EnvGod

Secure Environment Variable Vault

Online SaaS vault with API & npm SDK

Security-First • Developer-Friendly • Zero-Knowledge

Shutterstock Visual Unit IMAGE ID: 26 www.shutte

Executive Summary

Secure SaaS Vault

Online vault for environment variables with API and npm SDK

Three-Tier Architecture

Dashboard web app · Vault API · npm SDK

Control & Data Planes

Control Plane: Dashboard management
Data Plane: Secret retrieval

Security-First Design

Zero plaintext export · Strict scoping · Audit logging

Problem Statement

⚠ .env files are a security nightmare

Committed to Git

Secrets pushed to public and private repositories

Shared in Plain Text

Copied to Slack, email, chat across teams

Scattered Across Environments

Different versions for dev, staging, production

No Centralized Control

Difficult rotation, no audit trail or visibility

Target Users

Who Uses EnvGod

Dev Teams

Local development · Environment variable management

CI/CD Pipelines

GitHub Actions · GitLab CI · CircleCI

Production Workloads

Web applications · APIs · Serverless functions

Security Teams

Audit logging · Compliance · Access control

Goals and Non-Goals

Goals

- ✓ Reduce secrets in git
- ✓ Enforce scoping (project + env + service)
- ✓ Provide audit trail
- ✓ Security-first design

Non-Goals

- ✗ Replace all secret management
- ✗ Support every cloud provider
- ✗ Feature parity with enterprise vaults
- ✗ Client-side browser secrets

Core User Journeys

From Dashboard to Runtime

1. **Register** & Create Project →
2. **Store** Secrets →
3. **Generate** API Key →
4. **Install** SDK →
5. **Exchange** for JWT →
6. **Fetch** Bundle

JWT TTL: **5-15 minutes**

Product Scope V1

What Ships in V1

Dashboard Web App

- User authentication
- Project management
- Secret storage
- Audit logging

Vault API

- POST /v1/auth/exchange
- GET /v1/bundle
- Scoped access control
- JWT validation

envgod SDK

- JWT token exchange
- Bundle fetch automation
- Server-only enforcement
- Next.js safe-by-default

Azure Key Vault

- Envelope encryption
- AES-256-GCM at rest
- Per-project DEK
- KEK management

Rate Limiting

- Configurable per project
- Bundle fetch limits
- Distributed enforcement
- IP tracking

Kill Switch

- Immediate service stop
- Project-wide scope
- Audit trail included
- Emergency response

Out of Scope V1

What's Coming in V2

High-Assurance Mode

Coming Soon

- Zero-knowledge encryption
- Client-side key management
- Proof-of-possession

RBAC Enhancement

Coming Soon

- Team roles & permissions
- Approval workflows
- Service-level access control

Multi-Cloud Support

Coming Soon

- AWS KMS integration
- GCP KMS integration
- DigitalOcean support

V2 Roadmap planned for **Q3 2025**

System Architecture

Control Plane vs Data Plane

Control Plane

- Dashboard Web App
- API Key Management
- Projects / Envs / Services CRUD
- Audit Logging

User-facing · Configuration · Management

Data Plane

- Vault API Endpoints
  `POST /v1/auth/exchange · GET /v1/bundle`
- JWT Validation
- Secret Retrieval
- Rate Limiting

Runtime · Scoped Access · Encrypted

Data Model

Core Entities

Projects

- id (UUID)
- name (string)
- owner_id (UUID)

Environments

- id (UUID)
- project_id (FK)
- name (string)

Services

- id (UUID)
- project_id, env_id (FK)
- name (string)

Secrets

- id (UUID)
- key (string)
- encrypted_value (blob)

API Keys

- id (UUID)
- key_hash (string)
- scopes (JSON)

Audit Logs

- id (UUID)
- action (string)
- timestamp (datetime)

Security Architecture

Defense in Depth

1 At Rest

- AES-256-GCM encryption
- Envelope encryption (Azure Key Vault)
- No plaintext storage

Per-project DEK wrapped by KEK

2 In Transit

- TLS 1.3 for all HTTP
- Mutual TLS optional
- Secure API authentication

Certificate pinning & validation

3 Access Control

- Strict 3-level scoping
- JWT TTL: 5-15 minutes
- RBAC with project owner role

Project + Environment + Service scope

Auth & Access Flow

API Key to JWT to Bundle

Step 1: Token Exchange

```
Application
      |
      v
SDK
      |
      v
POST /v1/auth/exchange
      | (API Key)
      v
Vault API
      |
      +-- Validate scopes
      +-- Generate JWT
      |   (5-15 min TTL)
      v
SDK
      |
      v
Store JWT locally
```

