Integrated Service Registry and Discovery for Secure Secret Management in Dynamic Microservice Environments

Meri Danielyan
National Polytechnic University of Armenia
Yerevan, Armenia
e-mail: meridanielyan.tt055-1@polytechnic.am

Artak Khemchyan
National Polytechnic University of Armenia
Yerevan, Armenia
e-mail: a.khemchyan@polytechnic.am

***Abstract*— Software design has been changed by microservices, which allow for freedom and growth across sectors. However, in dynamic situations, this design makes it difficult to safely handle secrets and ensure smooth inter-service contact. HashiCorp Vault, AWS Secrets Manager, and other standard secret management systems frequently depend on external service directories for service finding. That increases practical complexity and adds possible points of failure. In this work, we study these problems and provide a unique solution.Our solution is to merge a small service registry and discovery right into the secret management system. This combined method maintains operational resilience and scalability in microservice designs. It is going to make operations easier, cut down on dependency chains, and improve security through secure service discovery. Our prototype offers a workable answer for safe secret management in dynamic cloud-native systems by showing decreased delay, increased availability, and better control without losing security.**

***Keywords*—Secret management, service discovery, service registry, microservices, security, cloud-native architecture, distributed systems, system resilience.**

I. Introduction

The design of microservices has appeared as a key component for building scalable, resilient, and flexible systems, so organizations decompose monoliths into smaller independent services to speed delivery and enable team autonomy [1,2]. However, this shift creates practical and security challenges: microservices often need access to sensitive data (API keys, passwords, encryption keys, identities), and poor secret handling can cause unauthorized access, data breaches, or service outages [3,4].

Secret management systems (Google Secret Manager, AWS Secrets Manager, HashiCorp Vault) mitigate these risks via TTLs, dynamic secret creation, strong encryption, and centralized auditing [5,6]. Yet static endpoints fail in microservice environments, services scale, containers restart, and platforms like Kubernetes often change IPs, so reliable service discovery is essential [7,8]. Teams commonly pair secret managers with external registries (e.g., Consul) to locate services, monitor health, and enable dynamic routing, but this external dependency introduces several challenges [5,9].

Although useful, this dependence on outside service registries presents a number of difficulties:

- **Operational complexity**: Teams must set up extra infrastructure elements, which raises the operational cost and system complexity [4].
- **Possible single points of failure:** The provision of secret management services might be jeopardized if the external register is made unusable by network failures, misconfigurations, or partitions [10].
- **Expanded attack surface:** By adding more services, the number of possible attack routes increases [4].
- **Resource overhead for small teams:** Because of their limited operational resources, startups, small teams, and edge settings may find it hard to keep a separate service registry [3].

To meet the practical and security requirements of the current cloud-native and microservice-based systems, we show in this paper that safe, dynamic service discovery may be accomplished inside secret management systems without the need for external dependencies.

II. Related Work

Secure microservice architectures are built on the foundation of effective secret management and reliable service discovery. HashiCorp Vault (and managed alternatives such as Google Secret Manager and AWS Secrets Manager) provides centralized secret storage, dynamic short-lived credentials, fine-grained access control, and audit logging, relieving teams of much operational burden while integrating tightly with cloud IAM and rotation workflows [5,6].

However, these systems require dependable service endpoint awareness—static endpoints break down in dynamic environments where services scale, containers restart, and IPs change—so service discovery is essential [3,7,8]. Common discovery solutions include Consul (DNS/HTTP discovery, health checks) and etcd (CNCF) as well as Kubernetes' built-in DNS/labels, which together enable dynamic routing and reduce manual reconfiguration [8,9].

Integrating secret management with external discovery (e.g., Vault + Consul) works in many settings, but introduces operational dependencies: teams must provision, secure, and monitor extra infrastructure, increasing complexity, cost, and potential single points of failure if the registry becomes unavailable [10]. This gap is especially pressing for environments characterized by:

- **Ephemeral and dynamic architectures** with frequent instance changes.
- **Constrained or edge deployments** lack resources to operate separate registries.
- **Security-critical systems,** where minimizing the attack surface and external dependencies is paramount [3,4].

