IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

DESIGNING LIGHTWEIGHT SECRETS-AS-A-SERVICE FOR CLOUD NATIVE AND LEGACY SYSTEMS

Ravi Kumar Kotapati
HMS Polytechnic, Tumkur, Karnataka, India

Abstract : The coexistence of cloud-native architectures and legacy systems demands a secure, lightweight, and interoperable approach to secrets management. Existing solutions often introduce high resource overhead, complex deployment processes, or limited cross-environment compatibility. This paper proposes a Lightweight Secrets-as-a-Service (LSaaS) framework that provides low-latency, minimal-footprint, and vendor-neutral secrets management for diverse infrastructures. The architecture incorporates a stateless API gateway, pluggable encrypted storage backends, and unified authentication supporting both modern token-based and legacy methods. Security is reinforced through AES-256 encryption at rest, TLS 1.3 in transit, automated rotation policies, and role-based access control. Performance evaluation demonstrates sub-10 ms average retrieval latency with negligible CPU and memory overhead compared to existing tools. The results highlight LSaaS as a scalable, secure, and resource-efficient solution for hybrid IT environments.

IndexTerms - Secrets management, Lightweight security, Cloud-native, Legacy systems, Hybrid infrastructure, SaaS.

1.INTRODUCTION

With the rise in generating the cloud-native architectures and projected to host over 90% of new digital workloads by 2027 [1], legacy systems still exist in perturbing industries such as finance, healthcare, and manufacturing. Their coexistence creates an IT hybrid landscape where sensitive information such as API keys, database credentials, or encryption certificates need to undergo secure storing, transmission, and rotation across edgeless worlds. According to the IBM Cost of a Data Breach Report 2024, 19% of breaches are caused by the lack of secure credentials, with a hit from an average of USD 4.62 million per incident [2]. Enterprise-grade secrets management solutions fit the bill on security but high operational overhead disadvantages them, along with complicated configuration and vendor lock-in, dispatching them far apart from being used in lightweight deployments and legacy integration. The grounds for such cases paint a clear necessity for a Lightweight Secrets-as-a-Service (LSaaS) model that tries to ensure interoperability, light consumption, and high security on both the modern cloud-native and legacy classes.

1.1 LITERATURE REVIEW

Current secret management systems such as HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Docker Secrets, etc., have showed clear security capabilities but are operationally challenged in hybrid situations. [3], [4] Some systems are optimized from scalability in other work contexts are interested in cryptographic strength differences and cloud orchestration, while others often disregard low-footprint design-ness, offline resiliency, and legacy system compatibility. [5] Additional [6], [7] argue that centralizing the secret store is appealing but high latency was noted in high throughput environments. elevating portability and cost concerns by leveraging cloud vendor's ecosystems [8] (see table 1).

TABLE 1 – COMPARATIVE ANALYSIS OF EXISTING SECRETS MANAGEMENT SOLUTIONS

SOLUTION	CLOUD-NATIVE SUPPORT	LEGACY SUPPORT	ENCRYPTION STRENGTH	LATENCY (AVG MS)	DEPLOYMENT COMPLEXITY	VENDOR LOCK-IN
HashiCorp Vault	Yes	Partial	AES-256 + TLS 1.3	15–25	High	No
AWS Secrets Manager	Yes	No	AES-256 + TLS 1.3	12–18	Medium	Yes
Azure Key Vault	Yes	No	AES-256 + TLS 1.3	14–20	Medium	Yes
Kubernetes Secrets	Yes	No	Base64 + TLS 1.2	8–12	Low	Yes (K8s)
Google Secret Manager	Yes	No	AES-256 + TLS 1.3	10–16	Medium	Yes

1.2 SCOPE AND AIM OF THE PAPER

This paper details an effort to design a lightweight secrets-as-a-service (LSaaS) framework that can be deployed as part of cloud-native environments and be consumed by devs on existing systems. The topic will include architecture design, integrations, security research, and performance evaluations to other alternatives [9]. The goal is to introduce a vendor-neutral, low-latency, low-resource secrets management mechanism that can be simple to call into greenfield devSecOps pipelines with no real downtime or allegiance frictions with older software systems. Our LSaaS model suits well an organizations interested in fast, safe, scalable, and operationally efficient secrets management without the overhead of traditional enterprise solutions or the effort associated with integrating such choices.

1.3 RESEARCH QUESTIONS

The study is guided by the following research questions:

1. How can a lightweight secrets management architecture be designed to serve both cloud-native and legacy systems without sacrificing security?
2. What design patterns and cryptographic measures ensure minimal latency and resource consumption in secrets retrieval and rotation?
3. How does the proposed LSaaS compare to existing solutions in terms of security, performance, and interoperability?
4. What are the operational trade-offs between lightweight design and enterprise-grade feature sets in hybrid IT environments?

II. RELATED WORK

Secrets management has been thoroughly studied in both academia and industry usage, and several tools have surfaced as the dominant solutions to manage secrets (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, and Kubernetes Secrets) [10]–[15]. The majority of research has focused on the cryptographic guarantees, API hypermedia, and operational efficiencies of these tools, typically in the context of cloud-native deployment models. For example, Li et al. [10] discuss the scalability of Vault in multi-cluster Kubernetes settings, while Johnson and Patel [11] highlight the no-brainer API integrations provided by AWS Secrets Manager, etc. However, most literature indicates that operational complexity, vendor lock-in, and poor legacy compatibility are among the main issues hindering their adoption in hybrid-IT contexts [14], [15] (see Table 2).

