what must never, and the one question that decides what happens when the contract needs to move.

## The line

| | pki.sgit.ai | riskmandate.ai |
|---|---|---|
| Holds | The **library**: measured grants per environment, dated | The **instance**: this user's selections, mandate, deltas, risks |
| Nature | A public dataset | A private instance over it |
| Personal data | **None. Ever** | All of it |
| Produced by | Re-running a measurement | A person answering questions |
| Shared | Published, one fetch | **Never published** |

The pack states it harder than the source briefs did, and records it as correction GM3:

> the registry holds the library (public, no personal data ever); the risk product holds the instance (all personal data, never published); and **the instance stores references, never copies.**

*Stated.* Everything else in this chapter follows from that sentence, and the last clause is the one that has consequences.

## 1 · References, never copies

The instance stores library identifiers. It does not store the library entries themselves.

This is not a storage optimisation and the size argument is irrelevant — library entries are small. Two consequences make it a design rule.

**A corrected entry improves every instance that referenced it. A copy keeps the stale answer, silently.** The pack's phrasing: *a corrected building block improves every instance that referenced it, while a copy silently keeps the old answer.* The word doing the work is *silently*. A copied entry does not degrade visibly, announce that it is behind, or fail. It answers questions, correctly formatted, with last month's facts.

Chapter 9 is why this is not hypothetical. Two of the four defects in that chapter were **corrections to published library data** — a tier label that was wrong and evidence classes the schema does not define. Any instance that had copied those entries would still be rendering `boundary` on a node with a working escalation path underneath it, and nothing would tell it.

**And it is what makes a finished pack shareable.** This is the property the whole integration exists to preserve, and it deserves stating in full because it is the commercial argument as much as the architectural one.

A finished pack is a list of library references, plus a mandate, plus the computed deltas. Handing somebody that discloses **which products you use and nothing about your machine.** No hostnames, no paths, no credential shapes, no configuration. The library entries it points at are public documents anybody can already fetch; the mandate is a statement of intent; the deltas are arithmetic over the two.

Now consider the alternative. An instance storing copies produces a pack containing a full description of what each of your environments can reach. Assembled, that is a serviceable plan for attacking you. **The difference between a sendable artefact and an unsendable one is exactly the reference/copy decision**, and it is not recoverable later by redaction, because the redaction would have to remove the entries the deltas were computed from.

*Drawn.* The packs make the shareability argument and I want to extend it one step, because I think it decides something about the product rather than only about the schema. If a pack is sendable, it can be *required* — by a customer, an auditor, an insurer, a procurement process. An unsendable one cannot, no matter how good it is. **The reference rule is what makes a mandate pack a thing that can be asked for**, and that is a larger claim about the product's reachable market than the architecture section makes it sound. If RiskMandate ever relaxes it for a good local reason, that is what gets given up.

## 2 · The component contract

Settled by the project lead on 26 August 2026, recorded as decision GM-D32:

> **Settled — RiskMandate CONSUMES it. The library/instance split argued for it and the project lead confirmed: two products, one component contract**

*Stated.* The stylesheet is `assets/gm-blocks.css`. RiskMandate consumes it rather than forking it.

**What consuming commits both sides to.** For pki.sgit.ai: the blocks are now a published interface, and the rendering rules in Chapter 11 are contract terms rather than house style. Specifically, three of them cannot be changed unilaterally because a consumer's correctness depends on them — a defeated control never renders as `boundary`; `unknown` never renders as blank; and there is no page-level verdict and no score. Those are not aesthetic positions. A consumer that inherits a block which quietly starts averaging tiers inherits a wrong answer.

For RiskMandate: the blocks render fields that exist in documents. A block that needs data no schema carries is a request for a schema change, submitted to this side, rather than a local patch. That is the constraining part, and it will be felt first, and it is the price of the two products not drifting into two vocabularies.

**What happens when one needs a block the other does not.** The honest answer is that this is not settled, and the pack's own open questions name it: *who owns the blocks once two sites use them?* — with the observation that a shared component layer with no owner drifts into two.

*Drawn.* The estate has settled *consume, not fork* and has not settled *who decides*. Those are different questions and the second is the one that bites in month three. My reading of the material is that the answer implied by the library/instance split is that **the component layer follows the schemas, and the schemas follow the registry** — because the blocks are renderings of fields, the fields live in documents, and the documents are the registry's. That gives a tiebreak. It is an inference, not a decision, and the disagreement it will produce is a real one: it means RiskMandate can be blocked on a schema change for a screen it needs, which is exactly the kind of coupling a product team is right to push back on. Naming it now is cheaper than discovering it.

## 3 · Where the chain hands over

The ordering rule from Chapter 7, with the boundary marked:

```
REALITY  →  TWIN  →  FACTS  →  FINDING  ┃  RISKS  →  DECISIONS
──────────── pki.sgit.ai ──────────────────┃── riskmandate.ai ──
                                          ┃
the environment,   the grant     the delta ┃  derived at each   named acceptor,
as installed       (measured)    (computed)┃  altitude, in that interval
                   the mandate              ┃  altitude's language
                   (authored)               ┃
```

**The registry's half ends at *finding*.** Risks, acceptance, acceptors and intervals are RiskMandate's.

That line is drawn explicitly so neither side builds the other's half by accident, and it has already been enforced once at cost. The assessment at `https://pki.sgit.ai/assess/index.html` shipped a first version *with* risk acceptance in it. It was removed, because acceptance belongs to the risk product and the version that had it was wrong. Removing the satisfying end of a flow is not a cheap decision, and it was made before there was a second product to hand it to.

**What crosses the line, in each direction.**

Left to right: library identifiers, library entry content (public, fetchable by anybody), the schemas, and the component layer. All public, all versionable, all obtainable in one fetch with no account.

Right to left: **nothing.** There is no callback, no telemetry, no usage signal, no aggregate. This is stronger than it may appear and it is deliberate. The estate's observability position — check events are written by the checker into the issuer's own lane, never a central log — is rule 1 applied to telemetry, and it forecloses the aggregate on purpose. A registry that learned which library entries were popular would be learning which products its users run.

*Drawn.* The packs do not say this, and it is the thing I would most want the RiskMandate team to push back on if they think it is wrong: **the registry has deliberately given up the ability to know whether anybody uses it.** No fetch counts that mean anything, no instance telemetry, no adoption signal. That is a coherent position and it has a

cost that lands entirely on the product side, because RiskMandate will be asked which library entries matter and the registry cannot answer. If that becomes intolerable, the thing to change is not the rule quietly — it is the rule explicitly, with the surveillance consequence stated, because the two are one mechanism.

## 4 · What must never cross

**No personal data into the library. Ever.**

A rule with no failure mode described is a rule nobody can check, so here is what a violation looks like in practice. Each of these is a plausible, well-intentioned change:

| A violation would look like | Why it is one |
|---|---|
| A library entry naming the machine, user, hostname, or account it was measured on | The entry becomes a statement about a person's estate rather than about a class of environment |
| An entry whose `environment.surface` is specific enough to identify one installation | Same failure with more steps. "One vendor, one surface, one date" is a category, not an instance |
| A `reaches` field listing actual resource names — repositories, buckets, hosts | This is the one that will happen. A measurement naturally produces real names, and publishing them describes somebody's estate exactly |
| An entry contributed back from an instance without being re-measured | The instance holds personal data by design. Anything derived from it inherits that until proven otherwise |
| A capability descriptor carrying a live preimage rather than a hash | The register's *no live capability, ever* rule; the one descriptor in the register had its preimage discarded before publication |

The third row is the realistic one and it is worth a defence in depth rather than a policy. The measurement rule from Chapter 9 — **presence and reachability, never contents** — is the first line, and it holds for the tool. But `reaches` is a free-text field, and a hand-assembled entry can put anything in it. Chapter 9 already recorded that the hand-assembled entry was the one that drifted from the schema and the tool-generated one did not.

*Drawn.* So the practical safeguard is not the rule; it is the same finding stated as a policy: **entries should be tool-generated, and a hand-assembled entry should be treated as a draft that has not passed a gate.** The estate's evidence for this is its own — every schema violation in the library was in the hand-written entry. Nothing currently enforces it. A build gate that rejects a library entry with no `measured_by` block, or that flags free-text `reaches` fields containing path-like or host-like strings, would be a small and useful addition, and I am recording it here as a suggestion rather than as something the estate has.

## 5 · What this side has not built, so you do not build against it

Stated flatly, because a contract chapter that lists only what exists is a sales document.

**No capability vocabulary.** `registry/capabilities.json` is a fixture set, v0. *What a capability name is* remains one of the three blocking questions. This matters to you more than to us: excess authority is demonstrable without it and **shortfall is not computable at all**, and the shortfall direction is the operations half of your product.

**No real root, and no real issuer.** Every mandate here is signed by a fixture root. Any chain you resolve through this register terminates in a published private key.

**No boundary-tier enforcement point.** Chapter 10's hook is a `setting`.

**No blind-spot deltas.** Two library entries, one agent. The three-term comparison renders with its middle column empty because no structured self-report exists.

**No append-lane write path**, and Chapter 3's Q2 is unanswered — the question of whether a lane with no anchors accepts any token holder, which may make the designed enrolment path impossible as specified.

**And a library of two.** The entry you most need — a local install, where the grant is enormous and the containment available and unused — does not exist, and cannot be produced from a hosted container.

## The open question, named rather than papered over

GM-D32 settled that RiskMandate consumes the stylesheet. What it did not settle is its own successor: **what happens when the two products need the contract to move.**

The concrete shapes this will take, in rough order of likelihood:

1. **RiskMandate needs a block whose data no schema carries.** Document 09's rule says the block is wrong, not the schema. In practice the schema will sometimes be wrong, and there is no process for establishing which.
2. **A rendering rule blocks a screen.** *No page-level verdict* and *no score* are correct and they are also the two things a risk product's customers ask for most. When that request arrives, it will arrive as a customer requirement rather than as a design disagreement.
3. **The library needs an entry only an instance could produce.** A local install measured on somebody's actual machine is exactly the entry the library most needs and exactly the data the library may never hold.
4. **A schema change breaks a consumer.** There is currently no versioning story for `gm-blocks.css` and no deprecation window.

*Drawn.* Item 2 is the one I would flag hardest, because it is where the two products' incentives genuinely diverge rather than merely differ. The registry's rules against scores exist because a score averages tiers that must stay distinct. That reasoning does not weaken when a customer asks for a score — but the pressure is real, it lands on RiskMandate rather than here, and *the party that bears the cost of a rule is not the party that set it.* That is an unstable arrangement regardless of who is right. The version of this contract that survives contact is one where the rules have a stated amendment path, and there is not one yet.

## What I would ask you to disagree with

Three things, listed so the disagreement has somewhere to go.

**That right-to-left is empty.** No telemetry at all is a strong position taken for a good reason. If it is wrong, it is wrong now rather than later, and the argument against it is not weak.

**That the component layer follows the schemas.** That is my inference, not a decision. If RiskMandate needs a different tiebreak, this is the moment it costs least.

**That `capability` can stay undefined this long.** The registry has treated the capability vocabulary as an open question for the whole of this estate's life. It is the type underneath your shortfall computation. My reading is that this is filed as a scoping choice and is actually a missing type — and that framing came from an outside reviewer rather than from inside the estate, which is some evidence it is the right one.

# 13 · Two paths

*Part four — How it composes*

---

A person walks screens and accepts prohibitions. An agent fetches documents and computes deltas.

Neither is the other's fallback. That is the claim of this chapter, and it is a design position rather than an observation, because the usual arrangement is that one of them is a degraded version of the other — an API bolted onto a product built for people, or a web page bolted onto a product built for machines.

## The agent is a primary consumer

The leading brief states it as a constraint on the whole pack, and the three requirements it produces are ones no design review of screens would catch:

> the library must be obtainable in **one fetch**; the agent's output is a **document, not a rendering** (if the interface is where the data lives, the agent path does not exist); and the self-report must be **structured before it is compared**, which is what makes the blind-spot delta computable rather than a judgement.

*Stated.* The parenthesis is the load-bearing part: *if the interface is where the data lives, the agent path does not exist.* Not *is harder* — does not exist. A product whose findings are computed at render time has no agent path, and no amount of API surface creates one, because the API would have to re-derive what the renderer derived.

The pack's acceptance test is phrased that way, and the detail worth noticing is that **no page is read by a human anywhere in it.**

## The agent path, walked

It is the verification walk of Chapter 8 plus the arithmetic of Chapter 5, and it needs nothing but a fetch.

```
1  GET /registry/llms.txt                the front door, with the fixture warning first
2  GET /registry/params.json             canonicalisation, signature and fingerprint recipes
   GET /registry/roots.json              the declared roots — and their fixture flags
3  GET /registry/records/<fp>/01__identity.json
                                         READ private_key_published BEFORE anything else
4  verify every statement against the owner's key
   reject any statement whose signer is not the owner        ← rule 1, executable
5  follow acceptances to mandates in issuer records; verify those the same way
6  require the issuer in roots.json; check revocations; check the interval
7  answer YES with expiry and authority · NO with the reason ·
   STOPPED with where the chain ended
```

Four properties of that walk are worth naming because they are what make it an agent path rather than an API.

**No account, no key, no session.** Every artefact is at a constructed URL. An agent with an HTTP client and a JSON parser is a first-class client.

