# **Securing Secrets in Kubernetes: AWS Secrets Manager vs. Sealed Secrets vs. Vault**

**Prepared for:** Chief Architect  
**Author:** [Your Name]  
**Date:** [Insert Date]

## **1. Executive Summary**

Managing secrets securely is critical for federal agencies due to strict compliance requirements (e.g., FISMA, FedRAMP). This report evaluates three approaches for securing secrets in an **on-premises Kubernetes (K8s) cluster**:

1. **AWS Secrets Manager (ASM) Integration**
2. **Sealed Secrets (Open Source)**
3. **HashiCorp Vault (Enterprise-Grade)**

The goal is to ensure secrets are **never exposed in CI/CD logs, pipelines, or Git repositories** while remaining accessible to pods in K8s.

## **2. Problem Statement**

* Secrets (API keys, DB credentials, certificates) must not be stored in plaintext in Helm charts, YAML files, or CI/CD logs.
* On-prem K8s clusters need a secure way to fetch secrets without manual intervention.
* Federal compliance requires auditability, encryption, and least-privilege access.

## **3. Solution Comparison**

| **Feature** | **AWS Secrets Manager (ASM)** | **Sealed Secrets** | **HashiCorp Vault** |
| --- | --- | --- | --- |
| **Encryption at Rest** | ✅ (AWS KMS) | ✅ (Cluster-local key) | ✅ (HSM/KMS) |
| **Dynamic Secrets** | ❌ (Static secrets only) | ❌ (Static only) | ✅ (Short-lived secrets) |
| **Access Control** | ✅ (IAM Policies) | ❌ (Cluster-wide key risk) | ✅ (Policies + RBAC) |
| **Audit Logging** | ✅ (CloudTrail) | ❌ (Limited) | ✅ (Detailed logs) |
| **On-Prem Compatibility** | ✅ (Requires AWS API access) | ✅ (Fully on-prem) | ✅ (Fully on-prem) |
| **CI/CD Safety** | ✅ (Secrets pulled at runtime) | ✅ (Encrypted in Git) | ✅ (Pulled at runtime) |
| **Compliance (FedRAMP)** | ✅ (FedRAMP Moderate/High) | ❌ (Self-managed) | ✅ (FedRAMP Certified) |

## **4. Recommended Approach: AWS Secrets Manager with Kubernetes Integration**

### **Why AWS Secrets Manager?**

* **FedRAMP Authorized** (Meets federal compliance requirements).
* **No Secrets in Git/CI Logs** – Secrets are fetched at runtime.
* **KMS Encryption** – Secrets encrypted at rest and in transit.
* **IAM & Resource Policies** – Fine-grained access control.

### **Implementation Steps**

#### **Option A: Using External Secrets Operator (ESO)**

1. **Deploy ESO in K8s**
2. sh
3. Copy
4. helm install external-secrets external-secrets/external-secrets
5. **Configure AWS IAM Role for Pods (IRSA) or K8s Service Account**
   * Use IAM roles for service accounts (if AWS EKS).
   * For on-prem, use **OIDC federation** or **instance profiles** (if nodes have AWS API access).
6. **Create a SecretStore (points to AWS Secrets Manager)**
7. yaml
8. Copy

apiVersion: external-secrets.io/v1beta1

kind: SecretStore

metadata:

name: aws-secret-store

spec:

provider:

aws:

service: SecretsManager

region: us-east-1

auth:

jwt:

serviceAccountRef:

1. name: external-secrets-sa
2. **Define an ExternalSecret (maps ASM secrets to K8s secrets)**
3. yaml
4. Copy

apiVersion: external-secrets.io/v1beta1

kind: ExternalSecret

metadata:

name: db-credentials

spec:

refreshInterval: 1h

secretStoreRef:

name: aws-secret-store

kind: SecretStore

target:

name: db-secret

data:

- secretKey: DB\_PASSWORD

remoteRef:

1. key: prod/db/password
   * Pods can now use db-secret without exposing it in CI/CD.

#### **Option B: CSI Driver with Secrets Manager**

* Uses the **AWS Secrets & Config Provider (ASCP) CSI driver** to mount secrets as volumes.
* Secrets are **never stored in etcd**, only in memory.

## **5. Alternatives: Sealed Secrets & Vault**

### **Sealed Secrets (Open Source)**

* **Pros:**
  + Encrypts secrets for Git storage (safe for CI/CD).
  + No dependency on AWS.
* **Cons:**
  + No dynamic secrets.
  + Single cluster key risk (if compromised, all secrets are exposed).

### **HashiCorp Vault (Enterprise Solution)**

* **Pros:**
  + Dynamic secrets (auto-expiring credentials).
  + Strong RBAC and audit logging.
  + FedRAMP-certified deployments available.
* **Cons:**
  + Higher operational overhead.
  + Requires Vault agents in K8s.

## **6. Final Recommendation**

| **Use Case** | **Recommended Tool** |
| --- | --- |
| **AWS-heavy environments** | ✅ AWS Secrets Manager + ESO |
| **GitOps + On-Prem K8s** | ✅ Sealed Secrets (if minimal AWS dependency) |
| **High-compliance (FedRAMP, dynamic secrets)** | ✅ HashiCorp Vault |

For **federal agencies**, **AWS Secrets Manager with External Secrets Operator** is the best balance of security, compliance, and ease of use.

## **7. Next Steps**

1. **Pilot AWS Secrets Manager + ESO** in a non-production cluster.
2. **Evaluate Vault** if dynamic secrets are required.
3. **Enforce IAM least privilege** and audit access via CloudTrail.