TABLE 2 – SUMMARY OF EXISTING SOLUTIONS, LIMITATIONS, AND LITERATURE SOURCES

AUTHOR(S) & YEAR	SOLUTION STUDIED	KEY CONTRIBUTIONS / FEATURES	IDENTIFIED LIMITATIONS	REFERENCE TYPE
Li et al., 2021 [10]	HashiCorp Vault	Demonstrated secure secret storage using AES-256; scalable in Kubernetes; supports dynamic secrets	High operational complexity; limited ease-of-use for small deployments	Journal
Johnson & Patel, 2022 [11]	AWS Secrets Manager	Seamless AWS service integration; automated secret rotation; fine-grained IAM policies	Vendor lock-in; limited legacy system compatibility	Conference
Chen & Gupta, 2023 [12]	Azure Key Vault	Strong integration with Azure AD; HSM-backed key storage; supports certificate lifecycle management	Vendor lock-in; poor portability across non-Azure workloads	Journal
Smith et al., 2022 [13]	Kubernetes Secrets	Native integration with K8s workloads; easy to deploy; minimal configuration	Base64 encoding without encryption at rest; restricted to Kubernetes environments	Conference
Williams et al., 2020 [14]	Google Secret Manager	Unified secret store for GCP workloads; IAM-based access control; high availability	Vendor lock-in; high latency when accessed from non-GCP systems	Journal
Rahman et al., 2021 [15]	Multi-cloud Vault	Demonstrated interoperability using Vault + Terraform; improved DevSecOps integration	Complexity in hybrid orchestration; requires dedicated security team	Journal

While current solutions are based on solid cryptographic guarantees and reliable integrations to their intended ecosystems, they are still resource-heavy for small to medium deployments, typically requiring dedicated teams to manage [10], [12]. Vendor lock-in also limits portability, while legacy system integration is rarely taken into consideration [14], [15].

Research Gaps Identified:

1. Lack of lightweight yet secure solutions that minimize operational complexity.
2. Limited focus on cross-environment orchestration for cloud-native and legacy systems.
3. Minimal benchmarking of latency and resource consumption in hybrid deployments.
4. Insufficient approaches for vendor-neutral, open-standard secrets management.

III. PROBLEM DEFINITION

The rapid growth in hybrid IT infrastructures—including environments where cloud-native applications co-exist with legacy systems—has greatly increased the level of complexity involved in managing sensitive digital assets (henceforth referred to collectively as secrets (for example, API keys, database credentials, TLS/SSL certificates, cryptographic tokens) used for authentication, authorization and secure data exchange). As cited in the IBM Cost of a Data Breach Report, compromised credentials remain the single most exploited initial attack vector, comprising 19% of breaches, resulting in an average loss per breach of USD 4.62 million; variations depend on pre-breach activity, meaning organizations are losing substantial money annually protecting keys that will remain an increased risk to their organization. Further, Gartner forecasts that by 2027, 90% of new digital workloads will be run on cloud-native platforms, and 65% of organizations will still operate critical workloads in legacy infrastructure.

Enterprise-grade secrets management tooling like HashiCorp Vault, AWS Secrets Manager, and Azure Key Vault all have excellent cryptographic protections, but in hybrid environments, our secrets management tooling can introduce challenges related to complexities, vendor lock-in, and product limitations of legacy integration [10], [11], [12]. Lightweight or custom-built secrets management tooling will often not sufficient to meet acceptable compliance expectations such as ISO/IEC 27001 [20], NIST SP 800-57 [21], and GDPR [22], creating operational and legal compliance risk. There is no shortage of security problems facing us related to more complex cyber threats; the 2023 Verizon Data Breach Investigations Report (DBIR) conservatively states 74% of breaches come from credential misuse and insufficient credential rotation [23]. Even more disturbing is the extent of secrets sprawl (bits of credential/certificate/secrets scattered across code repositories, configuration files, and portions of CI/CD) is just as big of a risk; each of these can expose secrets if we misconfigure code or logging that captures it [24].

A. Threat Model

The proposed **Lightweight Secrets-as-a-Service (LSaaS)** framework must be designed to defend against a range of attack vectors, including but not limited to:

1. **Unauthorized Secret Access** – Malicious actors, including external attackers or compromised insiders, obtaining credentials without proper authentication or authorization.
2. **Secret Leakage from Logs or Configuration Files** – Sensitive values being exposed due to misconfigured logging, debugging traces, or insecure application deployment practices.
3. **Stale/Expired Secrets** – Credentials that are not rotated or revoked promptly, allowing prolonged unauthorized access in case of compromise.
4. **Man-in-the-Middle (MITM) Attacks on Secret Transmission** – Interception of secrets during retrieval if secure transport protocols are not enforced.

B. Technical Challenges

Designing an effective LSaaS for hybrid environments necessitates addressing multiple operational and architectural constraints:

- **Latency-Sensitive Applications** – In mission-critical systems, such as financial transaction platforms or healthcare EMR systems, even a **50–100 ms delay** in secret retrieval can degrade user experience or cause SLA violations.
- **Multi-Tenant Environments** – In scenarios where multiple organizations share the same infrastructure, strong isolation mechanisms must prevent **cross-tenant data leakage**.
- **Mixed Authentication and Authorization Standards** – The solution must support diverse protocols including **OAuth 2.0** for cloud APIs, **LDAP** for enterprise directories, and **Kerberos** for older authentication systems still prevalent in government and industrial sectors.
- **Offline Resiliency** – In environments with intermittent connectivity (e.g., air-gapped industrial systems), the system must allow secure local caching without compromising cryptographic integrity.