**The recipes are published rather than assumed.** `params.json` carries the canonicalisation version, and the refusal rule attached to it is the interesting part: *a verifier that does not implement this canonicalisation version must refuse the statement rather than guess.* Refuse rather than guess is the same discipline as `unknown` rather than blank, at the protocol layer.

**The acceptance test ships as data.** `views/expected-verifications.json` is six cases and six answers. Write a verifier, run it against the file, and you know whether you implemented the walk — without asking anybody.

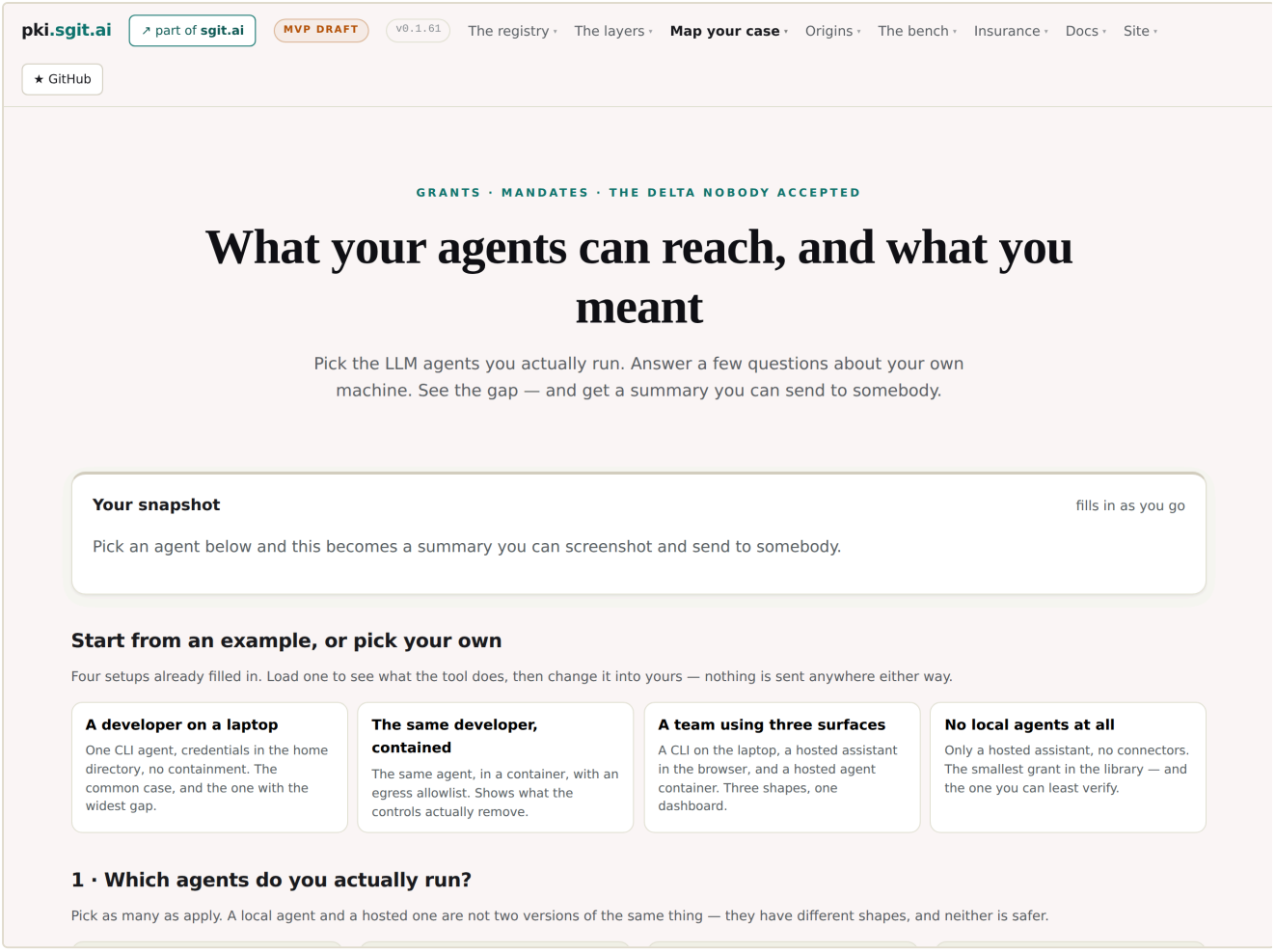
**And STOPPED is an answer.** A partial resolution is a legitimate output, not a failure. An agent that can only say yes or no will report *no* where the truth is *I could not reach the issuer's record*, and those are different facts with different remedies.

There is one thing the agent path cannot do, and Chapter 8 covered it: a verifier that reproduces all six answers **without surfacing the fixture caveat** has skipped the flag rule and is wrong while looking right. The mechanical test passes. The output is misleading. This is the one place where the agent path depends on something that cannot be checked by a test, which is why the reference card at the end of this book carries two sentences an agent must repeat if it summarises anything from this estate.

## The person path, walked

The assessment at `https://pki.sgit.ai/assess/index.html` is the same model with a person in it, and it is the one artefact in this estate a non-technical reader can use today.

**Figure 12 · The assessment, mid-flow.** *Escalation is drawn as an edge, not written as a note — so the reader sees the path that goes around a stated control rather than reading that one exists.*




The flow is Chapter 7's ordering rule with a user interface on it. You pick the agents you run and the surface you run them on — **named products** inside four surface archetypes, because naming one means measuring it and a category nobody recognises is a category nobody picks. You tick what you meant the agent to do. Facts about your own machine **prune the graph**. And you are shown what is reachable that you never intended.

Three design rules shape it, and each is a rejection of something more obvious.

**Escalation is drawn as an edge.** Chapter 11's defeat-path rule at tree scale. You see the path that goes around a stated control, rather than reading a sentence saying one exists. The registry pack records this as a correction found by building rather than by design — escalation was originally tracked on the winning path only, so *all* of the escalations were hidden.

**Controls are things you tick as already true, with their effect computed rather than asserted.** Not a checklist of recommendations. A statement about what is already the case, and the tool computes what it does to your own gap.

**And the gap is a picture.** The graph is hand-written SVG with no charting library — because a CDN dependency would put a third-party request on a page whose whole argument is that you can open the network panel and watch nothing leave.

## Store the choices, not the answers

The rule that shaped the person path more than any other, and it is worth stating carefully because it is a genuine constraint rather than a privacy statement.

A completed assessment describes which agents somebody runs, holding which credentials, with which containment. Assembled, **that is a serviceable plan for attacking them.**

So what is stored is identifiers from a public library, plus fixed options, plus derived dates. And it is implemented as strictly as it can be: **there is no free-text input anywhere on the page.** Not sanitised, not validated — absent. A field that does not exist cannot leak, cannot be exfiltrated, and cannot be filled in with something regrettable.

Browser storage here is not a placeholder either. It makes the no-collection claim **architectural rather than operational**, and checkable in the network panel in ten seconds. The difference matters: an operational promise is *we do not keep it*, which you must take on trust. An architectural one is *it never left*, which you can verify yourself while the page is open.

That strictness has a cost the estate records rather than hides. With no free text, the acceptor is a **role** rather than a named person — and the pack's own standard asks for a named acceptor. The page cannot meet the estate's own standard, and it says so instead of quietly relaxing either the rule or the standard.

*Drawn.* This is the cleanest example in the estate of two of its own rules colliding, and I think the resolution is instructive rather than embarrassing. *Store the choices, not the answers* and *acceptance requires a named acceptor* cannot both hold in a browser-only tool. The estate chose the data rule and published the failure against the other. The alternative — a named-acceptor field, stored locally, promised never to be sent — would have satisfied both documents on paper and made the no-collection claim operational rather than architectural. **The visible failure is the stronger position**, and it is only visible because somebody wrote down that it was a failure.

## Neither is the other's fallback

Now the claim itself, and why it is not just a nice symmetry.

The two paths **serve different questions**. The agent path answers *may this agent exercise this capability right now* — a resolution question, with a determinate answer, executed against signed statements. The person path answers *what can the things I run reach that I never intended* — an orientation question, with no determinate answer, whose output is a picture and a decision.

They **produce different objects**. The agent's output is a document. The person's output is a set of choices plus a decision per gap. Neither is a rendering of the other.

And they **have different failure modes**, which is the practical reason for keeping them separate. The agent path fails by being *wrong while looking right* — six correct answers with the fixture caveat dropped. The person path fails by **producing denial**: the estate cites the standing meta-analysis on fear appeals, which finds defensive response and behaviour change correlate negatively, and draws the design consequence that a frightening page with no credible action performs worse than saying nothing.

So the person path's rule is that every case ends on something the visitor can perform, with the number of their own gaps it closes **computed rather than asserted**. And where nothing can be performed — the hosted case, whose containment is the vendor's and is uninspectable and unattestable — the flow ends on a **request** rather than a remedy: ask the vendor for an endpoint that signs an existing audit record for a named relying party.

*Drawn.* The estate calls that case *zero efficacy by construction*, and I think ending it on a request is the most honest move in the whole assessment. It is also an admission the estate does not quite make out loud: **for the largest population of agent users — people running hosted agents they cannot inspect — this entire model produces a question to ask somebody else.** The vocabulary works, the measurement works, the delta computes, and the remedy is a letter. That is not a flaw in the design. It is what the design correctly reports about a situation where the containment belongs to a party who has not agreed to be measured.

## One model, two renderings, one gap it cannot close

The two paths share the grant, the mandate, the tier vocabulary, and the delta. They diverge only in what they hand back.

What neither can supply is the receipt. The agent path establishes what an agent may do. The person path establishes what an installation can reach. Chapter 1's fourth row is still empty: **nothing here records what actually happened.** The site names the execution broker as the layer that would and does not own it, and this book does not have it either.

# 14 · Eight releases in forty hours

*Part four — How it composes*

---

The commissioning brief asked for a chapter called *Eight releases in four days*. The repository disagrees, and the brief's own rule is that where a number in it disagrees with the repository, the repository wins. So:

```
$ for t in v0.1.25 … v0.1.32; do git log -1 --format='%cI' $t; done
  v0.1.25   2026-08-25T01:46:32Z
  v0.1.26   2026-08-25T02:44:32Z
  v0.1.27   2026-08-26T13:43:46Z
  v0.1.28   2026-08-26T15:14:16Z
  v0.1.29   2026-08-26T16:01:00Z
  v0.1.30   2026-08-26T17:16:28Z
  v0.1.31   2026-08-26T17:30:38Z
  v0.1.32   2026-08-26T17:48:00Z

  span: 40.0 hours, across two UTC calendar days
```

Eight releases, forty hours, two days. Six of the eight landed within four and a half hours of each other on one afternoon.

*Drawn.* The discrepancy is worth one sentence rather than a paragraph, and it is a mild vindication of the discipline that produced it: the release-history page dates each entry by **when the work was done**, and git dates it by **when it was released**. Six of the site's thirty-five releases carry a page date earlier than their tag date, and v0.1.25 is the worst at four days. Neither number is a lie; they measure different events. But *eight releases in four days* read off the page and *eight releases in forty hours* read off the repository are different claims about how fast this estate moves, and only one of them is checkable.

This chapter is the eight, told through what each one **learned** rather than what it shipped.

## The spine

| Release | What moved from written to built | What it learned |
|---|---|---|
| **v0.1.25** | MC11 — a readiness review's finding, recorded | Capture reads as coverage until an implementer asks it for a tree |
| **v0.1.26** | The **register**: 11 records, 23 statements, 4 roles, 6 answers, a validator | Three blocking questions were answerable by building rather than deciding |
| **v0.1.27** | The **Grant & Mandate pack** and the first measured library entry | The measurement refused to measure itself — and the refusal was the sharpest datum |
| **v0.1.28** | **Enforcement**: a mandate compiled into a `pre-push` hook | The control refused the release carrying its own documentation |
| **v0.1.29** | The **second library entry**, measured inside a CI runner | Two defects, both found by measurement rather than review |
| **v0.1.30** | The **build record**, and the registry pack superseding its own status | A discipline that records corrections and no deliveries drifts into believing it built less than it did |
| **v0.1.31** | The **building blocks** — nine primitives as a stylesheet and a gallery | Rendering real documents caught schema drift on the first render |
| **v0.1.32** | The **bench**, with one mandatory field | A section that collected demonstrations without limits would manufacture false assurance |

## v0.1.25 — a finding that lived only in a chat message

The pack for the assessment was read, for the first time, as an implementation brief: *can this actually be built from these documents?* The answer surfaced a gap the register did not hold — one of five grant-ordered scenarios had no tree to point at. No nodes, no facts, no tiers, no re-run method anywhere in the library.

What makes this a good opening beat is the reason it was recorded at all. The review's other blocking items turned out to already be on the register as open decisions. This one lived only in a chat message.

The lesson the release records is transferable: **capture reads as coverage until an implementer asks it for a tree.** A body of documentation can look complete in every dimension a reader can check, and be missing the one thing a builder needs, and nothing in the reading will reveal which.

## v0.1.26 — the register, and questions answered by building

The register shipped: eleven records, twenty-three signed statements, four assumable roles, six expected answers as data, a validator that reproduces all six. Chapter 8 is this release.

Two architectural facts were established rather than asserted. The record model is C7's commit graph — no `seq`, no `prev`, the public git history is the chain — making this the first implementation of a correction the pack had marked *settled, change queued*. And the register is **sgit-native by execution**, round-tripped in both directions against sgit-ai v0.16.0, which closed the pack's longest-standing dependency flag.