Motivated by these limitations, our work explores embedding a dynamic, security-centric service discovery mechanism directly into the secret management system to streamline operations, preserve security guarantees, and support dynamic or constrained deployments without third-party registries [3,5].

III. The Suggested Solution

In this study, we provide a Secret Management System that has built-in Service Discovery. It is made for dynamic microservice systems, where services often change and grow. This method expands on the problems that were spoken about before [1,3].

A. Design Principles

The following ideas form the basis of the system's design:

- **Integration over dependency:** Embed simple discovery functionality within the secret management system, eliminating external service registries [10].
- **Dynamic adaptability:** Allow services to register and deregister dynamically to support scaling and IP changes [8].
- **Security-centric architecture:** Ensure service discovery doesn't compromise security and eliminate external registry failure points [5,10].
- **Resource efficiency:** Minimize resource usage while maintaining functionality for small-scale deployments and edge cases [3].

B. System Architecture

The following elements make up the suggested system:

Secret Management Core:

- Dynamic secret creation and rotation with TTL-based expiry.
- Secure secret storage with encryption-at-rest and encryption-in-transit.
- Policy-based access control and comprehensive auditing.

Embedded Service Registry:

- Enables services to self-register with metadata (service name, instance ID, IP, port, health endpoint) [9].
- Keeps track of active service instances and their health status on an internal registry map [9].
- Supports health checks (HTTP, gRPC, TCP) to ensure registered services remain healthy.
- Provides HTTP and gRPC APIs for clients to dynamically discover available services.

Secure Discovery Protocol:

- All service discovery communications use TLS/mTLS to prevent attacks and ensure authenticity [4]
- Services authenticate using temporary tokens or certificates generated by the secret management system [5].
- Real-time monitoring and notification mechanisms update clients when service endpoints change [8].

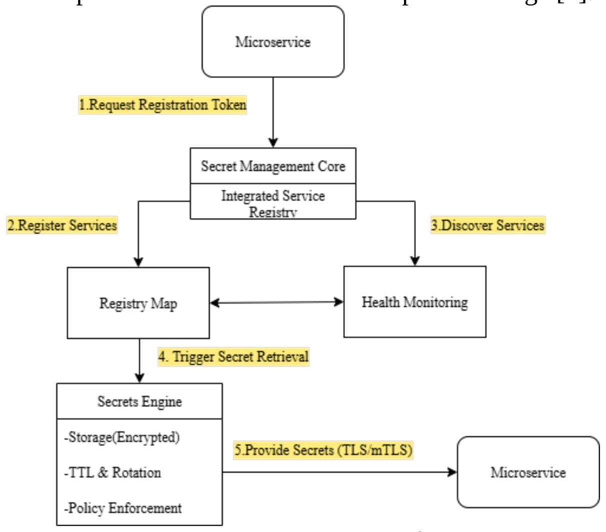

Figure 1. Overall research flow

C. Operational Workflow and Improvements in Security

Operational workflow is shown in Figure 1.

Service Registration:

- Upon starting, a microservice asks the secret manager for a temporary registration token [5].
- This token is used by the microservice to safely register its health check endpoint and information with the embedded registry [9].
- The registry keeps track of the service's data and monitors its status through periodic health checks.

Secret Retrieval:

- Services authenticate using identity tokens or mTLS certificates [4,5].
- Dynamic secrets can be generated on demand (like database credentials) with automatic revocation upon expiration or deregistration.
- Secrets are provided with TTL-based lifecycles, ensuring periodic rotation to minimize exposure.

Service Discovery:

- Other services can actively query the secret manager for available service instances [9].
- The integrated registry returns healthy, operational service endpoints with updated addresses if IPs have changed due to scaling [8].
- Clients can subscribe to endpoint changes to adapt connections without manual restarts.