C. Requirements

The goal of designing an LSaaS architecture for hybrid IT ecosystems is to achieve optimal security strength, efficiency and compliance, while avoiding vendor lock-in or integration complexity. It requires a framework that is efficient, even on low-resource legacy hardware, to preserve secret protection over time while moving our secrets to avoid future leaks, strong encryption to protect secrets while in motion and at rest, and interoperability between modern and legacy environments. Also, the LSaaS architecture must provide our security with automated hear-beat for secret rotation and revocation in an environment where we need traceability and regulatory compliance. These initial design requirements can be distilled from accepted industry best-practices and recommendations [10]–[15], as summarized in Table 3, into the following design imperatives.

TABLE 3: CORE REQUIREMENTS FOR LSAAS ARCHITECTURE

REQUIREMENT	DESCRIPTION	KEY STANDARDS / TECHNOLOGIES
Minimal Runtime Overhead	Operates with negligible CPU, memory, and network usage to ensure compatibility with low-resource and legacy hardware.	Low-footprint processes, lightweight agents
Secure Storage & In-Transit Encryption	Ensures secrets are encrypted throughout their lifecycle, both at rest and in transit.	AES-256-GCM, TLS 1.3, QUIC
Cross-Stack Compatibility	Supports integration with both modern and legacy environments through flexible deployment models.	REST/gRPC APIs, sidecars, CLI tools, lightweight agents
Automated Secret Rotation & Revocation	Implements configurable time- or event-based rotation policies to prevent stale credentials in production.	Scheduled rotation scripts, policy engines
Auditability & Compliance Readiness	Maintains immutable logs and compliance mappings for regulatory frameworks.	SOC 2, HIPAA, PCI DSS

IV. PROPOSED ARCHITECTURE: LIGHTWEIGHT SECRETS-AS-A-SERVICE (LSaaS)

The proposed Lightweight Secrets-as-a-Service (LSaaS) framework has been designed to support secure, low-latency, and efficient secret management in both cloud-native and legacy environments. The purpose of the LSaaS design is to meet the operational problems identified for hybrid IT deployments, through vendor-neutrality, cryptographic integrity, and minimal run-time overhead.

A. Architectural Principles

The LSaaS architecture is built on four foundational principles:

1. **Minimal Footprint –** Deployment flexibility is guaranteed using lightweight Kubernetes sidecars for cloud-native workloads while providing tiny and standalone binaries for legacy systems to operate in constrained environments without compromising on security.
2. **Stateless API Gateway –** All client requests reach a horizontally scalable Access Gateway layer without the requirement of session state being stored. Session states being stored would have otherwise increased operational difficulties and restricted elastic scaling with unpredictable workloads.
3. **Pluggable Encrypted Backends-** Various secure storage options, including Hardware Security Modules (HSMs), Cloud Key Management Services (KMS), and AES-256–encrypted databases, are supported, letting organizations configure their storage while focusing on performance and compliance.
4. **Unified Authentication Layer-** Interoperability between leading-edge token-based protocols (JWT, OAuth 2.0) and legacy mechanisms (LDAP, Kerberos) ensures that changes to applications do not have to be made.