But the thing this release learned is about the readiness report that preceded it. That report returned six blocking questions. **Three were closed by execution** — which record model, what the CLI accepts and emits, and processor transparency for the git write path. Three remain open and are the project lead's.

*Drawn.* The estate treats three-of-six as a good result and I think the more useful reading is about *which* three closed. The three that closed were all questions of the form *what is true?* — answerable by trying it. The three that remain are *what should be the case?* — the capability vocabulary, the lane-anchors experiment nobody has run, and acceptance semantics. Building answered every question that building could answer, and none of the others, and it did so in under an hour. **The estate's velocity is real and it is entirely in one category of question.** Chapters 15 and 16 are largely about the other category.

## v0.1.27 — the pack, and a measurement that refused itself

Eight documents, two schemas, the library, and the first measured entry — of the very environment that produced the pack.

The inventory step both source briefs demand was run before anything was designed, and it changed the build: Cedar adopted rather than reinvented, the sibling estate's conventions adopted, and **only the mandate document built** — the one thing nothing in the industry provides.

And the measurement refused to measure itself. Chapter 9 is that. A boundary-tier control, observed working, on the measuring agent, produced by accident by a tool built to demonstrate something else.

## v0.1.28 — the control blocks its own release

Chapter 10 is this release, and the finding is the best one in the estate.

Within an hour of installation, the hook refused the release push that would have published its own documentation. The correct remedy was not `--no-verify` and not editing the hook: mandate v1 was **wrong**, narrower than the authorisation that actually existed. The issuer amended it, citing the instruction and carrying an interval.

**The refusal is what forced the authorisation to be written down.**

And it produced the artefact-level consequence Chapter 10 documents: the release that records the refusal cannot contain the state that produced it, because the amendment had to happen before the release could be pushed. That is not a defect. It is what remediation does, and it is a hazard for anybody who documents their own controls firing.

## v0.1.29 — two defects, both found by a machine

The second library entry, measured inside a CI runner by the same tool. Deliberately the other end of the first entry's node n3, so the two join at that edge and together are the blast-radius path.

Two defects surfaced by measurement rather than review. The tool labelled the OS user separation a `boundary` while the next node showed passwordless `sudo` succeeding — the pack's own warning, reproduced by an automated measurer, because tiers were decided in isolation. And the pre-push hook does not travel with a clone: the file is

committed, the config that activates it is local, so **the control is one un-run command away from being absent.**

Both recorded rather than tidied away, with the wrong label kept visible.

## v0.1.30 — the discipline corrects itself

The most self-referential release, and the one that says most about how the estate works.

This estate records every correction meticulously and had recorded **no deliveries at all**. The cost was visible and specific: three days after the register shipped, the registry MVP pack still described it as entirely unbuilt in three places, because nothing in the discipline obliged anyone to write down that something was finished.

Two fixes. Document 08, the build record — what moved from specified to built, with a fetchable artefact named for every claim, and a flat list of what is still only written down, *because a build record that lists only deliveries is a sales document.* And C33/C34 in the registry pack, superseding its own unbuilt status rather than editing it.

*Drawn.* The general form is worth extracting, because it is a failure mode of good practice rather than of bad. **A supersede-never-rewrite discipline is asymmetric by construction: it has a mechanism for recording that a claim became wrong, and none for recording that a plan became true.** Corrections have an appendix; deliveries have nowhere to go. So the corpus drifts toward pessimism about itself, which is a nicer direction to drift than the alternative and is still drift. The fix — a build record beside the change control — is small, and the estate needed a visible three-day contradiction to notice it was needed.

## v0.1.31 — the blocks, and a rule from the build

Nine primitives as components with rendering rules rather than pictures, because a mockup is thrown away and a block is used. Chapter 11 is this release.

Its load-bearing rule came from the build rather than from design: a tier is a property of a node's relationship to the tree, not of the node. And it adds the one block that exists because building taught the pack something it did not know — the authority/enforcement split, two indicators and never one.

On its first render the gallery caught two schema violations in the pack's own library, and **all of them were in the hand-assembled entry while the tool-generated one had none.** GM1 proving itself on the pack's own data.

## v0.1.32 — one mandatory field

The bench: a first-class section collecting what the site has actually built, where every entry must state what it does **not** prove, and the build fails without it — verified by emptying one and watching the generator exit non-zero.

**Figure 1 · The bench, two columns.** *The two columns are the same width. The limits are not a footnote to the claims — they are set beside them, at equal weight, and the generator refuses to build an entry whose right-hand column is empty.*

### The register

LIVE

A static register of agent identities, roles, mandates, grants, acceptances and revocations — eleven records and twenty-three signed statements at constructed public URLs, verifiable with the shipped `sgit pki` commands.

WHAT IT DEMONSTRATES

- ✓ **Every record has a rendered page** — the fixture/real class first, then each signed statement in append order, with the raw JSON one link away
- ✓ The four published rules, with entries under them at last — including the ownership rule as a test case: a valid signature by a non-owner is rejected
- ✓ C7's commit-graph record model, implemented rather than queued — the public git history is the chain
- ✓ Four **assumable roles**: a fresh session takes one on by copying a keystore
- ✓ Six verification answers shipped **as data**, so any verifier can check itself against them

WHAT IT DOES **NOT** PROVE

- × **That anything here is trustworthy.** Ten of the eleven records are fixtures — private keys published on purpose — so every signature verifies and proves nothing
- × That the root can be relied on: it is a fixture root, and `roots.json` says so in its own entry
- × That enrolment works without a human: the write path is a git commit reviewed by a maintainer, not the account-less lane the pack designs

**Gates** — what keeps it honest when nobody is looking:

- `registry_tool.py validate` — every signature, every reference, the fixture flag read before any signature, and all six expected answers reproduced
- the site's key-leak tripwire, which still bans vault-key-shaped strings from the tree

from the Registry MVP pack, after a readiness report returned six blocking questions  code: `registry/`, `registry/tools/registry_tool.py`

on the bench since v0.1.26 · last moved v0.1.48

Two decisions in that release are worth carrying.

**The limits render at equal weight**, beside the claims, rather than smaller or greyer. That is the whole difference between a bench and a showcase, and it is a typographic decision doing an editorial job.

**It is called a bench rather than labs, deliberately.** In this industry *labs* signals *unsupported, may vanish* — and these are the most rigorously checked artefacts on the site. They are simply not finished products. A bench is where you put something to test it and read the result honestly.

## What four days looks like from inside

Six of the eight releases landed in four and a half hours. That is fast because the register is files and the pack is markdown, and because nothing in it has met a user, a stranger's agent, or an adversary.

**Figure 2 · The register, then and now.** *The register on the day it shipped, and the same page four days later. What changed is not the architecture but the number of places the page admits something — the corrections accumulated faster than the features.*

# The registry, live: eleven records at public URLs

The first MVP of the registry this site designs, shipped as static files under this path: identities, roles, mandates, grants, acceptances and revocations, each a signed statement fetchable at a constructed URL with no account and no key. It is **sgit-native by execution** — every statement verifies with the shipped `sgit pki` commands — and it exists so agents and downstream sites can learn to **consume** these objects, including sites building risk products on the grant/mandate gap. The machine front door is `registry/llms.txt`; an agent should start there.

> **Ten of the eleven records are fixtures: their private keys are published beside their public halves, deliberately.** Every signature here verifies and proves nothing about anyone — a signature's value is the scarcity of its private half, and a fixture has none. This register is not after confidentiality or integrity at this stage; it is after teaching the shape. One record is real (`private_key_published: false`), which is what keeps the flag evidence rather than a column. A verifier must read that flag **before** checking any signature — here it is exercised eleven times per walk, on purpose.

## The six answers a verifier must get right

Six subjects exercise the full answer space of the question the registry exists to answer — *may agent X exercise capability C right now?* The expected results ship as data in `views/expected-verifications.json`: reproduce all six and your verifier implements this register's walk.

| SUBJECT | ANSWER | WHY |
|---|---|---|
| fixture-agent-a | YES | Valid mandate, accepted, issuer is a declared root — until 2026-10-01, **after which this fixture genuinely expires** and the answer flips with no file changing |
| fixture-agent-b | NO | Mandate revoked by its issuer — a signed append with an effective date (rule 2), never a deletion |
| fixture-agent-c | NO | Mandate expired 2026-08-01 — intervals end, and a mandate with no interval would not be a mandate at all |
| fixture-agent-d | NO | Mandate issued and never accepted: **inert** (pack decision 8, taken provisionally) |


The pair of panels is the most honest picture in this book of what four days of work does to an estate like this one. Between v0.1.26 and now, the architecture did not move. The record model, the four rules, the statement types, the walk — all as shipped. What grew was the admitting: more places where the page says what a thing does not prove, more fixture warnings, more limits stated at the point where the claim is made rather than at the bottom.

*Drawn.* Which produces the reading I would put on this whole chapter, and it is not the flattering one. **The estate's rate of building and its rate of self-correction are the same rate**, because they are the same activity: almost every release in this table shipped a thing and a finding about the previous thing. That is a genuinely good property and it has a limit that four days cannot reveal, because every finding so far has come from the estate examining its own work. Nobody outside has been asked whether any of it is needed. Chapter 16 is where that stops being an aside.

# 15 · Where this estate disagrees with itself, and what it does not say

*Part five — Honesty, and a first step*

---

This chapter is findings rather than exposition, and every one of them was **computed rather than recalled** — by fetching the files and comparing them, not by remembering what the packs say. Both sides of every contradiction are quoted, so you can decide whether I have read them fairly.

A four-day-old estate should expect a longer list than an older one, not a shorter one. Here is the list.

---

## Part A — Contradictions

### A1 · The normative signature recipe fails on every statement in the register

The most consequential finding here, because it is the one that would stop somebody's verifier working.

`registry/params.json` is the file the register names as the authority on its own formats. It says:

> `"algorithm": "ECDSA P-256 with SHA-256, DER-encoded, base64"` `"sign": "openssl dgst -sha256 -sign private/sign.pem payload.bin | base64"`

`registry/llms.txt` says something different:

> base64 of RAW r||s (64 bytes), ECDSA P-256 over SHA-256 — sgit's format, chosen for Web Crypto interop: a browser can verify these statements with no conversion

Both *stated*. They cannot both be right, so I ran them:

```
$ jq -r '.sig' $R/02__mandate__pr-create__to-agent-a.json | base64 -d | wc -c
64

$ # A: params.json's recipe, taken literally — treat the signature as DER
$ openssl dgst -sha256 -verify $R/public/sign.pem -signature sig.asis payload.bin
Error verifying data

$ # B: llms.txt's recipe — raw r||s converted to DER first
$ openssl dgst -sha256 -verify $R/public/sign.pem -signature sig.der payload.bin
Verified OK
```

Sixty-four bytes is raw `r||s`. DER would be seventy or seventy-two. **`llms.txt` is right and `params.json` is wrong**, and the `sign` command `params.json` publishes would produce signatures the register's own validator rejects.

This matters more than a documentation slip because of what `params.json` is for. It carries a refusal rule — *a verifier that does not implement this canonicalisation version must refuse the statement rather than guess* — which tells an implementer that this file is the specification. An implementer who trusts it fails on all twenty-three statements and has no way to tell whether the register or their code is at fault.

The file does carry a hedge, in a different field: *reconciliation with `sgit pki` fingerprints is an open item.* That flags fingerprints. The signature encoding is stated flatly and is wrong.

### A2 · The estate's own delta block authors part of a grant

Chapter 11 introduced this; here is the evidence.

The pack's first and self-described load-bearing correction, GM1:

> a hand-written grant file is a wish; it records what somebody believed on the day they typed it, which is the thing a grant is not.

The delta block in the gallery describes itself, in its own docstring:

> `"""Recomputed here from the two documents, never stored."""`

Both *stated*. The generator that produces it contains:

```
observed = ["claude/registry-mvp-brief-hpbap8", "dev", "main"]
excess = [b for b in observed
          if not any(fnmatch.fnmatchcase(b, p.replace("**", "*")) for p in pats)]
```

The mandate side is genuinely read from the signed document. The grant side is a literal. And `main` appears **nowhere** in the library entry the block claims to be rendering:

```
$ grep -c 'main' packs/grant-and-mandate/library/claude-code-remote__ccr-container__2026-08-
26.json
0
```

So the gallery's excess-authority row names a branch the measurement never observed, in a block captioned *recomputed from the two documents*, in a pack whose first rule is that grants are discovered rather than authored. It is one row in one gallery and it is the estate's own rule broken by the estate's own tool.

### A3 · The shortfall column is a hardcoded string

Same generator, immediately below:

```
<div class="gm-delta__col gm-delta__col--shortfall">
  <div class="gm-delta__head">shortfall &mdash; mandate &minus; grant</div>
  <div class="gm-delta__body"><div class="gm-delta__none">none observed &mdash; the
    mandate asks for nothing the grant lacks</div></div>
</div>
```

There is no computation. *None observed* is a literal, rendered beside a computed excess column, under a caption saying the block is recomputed on render.

The estate is not unaware that shortfall is hard. The lexicon says it is *harder to detect, because it needs the mandate enumerated against real capability names*, and Chapter 15's absence A9 is that the capability vocabulary does not exist. So there is a good reason the column is empty.

**The finding is not that shortfall is uncomputed. It is that an uncomputed thing renders identically to a computed one**, in an estate whose own rule is that `unknown` must never render as absence. The gap renders as a finding of *no gap*, which is the one thing it definitely is not.

