*Human Factors in Design, Engineering, and Computing*, Vol. 238, 2026, 1973–1985
https://doi.org/10.54941/ahfe1008425

# Threshold-Based Key Management Across Heterogeneous Multi-Cloud Key Management Systems

**Musa Adamou[1], Ivan Nkwawir[2], and Amir Schur[1]**

[1]National University, San Diego, CA 92123, USA
[2]George Washington University, Washington, DC 21211, USA

## ABSTRACT

As organizations increasingly adopt multi-cloud architectures, cryptographic key management has emerged as a critical security challenge. Although workloads may be distributed across multiple cloud service providers (CSPs), cryptographic authority often remains concentrated within a single provider's Key Management System (KMS), creating potential single points of failure, vendor lock-in, insider threat exposure, and operational dependencies. Existing threshold cryptography research demonstrates the benefits of distributed trust; however, many proposed solutions assume homogeneous environments or rely on centralized intermediaries that diminish the resilience advantages of multi-cloud deployment. This research introduces a threshold-based cross-cloud key management architecture that distributes cryptographic trust across heterogeneous cloud-native KMS platforms, including AWS Key Management Service, Microsoft Azure Key Vault, and Google Cloud KMS. The proposed architecture is embodied in the Benzene Ring Secure Cross-Cloud System (BRSCCS), in which a secure orchestration layer serves as the protected nucleus. At the same time, independent cloud providers serve as separate trust domains holding distributed key shares. Using Shamir's Secret Sharing, a 256-bit AES master key is divided into multiple shares under a 2-of-3 threshold policy, ensuring that no individual cloud provider possesses sufficient information to reconstruct the complete key. The architecture incorporates several security-enhancing components designed for practical deployment in modern cloud environments. A confidential orchestration layer operating within a Trusted Execution Environment enables transient key reconstruction while minimizing key exposure. Federated identity services leverage short-lived credentials and workload identity federation to eliminate long-term secrets. Dynamic share rotation mechanisms periodically refresh distributed shares without modifying the underlying cryptographic key, reducing the impact of potential compromises.

**Keywords:** Multi-cloud security, Key management systems, Threshold cryptography, Shamir's secret sharing, Cloud security, Confidential computing, Distributed trust, Post-quantum cryptography

## INTRODUCTION

Cloud services have become integral to the global critical infrastructure sector's information technology. As cloud adoption has increased, particularly in financial institutions, financial regulators have raised concerns about

Received August 20, 2026; Revised October 2, 2026; Accepted October 8, 2026; Available online December 1, 2026

© 2026 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

potential concentration risk associated with cloud migration (Scott et al., 2023). Scott et al. (2023) argued that the goal of multi-cloud adoption is not to eliminate centralization but to manage concentration risk and improve resilience. Similarly, the European Securities and Markets Authority (ESMA, 2021) warned that cloud outsourcing may increase systemic risk when many organizations become dependent on a small number of cloud providers. These concerns have contributed to increased interest in multi-cloud architectures as a strategy to improve resilience and reduce dependence on individual cloud providers. Vendor lock-in remains a significant challenge in cloud computing due to the lack of standardization and interoperability across cloud providers (Opara-Martins et al., 2016). As cloud services have evolved into utility platforms supporting distributed applications and IoT ecosystems, organizations increasingly seek architectures that reduce dependence on individual providers (Alonso et al., 2023). Although multi-cloud approaches can improve resilience, many existing cloud services and architectures still rely on provider-specific capabilities, which can introduce operational dependencies and single points of failure (Karanjai et al., 2024).

This research addresses concentration risk by focusing on cryptographic control rather than infrastructure distribution alone. Given that concentration risk, vendor lock-in, and centralized cryptographic control remain unresolved, this research proposes a threshold-based key management approach across heterogeneous multi-cloud KMS systems, in which cryptographic keys are fragmented and distributed among multiple independent cloud providers.

## Problem Statement

Current multi-cloud environments face two related challenges: limited interoperability among heterogeneous cloud KMS platforms and continued reliance on provider-specific key management systems. Although workloads may be distributed across multiple clouds, cryptographic control often remains concentrated within individual providers, creating risks associated with vendor lock-in and single points of failure. This research addresses the problem through a threshold-based key management approach for heterogeneous multi-cloud KMS environments. Rather than concentrating cryptographic control within a single provider or homogeneous KMS platform, the proposed approach distributes key shares across multiple independent cloud KMS providers.

