EcomLabs

STEP-BY-STEP GUIDE

Protect your .env from AI coding agents

The settings.json deny rules that stop Claude Code from reading your secret files, what they cover and what they miss, where each key should live, and the checklist for rotating a key that leaked.

FOR	Projects where you work with Claude Code and keep API keys in a .env file
TESTED WITH	Node 22.22.3 · Ajv 6 against the SchemaStore schema · git 2.54 · local help of Google Cloud SDK 581.0.0
UPDATED	November 19, 2026
DOCS	code.claude.com/docs/en/permissions
ARTICLE	ecomlabs.dev/g/llaves-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What a Read deny rule blocks
Step 01	The project settings.json
Step 02	A safety net for every project
Step 03	Check that it works
If it leaked	Rotation checklist
When you finish	Checklist

BEFORE YOU START

What a Read deny rule blocks

Claude Code reads files in your working directory without asking, and commands such as `cat` or `grep` run without a prompt. A `Read` deny rule changes that for the paths you name. Per the permissions docs (checked on November 19, 2026):

- **It covers** Claude's file tools (Read, and on a best-effort basis Grep and Glob), `@` mentions, IDE context, the commands Claude Code recognizes in Bash (`cat`, `head`, `tail`, `sed`, `tee`) and redirects such as `< .env`. It also blocks Edit and Write on the same path.
- **It does not cover** a command that reads without naming the file, such as `grep -r pattern .`, a Python or Node script that opens it itself, or variables already in your terminal's environment: do not start the agent inside `railway run` or `railway shell` linked to production, and consider `CLAUDE_CODE_SUBPROCESS_ENV_SCRUB=1`.
- **Where each key lives:** names without values in `.env.example`; test keys in `.env.local`, out of git; real keys only in the secret manager (Railway sealed variables, Google Cloud Secret Manager).
- The docs do not mention PowerShell's `Get-Content`. On Windows, add the PreToolUse hook from `ecomlabs.dev/g/hook-en`.
- A `.claudeignore` file has no effect, and `.gitignore` only changes the `@` file picker.

STEP 01

The project settings.json

Save this as `.claude/settings.json` at the repo root and commit it, so the whole team gets the same rules. Validated with Ajv against the SchemaStore schema:

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "deny": [
      "Read(.env)",
      "Read(.env.*)",
      "Read(!.env.example)",
      "Read(*.pem)"
    ]
  }
}
```

- A bare filename matches at any depth: `Read(.env)` is the same as `Read(**/.env)`.
- Order matters: `Read(!.env.example)` comes after `Read(.env.*)`. A `!` listed first carves nothing out, and it only carves from rules in the same file.
- If you already have a `settings.json`, add the rules to its `deny` list; do not replace the file.
- The file is strict JSON: a `//` comment or a trailing comma breaks it.

STEP 02

A safety net for every project

In `~/.claude/settings.json`, a path with one leading slash starts at `~/.claude`, not at your project. To cover any folder, use `//`, the filesystem root (on Windows, every drive):

```
{
  "permissions": {
    "deny": [
      "Read(//**/.env)",
      "Read(//**/.env.*)"
    ]
  }
}
```

> **CAUTION**
> The `.env.example` carve-out does not work here: a `!` cannot reach rules anchored with `//`, and a project file cannot reopen what your user file blocks. With this net, the agent cannot see the template either; if it needs it, keep only the Step 01 rules.

STEP 03

Check that it works

1. Open Claude Code in the project and type `/permissions`. The dialog lists every rule and the file it comes from.
2. Ask it to read `.env` and to run `cat .env`. Both requests should end up blocked. Use a `.env` with fake values for the test.
3. Ask it to read `.env.example`: that should work.
4. In the terminal, make sure git holds no real `.env`:

```
git ls-files | grep -i "\.env"
git log --all --oneline -- .env
```

The first line should only print `.env.example`. The second should print nothing; if it lists commits, the key is in the history even if `.gitignore` lists it today: go to the rotation checklist.

IF IT LEAKED

Rotation checklist

A key the agent read sits in the local transcript (`~/.claude/projects/`, 30 days by default) and travelled to the model. A key that reached a commit is in every clone. Deleting is not enough: you rotate it.

1. **If it moves money or customer data, revoke it now** at the provider, even if the app fails for a few minutes.
2. **Create the new key.** In Stripe, rotating revokes the old key and creates a replacement; with a future expiry both work for up to 7 days, so after a leak pick an immediate expiry.

3. **Store it in the secret manager.** In Railway, edit the variable and deploy the staged change. In Secret Manager with Cloud Run (illustrative versions):

```
gcloud secrets versions add PAYMENTS_KEY --data-file=-
gcloud run services update my-service \
  --update-secrets=PAYMENTS_SECRET_KEY=PAYMENTS_KEY:2
gcloud secrets versions disable 1 --secret=PAYMENTS_KEY
```

4. **Check the app works** with the new key.
5. **Revoke the old key at the provider.** Disabling the version in Secret Manager does not invalidate it.
6. **Review the provider's logs** from the day the key could have left.
7. **Remove the key from wherever it landed**: your `.env`, files or commits the agent wrote. If it is in the history, follow GitHub's guide to removing sensitive data; after rotating, not before.
8. **Close the door**: the Step 01 deny rules and only test keys on your machine.

> **CAUTION**
> The three gcloud commands were checked against the local help of Google Cloud SDK 581.0.0 and Google's docs, not run against a project. Replace the names with yours and always pin a version: Cloud Run environment variables resolve when the instance starts.

WHEN YOU FINISH

Checklist

- [ ] `.claude/settings.json` blocks `.env`, `.env.*` and `*.pem`, with the `.env.example` carve-out last.
- [ ] `/permissions` shows the rules, and the agent can neither read nor `cat` the `.env`.
- [ ] On Windows, the PreToolUse hook covers PowerShell.
- [ ] Your local `.env` holds only test or read-only keys.
- [ ] Your folder is linked to a development environment.
- [ ] Real keys live in the secret manager, sealed or pinned to a version.
- [ ] `git log --all -- .env` prints nothing.
- [ ] For each key you know where to rotate it and where to read its logs.