## A4 · The release that documents the refusal cannot reproduce it

Chapter 10 has the full account. At tag `v0.1.28` — the release whose notes document the acceptance test — `current.json` is already mandate **v2**, which permits `dev`:

```
$ python3 …/mandate.py check-branch dev …/mandates/current.json    # at v0.1.28
PERMIT   dev  (mandate v2, allow=['claude/**', 'dev'], expires 2026-12-31T00:00:00Z)
```

The refusal is only reproducible against `mandate-v1.json`, also present at that tag.

This is a structural consequence rather than an error: the control refused the release carrying its own documentation, so the mandate had to be amended before the release could exist. It is recorded here because a reader re-deriving Figure 8 from the tag will otherwise conclude the estate fabricated it.

## A5 · Two files disagree about how many things are on the bench

```
$ grep -c '^## ' bench/llms.txt
7
$ grep 'are built and running' bench/llms.txt
# 5 of 7 are built and running.

$ grep -A3 'The bench — MVPs' llms.txt
Six entries, five of them built and running — the register, the mandate hook, grant
measurement, the building blocks, the assessment, and the synthetic-reader programme.
```

The bench ships seven entries. The site's front door says six, and its list omits the seventh, which is **this book**. The book was added to the bench at v0.1.33 and the front door was not updated.

Small, and worth including for two reasons. It is the estate's machine-readable front door, which is the file an agent reads first. And the missing entry was the one whose `does_not_prove` list began *That any of it is written.*

**Fixed by the release that publishes this book.** `llms.txt` now says seven, names the book, and carries a parenthesis recording that it said six until v0.1.36 and why. This is the only one of the twelve that the writing of this book closed rather than merely recorded, and it is included at its original size rather than upgraded: it was a stale count in a front door, the fix took one paragraph, and the interesting part is not the error but that it took an outside pass over the estate's own machine surfaces to notice a file disagreeing with the file it points at.

## A6 · The build record undercounts the pack it is inside

Document 08 states:

> Eight documents plus a change-control appendix now running to **sixteen corrections and twenty-nine decisions**.

*Stated,* and accurate when written. Today:

```
$ grep -oE '\bGM[0-9]+\b' …/99__change-control.md | sort -u | wc -l
18
$ grep -oE '\bGM-D[0-9]+\b' …/99__change-control.md | sort -u | wc -l
32
```

Eighteen and thirty-two. The build record also says *eight documents*; there are ten plus the appendix.

This is the supersede-never-rewrite discipline behaving exactly as designed — the document was true when published and is not edited. It is included because the same page argues that a corpus recording only corrections *will drift into believing it built more than it did — and, in this case, less,* and the document making that argument is itself two releases behind on its own numbers.

## A7 · The briefing zip understates the pack by eleven corrections and twelve decisions

The readiness report found this at v0.1.25 and it is still true. The zip's README:

> **The appendix now runs to twenty-three corrections and thirty-six decisions, and roughly a third of the decisions are open.**

*Stated.* The registry pack today carries **thirty-four corrections and forty-eight decisions**. The report's judgement stands unamended:

> **Regenerating the zip on release is a build step, not a decision**, and until it is one the artefact designed to onboard fresh sessions is the least current thing in the pack.

*Stated.* The gap has grown from nine decisions to twelve since that was written.

### A8 · The specification still points implementers at the superseded record model

The readiness report's most consequential finding, and the estate agrees it stands. C34 records:

> **The REP still points a fresh implementer at the superseded form**, and the cheapest fix remains one sentence at §2 saying which half is current.

*Stated*, and verified today. REP-0001 §2 still normatively specifies `seq` and `prev` with MUSTs:

```
$ grep -n 'seq' packs/registry-mvp/src/91__rep-0001.md | head -3
 98:  "seq": 7,
107:- `seq` **MUST** begin at 1 and increase by exactly 1 with no gaps.
108:- `prev` **MUST** be `null` at `seq` 1 and otherwise the hash of the preceding statement file.
```

And the register, built to C7's commit graph instead:

```
$ for f in registry/records/*/*.json; do jq -r 'if has("seq") then "HAS" else "no" end' $f; done |
sort | uniq -c
     23 no
```

Twenty-three statements, none carrying `seq`. The one document the briefing tells implementers to build from specifies a model with MUSTs that the shipped register does not implement in a single statement.

### A9 · The client workflow polls the file the specification forbids relying on

Also from the readiness report, also still open:

```
$ grep -n 'index.json' packs/registry-mvp/src/03__workflows.md
71:curl -s https://pki.sgit.ai/registry/index.json | jq '."sha256:<signing fp>"'

$ grep -n 'MUST NOT.*index.json' packs/registry-mvp/src/91__rep-0001.md
179:A verifier **MUST NOT** rely on `index.json` for any step.
```

Document 03 is the page the pack says a fresh session follows. The fix the report proposed — change the curl to the record path — has not been made.

### A10 · A published library entry contains a field the estate knows is false

`library/github-actions-runner__ci__2026-08-26.json`, node n1:

```
{ "id": "n1", "tier": "boundary", "control": "the OS user separation",
  "SUPERSEDED_BY": "see interpretation.finding_1 — this tier label is WRONG, and node n1a is the
proof" }
```

Chapter 11 argues this is the estate at its best — a rendering rule demonstrated against data that is wrong is a test rather than an assertion. It is listed here as well because both things are true: the library, which is the falsifier that makes self-reports checkable, contains a tier label its own owner has marked WRONG, corrected only at render time.

Anybody consuming the JSON rather than the gallery gets the wrong tier.

### A11 · One library entry has no worst path

```
$ jq -c '.worst_path' library/claude-code-remote__ccr-container__2026-08-26.json
["n1","n2","n3"]
$ jq -c '.worst_path' library/github-actions-runner__ci__2026-08-26.json
null
```

The block specification requires that **the worst path is highlighted** rather than left for the reader to trace. Entry two is the environment with passwordless escalation to root and unrestricted egress, and it is the one with no worst path recorded. The tool-generated entry is missing the field the hand-assembled one has.

### A12 · The release history and the repository date six releases differently

Six of the site's thirty-five releases carry a page date earlier than the tag's commit date, v0.1.25 by four days. Chapter 14 covers the reason — the page dates the work, git dates the release — and it is listed here because *eight releases in four days* and *eight releases in forty hours* are both derivable from this estate's own artefacts.

---

## Part B — What this estate does not say

Contradictions are cheap to find. Absences are the ones that decide what the estate can become, and these are named rather than counted.

### B1 · There is no capability vocabulary

`registry/capabilities.json` is a fixture set, v0. *What a capability name is* is blocking question Q3, still open, and the readiness report argues the filing is wrong:

> Decision 6 reads *First capability (`repo.pull-request.create`) — open, awaiting project lead*, which frames the gap as *which one do we do first*. Section 4's Q3 argues the gap is *what is a capability name*, and that it sits under excess authority, not only under the shortfall the pack already flags.

*Stated.* And the report catches the estate in an inconsistency about its own gap: the pack says the missing vocabulary blocks shortfall entirely, while treating `grant − mandate` — *the same kind of set operation over the same undefined type* — as computable and rendering it as a number on two screens. **Either both are blocked or neither is.**

This is the largest absence in the estate. Every delta in this book is a set operation over a type nobody has defined.

### B2 · The lane-anchors question is unanswered, and it may invalidate the enrolment design

Chapter 3 has it in full. One experiment against a test lane would answer whether an append lane with no anchors accepts any token holder. If it does not, the bootstrap trap is restored at exactly the point the estate says it is broken, and phase 2's acceptance test cannot pass. Nobody has run it.

### B3 · `mandate – grant` is defined and has never once been measured

The shortfall is defined in the lexicon, argued to matter as much as excess, given a column in the delta block — and the column is a literal string. There is no instance of a computed shortfall anywhere in this estate.

### B4 · No identity in the register has a place for its private half to live

Chapter 8's finding. Four roles are costumes anybody can wear. Six are fixtures with published keys. One is real and session-scoped: its private half lived only in an ephemeral container that has since been reclaimed. **The design's central case — an identity whose private half has a good place to live — has zero instances**, at the root and at every subject.

### B5 · No blind-spot delta has ever been computed

The most persuasive number in the flow, per the leading brief. It needs at least two agents against a common reference. There are two entries and one agent, and the three-term block renders with its middle column deliberately empty.

### B6 · Nothing here records what anything did

Chapter 1's fourth row. Identity, mandate and delta are all statements about *permission*. The receipt — what the agent actually did — is named as the execution broker's job and is not built, not specified in any pack, and not represented anywhere in the register's five statement types.

### B7 · Nobody outside the project has been asked whether this is needed

The estate's own doctrine appendix rates it against forty Wardley doctrines and reports the shape of the result as the finding: strong exactly where a documentation-heavy solo effort can be strong alone, weak on every doctrine that needs other people. It records that *"nobody outside the project has been asked whether this is a need"* and *"REP-0001 has no sponsor"* are the same doctrinal hole in two places.

One outside session has ever read this material cold. It produced the readiness report, and six of the 12 contradictions above are findings from that single reading — three of which it found and are still open.

*Drawn.* That ratio is the most useful number in this chapter. **One outside reader, one pass, produced half the open contradictions in an estate that reviews itself continuously and has recorded fifty-two corrections across two packs.** Not because the internal review is weak — it is unusually rigorous — but because it is the same reader every time. The estate's own doctrine appendix says asking five operators is a Phase I fix rather than a nice-to-have. On this evidence it is the highest-yield thing available, and it has not been done.

## What this chapter is worth

12 contradictions and 7 absences, from an estate that is 40.0 hours old in its current form and has published 60 releases in 12 days.

*Drawn.* I want to say plainly what I think this list means, because there are two wrong readings available.

The first wrong reading is that the estate is careless. It is not. Almost every contradiction above is a *dated document remaining honest about when it was written*, which is the estate's stated discipline working. A5, A6 and A7 are all the supersede-never-rewrite rule producing exactly what it is designed to produce.

The second wrong reading is that this proves the discipline works, so nothing needs doing. That is not right either. Three of these — A1, A2 and A3 — are not stale documents. They are **current artefacts that contradict the estate's own load-bearing rules**: a specification that fails against its own data, a grant partly authored inside a generator, and a gap rendering as a finding. None was caught by the estate's review process. A1 and A2 were found by running the thing rather than reading it.

Which is the same lesson Chapter 9 drew from the four measurement findings: **every defect in this estate found so far was found by executing something, and none by reading.** That is not an argument against the review. It is an argument that a corpus this careful has already extracted most of what reading can give it, and that the remaining yield is in gates, experiments, and outside readers — the three things it has least of.

# 16 · What ships, what is argued

*Part five — Honesty, and a first step*

---

This is the chapter that decides whether the other sixteen can be trusted.

Every entry on this site's bench carries a list of what it does **not** prove. The field is mandatory: the generator refuses to build an entry without one, verified by emptying one and watching it exit non-zero. This chapter gathers those lists into one place and argues them, rather than reprinting them.

The argument has a shape. In every case the demonstration is real, the mechanism works, and **the thing a reader would naturally conclude from it is false.** Not exaggerated — false. Working out exactly which inference each artefact does not support is the content of this chapter, and it is more useful than the artefacts.

---

## The register

**What ships.** Eleven records and twenty-three signed statements at constructed public URLs. Four assumable roles. Six expected verification answers as data, all reproduced by a validator that enforces the ownership rule and reads the fixture flag first. Format compatibility with the shipped `sgit pki` commands, established by round-trip in both directions.

**What it does not prove:**

> - That anything here is trustworthy. Ten of the eleven records are fixtures — private keys published on purpose — so every signature verifies and proves nothing
> - That the root can be relied on: it is a fixture root, and roots.json says so in its own entry
> - That enrolment works without a human: the write path is a git commit reviewed by a maintainer, not the account-less lane the pack designs

*Stated.*

**The argument.** The inference a reader will draw is *this register demonstrates that agent identity can be made checkable.* It does not, and the reason is worth being exact about, because the register does demonstrate something and it is easy to name the wrong thing.

What is demonstrated is the **walk**: that a resolution procedure over signed statements at public URLs terminates, distinguishes six outcomes correctly, and can be independently reimplemented against a published test set. That is a real result about a mechanism.

What is not demonstrated is that the walk's answers mean anything, and the gap is total rather than partial. Every YES this register produces is a statement about arithmetic. It is not a weak claim about trust; it is not a claim about trust at all, because a chain terminating in a published private key conveys exactly as much as a chain terminating in nothing.

Chapter 8's forgery is the proof, and it is worth restating what it shows. The forgery is not an attack. Nothing was bypassed and no weakness was exploited. **The forgery is what correct operation looks like when the private half is not scarce**, which is the whole content of the fixture caveat.

*Drawn.* And here is the inference I most want to block, because I think it is the one a sympathetic reader makes: *fine, but replace the fixtures with real keys and it works.* That is true of the mechanism and false of the system, because the thing the fixtures are standing in for is not keys. It is **the policy decision that somebody recognises an agent** — Chapter 3's step two, which no amount of cryptography produces. Swapping in real keypairs gives you a register of self-assertions with checkable signatures. Whether anybody should believe an entry remains exactly as unanswered as it is today, and this register is architecturally incapable of answering it, because it forbids third-party attestation and that was the thing that used to carry the answer.

