LEARN · THE STACK

Zero-Secret Architecture: How to Minimize Your Attack Surface

Every API key, token, and certificate your team holds is a liability you must defend, rotate, and audit forever. This piece shows how Pillar runs an entire production platform with approximately six secrets, and how you can rebuild your stack around the same principle: if a secret can be eliminated, eliminate it.

---

10 min read

Last updated: June 10, 2026

> *Every secret you hold is an attack surface and a maintenance burden. The fewer secrets you handle, the safer your operation and the simpler your ops. Pillar designs for zero-secrets-where-possible, and this piece explains how to do the same in your business.*

The thesis

- Why secret count is a better security metric than "do we use vault X"
- The Zero-Secret Stack framework: a five-pillar mental model for eliminating credentials
- How Pillar replaced Stripe API keys, mail credentials, database logins, and SSH keys with hosted equivalents
- Concrete swaps you can make this week (Stripe Payment Links, Formspree, Cloudflare Access, static JSON)
- How to isolate the secrets you cannot eliminate so a single leak does not topple the stack

01 — The Zero-Secret Stack

The Zero-Secret Stack

Most security advice tells you how to store secrets better. The Zero-Secret Stack inverts the question: which secrets do you not need to hold at all? Each pillar below pushes a category of credential off your machines and onto a hosted boundary you do not have to defend, rotate, or audit. The remaining secrets are few enough to count on one hand.

1

Pillar 1: Outsource the Payment Boundary

Payment processing is the single most common reason startups handle a high-value API key. Replace server-side Stripe integration with Stripe Payment Links or Stripe-hosted Checkout, so the secret-key half of the integration lives on Stripe's servers, not yours. The only piece you keep is a webhook verification secret, which has no spending power on its own.

2

Pillar 2: Eliminate the Backend, Eliminate the Backend Keys

Every server you operate brings a fleet of secrets along for the ride: SSH keys, database credentials, mail provider keys, queue tokens, log shipper tokens. A static-first architecture deployed to Cloudflare Pages, Vercel, or Netlify removes the server entirely, and with it removes the credentials it would otherwise require. The build pipeline runs locally or in CI; nothing on the live edge has a writable secret.

3

Pillar 3: Outsource Forms and Email

Contact forms and transactional email are the second most common source of leaked credentials, because every junior engineer wires SendGrid or Mailgun keys into the frontend at some point. Instead, route forms through a hosted intake service like Formspree, which holds its own delivery credentials behind its own boundary. Your site has no mail key to leak.

4

Pillar 4: Authenticate Without Storing Auth

Most admin dashboards do not need their own login system, which is a good thing because login systems require session secrets, password hashes, and a database to put them in. Cloudflare Access, Tailscale, and similar identity-aware proxies let you put any URL behind your existing Google or GitHub SSO without writing or storing a single auth token yourself. The identity boundary lives in the provider, not in your code.

5

Pillar 5: Isolate the Irreducible Few

After eliminating what you can, a small number of secrets will remain, typically webhook verifiers, deploy CLI tokens, and a domain registrar API. Put each in the smallest possible blast radius: one secret per Cloudflare Worker environment, deploy tokens that live only on developer laptops via wrangler, and registrar keys held offline. Rotate quarterly; never commit to a repository.

02 — The data.

~6

Total production secrets handled by Pillar across the entire stack

PILLAR INTERNAL STACK AUDIT, 2026

20–50

Typical secret count for a comparable SaaS startup (Stripe API key, database creds, mail keys, SMS keys, analytics, CDN, DNS API, etc.)