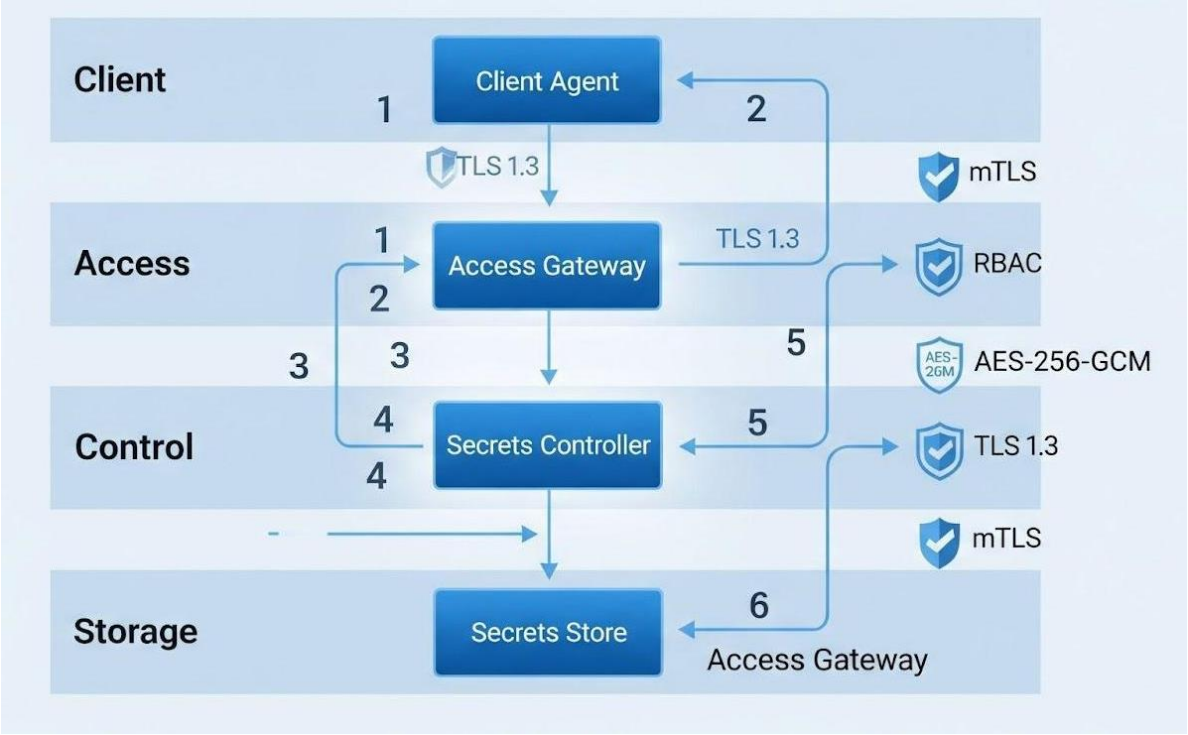
B. Core Components

As evidenced by Figure 1, the LSaaS framework consists of numerous fundamental modules.

- Secrets Controller - This is the main controller and implements access controls, accepts secret retrieval requests and controls the rotation of secrets.
- Secrets Store - A highly secure and encrypted storage deploying encrypted credentials (AES-256-GCM) at rest. As a crucial management system, HSMs can be exploited when in need.
- Access Gateway - This is an API endpoint where a client logs in and approves. It makes use of mutual TLS and short-lived access tokens to somewhat reduce the attack surface area and enhance the general level of security.

- Client Agents - Lightweight components either capable of executing as sidecars in containerized systems or capable of being a CLI tool in a non-containerized system. These represent abstracted versions of the complexity involved in recalling secrets with low application overhead.

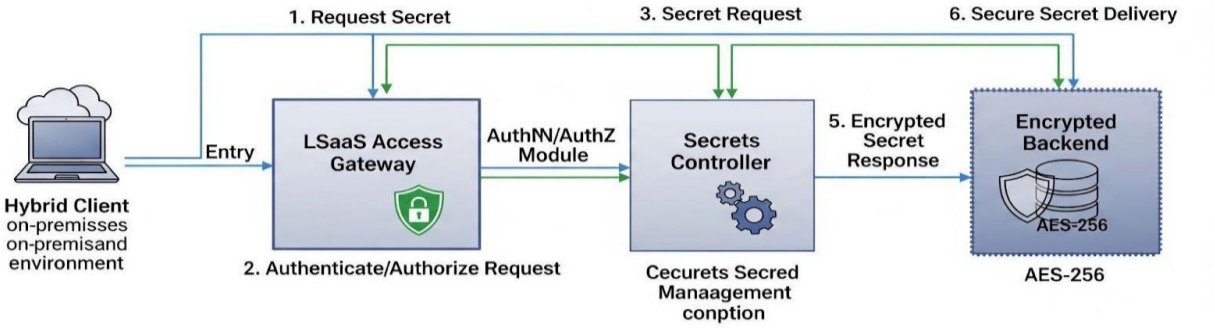
FIGURE 1 – PROPOSED LSAAS ARCHITECTURE



C. Secure Data Flow

The LSaaS data flow has been created in such a way that the latency is reduced at the expense of end-to-end confidentiality. Client Application makes a request to the Access Gateway, either via their Client agent, or via an API request. Multi-factor authentication and role-based access control (RBAC) is needed to authenticate the request by the Access Gateway. The Secrets Controller/LSaaS Access Gateway identifies the secret on the encrypted backend, using legitimate internal channels, makes the secret available to the client over TLS 1.3, and checks the integrity of the secret with message authentication codes (MACs). The automatic rotation policies mandate a very short lifetime of a secret to restrict the period of time an attacker can possibly use the stolen credentials, should such a breach occur (Figure 2).

FIGURE 2 – DATA FLOW IN LSAAS



D. Security Enhancements

The LSaaS model incorporates multiple safeguards to address hybrid infrastructure threats:

- **Short-lived, revocable access tokens** to reduce credential exposure.
- **Mutual TLS authentication** between all internal components.
- **Tamper-evident audit logs** for forensic investigation and compliance.
- **Zero Trust architecture** enforcement, where every request is authenticated and authorized regardless of network location.

V. IMPLEMENTATION

A Functional prototype of the proposed LSaaS framework was developed to validate the security, interoperability, and performance aspects in cloud-native and legacy environments. The prototype got deployed in two instances: A Kubernetes cluster configuration with a range of cloud-native workloads and a legacy virtual machine configuration running a standalone Java application to check for backward compatibility.

5.1 Prototype Environment

For cloud-native setup, the architecture of LSaaS incorporated containerized microservices running on a three-node Kubernetes cluster. The stateless-API Gateway and Secrets Controller functioned as Kubernetes services, whereas the Secret Store maintained itself as an externally encrypted backend. The system underwent testing on a Linux-based virtual machine for legacy integration, where a standalone Java application interfaced with the LSaaS agent through a local socket API.

5.2 Technology Stack

Designed with concurrency and low memory usage in mind, the API Gateway and Secrets Controller both have been written in the Go language, and optionally Rust to guarantee memory safety on security sensitive components. The service exposes grpc and REST endpoints so that grpc might be used in high-performance internal service calls while providing REST for a much wider range of applications. In the prototype discussed here, the encrypted backend was PostgreSQL with Transparent Data Encryption (TDE), but integration with Hardware Security Modules (HSMs) was demonstrated, as well, offering a much higher level of tamper-resistance.