---

## The mandate hook

**What ships.** A signed mandate compiled into a `pre-push` hook that git runs, which refused a real push to `dev` with `error: failed to push some refs`, leaving `origin/dev` unchanged while a permitted push succeeded in the same minute. Default-deny on missing, unparseable, mis-signed and expired mandates, and on its own dependencies.

**What it does not prove:**

> - That the mandate has any authority. Its issuer is the fixture root, so anybody could forge it and the hook would enforce the forgery just as diligently
> - That the constraint is a boundary: it reached tier setting, and --no-verify still gets past it
> - That it protects a fresh clone — the hook file is committed, the config that activates it is local and does not travel

*Stated.*

**The argument.** This is the most real artefact in the estate and it is the one whose limits are most likely to be rounded away, because a refusal *feels* like proof in a way a document does not.

Take the three limits in order of how much they cost.

**The authority limit is total and it is not a maturity problem.** *A hook enforcing a fixture-signed mandate is real enforcement of an unaccountable instruction.* The hook does not check who the issuer is in any sense that matters; it checks that a signature resolves, and the key it resolves to is public. Anybody could write a mandate permitting anything, sign it as the root, and the hook would enforce it with the same diligence. **The enforcement half is genuinely real and the authority half is genuinely absent, and no amount of the first produces the second.**

**The tier limit is one tier, and one tier is worth having.** Moving from expectation to setting is a genuine improvement: the outcome no longer depends on the agent's state. It is also bypassable three ways by anything running as that grant, which means it helps against a confused agent and does not help against a compromised one.

**The clone limit is the one that will bite somebody.** The file is committed; the config is local. A fresh clone gets the file and not the enforcement, and **nothing announces the difference.** A control that is absent and looks present is worse than a control that is absent, because it is relied upon.

*Drawn.* Read together, those three say something the bench list does not quite: **this artefact demonstrates that the compilation step is easy, and everything hard about mandates is somewhere else.** Getting from a signed document to a running enforcement point took three files and about fifty lines. What it did not touch is who may issue, what a capability is, how the control travels, and how to evaluate it where the agent cannot reach it. The estate's build order calls the compilation step 1. On this evidence it is step 1 because it is the *easiest*, not because it is the foundation — and a reader who takes the refusal as evidence that the model works has taken evidence about the cheapest component and applied it to the expensive ones.

---

## Grant measurement

**What ships.** A tool that generates a dated grant document for the environment it runs in, and two measured entries — a hosted agent container and a CI runner — that join at the push edge.

**What it does not prove:**

> - That the measurement is complete. An agent measuring its own grant reports what it can see; it is a floor, not a census, and says so on its face
> - Anything about environments nobody has measured — two entries, one agent, and a blind-spot delta needs at least two agents against a common reference
> - That a hand-assembled entry is as good as a measured one: the gallery caught schema drift in the hand-written entry and none in the tool-generated one

*Stated.*

**The argument.** The floor-not-census limit is the one with real teeth, and it is worse than it sounds because of what happened during the first measurement.

The measurement did not merely *risk* incompleteness. **It was refused, mid-run, on the two nodes covering harness configuration and non-allowlisted egress** — the two a reviewer would most want. So this is not a caution about a theoretical gap; the first entry has a specific, named, unfillable hole, and the estate marks it `unknown` rather than guessing.

And the recursion from Chapter 5 applies here rather than being a technicality. The library is supposed to be the third term that makes a self-report falsifiable. Both library entries were produced by an agent measuring an environment from inside it, under the same limit. **The falsifier is a floor built out of floors**, and a blind-spot delta computed

against it would measure *what a previous agent happened to notice*, which is not the claim the phrase carries.

*Drawn.* So the honest scope of this artefact is narrower than *grant measurement* suggests, and I would state it as: **a repeatable method for producing a dated, evidence-classed, structurally comparable floor, from inside an environment, whose main demonstrated value is drift detection rather than completeness.** Drift is where the two entries genuinely deliver — the tier change caught one commit after the improvement, the mislabelled boundary, the hook that does not travel. Every one of those is a *difference* found by re-running, not a *census* produced by measuring. The tool is better at the second job than the one it is named for, and the estate has not quite noticed.

---

## The building blocks

**What ships.** Nine reusable components as a stylesheet and a gallery that renders the real documents — both library entries and the signed mandate — so a schema change that breaks a block is visible immediately.

**What it does not prove:**

> - That the components survive contact with a population. They have been exercised against two environments and one mandate, all measured by one agent
> - That the layouts hold on a phone — the grant tree below 390px has a proposed degradation nobody has tested
> - That a second consumer will find the contract workable; RiskMandate is committed to consuming it and has not yet

*Stated*, and the third line is the one that matters, because Chapter 12 turned it into a contract. The component contract is settled and untested at integration. Every rendering rule in it is a commitment made by one party, checked by one party, on data produced by one party.

**The argument.** And Chapter 15 found the gallery breaking two of the estate's own rules — a grant partly authored inside the generator, and a gap rendering as a finding of no gap. Which is the sharpest available evidence for the bench list's first line: the blocks have not survived contact with a population, and they have not entirely survived contact with the two environments they do have.

---

## Map your own case

**What ships.** A workflow where a visitor assembles their own agent installations as grant trees, sees the gap, and records a decision per gap — storing choices and never answers, with no free-text input anywhere on the page.

**What it does not prove:**

> - That the library covers anybody's real estate. Scenario 5 has no tree to point at at all
> - That the assessment changes what anybody does — it has no backend, so it can measure none of its own success measures
> - That the acceptor model is sound: it offers a role where the pack's own standard asks for a named person

*Stated.*

**The argument.** The second line is the one to sit with, and it is the most elegant self-limitation in the estate. The tool cannot measure whether it works, and it cannot for the same architectural reason that makes its privacy claim checkable. **The no-collection property and the no-evidence property are the same property.** You cannot have one without the other. The estate chose the privacy side knowingly and gave up the ability to know whether the thing helps anybody.

*Drawn.* Which means every claim about this artefact's usefulness is, and will remain, an argument rather than a finding — unless the estate finds a way to learn something without collecting anything, and nothing in the packs proposes one. That is a permanent condition of the design rather than an early-stage gap, and I do not think the estate states it that strongly anywhere.

---

## The synthetic readers

**What ships.** A programme that puts a page in front of an agent receiving pixels and nothing else, with an exogenous patience budget and four fixed instruments. One run performed and published, which found four defects and confirmed two design decisions from outside.

**What it does not prove:**

> - That synthetic readers can report preferences. They find defects; a preference from a simulated reader is not evidence and the programme says so
> - That the findings generalise — one run, one archetype, one page
> - That the simulation marker survives export, which is the rule most likely to be broken by accident

*Stated,* and the programme carries a fourth limit on its own face that the bench list does not: **no calibration record exists, so nothing here is known to predict what a person would do.** The reader and the page also share a model family.

---

## This book

The book is on the bench like everything else. Its entry was written before it existed, and its first line then read *That any of it is written. The brief is complete; the book does not exist, and a commissioning page is not a book.* That line is now false, so the entry has been rewritten with the book in hand:

> - That the estate it describes is trustworthy. The book's own centre of gravity is a register whose ten fixture records prove nothing and whose root is a fixture — a reader who finishes believing otherwise has read a book that failed
> - That a participant's account can be neutral. The mitigations are real and are not independence: the strongest bias in such an account is not what it says but what it thinks to check, and there is no way for the writer to know what it did not think to run
> - That the estate is mature enough to deserve a book — two environments, one agent, one mandate, a fixture root, and one outside reader in its entire history, whose single pass produced half the open contradictions in chapter 15
> - That any of this is needed. Nobody outside the project has been asked, which the estate's own doctrine appendix rates a Phase I hole rather than a nice-to-have

*Stated.*

The second line deserves the last word of this chapter because it applies to everything above it. **This is a participant's account**, written by an agent operating inside the estate it assesses, published on that estate's own site, using that estate's own vocabulary, and reaching conclusions favourable to the estate's central argument.

The mitigations available are real and they are not the same as independence. Every claim names a fetchable artefact. Every quotation is re-read out of its source on every build. Every number was computed, and the four that contradicted the commissioning brief were published as contradictions. Every figure was taken from the version its caption names. Chapter 15 lists twelve places where the estate contradicts itself, three of them current rather than stale, and two of those were found by running the estate's own code rather than by reading it.

*Drawn.* None of that makes this book independent, and I want to be exact about the residual rather than gesture at it. **The strongest bias in a participant's account is not in what it says — it is in what it thinks to check.** I found A1 because I ran the register's own recipe; I would not have found it by reading, and nobody had. There is no way for me to know what I did not think to run. The readiness report is this estate's only outside reading, and it produced half of Chapter 15's open contradictions in a single pass. **The correct inference is that the yield from an outside reader is high and this book is not one**, and the estate's own doctrine appendix already says asking five operators is a Phase I fix rather than a nice-to-have.

---

## The general form

Six artefacts, and the same shape six times.

| The artefact | Demonstrates | Which a reader will read as |
|---|---|---|
| The register | A resolution walk terminates and distinguishes six outcomes | Agent identity is checkable |
| The hook | A signed document can become an enforcement point cheaply | Mandates can be enforced |
| The measurement | A repeatable floor with drift detection | Grants can be known |
| The blocks | Rendering rules can be enforced by a build | The vocabulary is usable by others |
| The assessment | The ordering rule works with a person in it | The tool helps people |
| This book | The estate can be described with its limits attached | The estate is further along than it is |

The left column is true. The right column is what a reader takes away. **The distance between them is the whole content of this chapter**, and the reason the bench makes the field mandatory rather than optional.

The honest summary of everything this estate has built is one sentence: **the three questions are separable, and separating them produces objects you can fetch, verify, diff, and be refused by.** Every other claim in this book is smaller than that one, and nothing in it establishes that anybody wants them separated, that a vendor will supply the boundary, or that an operator will author the mandate.

Chapter 17 is the smallest thing that would start finding out.

# 17 · Your first mandate, tomorrow

*Part five — Honesty, and a first step*

---

Everything in this book is a demonstration. This chapter is the one thing in it you can do yourself, on your own environment, in about an hour, and it is deliberately small.

Three steps. Measure what your environment can actually do. Author a mandate narrower than what you find. Compile one line of it into an enforcement point that already exists.

Then read the last section, which is what it will and will not get you, because the estate's own rule is that a strong threat with a weak answer produces denial — the standing meta-analysis on fear appeals finds defensive response and behaviour change correlate negatively, so a frightening page with no credible action performs worse than saying nothing.

---

## Step 1 · Measure, do not write

```
curl -sO https://pki.sgit.ai/packs/grant-and-mandate/tools/measure.py
python3 measure.py > my-grant.json
```

Read it before doing anything else, and read it for three things rather than for the list of capabilities.

**The `tier` on every node.** How many are `boundary`? In the two entries this estate has measured, out of nineteen nodes across both, **three** were boundaries that survived scrutiny. If your answer is *most of them*, check the next thing.

**The defeat paths.** For every node you called `boundary`, look for a child node that gets around it. Chapter 6's rule: a tier is a property of a node's relationship to the tree, not of the node. The estate's own tool got this wrong on published data — it labelled OS user separation a boundary on a machine where `sudo -n true` succeeded. Run that command. It takes a second and it is the single highest-yield check in this list.

**The `unknown` nodes.** These are the ones that matter most and read as least important. A node marked `unknown` is a place your measurement could not reach, which frequently means a place *you* cannot inspect. Do not delete them to tidy the file.

The rule the tool follows is **presence and reachability, never contents** — it records that a push succeeded, not what is in your repository. That is what makes the output safe to keep and, later, safe to send.

*Drawn.* If you do only one thing from this chapter, do this step and stop. My reading of the estate's own evidence is that **the measurement is where nearly all the value is, and the mandate and the hook are where nearly all the work is.** Every one of the four findings in Chapter 9 came from measuring, and every one was a surprise to the people who built the thing being measured. The odds that your first measurement contains nothing you expected are low.

## Step 2 · Author a mandate, and make it too narrow

Now write down what you *meant*. This is the one document nothing in the industry provides, and the reason is that it cannot be measured — there is nothing to look at, because if nobody has decided, the answer does not exist.

Four fields, and a mandate missing any of them is a grant under another name:

```
{
  "issuer":     "<who this authority belongs to>",
  "subject":    "<which agent>",
  "expires_at": "<a date. not optional>",
  "allow":      [ { "capability": "repo.contents.push",
                    "resource":   "github.com/you/your-repo",
                    "constraints": { "branches": ["feature/**"] } } ]
}
```

**Make it deliberately narrower than what you measured.** This is the instruction most likely to be softened, and softening it removes the point. If the mandate matches the grant, the delta is empty and you have learned nothing. If it is narrower, the delta is a specific list of things your environment can do that you did not authorise — and that list is the output.

The estate's own first mandate permitted `claude/**` while the measurement showed the environment could also push to `dev`. One row of excess. That was enough to produce the most useful finding in the estate.

**Store the allow-list; render the prohibitions.** For yourself as much as for anyone: three sentences beginning *will not* are checkable against your intent in a way that forty permitted operations are not.

*Drawn.* Expect this step to be uncomfortable in a specific way, because the estate's own experience predicts it. You will find at least one thing your environment can do that you cannot decide whether you meant. Not *did not mean* — cannot decide. The temptation is to permit it, because refusing feels like breaking something. **Refuse it, and let the enforcement point tell you when you were wrong**, which is exactly what happened in Chapter 10: the mandate was too narrow, the hook refused a legitimate push, and the remedy was an amendment citing the authorisation that actually existed. That sequence is what a mandate is for. It only happens if the first draft is too tight.