## Background and Related Work

Chandramouli et al. (2013) revealed that managing cryptographic keys is more complex in cloud environments than in enterprise IT environments due to differences in the ownership and control of the infrastructure on which both the KMS and protected resources reside. Opara-Martins et al. (2016) explained that vendor lock-in is a major barrier to the adoption of cloud computing, due to the lack of standardization. Their findings exemplified the importance of interoperability, portability, and standards in cloud computing. While the solutions proposed by Opara-Martins et al. address vendor lock-in and interoperability concerns, they do not fully address cryptographic

data protection. Huang and Chen (2018) noted that cryptographic keys are used to encrypt data, making key management a paramount issue. Zatlavi (2024) compared customer-managed key implementations across Azure, AWS, and Google Cloud (GCP). Azure associates a managed identity with the storage account for encryption operations, AWS relies on the credentials of the requesting principal during KMS decryption, and Google Cloud uses a service agent that performs cryptographic operations on behalf of storage services. Rai (2026) compared AWS KMS, Azure Key Vault, and Google Cloud KMS based on architecture, compliance levels, and service integration. Rai noted that AWS KMS uses a fully managed, multi-tenant HSM architecture, whereas AWS CloudHSM provides a dedicated, single-tenant deployment option.

Krishnan (2024) reviews AWS KMS, Azure Key Vault, and Google Cloud KMS and discusses their approaches to key storage, key rotation, and access control. CSPs offer integrated KMS solutions as part of their platforms, in which cloud-based KMS can be considered a service offering that the customer does not have to manage the underlying software or hardware (Krishnan, 2024). Amazon Web Services provides the AWS Key Management Service. This managed key management service enables the creation and management of cryptographic keys and their use to encrypt AWS services and applications (Hakatemia, 2023). Similarly, GCP provides Cloud Key Management Service, which allows users to create, import, and manage cryptographic keys and perform cryptographic operations in a single centralized cloud service (Krishnan, 2024; Rai, 2026). Across all providers, the importance of key management is emphasized in security guidance, noting that cryptographic operations enable secure communication and data encryption at rest, and that the security of these operations all relies on proper key management (NSA/CISA, 2024). These three cloud platforms serve as independent trust domains for the proposed multi-party threshold architecture.

The multi-party paradigm of threshold cryptography enables threshold schemes that apply principles of secure multiparty computation (MPC) to achieve protocols for the secure distribution of trust in the operation of cryptographic primitives (NIST, 2026). In a threshold cryptosystem, the private key is typically distributed among the parties that constitute the system via cryptographically secure secret sharing, with the goal of generating a private key that is jointly held by the parties (Cachin, 2023). Shamir (1979) introduced secret sharing as a method for dividing a secret into multiple shares such that the secret can be reconstructed only from a required subset of shares. NIST describes threshold cryptography as an approach in which a secret key is distributed across multiple parties, and a threshold number of parties must participate in a cryptographic operation. Le Mouël et al. (2024) similarly describe threshold cryptography as the distribution of cryptographic capability among multiple participants.

Honzik (2024) describes Shamir's Secret Sharing (SSS) as a secret sharing algorithm developed by Adi Shamir in 1979 that can be used to introduce a threshold requirement for protecting Bitcoin. SSS allows users to split a key into several distributed shares, with only a certain threshold of the shares needed to reassemble the key. Honzik (2024) notes that SSS can be used to design quorums such as 2-of-3 or 3-of-5, like multisig. Honzik (2024)

further explains that the private key must exist in one place at one time before it is first split into shares and again when the shares are recombined to approve a withdrawal. According to Honzik (2024), these vulnerabilities create temporary single points of failure, meaning that SSS itself does not offer truly institutional-grade security, unlike multisig or MPC. Honzik (2024) explained that MPC allows multiple participants to jointly perform a computation while each participant's contribution remains private. When applied to bitcoin custody, MPC involves distributed shares, such as SSS; however, the shares are neither split from a private key nor used to reconstruct it. Instead, multiple parties compute a single signature directly from a threshold of their shares. Desmedt (2025) described secret sharing as a method in which a dealer shares a secret among $n$ parties, represented as $P_1, P_1, \ldots, P_n$. Authorized subsets of parties can reconstruct the secret.

The Cloud Security Alliance (CSA) described multi-cloud KMS as a challenge involving multiple CSPs, applications, and environments. The publication compares key-management models, including BYOK, HYOK, customer-managed KMS, customer-held KMS, and third-party multi-cloud KMS solutions (CSA, 2025). The CSA describes several cloud key responsibility models. In the Provider-Managed Key (PMK) model, the CSP generates, stores, uses, rotates, and backs up the encryption keys. In the Customer-Managed Key (CMK) model, the customer directs key lifecycle activities such as generation, usage, and rotation, while the CSP typically stores and uses the keys within its Key Management Service (KMS) or Hardware Security Module (HSM). CSA further describes Bring Your Own Key (BYOK) as a variant of CMK in which customers generate key material externally and securely import it into the cloud provider environment. Hold Your Own Key (HYOK) maintains customer control of keys within a customer-owned KMS, while data may remain stored in the cloud provider environment.