**Attachments:**

* [AWS Secrets Manager FedRAMP Compliance Documentation](https://aws.amazon.com/compliance/fedramp/)
* [External Secrets Operator GitHub](https://github.com/external-secrets/external-secrets)
* [HashiCorp Vault FedRAMP](https://www.hashicorp.com/blog/hashicorp-vault-attains-fedramp-moderate-authorization)

## 

## **Proposal: Secure Secrets Management for On-Prem Kubernetes Clusters**

### **Objective**

To securely manage and deliver secrets (API keys, credentials, certificates) into pods running in an **on-prem Kubernetes cluster**, without exposing sensitive values in CI/CD pipelines or logs. The solution must meet **federal agency requirements** for security, auditability, and scalability.

## **Option 1: AWS Secrets Manager Integration**

### **Overview**

AWS Secrets Manager is a managed secrets storage service offering fine-grained access control, automatic rotation, and audit trails via AWS CloudTrail. Integrating it with on-prem Kubernetes clusters is feasible using tools like the [AWS Secrets & Configuration Provider (ASCP)](https://github.com/aws/secrets-store-csi-driver-provider-aws), which works with the **Secrets Store CSI Driver**.

### **How It Works**

1. **Secrets Storage**: Secrets are stored securely in AWS Secrets Manager, encrypted at rest with KMS.
2. **Kubernetes Integration**:  
   * Use the **Secrets Store CSI Driver** along with the **AWS provider**.
   * Deploy the driver in the on-prem cluster.
   * Configure Kubernetes SecretProviderClass resources to map AWS Secrets Manager secrets to in-cluster volumes or Kubernetes Secrets.
3. **Security**:  
   * Pods access secrets as mounted files or Kubernetes secrets.
   * **No secret values are exposed in Git or CI/CD pipelines**, since secrets are pulled at runtime by the CSI driver.
4. **Authentication**:  
   * Use **IAM Roles Anywhere** or **IRSA for EC2** if there's a hybrid setup.
   * On-prem clusters authenticate to AWS using x.509 certificates (e.g., via IAM Roles Anywhere).

### **Pros**

* Centralized secrets storage and management via AWS.
* Automatic rotation and fine-grained IAM access.
* FIPS-compliant encryption and audit trails (CloudTrail).
* Runtime secret injection (not in container images or CI/CD).

### **Cons**

* Adds AWS dependency to on-prem systems.
* Requires secure IAM identity federation (IAM Roles Anywhere) setup.
* Potential latency if pulling secrets across cloud/on-prem boundary.

## **Option 2: HashiCorp Vault**

### **Overview**

HashiCorp Vault is a highly flexible secrets management tool that can be deployed on-prem or in the cloud. Vault is well-suited for high-security environments (such as federal agencies) and supports dynamic secrets, PKI, and audit logging.

### **How It Works**

1. **Secrets Storage**: Secrets stored in Vault, encrypted at rest and in transit.
2. **Kubernetes Integration**:  
   * Use **Vault Agent Injector** to auto-inject secrets into pods as environment variables or files.
   * Pods authenticate using **Kubernetes Service Account JWTs**.
3. **CI/CD Integration**:  
   * Secrets are **not stored in pipelines**; pods pull secrets dynamically from Vault at runtime.

### **Pros**

* Full control over secrets lifecycle.
* Supports dynamic secrets and secret rotation.
* Flexible deployment (on-prem, multi-cloud).
* Strong audit logging and access policies (via Sentinel or ACLs).
* FIPS 140-2 validated versions available for federal use.

### **Cons**

* Requires self-hosting, backup, and HA setup.
* Operational complexity (initial learning curve, maintenance).
* May require HSM integration for compliance in some environments.

## **Option 3: Sealed Secrets**

### **Overview**

Bitnami Sealed Secrets encrypt Kubernetes Secrets into SealedSecret CRs that can be safely stored in Git. A controller inside the cluster decrypts and applies them at runtime.

### **How It Works**

1. **CI/CD creates a SealedSecret** from a plaintext secret and a public key.
2. The encrypted SealedSecret is committed to Git.
3. At deploy time, the Sealed Secrets controller decrypts and creates a Kubernetes Secret.

### **Pros**

* GitOps-friendly and simple to use.
* Secrets are encrypted and versioned in Git.
* Easy for developer workflows.

### **Cons**

* Secrets are **still created in CI/CD**, albeit encrypted.
* No secret rotation mechanism.
* Not suitable for runtime secret fetching.
* Less ideal for dynamic secrets or compliance-heavy environments.

## **Recommendation**

For a **federal agency context**, where **security, auditability, and runtime control** are critical:

### **✅ Primary Recommendation: HashiCorp Vault (on-prem or cloud-hosted)**

Vault offers:

* Fine-grained access control.
* Secrets injection at runtime.
* Dynamic secrets and PKI support.
* Strong auditing and compliance features.
* Deployment flexibility (air-gapped, HSM-integrated, or SaaS).

It is particularly well-suited for meeting **federal compliance requirements** such as FISMA, FedRAMP, or NIST 800-53.

### **✅ Secondary Option: AWS Secrets Manager + CSI Driver (if AWS is already used)**

This is a viable option **if AWS is already in your stack** and cross-environment IAM integration is acceptable. It provides managed convenience but introduces dependency on AWS infrastructure and IAM federation setup.

### **❌ Not Recommended: Sealed Secrets for Federal Agency Use Cases**

While simple and GitOps-aligned, Sealed Secrets lacks rotation, dynamic secrets, and runtime delivery. It's not ideal for high-security environments.

## **Conclusion**

To meet the stringent requirements of a federal agency, **Vault stands out** as the most secure and flexible solution. For environments already embedded with AWS, integrating **AWS Secrets Manager with the Secrets Store CSI Driver** offers a strong alternative — provided IAM and network security concerns are addressed.