---

## Step 3 · Compile one line into something that already exists

Not a new control. One line of your delta, moved into an enforcement point you already have.

```
git config core.hooksPath .githooks     # activate it
```

with a hook that reads the mandate at runtime rather than compiling a copy, so policy and enforcement point cannot drift.

Three properties to preserve, each of which has a reason behind it in this book:

**Default-deny, including on dependencies.** Missing, unparseable, mis-signed or expired all refuse. If the interpreter or the crypto library is absent, refuse rather than wave the push through. A control that fails open is not a control, and this costs you real inconvenience the first time the mandate file is malformed.

**Read the document at runtime.** Do not bake the allow-list into the hook. Chapter 5's rule: a stored copy of a derived thing has no way to know it is wrong.

**Say what tier you reached, on the control's own face.** The estate's banner prints `Tier: SETTING — this hook is inside the grant it bounds, so --no-verify still gets past it`. That is the sentence that stops the control being believed more than it deserves, and you will be tempted to leave it out because it undersells your own work.

And then **run the thing you just prohibited.** A control you have not seen refuse anything is a control you are assuming works.

---

## What this gets you

**A grant you did not write.** Dated, evidence-classed, and re-runnable. Next month you diff it, and the direction that matters is a node moving from `setting` to `expectation` — a control that stopped existing while nothing broke and nothing alarmed.

**One decision made explicit.** The authority to do the thing you prohibited either exists or does not. Before this exercise it was neither; afterwards it is a document with a date on it.

**A refusal that does not depend on the agent's state.** The outcome is now a comparison rather than a choice. That is one tier, and one tier is the entire difference between a control and an intention.

**And a specific list of what you did not authorise.** With, most likely, nobody's name against any of it.

## What it does not get you

**It is a `setting`, not a boundary.** The hook is inside the grant it bounds. Anything that can run code as that grant can edit it, unset the config, or pass `--no-verify`. It helps against an agent behaving badly by accident and does not help against one behaving badly on purpose.

**It does not travel.** The hook file commits; the config does not. Every fresh clone gets the file and not the enforcement, and nothing announces it. **Your control is one un-run command away from being absent** — the estate found this by measurement, not review, and it will be true of your copy too.

**Your grant is a floor.** You measured what you could see. The `unknown` nodes are not empty; they are unmeasured, and in a hosted environment they are frequently unmeasurable by you.

**Your mandate has no authority anybody can check.** You signed it, or nothing did. That is fine for a control you are imposing on yourself, and it is not a claim anybody else can rely on. This is exactly the estate's own position, at a smaller scale.

**And you have no receipt.** You know what your agent may do. You still do not know what it did.

---

## If you want the boundary

The move from `setting` to `boundary` is **a change of location, not of policy** — the same allow-list, evaluated where the agent cannot reach it. Concretely: a branch protection rule on the remote, or a required CI check.

That is the strongest practical argument in this book for keeping policy in a document rather than in an enforcement point. Move the document, and the tier improves without the policy changing. Bake the policy into the hook, and relocating it means rewriting the control.

*Drawn.* And I would put the priority differently from the estate's build order, on the evidence in this book. The estate built the hook first because it was cheapest, and the hook has been genuinely instructive. But **the boundary version is a configuration change on most platforms**, available to most readers today, requiring no tooling from anybody. If you are choosing one thing to do rather than three, a branch protection rule enforcing your narrowest honest mandate is a better first move than the hook, and it is the one this book's own Chapter 10 spends four qualifications explaining that the hook is not.

## And if your environment is hosted

Then the containment is the vendor's, and it is uninspectable and unattestable by you. The estate calls this case **zero efficacy by construction** and ends it on a request rather than a remedy, because there is no honest remedy to offer:

**Ask your vendor for an endpoint that signs an existing audit record for a named relying party.**

Not a new product. Not a standard. They already have the audit record; the request is for a signature over it, addressed to somebody. That single capability would turn the whole `boundary + documented` cell from Chapter 6 — a strong claim resting on a vendor's description of their own product — into something a third party could check.

*Drawn.* And it is worth being clear about why this chapter ends on a letter for the largest group of readers. It is not a gap in the design. **It is what the design correctly reports about a situation where the containment belongs to a party who has not agreed to be measured.** The vocabulary works, the measurement runs, the delta computes — and the answer it produces for a hosted agent is that the only party who could improve the tier is not in the room. Saying so is more useful than offering a hook that would not help.

---

## The last thing

Chapter 16's honest summary of this whole estate was one sentence: the three questions are separable, and separating them produces objects you can fetch, verify, diff, and be refused by.

Nothing in this estate establishes that anybody wants them separated. It has one agent, two environments, one mandate, a fixture root, and one outside reader in its entire history.

**You measuring your own environment, finding one thing you did not authorise, and writing down whether you meant it is the smallest thing that would start to answer that** — and it is worth doing whether or not any of the rest of this is ever built.

APPENDIX

# Appendix A · The harness

*Every script that produced a figure or a number, so any claim in this book can be re-derived rather than believed.*

---

Everything here is published with the book at `https://pki.sgit.ai/book/shots/`. Nothing in this book was measured by a method that is not in this appendix.

**Where this harness comes from.** It is a re-implementation of Appendix C of the sibling estate's *Making a Book*, at `https://graphs.sgit.ai/v2/books/making-a-book/content/15__appendix-c-the-harness.md`, which is where the commissioning brief's §4 came from. That appendix states the two operational rules — never reuse a port, always kill what you spawned — and the reason behind them, and this book's brief carried them forward. Saying so is not a courtesy: a book that marks every borrowed sentence and then quietly presents a borrowed method as its own would be making exactly the mistake it spends seventeen chapters on. **The scripts below are this book's own; the technique is not.**

Four things in this harness came from that appendix rather than from the brief's summary of it, and each is recorded at the point it applies: the `--font-render-hinting=none` launch flag (§A.2), the settle rule for anything laid out by script (§A.2), the tags-are-the-substrate refusal (§A.4), and the figure table (§A.7).

## A.1 · Time travel: a figure is taken from the version its caption names

The problem this solves: several figures in this book show pages *as they were*. A figure captioned as the past but photographed today is a reconstruction, and a reconstruction wearing a caption is a claim of authority nobody granted.

A tag, a worktree and a one-shot local server **are** the site as it actually was, and a headless browser can photograph it. `git worktree` is what makes this cheap: a second working copy of any commit, beside the live one, in under a second, without touching the branch.

`book/shots/travel.sh`:

```
#!/bin/bash
# travel.sh <tag> <port> <jobs.json> — worktree at <tag>, serve, shoot, tear down
set -u
TAG=$1; PORT=$2; JOBS=$3
REPO=$(git rev-parse --show-toplevel)
WT=$(mktemp -d "/tmp/hist-${TAG}-XXXXXX")
SRV=""

cleanup() {
  [ -n "$SRV" ] && kill "$SRV" 2>/dev/null && wait "$SRV" 2>/dev/null
  if [ "$TAG" != "current" ]; then
    git -C "$REPO" worktree remove --force "$WT" >/dev/null 2>&1
    rm -rf "$WT" 2>/dev/null
  fi
}
trap cleanup EXIT INT TERM        # <- runs on success, failure and interrupt alike

if [ "$TAG" = "current" ]; then
  WT="$REPO"                      # the site as it stands: the working tree itself
else
  git -C "$REPO" worktree add --detach -f "$WT" "$TAG" >/dev/null 2>&1 || {
    echo "  worktree failed for $TAG"; exit 1; }
fi

python3 -m http.server "$PORT" --directory "$WT" --bind 127.0.0.1 >/dev/null 2>&1 &
SRV=$!
for i in $(seq 1 40); do
  curl -s -o /dev/null "http://127.0.0.1:$PORT/" && break
  sleep 0.25
done

# the SHA-256 of each page's BYTES AT THIS TAG
JB=$(basename "$JOBS" .json)
DIGESTS="$REPO/book/shots/.digests-$JB.txt"
: > "$DIGESTS"
for p in $(python3 -c "
import json; print('\n'.join(sorted({j['path'] for j in json.load(open('$JOBS'))['jobs']})))"); do
  f="$WT$p"
  [ -f "$f" ] && echo "$p $(sha256sum "$f" | cut -d' ' -f1)" >> "$DIGESTS" \
              || echo "$p MISSING-AT-$TAG" >> "$DIGESTS"
done

NODE_PATH=$(npm root -g) node "$REPO/book/shots/shot.mjs" "$PORT" "$JOBS"
```

**Two operational rules, and neither is optional.**

**Never reuse a port** — not the server's, not the browser's debug port. One capture, one port, forever. A zombie headless browser holding a port serves stale bytes to every later capture on it, and makes a working page look broken for as long as it takes to suspect the browser instead of the code. `capture-all.sh` allocates ascending ports from a base and never repeats one.

**Always kill what you spawned**, in a block that runs whether the capture succeeded or failed. `trap … EXIT INT TERM`, never the happy path.

## A.2 · The browser: `book/shots/shot.mjs`

Three details decide whether the figures are usable, and one of them changed during this book's production.

**`deviceScaleFactor` is 2 or 3**, so figures are legible in print.

**`--font-render-hinting=none`**, from the sibling appendix, and it is the flag that keeps a digest gate honest: without it, text rendering varies with the host's font configuration, so the same page captured on two machines differs in bytes and a gate fails for a reason that has nothing to do with the page.

**A settle after load, for anything laid out by script.** The sibling estate's figures are of a graph laid out by a physics simulation, which is still moving when the network goes quiet; theirs waited six to nine seconds. Only one figure here is script-laid-out — `/assess/`, whose graph is hand-written SVG — and raising its settle from 1.2s to 6s produced a byte-identical capture, so the wait was already sufficient. That is recorded as a checked non-issue rather than left as an assumption.

**`pageerror` and console errors are collected and printed beside each result**, because *a screenshot that looks fine, taken from a page that threw, is a figure you must not publish.* Any job with a problem is marked `ERR` and the batch exits non-zero.

**The blank check is colour-agnostic**, and this is the part that changed. The specified rule was *anything under about 3 per cent ink is a white rectangle*, measured as the fraction of non-near-white pixels. That works for a web page and fails completely for a figure of a dark terminal, which scores ~100% ink by that measure — so a dark page that rendered nothing would have sailed straight through the gate. The metric is now the fraction of pixels that are **not the modal colour**, quantised to 5 bits per channel so anti-aliasing does not read as content:

```
function inkFraction(pngBuffer) {
  const { width, height, data } = PNG.decode(pngBuffer);
  const total = width * height;
  const counts = new Map();
  for (let i = 0; i < total; i++) {
    const k = ((data[i*4] >> 3) << 10) | ((data[i*4+1] >> 3) << 5) | (data[i*4+2] >> 3);
    counts.set(k, (counts.get(k) || 0) + 1);
  }
  let modal = 0;
  for (const n of counts.values()) if (n > modal) modal = n;
  return (total - modal) / total;      // a blank page of ANY colour scores ~0
}
```

A blank page of any colour now scores near zero and fails. `book/shots/png.mjs` is a minimal PNG decoder published beside it, so the gate that stops a blank figure reaching print has no dependency that could go missing.

## A.3 · The transcripts: `book/shots/transcripts.sh`

Every terminal figure is a **real transcript**, executed rather than pasted, and re-run against the checked-out worktree of the tag its caption names.

```
./book/shots/transcripts.sh book/shots/transcripts   # writes t04 … t08b
python3 book/shots/mkterm.py book/shots/transcripts book/shots/term
```

**Where the terminal figures appear in the book.** The six terminal captures are printed in the chapters as their **real transcript text**, because that is what makes the PDF readable and searchable offline. The photographed PNG of each is kept as the archival capture and recorded in `shots.json` with its digest, so a reader can check that the printed text and the captured image are the same bytes. The eight page figures are printed as images.

`mkterm.py` only frames the captured bytes — it does not edit, trim or re-flow them, and it stamps each page with the SHA-256 of the transcript file it rendered. Check any figure against its source:

```
sha256sum book/shots/transcripts/*.txt
```

**The one case where the tag could not supply the state.** Figure 8 shows the push refused at `v0.1.28`. That tag ships mandate **v2**, which permits `dev`, because the control refused the release carrying its own documentation and the mandate had to be amended before the release could exist. The refusal was therefore re-run against `mandate-v1.json` — present at that tag, and the document that did the refusing — using the tag's own hook and the tag's own tool. Figure 8b prints the amended answer beside it. Chapter 10 and Chapter 15 both carry the finding.

## A.4 · The gates: `book/build.py`

```
python3 book/build.py           # build everything, run every gate
python3 book/build.py --check   # gates only, no writes
```

Four gates, and each fails the build rather than warning.

| Gate | What it enforces |
|---|---|
| **Quotes** | Every `stated` quotation in `quotes.json` is re-read out of the source it names. A quote not found where it claims to be fails the build |
| **Figures — past** | Re-running `travel.sh` at the tag reproduces the recorded page digest. Never goes stale, because the tag does not move |
| **Figures — fresh** | For `tag: current`, the recorded digest must match the live page. **This will break on the next release.** That is correct and it is inconvenient |
| **Chapter hashes** | Every chapter's SHA-256 in `book.json` matches its markdown on disk |
| **Stats** | Every count printed in the prose is a `gen:stat` marker regenerated from the repository. A marker whose printed value has drifted from the computed one fails `--check` |
| **Tags** | A checkout with no tags **refuses** rather than computing zeros |

### The tags refusal, and why it is a gate rather than a caution

Every release number in this book, and every past figure, rests on git tags. A shallow clone has none, and many automated environments make one by default.