## Research Methodology

This study applies Design Science Research Methodology to develop and evaluate threshold-based key-management architecture for heterogeneous multi-cloud environments. Design Science Research is appropriate when knowledge and understanding of a problem and its solution are developed through the creation and application of an artifact (Hevner et al., 2004). Hevner et al. identified design science paradigm as a research methodology concerned with creating and evaluating innovative artifacts intended to address identified problems.

The research process was organized in accordance with the Design Science (DS) Research Methodology described by Peffers et al. (2007), which includes problem identification and motivation, definition of solution objectives, design and development, demonstration, evaluation, and communication. In the present study, the problem-identification stage examined cryptographic concentration risk, vendor dependence, limited interoperability, and fragmented key-management controls in heterogeneous multi-cloud environments. The solution objective was to develop an architecture that distributes cryptographic authority across independent cloud trust domains

without requiring any single provider to hold the complete master key. Peffers et al. (2007) stated that DS is of importance in a discipline oriented to the creation of successful artifacts.

During the design-and-development stage, the Benzene Ring Secure Cross-Cloud System (BRSCCS) was developed as the research artifact. The architecture assigns separate key shares to three representative trust domains corresponding to AWS Key Management Service, Microsoft Azure Key Vault, and Google Cloud KMS. A 256-bit AES master key is divided through Shamir's Secret Sharing under a 2-of-3 threshold configuration. The secure orchestration layer is responsible for coordinating share retrieval, threshold reconstruction, authorized cryptographic use, share rotation, and lifecycle-management activities. The demonstration and evaluation stages were conducted through an early Python prototype executed in Google Colab environment. The prototype implemented Shamir's Secret Sharing over GF($2^8$), using the AES reduction polynomial 0x11B, and reconstructed the secret through Lagrange interpolation at zero. Three instances of a mock cloud KMS class represented the participating AWS, Azure, and Google Cloud trust domains. Each simulated provider protected its assigned share using a separately generated AES-256 root key and AES-GCM authenticated encryption.

The prototype evaluation examined key-share generation, reconstruction using the required threshold, operation during simulated provider outages, refusal of reconstruction below the threshold, AES-GCM encryption and decryption, key-share rotation, separation between old and refreshed share sets, and implementation cross-validation. The custom Shamir implementation was cross-validated against the PyCryptodome secret-sharing implementation and exercised using 20 randomly generated 256-bit keys. The evaluation was conducted in a local simulated environment. It demonstrates the functional behavior of the core threshold mechanism but does not constitute a live implementation across AWS, Microsoft Azure, and Google Cloud. Live cloud integration, Trusted Execution Environment deployment, formal security verification, penetration testing, and operational performance evaluation remain outside the scope of the present study.

## Prototype Validation and Results

Consistent with the demonstration and evaluation activities of Design Science Research Methodology, the BRSCCS artifact was evaluated through controlled prototype scenarios. Peffers et al. (2007) distinguish demonstration of an artifact's use from evaluation of how well the artifact supports the identified solution objectives. The present evaluation focused on the functional properties of the BRSCCS threshold mechanism in a simulated multi-cloud environment. The prototype first verified that each of the three possible two-share combinations reconstructed the original 256-bit AES master key. The tested combinations represented AWS KMS and Azure Key Vault, AWS KMS and Google Cloud KMS, and Azure Key Vault and Google Cloud KMS. The prototype also successfully encrypted and decrypted an AES-GCM-protected test payload during normal operation. Availability behavior was

examined by changing the availability state of the simulated providers. When the simulated AWS provider was unavailable, the orchestrator collected the remaining shares from the simulated Azure and Google Cloud providers and successfully decrypted the test payload. When both the simulated AWS and Azure providers were unavailable, only one share remained accessible. The orchestrator refused reconstruction because the required threshold of two shares was not satisfied. These results demonstrate functional threshold enforcement and single-provider-outage tolerance within the local simulation.

The prototype also evaluated dynamic share rotation. A refreshed set of shares was generated for the existing master key and distributed to the simulated providers. The refreshed Google Cloud share differed from the previous share, while data encrypted before rotation remained decryptable using the refreshed share set. In a separate scenario, an old Google Cloud share was combined with a refreshed Azure share. The resulting value did not equal the original master key. This test demonstrates separation between the specific old and refreshed share sets evaluated in the prototype. It does not constitute a comprehensive analysis of all possible share-version or rollback attacks. The custom GF($2^8$) Shamir implementation was cross-validated against PyCryptodome's secret-sharing implementation. Because the PyCryptodome implementation processed 16-byte secrets, the 32-byte AES-256 key was evaluated as two 16-byte portions and successfully reconstructed. The custom implementation was also exercised using 20 randomly generated 256-bit keys, with successful reconstruction from the tested two-share combinations. The prototype includes a procedure for measuring local reconstruction latency across 200 iterations. Numerical latency values are not reported because the available prototype record does not include the resulting meaning and 95th-percentile values.