INDUSTRY COMPOSITE ESTIMATE

0

Server-side Stripe API keys on Pillar infrastructure (payment flow is Stripe-hosted Checkout)

PILLAR INTERNAL STACK AUDIT, 2026

1

Cloudflare Worker holding a webhook verification secret, isolated from the rest of the stack

PILLAR INTERNAL STACK AUDIT, 2026

0	0
Production databases, production servers, or long-lived SSH keys on Pillar's stack	Mail provider API keys on Pillar's frontend (form intake delegated to Formspree)
PILLAR INTERNAL STACK AUDIT, 2026	PILLAR INTERNAL STACK AUDIT, 2026

Why secret count is the metric that matters

Security audits often focus on whether secrets are stored properly: in a vault, encrypted at rest, scoped to least privilege. Those things matter, but they answer a smaller question than the one founders should be asking. The right first question is: how many secrets do we hold at all? A secret you do not have cannot leak, cannot be rotated wrong, cannot be committed to a public repository at 2am by a tired contractor, and cannot be exfiltrated by a malicious npm package. The cheapest defense is to not need the defense.

Every secret carries three ongoing costs. First, a leak cost: the blast radius if it escapes, measured in dollars (a Stripe live key) or in reputational damage (a mail API key used to phish your customers). Second, a rotation cost: the engineering time to swap it everywhere it is referenced, multiplied by the number of times per year you should be rotating. Third, an audit cost: the documentation, access reviews, and compliance evidence you need to prove you are managing it correctly. Cut the count from forty to six, and you cut those three costs by roughly the same factor.

This is why Pillar tracks its production secret inventory as a single small list rather than as a directory in a vault. The list fits in a sentence: a webhook verification secret in one Cloudflare Worker, a wrangler deploy token on the developer laptop, a registrar API credential held offline, and a small number of similar single-purpose tokens. Each one exists because there was no hosted equivalent we could outsource the boundary to. That number is the number we defend.

Pillar's secret inventory, walked through end-to-end

Start at the customer's browser. A visitor lands on a Pillar Studio page, reads about a domain, and clicks the buy button. That button is a Stripe Payment Link, which is a URL hosted by Stripe. There is no Stripe API key on the Pillar side of that handoff, because the entire checkout flow runs on Stripe's domain, with Stripe's keys, on Stripe's servers. When the transaction completes, Stripe fires a webhook at a single Cloudflare Worker we run for exactly this purpose, and that Worker verifies the signature using a webhook secret stored only in the Worker's environment variables. The webhook secret has no spending power; it only proves Stripe sent the message.

Now look at the contact form. Visitors who want to reach the team submit through an intake form whose action attribute points at Formspree. Formspree holds the credentials that send the resulting email; the static page on Pillar holds nothing more sensitive than a generic endpoint URL. There is no SendGrid key, no Mailgun key, no SMTP password anywhere on the Pillar side of that flow. If our frontend repository leaked tomorrow, no mail-provider key would leak with it, because no mail-provider key exists.

Deployment uses Cloudflare Pages, driven from a developer laptop by the wrangler CLI. The wrangler token lives in the operator's local keychain and is never shipped to a CI server, a build environment, or a shared machine. The published site is static HTML, CSS, and a handful of JavaScript files; it contains no runtime secrets because there is no runtime to hold them in. Even the deploy credential is short-lived in practice, because rotating it is a one-line CLI command that touches a single keychain entry. The final piece is the domain registrar account at Dynadot, whose credentials are held with the same hygiene as personal banking: long password, hardware-key second factor, no API key issued unless a specific automation needs it.

A migration playbook for an existing stack

If you are operating today with the typical 20-to-50 secret inventory, the migration is mostly a matter of replacing each secret-bearing service with a hosted equivalent that holds the secret for you. The order matters: start with the highest-leverage replacements, because each one removes not just a credential but the operational tax that came with it. Replace your server-side Stripe integration with Stripe Payment Links or Stripe-hosted Checkout, and the live key plus the rotation policy plus the PCI scope all leave at once. Replace your custom mail integration with Formspree for intake and a hosted transactional provider's customer-managed templates for outbound, and the mail key plus the bounce monitoring plus the deliverability complexity all leave at once.

The next category to attack is the backend itself. Most startup backends exist because the team assumed they were necessary, not because a specific feature requires server-side state. A surprising fraction of marketing sites, documentation sites, content sites, and even commerce sites can be rebuilt as static deployments on Cloudflare Pages or an equivalent edge platform, with the small dynamic pieces moved into individual Workers or Functions. Each backend you delete takes a database credential, a mail relay credential, and a server SSH key with it. The replacement is often faster, cheaper, and easier to reason about, in addition to being smaller in attack surface.