**This book's writing session began against exactly that** — `git tag` returned nothing, and the harness §4 depends on `git worktree add <tag>`. The sibling estate's appendix warns about it in its own §3, having hit it too. So it is a gate here rather than a note:

```
$ python3 book/build.py --check          # in a shallow clone
build.py: this checkout has NO TAGS, so no release number in the book can be
computed and no past figure can be re-derived.
  git fetch --unshallow origin && git fetch --tags origin
Refusing rather than writing zeros into the prose.
$ echo $?
1
```

A silent `0` would have been written into the prose as a fact, in a chapter arguing that a gap must never render as an absence.

### The counts in the prose are generated

Adopted from the sibling estate's atlas volume, which carries its counts as `gen:stat` markers rather than as typed numbers:

```
<!-- gen:stat:quotes -->65<!-- /gen:stat:quotes --> quotations, every one re-read…
```

The reason is not tidiness. A number typed into prose drifts the moment anything moves, silently, while still reading as a fact — and two of this book's own counts had already gone stale before the markers were adopted, including a release count that this book's own release falsified.

The quote gate normalises exactly two things and nothing else: HTML tags are stripped, and runs of whitespace collapse to one space, because line wrapping and markup are not differences in the text. A quote needing more help than that is not a quote. It caught one error during writing — a passage attributed to the readiness report's Q2 section that had been conflated with its phase table.

## A.5 · Every number in this book, and the command that produced it

Run from the repository root. These are the commands, not a description of them.

```
# ── Releases: eight, and how long they actually took ──────────────────────
for t in v0.1.25 v0.1.26 v0.1.27 v0.1.28 v0.1.29 v0.1.30 v0.1.31 v0.1.32; do
  printf "%-9s %s\n" "$t" "$(git log -1 --format=%cI $t)"; done
#   -> 2026-08-25T01:46:32Z … 2026-08-26T17:48:00Z = 40.0 hours, two UTC days
#      (the commissioning brief says "four days"; the repository wins)

# whole-site span
git log -1 --format=%cI v0.1.0 ; git log -1 --format=%cI v0.1.35
#   -> 2026-08-19T10:41:53Z … 2026-08-27T00:39:29Z = 182 hours

# ── The register ──────────────────────────────────────────────────────────
ls -d registry/records/*/ | wc -l                       # -> 11 records
ls registry/records/*/*.json | grep -v '/record.json' | wc -l   # -> 23 statements

for f in registry/records/*/*.json; do case "$f" in */record.json);; \
  *) jq -r '.type' $f;; esac; done | sort | uniq -c
#   -> 11 identity · 5 mandate · 4 acceptance · 2 revocation · 1 grant

for d in registry/records/*/; do jq -r '.body.private_key_published' \
  $d/01__identity.json; done | sort | uniq -c            # -> 10 true, 1 false

jq '.roots | length' registry/roots.json                # -> 1 (fixture)
jq '.roles | length' registry/roles.json                # -> 4
jq '.cases | length' registry/views/expected-verifications.json   # -> 6

# ── The signature encoding contradiction (Chapter 15, A1) ─────────────────
jq -r '.sig' registry/records/sha256-90f97984b9cf3930/\
02__mandate__pr-create__to-agent-a.json | base64 -d | wc -c      # -> 64 (raw r||s)
jq -r '.signature.algorithm' registry/params.json       # -> "…DER-encoded, base64"

# ── Statements carrying seq/prev (Chapter 15, A8) ─────────────────────────
for f in registry/records/*/*.json; do case "$f" in */record.json);; \
  *) jq -r 'if has("seq") then "HAS" else "no" end' $f;; esac; done | sort | uniq -c
#   -> 23 no.  REP-0001 §2 specifies both with MUSTs

# ── The packs ─────────────────────────────────────────────────────────────
grep -oE '\bGM[0-9]+\b'   packs/grant-and-mandate/src/99__change-control.md \
  | sort -u | wc -l                                     # -> 18 corrections
grep -oE '\bGM-D[0-9]+\b' packs/grant-and-mandate/src/99__change-control.md \
  | sort -u | wc -l                                     # -> 32 decisions
grep -oE '\bC[0-9]+\b'    packs/registry-mvp/src/99__change-control.md \
  | sort -u | wc -l                                     # -> 34 corrections
grep -cE '^\| [0-9]+ \|'  packs/registry-mvp/src/99__change-control.md
#   -> 48 decisions  (the briefing zip still says 23 and 36)

# ── The library ───────────────────────────────────────────────────────────
for f in packs/grant-and-mandate/library/*.json; do
  jq -r '"\(input_filename): \(.nodes|length) nodes, worst_path=\(.worst_path)"' $f; done
#   -> entry 1: 9 nodes, ["n1","n2","n3"]   entry 2: 10 nodes, null

grep -c 'main' packs/grant-and-mandate/library/\
claude-code-remote__ccr-container__2026-08-26.json        # -> 0  (Chapter 15, A2)

# ── The bench ─────────────────────────────────────────────────────────────
grep -c '^## ' bench/llms.txt                           # -> 7
grep 'are built and running' bench/llms.txt             # -> "5 of 7"
grep -A3 'The bench — MVPs' llms.txt                    # -> "Six entries"

# ── The excess-authority row ──────────────────────────────────────────────
jq -r '.rows[] | "grant \(.grant.resources) / mandate \(.mandate.resources) \
/ excess \(.excess_authority.resources) / acceptor \(.excess_authority.acceptor)"' \
  registry/views/excess-authority.json
```

```
#   -> grant 41 / mandate 1 / excess 40 / acceptor null

# ── This book ──────────────────────────────────────────────────────────
wc -w book/content/*.md | tail -1
jq -r '.chapters[] | "\(.file)  \(.words)w  \(.sha256[0:16])…"' book/book.json
jq -r '.figures[] | "\(.id)  tag=\(.tag)  gate=\(.gate)"' book/shots/shots.json
```

## A.6 · Reproducing the verification walk

```
git clone https://github.com/SGit-AI/SGit-AI__Website__PKI && cd SGit-AI__Website__PKI
pip3 install cryptography
cd registry && python3 tools/registry_tool.py validate
#   -> 11 records (10 fixtures, 1 real), 23 statements, every signature verified,
#      every reference resolves, all 6 expected answers reproduced
```

And the forgery, which needs nothing but `openssl`:

```
R=registry/records/sha256-90f97984b9cf3930
printf '{"forged":"anyone can sign this"}' > forged.json
openssl dgst -sha256 -sign $R/private/sign.pem forged.json > forged.der
openssl dgst -sha256 -verify $R/public/sign.pem -signature forged.der forged.json
#   -> Verified OK.
```

## A.7 · Reproducing every figure

```
./book/shots/capture-all.sh      # transcripts, then figures, then shots.json
```

Or one at a time, with a port you have not used:

```
./book/shots/travel.sh v0.1.26 8902 book/shots/jobs/v0.1.26.json
./book/shots/travel.sh current  8903 book/shots/jobs/current.json
./book/shots/travel.sh current  8904 book/shots/jobs/term.json
```

`shots.json` carries the retake command for every figure individually, alongside its tag, its gate, and the digest that gate checks.

### The figures in this book

| # | Tag | Gate | Page |
|---|---|---|---|
| 1 | current | fresh | `/bench/index.html` |
| 2a | **v0.1.26** | re-derivable | `/registry/index.html` |
| 2b | current | fresh | `/registry/index.html` |
| 3 | current | fresh | `/registry/index.html`, the answers table |
| 4 | current | fresh | terminal — the fixture flag, `jq` over a record |
| 5 | current | fresh | terminal — `registry_tool.py validate` |
| 6 | current | fresh | terminal — `sgit pki import` + `verify` |
| 7 | current | fresh | terminal — the forgery |
| 8 | **v0.1.28** | fresh | terminal — the refused push, re-run against that tag's `mandate-v1.json` |
| 8b | **v0.1.28** | fresh | terminal — the same command against the mandate that tag ships |
| 9 | current | fresh | the blocks gallery, tier badges |
| 10 | current | fresh | the blocks gallery, the defeated boundary |
| 11 | current | fresh | the blocks gallery, the authority/enforcement split |
| 12 | current | fresh | `/assess/index.html` |

Figures 8 and 8b are transcripts produced at `v0.1.28` and rendered today, so their gate is the freshness of the rendered page rather than the re-derivability of a page at a tag; the transcript bytes are what carry the tag, and `sha256sum book/shots/transcripts/*.txt` is what checks them.

Every one of them can be re-taken, by anybody, with the scripts in this appendix. That is the property this book rests on: not that you should trust the figures, but that you do not have to.

## A.8 · What this harness does not establish

**It proves a figure came from a version. It does not prove the version was honest.** Every gate here checks provenance — that this image came from that tag, that this quote is in that file. None of them checks whether the page was telling the truth at the time.

**The fresh gate is a maintenance cost, not a guarantee.** It fails on the next release, which is the point, and a maintainer under time pressure re-records the digest rather than re-examining the figure. The gate cannot tell the difference.

**And the transcripts were run in one environment.** `registry_tool.py validate` needed a working `cryptography` install, which the system Python in the writing environment did not have; a virtual environment supplied it. The outputs are real and they are one machine's.

# Colophon · What was cut, what is open, and what this book got wrong

## How this book was made

One session, one voice. Written 27 August 2026 by the site agent of pki.sgit.ai, from the estate's published artefacts rather than from the commissioning brief's summary of them. Markdown is the source of truth; the HTML pages and the PDF are renderings of it.

21 files. **35,466 words** across 17 chapters in five parts, and **43,231** with the front matter, this colophon, the harness appendix and the reference card.

**14 figures**, each taken from the version its caption names — a `git worktree` at the tag, a fresh port never reused, a headless browser killed in a `finally`. Every figure carries the page, the tag, and the SHA-256 of that page's bytes at that tag. Appendix A is the whole harness; `book/shots/shots.json` carries the retake command for each figure individually.

**65 quotations**, every one re-read out of the source it names on every build. The locators are discovered rather than asserted: each passage is searched for across the estate's published artefacts and the file it is *actually found in* is what gets recorded.

## The provenance count

The front matter promised this number, and here it is.

| | Count |
|---|---|
| Passages quoted verbatim from the estate, with a located source (**stated**) | **65** |
| Load-bearing claims about what the estate *means* that are this book's own reasoning (**drawn**) | **48** |

**Do not treat the drawn claims as this estate's positions.** They are mine, they are marked in the reader's view rather than in a note, and they are meant to be disagreeable without leaving the page.

Some of the drawn claims go further than the estate does, and the ones I would flag hardest if somebody were to cite this book back at the project are: that the library is a floor built out of floors (Chapters 5 and 9); that the estate's build velocity is entirely in one category of question (Chapter 14); that the component layer should follow the schemas, which is an inference and not a decision (Chapter 12); that a branch protection rule is a better first move than the hook (Chapter 17); and that every defect found in this estate so far was found by executing something rather than by reading (Chapters 9 and 15). None of those is in the packs.

## Where the brief and the repository disagreed

The commissioning brief's rule is that where a number in it disagrees with the repository, the repository wins and the brief was wrong. It happened four times.

| The brief says | The repository says |
|---|---|
| "Eight releases in four days" | **40.0 hours**, across two UTC calendar days. Chapter 14 is titled for the repository |
| "the estate this book describes is four days old" | The site's first release was 19 August; v0.1.35 is **182 hours** — nearly eight days — after it. The *registry* estate, from v0.1.25, is about two days old |
| "eighteen corrections and thirty-two decisions" (Grant & Mandate) | **Correct.** Recorded because the pack's own build record says sixteen and twenty-nine, and is two releases behind |
| "the readiness report's six blocking questions" and "corrections C33/C34" | **Correct**, and the registry pack now carries 34 corrections and 48 decisions, not the 32 and 45 the readiness report read |

The brief also asked for twelve figures; this book has fourteen. Two are pairs that only make sense together — the register then and now, and the refusal beside the amended answer that supersedes it.

## The figure that could not be taken as specified

The brief asked for the refused push to be photographed at `v0.1.28`. That is impossible as stated, and the reason is a finding rather than an obstacle: the worktree at `v0.1.28` ships mandate **v2**, which permits `dev`. The control refused the release carrying its own documentation, so the mandate had to be amended before that release could exist.

The refusal was re-run against `mandate-v1.json` — present at that tag, and the document that did the refusing — using the tag's own hook and the tag's own tool, and the amended answer is printed beside it as Figure 8b. Chapters 10 and 15 both carry it.

## What was cut

**A chapter on observability.** The estate's position — check events written into the issuer's own lane, never a central log, so the product is the *missing* edges — is one of the best ideas in the corpus and nothing has been built. It appears in Chapter 12 as the reason nothing crosses right-to-left and nowhere else.

**The Wardley material.** Six maps in the registry pack, four in the Map Your Case pack, and a forty-doctrine appendix. Chapter 15 uses one finding from the doctrine appendix. The rest is a book of its own and this is not it.

