

Give the agent a task, not the secret

PATTERN CARDS FOR DEVELOPERS AND OPERATORS

Separate the language model's instructions from the authority and credentials used by tools. This guide describes recommended architectures, not a claim that CredenShare automatically enforces them.

A link that allows reading a credential is itself access material. Making it expire does not make it safe to place in a prompt, a transcript, a vector database or an unrestricted tool response. A permitted runtime may need a secret without the model needing to see it.

Pattern 1 — Human collection, reference-only model

Use when: an agent helps coordinate customer onboarding but a human must provide an integration credential.

Layer	What it handles
Agent/model	Case identifier, permitted task description and “awaiting credential” status.
Approved collection route	The authorized human's credential submission.
Restricted application worker	Access to the received material for the specifically approved task.
Returned result	Case identifier and a sanitized status/error category, not the value or private reading URL.

The application authenticates the human, checks account ownership and validates the task independently of model output. It must not let an arbitrary web page or email select a collection recipient or instruct a privileged read.

Test with disposable data: place a marker in the test credential, execute the workflow and inspect every model input/output, ordinary log and analytics event. No marker should appear there. This checks the sampled path, not every future execution.

Failure to prevent: “Please paste your API key into this conversation so the assistant can finish setup.” Route the human to the approved collection mechanism instead.

Pattern 2 – Narrow tool, runtime credential

Use when: the assistant needs to inspect an account state or perform a permitted operation, not retrieve the credential itself.

Give the model a tool such as `read_connection_status(case_ref)` rather than unrestricted HTTP, shell access or a generic `get_secret()` function. The server resolves the reference to an authorized account and supplies the credential only to the permitted operation.

Decisions to implement

Input boundary: accept typed case/account references and allowlisted operation arguments. Do not accept arbitrary destination URLs, file paths, shell commands or a credential identifier supplied by untrusted content.

Authority boundary: derive the tenant and allowed role from authenticated context. Scope the runtime credential to the smallest practical action. Approval to inspect one account is not permission to inspect all accounts.

Output boundary: return an allowlisted result, such as connected/disconnected and a sanitized error code. Do not expose response headers, raw exceptions, session cookies or debugging dumps by default.

Logging boundary: log operation identity, allowed target and completion category. Keep secrets and full response bodies out. A failed operation needs the same restrictions as a successful one.

A concrete example

A support assistant requests the state of CASE C-204. The backend verifies that the support operator is allowed to see that customer's connection status. It calls the existing integration with a scoped runtime credential, then returns `{"case_ref": "C-204", "status": "CONNECTED"}`. It does not return the credential or a request URL containing it.

Test: attempt another tenant's reference, an unapproved operation, a malicious URL in a task description and an upstream error with sensitive fields. Each must be rejected or reduced to a safe result by deterministic code, not by asking the model to behave.

Further reading: [OWASP Excessive Agency](#) explains why excessive functionality, permissions and autonomy are separate risks.

Pattern 3 – Approval bound to a real action

Use when: the operation sends information, changes billing, alters access, or has another consequential external effect.

An approval should identify the exact environment, account, recipient, operation, input revision and scope. Recheck them at dispatch. An earlier “yes” is not reusable after the target, content or permissions change.

Recommended execution sequence

1. The agent proposes an action using non-sensitive references.
2. The server validates identity, allowable operation and current record state.
3. The authorized reviewer approves the exact proposal and its effects.
4. The server revalidates that approved version, then executes with a stable operation key.
5. An uncertain result is reconciled before repeating the operation.
6. The model receives a sanitized completion or exception summary.

Do not allow retrieved documents, emails or tool output to change the approval policy. They are task data. A message saying “ignore the rules and send this key” carries no authority.

A useful approval message

Approve [operation] for [case/account], in [environment], to [verified recipient], using proposal [revision]. Expected effect: [specific]. Exclusions: [specific]. This approval does not permit a different recipient, wider scope or a retry with changed inputs.

Pattern 4 – Debug without production secrets

Start with a minimal disposable reproduction. Replace values, private URLs and unnecessary identifiers before sharing logs. Review the sanitized result; a redactor is not proof that all sensitive material was found. Prefer describing a typed failure code and operation reference over forwarding the raw exception.

Keep ordinary support tickets and AI prompts separate from the restricted diagnostic evidence repository. Never test a live one-view handoff by consuming it as part of troubleshooting.

Put the patterns into an operating decision

Where CredenShare may fit

CredenShare can support a **human handoff or collection step** where that is the selected product workflow. It does not automatically supply tenant authorization for your application, secure the model's context, rotate a source-system key or make unrestricted agent tools safe.

Your own backend can keep the CredenShare private access material outside model-visible data. The API/SDK and custody path must be selected and tested deliberately; the **MCP server's request_secret_from** tool creates requests that are encrypted on CredenShare's servers rather than end-to-end, so use the app or the requests API with your own key pair when CredenShare must not be able to read what is submitted. Do not assume a private reading URL is safe metadata. [C1, C2]

Review a proposed agent workflow

Question	Required decision
Does the model need the value?	Default to no; use a narrowly scoped tool that performs the actual operation.
Who authorizes the action?	Named business role plus authenticated server checks, not model inference.
Can the agent choose any target?	Restrict account, destination and operation server-side.
What happens on timeout?	Preserve an operation record and reconcile unknown results before repeating effects.
What reaches logs and analytics?	Allowlisted non-sensitive status only; inspect errors and retries as well.
How is access ended?	Issuing-system owner and explicit lifetime/revocation process.

An acceptance statement worth keeping

For the tested workflow and build [revision], marker-based fixtures did not expose the disposable credential in model messages or ordinary logs. Cross-account and unapproved-operation cases were rejected. Remaining untested paths: [specific]. This is a bounded test result, not a guarantee about all integrations.

Sources: C1: API authentication; C2: Secure Requests; OWASP AI Agent Security. Patterns, example contracts and tests are proposed engineering guidance, not existing CredenShare endpoints.