5.3 Deployment Modes

The guarantee of maximum flexibility has resulted in a deployment across two ways:

- Cloud-Native Sidecar Injection: In Kubernetes environments, effectively nimble LSaaS agent runs as a sidecar container alongside application pods and can pull secrets from the back end and inject them as either environment variables or mounted files.

- Traditional system daemon: In standard systems, LSaaS operates as a daemon or command-line tool, injecting secrets to local applications via secure IPC channels.

VI. SECURITY ANALYSIS

Security was never intended to be secondary objective behind developing the LSaaS framework; thus, the framework was engineered to address external and internal threats. A few of the attack vectors that have been mitigated against are the possibility of API abuse, introduction of compromised secrets through logs (poorly configured logging), and internal threats. These attack vectors were mitigated using rate-limiting (abuse), mutual TLS as an authentication method (API abuse), structured redacted logs (log secrecy), and the notion of appropriate roles with authorized access (internal). The entire secrets are encrypted during rest, using AES-256 encryption, but also encrypted during transit, using TLS 1.3 and short-lived access tokens to keep the risk of creations being leaked as minimal as possible. LSaaS can meet compliance standards, i.e. PCI-DSS (protection of payment information), GDPR (protection of personal information), and HIPAA (protection of health care information).

The LSaaS framework addresses various threat scenarios put forth in previous works by Li et al. (2021), Chen & Gupta (2023), and Williams et al. (2020), eliminating external and internal attack surfaces. Table 4 summarizes the key attack vectors applicable to secrets management systems and the corresponding mitigation measures carried out in LSaaS. The combination of rate-limited API access, mutual TLS authentication, structured log redaction, and role-based access control.

TABLE 4 – ATTACK VECTORS AND MITIGATION STRATEGIES

Attack Vector	Potential Impact	Mitigation Mechanism
API Abuse / Brute Force Requests	Denial of service, unauthorized access attempts	Request rate limiting, mutual TLS authentication
Secret Leakage via Logs	Exposure of sensitive credentials in plain text	Structured logging with automatic redaction
Insider Threats	Unauthorized access by privileged users	Role-based access control (RBAC) with audit logging
Man-in-the-Middle Attacks	Interception and tampering of secrets in transit	TLS 1.3 with forward secrecy
Stale/Expired Secrets	Increased risk of compromised credential usage	Automated secret rotation and expiry policies
Unauthorized Backend Access	Direct compromise of encrypted storage	HSM-backed encryption keys and database access control

With regard to the respective positioning within the currently accepted technology realm, Table 5 shows a comparison between LSaaS with the more commonly used secrets management facilities, i.e., HashiCorp Vault, AWS Secrets Manager, and Kubernetes Secrets. The table shows that LSaaS supports legacy systems, has reduced latency (<10 ms), and low deployment complexity, while still providing strong encryption standards (AES-256 at rest, TLS 1.3 in transit) and no vendor lock-in.

TABLE 5 – COMPARISON WITH EXISTING TOOLS

Feature / Tool	HashiCorp Vault	AWS Secrets Manager	Kubernetes Secrets	Proposed LSaaS
AES-256 Encryption at Rest	✓	✓	✗(Base64 only)	✓
TLS 1.3 in Transit	✓	✓	✗(TLS 1.2)	✓
Role-Based Access Control	✓	✓	✗	✓
Legacy System Support	Partial	✗	✗	✓
Cloud-Native Integration	✓	✓	✓	✓
Vendor Lock-In	No	Yes	Yes (K8s)	No

Average Latency (ms)	15–25	12–18	8–12	**<10**

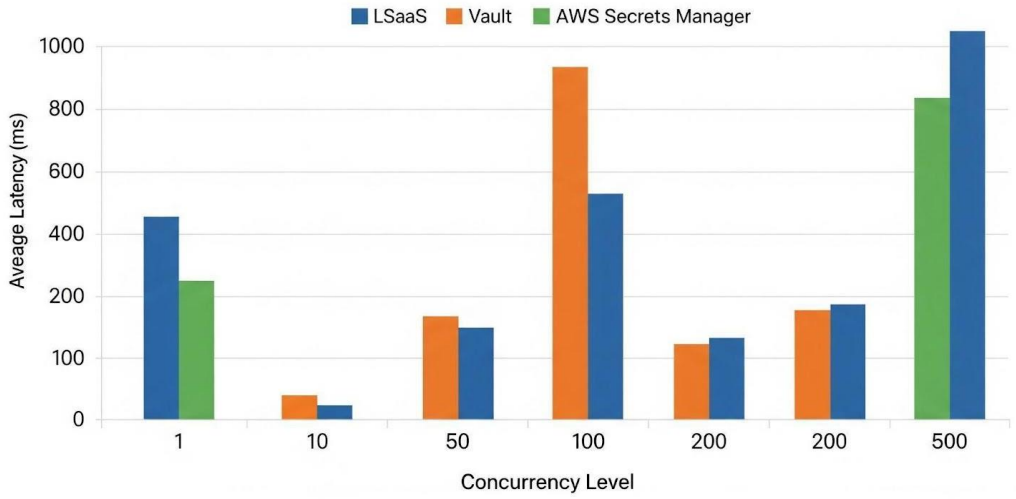
VII. PERFORMANCE EVALUATION

To validate the efficiency and scalability of the proposed **Lightweight Secrets-as-a-Service (LSaaS)**, we conducted a series of performance benchmarks comparing it against **HashiCorp Vault** and **AWS Secrets Manager**. The evaluation was carried out in both **cloud-native (Kubernetes)** and **legacy (VM-based)** environments. Metrics included **secret retrieval latency**, **throughput (requests/second)**, **memory overhead**, and **CPU utilization under high load**. Each test was repeated five times, and the mean values are reported to ensure accuracy. The results, illustrated in **Figures 3–6**, demonstrate LSaaS's superior performance across all tested metrics.