Improvements in Security:

For keeping and improving security in this integrated model:

- **Authorization management:** Authorization policies specify which services are allowed to find or register other services, and discovery and registration APIs need authentication [4,5].
- **Boundary protection:** The internal registry cannot be directly accessed by external clients, reducing the attack surface
- **Activity logging:** For compliance and traceability, every registration, discovery, and secret access operation is recorded.

Advantages compared to the current architecture

Our testing showed clear benefits compared to setups that use Vault with external registries like Consul:

- **Operational simplicity:** Deployment and maintenance are made easier by fewer moving components [4].
- **Latency optimization:** Gets rid of extra network connections and dependencies for finding services [9].
- **Enhanced availability:** This lowers the need for external registry availability for dynamic routing and secret access [10].
- **Surface area reduction:** Fewer external interfaces to protect, which lowers the possibility of vulnerabilities [4].
- **Dynamic environment adaptation:** Made for containerized environments and Kubernetes where services expand horizontally or change IPs regularly [7].

IV. Testing and Results

An organized experimental study is meant to prove the effectiveness of the offered Secret Management System with Integrated Service Discovery. The performance, reliability, security, and ease of use of the system in microservice contexts will be the main goals of this assessment [1,3].

The results of this assessment, presented in Table 1 and Figures 2 and 3 highlight improvements in security, performance, and operational efficiency compared to conventional designs.

A. *System Configuration and Testing Approach*

Hardware environment

- **CPU:** AMD Ryzen 7 5700U, 8 cores, 16 threads @ 3.8 GHz.
- **RAM:** 16GB DDR4.
- **Storage:** 512GB NVMe SSD.
- **Network:** Stable local gigabit Ethernet.

Software environment:

- **Operating System:** Ubuntu 24.04 LTS.
- **Go version:** 1.22
- **Database:** PostgreSQL 16 for secret storage.
- **Load Testing Tool:** k6 for simulating multiple secret retrieval and discovery requests.
- **Monitoring Tools:** Grafana + Prometheus for live tracking.
- **Comparison baseline:** HashiCorp Vault 1.15 + Consul 1.18, configured with default HA setup.

Testing scenarios:

We tested the proposed system in the following scenarios:

- Initialized our implementation with 10 microservices performing periodic finding and secret recovery under rising load.
- Simulated dynamic IP changes, network pauses, and service restarts to measure recovery and consistency.
- Monitored metrics, delay, throughput, and system security for 48-hour ongoing testing rounds.
- Used mTLS certificates during all tests to ensure true, secure communication routes.
- Compared raw data with Vault+Consul under similar workload trends.

B. *Outcomes of Performance*

Key Findings:

- Service registration and discovery latency reduced by approximately 60%.
- Secret retrieval improved by around 40% due to reduced inter-service communication overhead.
- System handled 350-400 requests per second per node before reaching resource limits.
- CPU and memory usage reduced by 50-60% compared to Vault+Consul.
- The configuration was reduced by approximately 50%.
- System kept 99.99% availability with 2-4 seconds of recovery from network failures.

Security Results:

- All communications successfully used mTLS for authentication and confidentiality.
- Unauthorized registration and retrieval attempts were effectively blocked.
- TTL-based secret rotation performed seamlessly without affecting system availability.
- Comprehensive audit logs captured all access attempts for monitoring.