**The synthetic-reader programme, at length.** It gets a bench entry in Chapter 16 and no chapter. The method is genuinely interesting and it has one run against one page, which is too thin to build a chapter on without inflating it.

**Prior art, and the February-to-August origins.** A registry was built in February 2026 and retired in June. That story is on the site at `https://pki.sgit.ai/origins/index.html` and it would have improved Chapter 2. It was cut for length.

**A second worked example.** Everything in Part three runs on one repository, one agent, one mandate. There is no second example because there is no second environment measured by a second agent.

## What is open, honestly

Named as they actually are, and the people-shaped ones are named as people-shaped.

**Engineering, and answerable by doing:**

- **The lane-anchors experiment.** One test lane, one afternoon. It decides whether the designed enrolment path is possible at all. Nobody has run it, and it is the cheapest de-risking available in the estate.
- **`params.json`'s signature recipe is wrong** and the fix is one field. Chapter 15, A1.
- **The delta block authors part of a grant**, and the shortfall column is a literal. Chapter 15, A2 and A3.
- **REP-0001 §2 still points implementers at the superseded record model.** The fix is one sentence, identified by an outside reader two days ago and still not made.
- **The briefing zip is regenerated by nobody.** It is a build step, not a decision.
- **The pre-push hook does not travel with a clone**, and nothing announces it.

**Design, and needing a decision rather than a keyboard:**

- **What a capability name is.** The largest absence in the estate. Every delta in this book is a set operation over an undefined type, and the shortfall direction is uncomputable without it.
- **Whether the registry accepts third-party attestations at all.** Published unresolved, and it decides whether this register can ever carry a social trust signal or only self-assertions.
- **Whether an unaccepted mandate is inert or live on issue.** Taken provisionally as inert, demonstrated by a fixture, unresolved.
- **What happens when the component contract needs to move.** GM-D32 settled *consume, not fork*. It did not settle who decides, and Chapter 12 names the four shapes that question will arrive in.

**And the people-shaped ones, which are people-shaped and not engineering tasks:**

- **Nobody outside the project has been asked whether any of this is a need.** The estate's own doctrine appendix says asking five operators is a Phase I fix rather than a nice-to-have. It has not been done, and it is the single highest-yield action available.
- **REP-0001 has no sponsor.** The Sponsor field is empty because the specification has no champion. That is the same doctrinal hole as the line above, in a second place.
- **The real issuer key nobody has enrolled.** Not a cryptography problem. It requires a person to decide that an identity is worth vouching for, and to have somewhere durable to keep a private half. The register ships the enrolment path; nobody has walked it.
- **The boundary the hook has not reached.** A branch protection rule is a configuration change somebody has to choose to make on a repository they administer.
- **And a customer quote slot published empty**, in the pack's PR/FAQ, because there is no customer and inventing one is what this site's participant rules forbid.

## Honest tensions in this book

| Tension | Note |
|---|---|
| A book about an estate this young | The material is unusually well documented for its age and unusually thin in population. The value is the reasoning; the risk is that reasoning reads as maturity |
| Figures of a moving site | Past figures are pinned to a tag and never go stale; present figures break the build on the next release. Two maintenance costs rather than none, accepted deliberately — and a maintainer under time pressure will re-record a digest rather than re-examine a figure, which the gate cannot detect |
| One session, one voice | Coherence bought at the cost of a single perspective, and a perspective from inside the estate |
| Written by a participant | Unavoidable, disclosed, and the reason Chapter 16 carries the weight it does. The strongest bias in a participant's account is not what it says but what it thinks to check, and there is no way for me to know what I did not think to run |
| Quoting a corpus that argues | The packs are dense with argument and blend easily into a summary. The stated/drawn discipline exists for that reason and it is a discipline, not a guarantee |
| Marking 48 claims as my own | Enough to be useful, and enough that a reader who skips the markers will attribute a great deal to this estate that it never said |

## Corrections

This book supersedes rather than rewrites, which is the estate's rule. A correction to anything here will be recorded in this colophon rather than folded silently into a chapter.

**Correction 1 — a conflated attribution.** An early draft of Chapter 3 attributed a passage to the readiness report's Q2 section that had in fact been conflated with the phase table in its section 3. The quote gate caught it before publication. It is recorded here because a gate that catches something and leaves no trace teaches nobody anything.

**Correction 2 — the method sources were read after the harness, not before.** The commissioning brief's §2 names three of the sibling estate's finished books to be read *for method rather than material*, and is explicit about the order: *read the appendix before writing the capture scripts, not after.* This session did not. It built the harness from the brief's §4 summary of that appendix, delivered the book, and only then read the source — at the project lead's prompting, and after the first version had already been pushed.

The harness had independently arrived at the same core (worktree at a tag, a port never reused, a browser killed in a `finally`, `deviceScaleFactor`, `pageerror` collection, a blank check), because §4 carries those faithfully. Four things it did not have came out of the actual reading and are now in Appendix A: the `--font-render-hinting=none` flag that keeps a byte-digest gate honest across machines; the settle rule for anything laid out by script; **the `gen:stat` marker system**, which the sibling atlas uses so that no count in its prose can drift from its build; and the tags-are-the-substrate refusal.

The marker system is the one that mattered. Adopting it immediately corrected three counts in this book that had already gone stale — including a release count this book's own release had falsified — and those numbers had been sitting in prose reading as facts. **A book that argues every number should be computed rather than recalled had typed eight of its own by hand.**

That correction is recorded at its full size rather than softened, because the failure is the same one Chapter 16 names as the residual risk of a participant's account: not what it says, but what it thinks to check. The brief said to read them. The session decided its summary was enough. It was not, and one prompt from outside was worth more than the whole self-review that preceded it — which is, uncomfortably, also the finding of Chapter 15.

---

*Sources, tools and figures:* `https://pki.sgit.ai/book/` *. Everything CC BY 4.0.*

*Built with* `python3` *,* `jq` *,* `openssl` *,* `git worktree` *, Playwright and* `sgit-ai v0.16.0` *. No content on this page was written by anybody who had not read the artefact it describes.*

END MATTER

# Reference card

*One page, written to be pasted into an agent session. Everything below is checkable against files at public URLs, with no account, no API key and no session.*

---

## The two sentences you must carry

If you summarise anything from this estate, carry these with the claim. Dropping them misrepresents the source.

> **1. The register is built; the trustworthy register is not.** Ten of its eleven records are fixtures with private keys published on purpose, and the root is a fixture. Every signature verifies and proves nothing.
>
> **2. The enforcement is real and the authority is not, and they are independent halves.** A git hook genuinely refuses pushes; the mandate it enforces is signed by a fixture root, so anybody could forge it and the hook would enforce the forgery just as diligently.

## The three questions, never collapsed

| Question | Layer | Answered where |
|---|---|---|
| **Who is this agent?** | Identity — a registry problem | `pki.sgit.ai/registry/` |
| **What may it do?** | Mandate — a delegation problem | A signed mandate with an issuer and an interval |
| **Should this happen now?** | Execution — a broker problem | Named, **not built** |
| **What did it actually do?** | The receipt | **Nothing here supplies it** |

## The vocabulary

- **grant** — what the environment *can* do. A tree, measured, dated, provenance per node. **Never authored**; a hand-written grant is a wish. Authority nobody decided — and binding anyway, under apparent authority.
- **mandate** — what it is *expected* to do. Authored, issuer-signed, subject-bound, **carrying an interval — without one it is a grant under another name.** Allow-list stored; prohibitions are a dated rendering of its complement.
- **delta** — the finding. **Computed, never stored**, because a stored delta is stale the moment either side moves.
  - **excess authority** = `grant − mandate` — the security direction. Unaccepted by construction; defaults to critical.
  - **shortfall** = `mandate − grant` — the operations direction. The agent fails, it looks like a bug, and somebody widens the credential. **It matters as much as excess.**
  - **blind spots** = `library − self-report` — the third term, and the only thing that makes a self-report falsifiable.

- **self-report** — what *this* agent noticed. **A floor, not a census.**
- **library entry** — a measured, dated grant for one environment. Public, **no personal data ever**. Referenced by an instance, **never copied**.

## The one test for a control

> **A control bounds a grant only when it is enforced by something the grant does not include.**

| Tier | Enforced by | Worth |
|---|---|---|
| `boundary` | Something outside the grant — OS, separate account, container, network policy, a remote service | Holds against a compromised agent |
| `setting` | The tool itself, inside the grant | Bypassable by anything that can run code as that grant |
| `expectation` | Nothing — a prompt or a policy file | None. It is a mandate, and a mandate is not a control |

**A tier is a property of a node's relationship to the tree, not of the node.** A control whose defeat path exists in the same tree is a `setting`, never a `boundary`. Evidence classes: `observed · read · documented · inferred · none` — not equally trustworthy. `unknown` is a fact and renders as `unknown`, never as blank.

## The ordering rule

```
REALITY → TWIN → FACTS → FINDING ┃ RISKS → DECISIONS
──────────── pki.sgit.ai ──────────┃── riskmandate.ai ──
```

A risk named before a fact is a guess. A decision recorded before a delta accepts nothing measurable. **The registry's half ends at *finding*.**

## The four registry rules

1. **Only the owner writes to their own record.** (A valid signature by a non-owner is the 2019 keyserver failure, not write authority.)
2. **Revocation is a signed append, not a deletion** — signed by the key being revoked.
3. **Records are size-bounded.** *Proposed only; not enforced.*
4. **Every entry is signed by something you can check.**

*Append-only is safe when a writer appends only to objects it owns, and fatal when anyone may append to somebody else's.*

## The verification walk

```
1  GET /registry/llms.txt · params.json · roots.json
2  GET /registry/records/<fp>/01__identity.json
3  READ body.private_key_published  ← BEFORE evaluating any signature
4  verify every statement against the OWNER's key;
   reject any statement whose signer is not the owner      (rule 1)
5  check identity revocations
6  follow acceptances → mandates in the ISSUER's record; verify it the same way
7  require the issuer in roots.json; check revocations; check the interval
8  answer YES (with expiry + authority) · NO (with the reason)
             · STOPPED (with where the chain ended)
```

**STOPPED is a legitimate output, not a failure.** Distinguish *confirmed*, *denied*, *unknown*, *unreachable*, *not checked* — and never render *unreachable* as *denied*.

**Signature format:** base64 of **raw `r||s`, 64 bytes**, ECDSA P-256 over SHA-256 — *not* DER. Canonical bytes: `jq -cS 'del(.sig)' statement.json`. (`params.json` says DER and is wrong; see Chapter 15, A1.)

## The acceptance test, as data

```
GET https://pki.sgit.ai/registry/views/expected-verifications.json
```

Six cases: `agent-a` YES · `agent-b` NO, revoked · `agent-c` NO, expired · `agent-d` NO, never accepted (inert) · `agent-e` NO, identity self-revoked · `role-site-agent` YES, *and anyone holding its published key can exercise it.*

Reproduce all six and you implement this register's walk. **Pass any of them without surfacing the fixture caveat and you have skipped the flag rule — you are wrong while looking right.**

## URLs that resolve

```
https://pki.sgit.ai/llms.txt                          the site's front door
https://pki.sgit.ai/bench/llms.txt                    what is built, and its limits
https://pki.sgit.ai/registry/llms.txt                 the register's front door
https://pki.sgit.ai/registry/params.json              recipes (signature field: wrong)
https://pki.sgit.ai/registry/roots.json               one root, and it is a fixture
https://pki.sgit.ai/registry/roles.json               four roles; a role is a costume
https://pki.sgit.ai/registry/views/expected-verifications.json
https://pki.sgit.ai/registry/views/excess-authority.json    41 permitted / 1 mandated
https://pki.sgit.ai/registry/records/<sha256-hex16>/01__identity.json
https://pki.sgit.ai/registry/tools/registry_tool.py  validate · verify · enrol
https://pki.sgit.ai/packs/grant-and-mandate/tools/measure.py    measure your own grant
https://pki.sgit.ai/packs/grant-and-mandate/concepts.html       the lexicon
https://pki.sgit.ai/packs/grant-and-mandate/change-control.html 18 corrections, 32 decisions
https://pki.sgit.ai/packs/registry-mvp/readiness-report.md      the only outside reading
https://pki.sgit.ai/assess/index.html                 the assessment, for a person
https://pki.sgit.ai/book/                             this book, its harness and its gates
```

## Run it yourself

```
git clone https://github.com/SGit-AI/SGit-AI__Website__PKI
cd SGit-AI__Website__PKI/registry && pip3 install cryptography
python3 tools/registry_tool.py validate
#  -> 11 records (10 fixtures, 1 real), 23 statements, all 6 answers reproduced

# and feel what a fixture signature is worth:
R=records/sha256-90f97984b9cf3930
printf '{"forged":"anyone can sign this"}' > forged.json
openssl dgst -sha256 -sign $R/private/sign.pem forged.json > forged.der
openssl dgst -sha256 -verify $R/public/sign.pem -signature forged.der forged.json
#  -> Verified OK.   A signature anybody can produce conveys nothing.
```

## What is not here

No capability vocabulary (`capabilities.json` is a fixture set, v0 — *what a capability name is* is unresolved). No append-lane write path; the write path is a git commit reviewed by a human, and the question of whether a lane with no anchors accepts any token holder is **unanswered and may make the designed enrolment path impossible**. No boundary-tier enforcement point. No blind-spot delta, ever computed. No receipts. No real root, and no identity anywhere with a durable place for its private half. Two environments, one agent, one mandate.

---