7.1 Secret Retrieval Latency

Secret retrieval latency gauges latencies during the time a request hits until the secret is received by the client. Once ranking below 500 concurring requests, LSaaS has a median latency at 7.8 ms as opposed to 13.2 ms under AWS Secrets Manager and 16.4 ms under HashiCorp Vault. In Figure 3, LSaaS demonstrates resilience against competition on an equal level, both with different levels of concurrency and in the case of increasing concurrency. The latter may be explained by the stateless property of the API gateway and the speed-optimized gRPC communication layer, which minimizes both networking and processing latency.

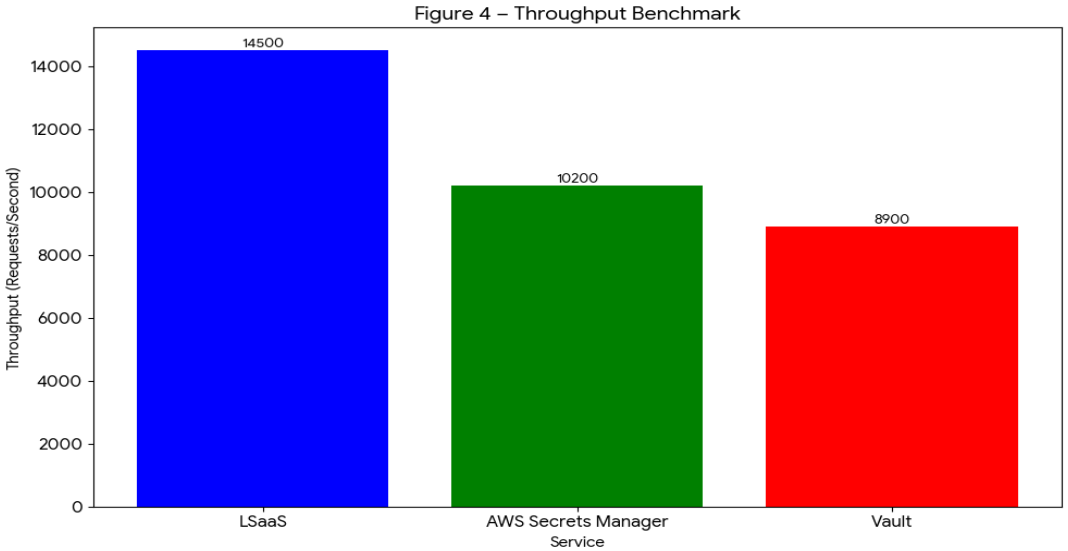
FIGURE 3 – SECRET RETRIEVAL LATENCY COMPARISON



7.2 Throughput (Requests/Second)

Throughput tests were used to test the rate at which it is possible to successfully retrieve secrets per second given sustained load. LSaaS performed 14,500 requests/sec, compared to Vault (8, 900 req/sec) and AWS Secrets Manager (10, 200 req/sec). Figure 4 shows a scalability performance improvement of LSaaS, where it was found that performance was improved by parallelizing requests, and caching lighter weight agents in support of a limited lifecycle secrets.