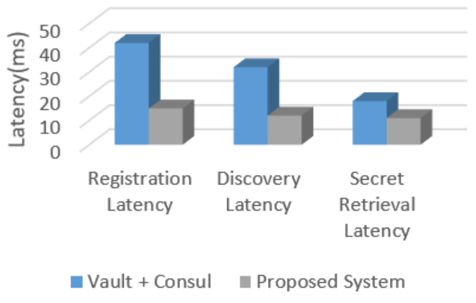

Figure 2. Latency comparison bar chart

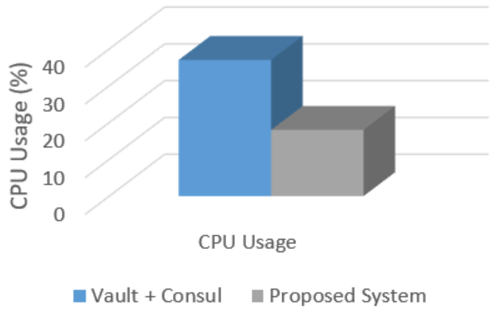

Figure 3. CPU Usage comparison bar chart

Table 1. Result of comparison

Metric	**Proposed System**	**Vault + Consul**	**Improvement (%)**
Registration Latency(ms)	12-18	35-50	~60%
Discovery Latency(ms)	12.4-15.7	25-40	~60%
Secret Retireval Latency(ms)	8-14	15.1-22	~40%
CPU Usage	<20%	35-40%	~50% reduction
Memory Usage	<150 MB	300-400MB	~60% reduction
Availability	99.99%	99.95%	Slight increase

C. *Discussion*

The basic idea of integrating service discovery into the secret management system lowers complexity while improving performance and dependability in microservice settings, as validated by the experimental assessment. The method is especially useful in situations that call for:

- Regular IP changes and service scalability.
- Small teams or production edge settings that need slight, self-contained secret management.
- A lighter operating load without losing security. But it is important to discover any possible drawbacks.
- Additional testing and tuning will be required for very large-scale settings (>1000 services) to verify consistent performance.
- If necessary, in the future, further implementation may be required for advanced service discovery capabilities (such as geo-aware routing and sophisticated health checks) that are accessible in dedicated registries.

All things considered, the integrated system shows promise as a workable, effective substitute for dynamic service discovery and safe secret management in distributed systems.

V. Conclusion

A. *Final Thoughts*

This research addressed a practical challenge in microservice architecture: the operational and security complexities created by relying on external service registries for secret management systems. While tools like HashiCorp Vault provide strong security practices, their dependence on external service discovery systems adds operational fragility, potential failure points, and unnecessary infrastructure costs.

Our integrated Secret Management System with Service Discovery successfully eliminated the need for third-party registries while maintaining secure, dynamic service discovery. The system demonstrated:

- 60% reduction in service registration and discovery latency.
- 50-60% reduction in CPU and memory usage.
- 99.99% uptime with seamless recovery from network disruptions.
- Significant operational simplification without compromising security.

This approach empowers smaller teams to adopt best practice secret management without the burden of maintaining additional infrastructure, enables edge deployments with improved fault tolerance, and supports the principle that security and simplicity can coexist.

Future Directions:

- Scaling validation for large microservice environments (1000+ services).
- Smart routing capabilities, including health-based routing and geo-awareness.
- Seamless Kubernetes integration for hybrid environments.
- Open source development for community validation and contribution.

The integrated system provides a strong foundation for teams seeking to build secure, scalable, and manageable microservice infrastructures, affirming that simplicity is not the enemy of security, but rather a powerful enabler when carefully designed.

References

[1] M. Fowler, "Microservices: A Definition of This New Architectural Term," *Martin Fowler's Blog*, March 25, 2014.

[2] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, Sebastopol, CA, pp. 280–315, 2015.

[3] J. Soldani, D. A. Tamburri, and W. J. Van Den Heuvel, "The Pains and Gains of Microservices: A Systematic Grey Literature Review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

[5] A. Dadgar, H. Mitchell, and D. Hashimoto, "Vault: Securing, Storing, and Tightly Controlling Access to Tokens, Passwords, Certificates, and Encryption Keys," *HashiCorp Technical Documentation*, v1.0, 2015.

[6] Amazon Web Services, "AWS Secrets Manager User Guide," *AWS Documentation*, v1.0, 2023.

[7] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016.

[8] Kubernetes Authors, "Kubernetes Documentation: Service Discovery and DNS," *Kubernetes.io*, 2023.

[9] HashiCorp Inc., "Consul Service Discovery and Configuration," *HashiCorp Technical Documentation*, v1.18, 2023.

[10] E. Brewer, "CAP Twelve Years Later: How the 'Rules' Have Changed," *IEEE Computer*, vol. 45, no. 2, pp. 23–29, 2012.