Step 2: Bundle Fetch

```
Application
      |
      v
SDK
      |
      v
GET /v1/bundle
      | Authorization: JWT
      v
Vault API
      |
      +-- Validate JWT
      +-- Check scopes
      v
Return bundle
      | (scoped env vars)
      v
SDK
      |
      v
Inject secrets
      | (process env)
      v
Run application
```

JWT TTL
5-15 minutes

Scoped Access
Project + Env + Service

Auto-Refresh
SDK manages tokens

SDK Design

envgod npm Package

Clean API

- Automatic JWT token exchange
- Secret retrieval bundle

Server-Only

- Next.js safe-by-default
- Prevents client-side access

Local Caching

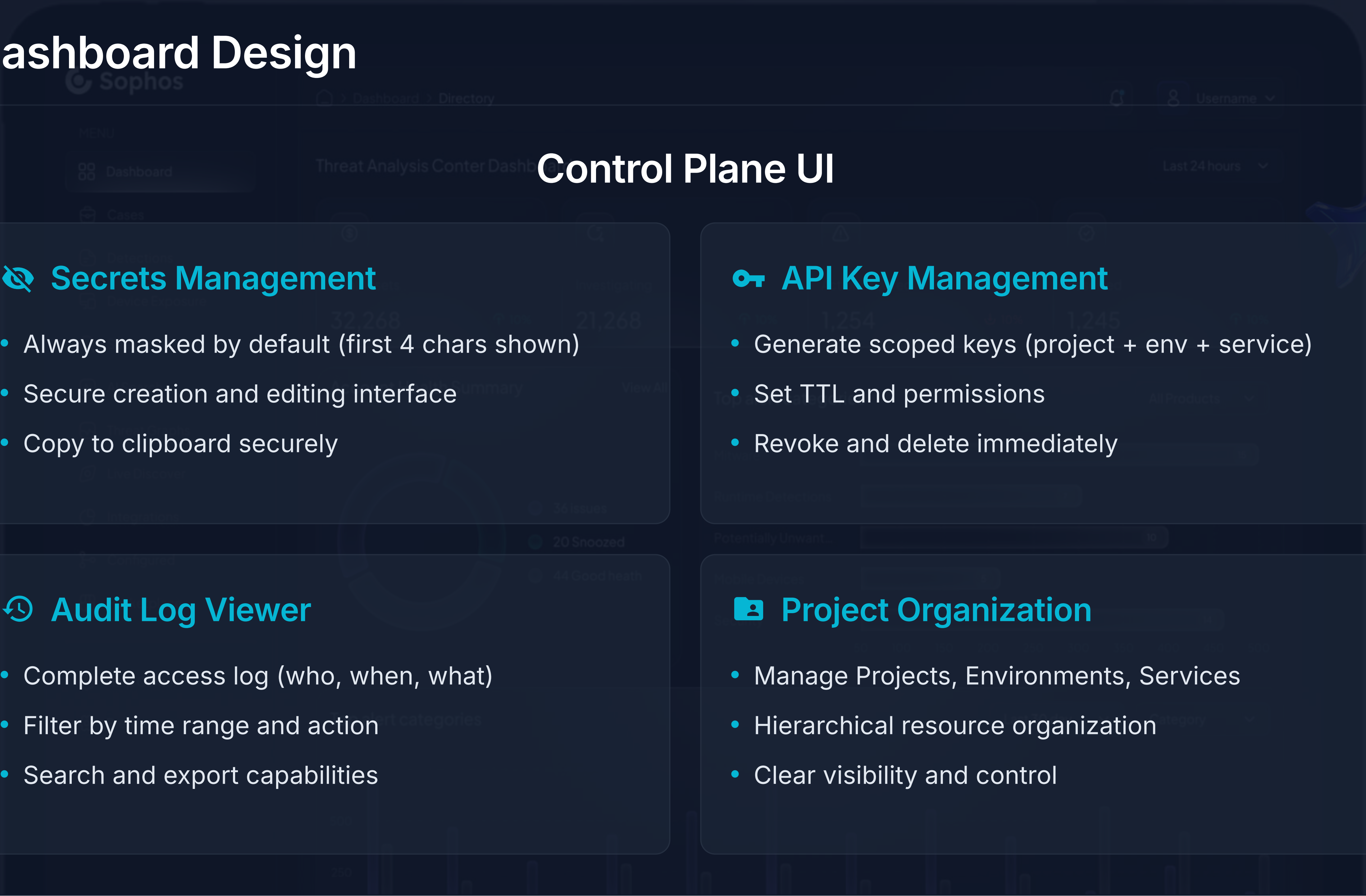
- JWT caching (5-15 min)
- Bundle cache with TTL

TypeScript Support

- Full type definitions
- IDE auto-completion

Required Environment Variables

`ENVGOD_API_URL` `ENVGOD_API_KEY` `ENVGOD_PROJECT` `ENVGOD_ENV` `ENVGOD_SERVICE`



Dashboard Design

Control Plane UI

Secrets Management

- Always masked by default (first 4 chars shown)
- Secure creation and editing interface
- Copy to clipboard securely