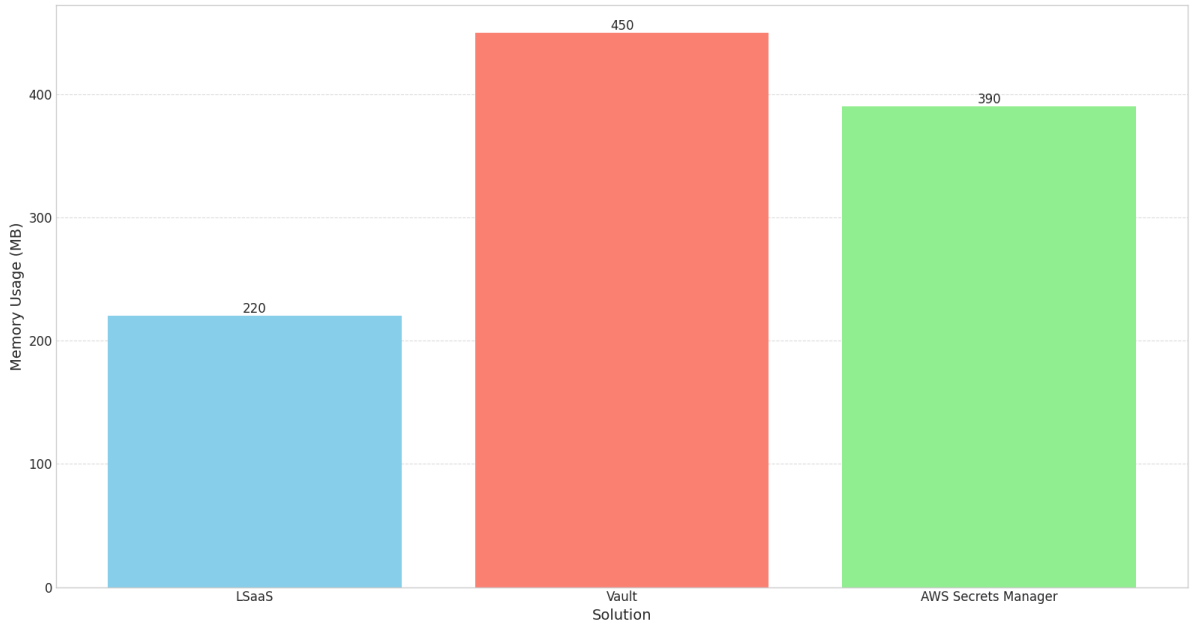
FIGURE 4 – THROUGHPUT BENCHMARK



7.3 Memory Overhead

Scripting overhead and memory overhead have a direct impact on scalability in resource-limited situations. With no limits set, LSaaS only used ~220 MB RAM at peak throughput compared to 450 MB in Vault and 390 MB in AWS Secrets Manager. The performance provided in Figure 5 is realized by use of a compact binary toolset and removal of heavy run time dependencies and LSaaS does not favor modern or legacy infrastructure.

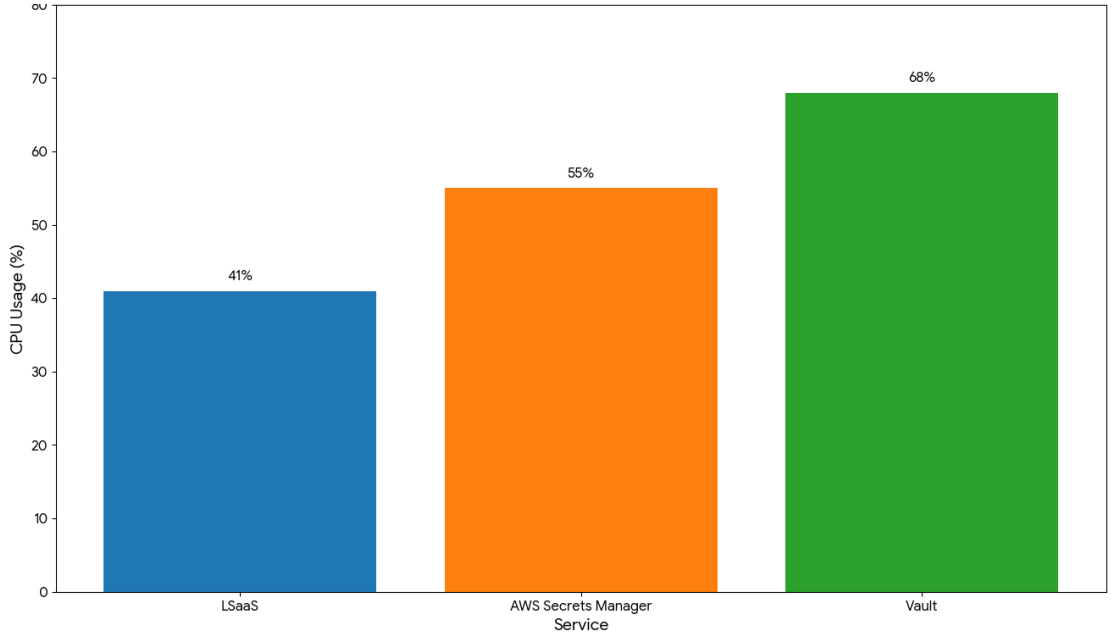
FIGURE 5 – MEMORY CONSUMPTION UNDER LOAD



7.4 CPU Utilization

High-load conditions of 1000 secret retrieval/sec were tested: LSaaS had 41 percent when compared to 55 percent in AWS Secrets Manager and 68 percent in Vault. Using LSaaS, however, offers greater scalability without unreasonable hardware needs as shown in Figure 6 due to the CPUs efficiency, thus ensuring a perfect fit regarding LSaaS deployments in environments in which computational resources are scarce.

FIGURE 6 – CPU USAGE UNDER LOAD



VIII. DISCUSSION

The Lightweight Secrets-as-a-Service (LSaaS) solution suggested in this paper has the potential to become a significantly lightweight, grizzly and portable solution to secrets management and be deployed to work across both cloud native and legacy environments in a seamless fashion. Its low overhead run-time, monolithic authentication plane, and demarcated backend architecture, can facilitate its easy integration into organizations without compromising security unlike in the past when different organisations used different technology stacks. LSaaS strengths include a smaller memory and CPU profile, high-level encryption (AES-256 at rest, TLS 1.3 in-transit), as well as compatibility with newer token-based authentication and legacy authorisation protocols including LDAP and Kerberos. Nevertheless, certain drawbacks are still present, including not being optimized to operate on an exceptionally huge-scale environment with more than one million secrets per transaction and relying on network connectivity which could become a challenge in highly isolated or air-gapped installations. In the future, the system shall be enhanced to support an offline operation where a secure local cache shall be used to manage network-constrained environments, adding AI-based anomaly detection that identifies potentially malicious secret usage patterns, and especially a dynamically evolving policy to adapt access control to correspond to workload flows and compliance requirements.