Last, deal with the secrets you cannot eliminate. For each remaining credential, ask three questions: what is the smallest environment that can hold it, how short can its lifetime be, and what is the blast radius if it leaks? A Stripe webhook secret belongs in one Worker's environment, rotates quarterly, and on leak only allows an attacker to forge webhook traffic to one endpoint. A wrangler deploy token belongs only on a developer laptop, rotates whenever a teammate leaves, and on leak only allows an attacker to deploy code (which your code review process catches). A registrar API key, if you use one, belongs offline and is only loaded into memory for the duration of a specific automation run. These boundaries do not eliminate the secret, but they convert it from a permanent liability into a contained one.

03 — Watch: a real walkthrough

04 — Your zero-secret audit, this week

Run this checklist against your own stack. Each item targets a specific category of credential that most teams can eliminate or shrink within a single sprint.

1. Inventory every secret. Open a private document and list every API key, password, token, certificate, and SSH key your team holds. Aim for a complete list; the surprise is usually the count.
2. Categorize each as eliminable, replaceable, or irreducible. For each entry write the hosted alternative that would let you delete it, or note that none exists.
3. Replace server-side Stripe with Stripe Payment Links or Stripe-hosted Checkout, keeping only a webhook verification secret in a single isolated Worker.
4. Replace custom mail integrations with Formspree for intake and customer-managed templates for outbound, removing the mail API key from your frontend and most of your backend.
5. Replace bespoke admin authentication with Cloudflare Access, Tailscale, or an equivalent identity-aware proxy, so you stop storing your own session secrets.
6. Audit your repositories with a tool like gitleaks or trufflehog to confirm no secret was ever committed in history; rotate anything found.
7. For the irreducible few, document each in a one-page runbook: where it lives, what its blast radius is, who rotates it, and when. If you cannot describe that in a paragraph, the secret is not isolated enough.

Frequently asked questions.

Is six secrets really a meaningful target, or is it just a marketing number?

It is a working count from an actual production audit, not a slogan. The point of citing it is not that six is a magic number but that it is one full order of magnitude below the industry norm of 20-to-50. The target for your own stack should be the lowest number you can reach without giving up features your users care about. If yours lands at eight, or twelve, the framework still did its job.

Doesn't outsourcing to Stripe, Formspree, and Cloudflare just move the secrets elsewhere?

Yes, and that is the point. Those vendors hold credentials inside boundaries they defend professionally, with security teams, SOC 2 audits, and bug bounties that are far larger than what most startups can run in-house. You are not eliminating the existence of secrets globally; you are moving them off your machines and onto the machines of organizations whose business model is defending them. The leak risk on your side drops to near zero for the categories you outsource.

What about the secrets I genuinely cannot eliminate, like a registrar API key?

Apply the isolation pillar from the framework. Put the credential in the smallest possible environment, give it the shortest practical lifetime, and document its blast radius. A registrar key that lives offline, is loaded only for a specific automation run, and is rotated quarterly is a small liability. A registrar key sitting in a long-lived environment variable on a shared server is a large one. The credential is the same; the architecture around it is what matters.

How do I do this without a major rewrite?

You almost certainly do not need a rewrite. The replacements in this piece are mostly drop-in: Stripe Payment Links replace a Stripe integration in a single sprint, Formspree replaces a contact form in an afternoon, and Cloudflare Access wraps an existing admin URL in minutes. The backend elimination step is larger, but it is also optional; even keeping your backend and only eliminating four or five frontend secrets is a meaningful improvement.

Does this approach scale to a larger company?

The principle scales; the specific vendors may change. A larger company will hold more secrets than Pillar does, because it has more product surface area and more compliance requirements, but the framework still applies: outsource what you can, eliminate the backend boundaries you do not need, and isolate the irreducible few. Many companies running at significant scale operate with fewer credentials than smaller competitors precisely because they took this approach early.