API Key Management

- Generate scoped keys (project + env + service)
- Set TTL and permissions
- Revoke and delete immediately

Audit Log Viewer

- Complete access log (who, when, what)
- Filter by time range and action
- Search and export capabilities

Project Organization

- Manage Projects, Environments, Services
- Hierarchical resource organization
- Clear visibility and control

Operational Controls

Production-Ready Ops

Rate Limiting

- Configurable per project
- Distributed enforcement
- IP tracking and geo-fencing

Anomaly Detection

- Location and frequency analysis
- Behavioral pattern recognition
- ML-powered threat detection

Key Revocation

- Manual and automatic revocation
- Immediate effect on revocation
- Bulk revocation tools

Kill Switch

- Project-wide service stop
- Emergency response capability
- Audit trail included

Alert Integration

Real-time notifications to your channels

Slack PagerDuty Email

Phased Delivery Plan

14 Weeks to Production

0 Architecture

1 week

- System architecture finalization
- Security review & threat modeling

→

1 Core MVP

4 weeks

- Dashboard auth & secret storage
- Vault API auth exchange + bundle

→

2 Production Ready

4 weeks

- Envelope encryption & audit logs
- Rate limiting & monitoring

→

3 Advanced Features

3 weeks

- Dashboard audit viewer
- Enhanced anomaly detection

→

4 Launch & GA

2 weeks

- Documentation & tutorials
- Security audit passed

✓ Phase 0-2
MVP

✓ Phase 3
Beta Ready

✓ Phase 4
GA Launch

Risks and Mitigations

Risk Management Strategy

API Key Leakage

- JWT exchange (5-15 min)
- Scoped access enforcement
- Immediate revocation & kill switch

Dashboard Compromise

- MFA for all users
- Complete audit logging
- Anomaly detection & alerts

Azure Key Vault Outage

- Backup KEKs stored securely
- Failover procedures tested
- Multi-region deployment

Secrets in Browser Bundles

- Server-only SDK enforcement
- Next.js safe-by-default
- Runtime checks prevent client access

Insider Threat

- Complete audit logs (metadata only)
- RBAC with project owner role
- Monitoring & behavioral analysis

Zero-Day Vulnerabilities

- Regular security patches
- Bug bounty program
- Incident response runbooks

FAQ — Part 1

Common Questions

Q1 Why not just use Azure Key Vault directly?

✓ V1:

Simplified API, scoped access, audit logs

Risk Reduction:

Dev-friendly abstraction over complex infrastructure

V2:

Multi-cloud support (AWS, GCP, DigitalOcean)

Q2 What happens if an API key leaks?

✓ V1:

Short-lived JWTs (5-15 min), scoped to project+env+service, immediate revocation

Risk Reduction:

Minimizes blast radius of leaked credentials

V2:

Client-side encryption (zero-knowledge architecture)

Q3 What happens if Vault API backend is compromised?

✓ V1:

No plaintext secrets, envelope encryption, audit logs

Risk Reduction:

Data encrypted at rest with KEK in Azure Key Vault

V2:

Zero-knowledge architecture (client-side encryption)

Q4 Can an attacker call API from anywhere (CORS limits)?

✓ V1:

No CORS on server-side SDK, rate limiting, IP tracking

Risk Reduction:

Controls unknown/unauthorized access sources

V2:

IP whitelisting and geo-fencing capabilities

FAQ — Part 2

Security & Operations

Q5 How do you prevent "download all secrets" abuse?

V1:

No bulk endpoint, strict 3-level scoping, rate limiting

Risk Reduction:

Impossible to fetch all secrets for a project

V2:

Per-secret approvals workflow

Q6 How do you ensure secrets don't end up in client-side bundles (Next.js)?

V1:

Server-only SDK enforcement, Next.js safe-by-default, runtime checks

Risk Reduction:

SDK fails on client-side, prevents bundle inclusion

V2:

Build-time validation and tree-shaking

Q7 How does this work on Vercel and serverless environments?

V1:

SDK auto-manages JWT, handles cold starts, stateless design

Risk Reduction:

Works everywhere with no infrastructure changes

V2:

Edge function optimization and caching

Q8 How do you handle secret rotation and key revocation?

V1:

Dashboard UI for manual rotation, kill switch, audit logs

Risk Reduction:

Immediate revocation and full traceability

V2:

Automated rotation schedules and secret rotation tools

FAQ — Part 3

Compliance & Performance

Q9 Do you store plaintext secrets anywhere (logs, DB, backups)?

✓ **V1:**

No plaintext anywhere, AES-256-GCM encryption, envelope encryption with Azure Key Vault

Risk Reduction:

Zero plaintext exposure in logs, DB, or backups

V2:

Zero-knowledge architecture (client-side encryption)

Q10 How do you handle performance (cold starts, caching)?