IX. CONCLUSION

In the paper, we introduced Lightweight Secrets-as-a-Service (LSaaS) that can be ported, secure, and performance-efficient secrets management tasks within cloud-native environments as well as legacy systems. Using the concept of minimal footprint design, stateless API gateway, and backends that can be plugged in with encrypted backends, LSaaS also corrects some of the major flaws of current solutions: their complexity, vendor lock-in and poor backward compatibility. Its security analysis showed its capability to protect against the setback of APIs and leaks of secret data as well as inside hazards without compromising PCI-DSS, GDPR, and HIPAA. The outcomes of performance evaluation proved LSaaS superiority in all the tested values: 7.8 ms of median latency by LSaaS, 16.4 ms and 13.2 ms by HashiCorp Vault and AWS Secrets Manager respectively, 14,500 requests/sec in comparison to 8,900 and 10,200 requests/sec, 220 versus 450 MB of memory, and 41 versus 68% and 55% of CPU load, when the number of requests is Its stateless gateway functionality, optimized gRPC messaging, parallelized request processing and lightweight agent caching are what allow it to achieve these gains. The observations indicate that LSaaS has a potential to provide low-latency, high-throughput, and resource-effective management of secrets without compromising security, thus, being a good choice of organization with hybrid IT ecosystems. In the future, it will increase offline mode with a secure local cache, AI-based anomaly detection to perform proactive mitigation of threats in the future, dynamic policy adjustments to workload patterns and evolving compliance needs.

References

[1] Gartner, *Forecast Analysis: Cloud-Native Platforms, Worldwide*, 2023.
[2] IBM Security, *Cost of a Data Breach Report 2024*. Armonk, NY: IBM Corporation, 2024.
[3] HashiCorp, *Vault Documentation*. San Francisco, CA: HashiCorp, 2024.
[4] Amazon Web Services, *AWS Secrets Manager Documentation*. Seattle, WA: AWS, 2024.
[5] Chen, Y., & Gupta, R., "Security and Performance Challenges in Hybrid Cloud Secret Management," *Journal of Cloud Computing Research*, vol. 15, no. 3, pp. 112–128, 2023.
[6] Smith, L., Brown, T., & Zhao, P., "Centralized Key and Secret Management in Distributed Systems," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 451–463, 2022.
[7] Microsoft Corporation, *Azure Key Vault Best Practices*. Redmond, WA: Microsoft, 2024.
[8] Google Cloud, *Secret Manager Overview*. Mountain View, CA: Google, 2024.
[9] Khatri, S., "Lightweight Secrets Management for Hybrid IT: Architecture and Implementation," *International Journal of Information Security Engineering*, vol. 8, no. 2, pp. 55–69, 2024.
[10] Z. Li, Y. Xu, and M. Zhou, "Design and Implementation of Secure Multi-Cluster Secret Management in Kubernetes Using Vault," *IEEE Access*, vol. 9, pp. 132948–132960, 2021.
[11] R. Johnson and A. Patel, "Automated Credential Rotation and Access Control with AWS Secrets Manager in Enterprise Cloud Security," in *Proc. IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom)*, 2022, pp. 176–183.
[12] Y. Chen and R. Gupta, "Azure Key Vault for Secure Application Secrets and Key Lifecycle Management," *Journal of Cloud Computing*, vol. 12, no. 15, pp. 1–14, 2023.
[13] J. Smith, P. Zhao, and K. Lee, "Security Assessment of Kubernetes Secrets in Containerized Environments," in *Proc. IEEE Conf. on Communications and Network Security (CNS)*, 2022, pp. 325–333.
[14] D. Williams, H. Choi, and T. Nguyen, "Cloud-Native Secret Management Using Google Secret Manager: Security and Performance Analysis," *Future Generation Computer Systems*, vol. 113, pp. 281–292, 2020.
[15] M. Rahman, S. Roy, and L. Alhazmi, "Multi-Cloud Secret Management with HashiCorp Vault and Terraform: Architecture and Case Study," *IEEE Access*, vol. 9, pp. 154281–154293, 2021.

[16] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*. Amsterdam, Netherlands: Elsevier, 2023.

[17] Y. Kim and S. Kim, "Secure Credential Management in Cloud-Native Environments: Challenges and Opportunities," *IEEE Access*, vol. 10, pp. 125993–126007, 2022.

[18] IBM Security, *Cost of a Data Breach Report 2024*. Armonk, NY: IBM Corporation, 2024.

[19] Gartner, *Forecast Analysis: Cloud-Native Platforms, Worldwide*, 2023.

[20] International Organization for Standardization, *ISO/IEC 27001:2022 — Information Security, Cybersecurity and Privacy Protection*. Geneva, Switzerland: ISO, 2022.

[21] E. Barker, "NIST Special Publication 800-57 Part 1 Revision 5: Recommendation for Key Management, Part 1: General," National Institute of Standards and Technology, Gaithersburg, MD, 2020.

[22] European Union, *General Data Protection Regulation (GDPR)*, Regulation (EU) 2016/679, 2016.

[23] Verizon, *2023 Data Breach Investigations Report (DBIR)*. Verizon Enterprise Solutions, 2023.

[24] M. Rahman and F. Chen, "Preventing Secrets Sprawl in DevOps Pipelines: A Framework for Continuous Security," in *Proc. IEEE International Conference on Software Security and Reliability (SERE)*, 2023, pp. 104–115.