✓ **V1:**

JWT local caching (5-15 min), bundle caching, stateless design

Risk Reduction:

Fast performance with minimal API calls

V2:

Edge caching, pre-warming, predictive prefetching

Q11 What about compliance (audit logs, access traceability)?

✓ **V1:**

Complete metadata audit logs, timestamped, searchable, exportable

Risk Reduction:

Full traceability of all secret access (who, when, what)

V2:

Compliance reports, SOC 2 preparation, retention policies

Q12 How do you handle outages/availability and rollback?

✓ **V1:**

Multi-region deployment, CDN, load balancer, kill switch

Risk Reduction:

99.9% uptime target, immediate rollback capability

V2:

Active-active architecture, disaster recovery, zero-downtime deployments

Success Metrics and Acceptance Criteria

Security Metrics

- ✓ Zero plaintext secrets in logs
- ✓ Zero API key reuse for data reads
- ✓ All secrets encrypted at rest
- ✓ Audit logs for 100% of secret reads
- ✓ Rate limiting active on all endpoints

Product Metrics

- ✓ Average time to first secret fetch <2 seconds
- ✓ 99.9% API uptime (monthly)
- ✓ 100% successful secret fetch within project scope
- ✓ Zero critical security incidents
- ✓ Successful secret rotation in <5 minutes

Acceptance Criteria

All security metrics passing · Beta users deploy to production · Documentation complete with tutorials · Security audit passed with no critical findings

Appendix — Roadmap V2 High-Assurance Mode

Zero-Knowledge

- Client-side encryption with user keys
- EnvGod never sees plaintext secrets
- Proof-of-possession enforcement
- Maximum security posture

Enhanced RBAC

- Team roles and permissions
- Service-level access control
- Approval workflows for secrets
- Fine-grained scoping

Enterprise Features

- SSO & SAML authentication
- SOC 2 compliance reports
- Advanced analytics & monitoring
- Dedicated support & SLAs

How to Install & Use EnvGod SDK

Key Features

Server-Only Execution

Secure by default — throws in browser environments

In-Memory Storage

Never writes secrets to disk or logs

Auto-Auth & Caching

Smart JWT exchange and token caching

Security Notes

- Explicitly checks for `window` object
- Secrets held in memory only — fresh fetch on restart
- SDK never logs secret values

Installation

```
npm install @rusamer/envgod
# or
pnpm add @rusamer/envgod
# or
yarn add @rusamer/envgod
```

Configuration

```
// Environment Variables

ENVGOD_API_URL = https://api.envgod.com
ENVGOD_API_KEY = sk_xxx
ENVGOD_PROJECT = myapp
ENVGOD_ENV = prod
ENVGOD_SERVICE = web
```

Usage (Node.js)

```
import { loadEnv } from '@rusamer/envgod';

async function main() {
  // Auto-Auth + Caching handled automatically
  const env = await loadEnv();

  console.log(env.MY_SECRET); // Accessed securely
}

main();
```

Why EnvGod?

The Right Choice for Modern Teams

Security-First Design

- Zero plaintext guarantee — never in logs, DB, or backups
- Scoped access (Project + Env + Service) minimizes blast radius
- Complete audit trail — who, when, what (metadata only)
- Short-lived JWTs (5-15 min) with automatic refresh

Developer-Friendly

- Clean, simple API — no complex infrastructure required
- Server-only SDK with Next.js safe-by-default
- Works everywhere: Vercel, Netlify, Docker, Kubernetes
- Fast setup — 5 minutes to production

Scoped Access Control

- Project → Environment → Service granularity
- Impossible to fetch all secrets in a single request
- Each API key scoped to specific project/service
- No bulk secrets download — prevents abuse

Enterprise-Ready

- Production-grade reliability with 99.9% uptime SLA
- Kill switch for immediate service interruption
- Built-in monitoring, rate limiting, and anomaly detection
- V2 roadmap: Zero-knowledge, multi-cloud, RBAC, SOC 2

Azure Key Vault

Enterprise vaults: Complex infrastructure, expensive, overkill for dev teams

EnvGod

Modern SaaS: Simple, secure, developer-friendly, built for today's stack

No One Will Hack Your Keys Any More

UNBREAKABLE PROTECTION FOR YOUR SECRETS

ZERO EXPOSURE

No plaintext anywhere
Not in git · Not in logs
Not in backups

SCOPED ACCESS

Each key locked to
Project + Environment
+ Service level

COMPLETE AUDIT

Every access tracked
Logged · Monitored
100% traceability

STOP KEY LEAKAGE NOW

MILITARY-GRADE ENCRYPTION · ZERO PLAINTEXT GUARANTEE

shutterstock
IMAGE ID: 26
www.shutte