**Table 1:** Prototype and architectural evaluation of BRSCCS.

| Property Evaluated | Evaluation Mechanism | Result | Validation Status |
|---|---|---|---|
| Two-share reconstruction | 2-of-3 Shamir threshold | Each tested two-share combination reconstructed the original AES-256 key | Demonstrated in local prototype |
| Below-threshold operation | Orchestrator threshold check | Reconstruction was refused when only one share was reachable | Demonstrated in local prototype |
| Normal cryptographic operation | AES-GCM encryption and decryption | Test data was successfully encrypted and decrypted | Demonstrated in local prototype |
| Single-provider outage | Simulated provider availability state | Decryption succeeded when the simulated AWS provider was unavailable | Demonstrated in local simulation |

*(Continued)*

**Table 1:** Continued.

| Property Evaluated | Evaluation Mechanism | Result | Validation Status |
|---|---|---|---|
| Two-provider outage | Simulated failure of AWS and Azure | Reconstruction was refused because only one share remained accessible | Demonstrated in local simulation |
| Key-share refresh | Generation of a fresh Shamir share set | Refreshed shares differed while previously encrypted data remained decryptable | Demonstrated in local prototype |
| Tested share-set separation | Combination of one old and one refreshed share | The tested combination did not reconstruct the master key | Demonstrated for the tested scenario |
| Implementation cross-validation | PyCryptodome secret sharing | The AES-256 key was reconstructed as two 16-byte portions | Demonstrated |
| Randomized testing | 20 randomly generated 256-bit keys | The tested share combinations reconstructed the generated keys | Demonstrated |
| Live provider heterogeneity | AWS, Azure, and Google Cloud APIs | Provider-specific interfaces were not implemented | Not empirically validated |
| TEE protection | Proposed confidential orchestration | The prototype used a standard Colab runtime | Not implemented |
| Cross-provider policy consistency | Unified policy engine and native controls | Policy enforcement was described architecturally | Not implemented |
| Operational performance | Local 200-run benchmark procedure | Live-cloud latency, throughput, scalability, and cost were not measured | Not operationally validated |

## Limitations

The BRSCCS study has several limitations. First, the research artifact was evaluated through simulated cloud trust domains rather than live AWS KMS, Microsoft Azure Key Vault, and Google Cloud KMS services. The common mock-provider interface supports evaluation of the threshold coordination logic but does not test provider-specific APIs, identity systems, access controls, quotas, service availability, authentication processes, or operational constraints. Second, the prototype was executed within Google Colab. Consequently, it does not provide representative measurements of cross-cloud reconstruction latency, network overhead, throughput, scalability, availability, or cost. Although a local reconstruction-latency procedure was included, local execution excludes network communications, identity federation, provider throttling, and live KMS processing.

Third, the Trusted Execution Environment described in the proposed BRSCCS architecture was not implemented. In the prototype, key reconstruction occurred within the Python orchestrator. Deletion of the

Python key reference after the operation does not demonstrate secure or verifiable erasure of key material from runtime memory. The security of a future TEE deployment would depend on the selected confidential-computing environment, attestation mechanism, memory protection, implementation integrity, and defined threat model. Fourth, the prototype demonstrates functional threshold behavior but does not constitute formal verification of either the custom Shamir implementation or the complete BRSCCS architecture. The comparison between a single share and the master key does not independently demonstrate information-theoretic secrecy. That property derives from the underlying construction of Shamir and its associated mathematical and randomness assumptions. The current evaluation also excludes malicious-share detection, side-channel attacks, replay attacks, rollback attacks, compromised-orchestrator behavior, credential compromise, denial-of-service attacks, formal security analysis, and penetration testing.

Fifth, the share-rotation experiment demonstrates that a refreshed share set can be generated for the same master key and that the tested mixture of an old share and a refreshed share does not reconstruct that key. The prototype does not implement atomic cross-provider rotation, distributed share-version management, rollback prevention, secure destruction of retired shares, recovery from partial rotation, or lifecycle audit logging. Finally, the prototype evaluates a 2-of-3 configuration involving three representative cloud providers. Other KMS implementations, larger threshold configurations, regulatory requirements, and production-scale lifecycle processes may require architectural adaptation. The findings therefore establish prototype-level functional feasibility for the evaluated threshold workflow rather than demonstrated operational effectiveness across production multi-cloud environments.

## Architecture Design: The BRSCCS Benzene Ring Model

The BRSCCS architecture distributes cryptographic key shares across AWS KMS, Azure Key Vault, and Google Cloud KMS. Each cloud provider operates as an independent trust domain, while a secure orchestration layer coordinates the retrieval of shares, threshold reconstruction, cryptographic operations, and lifecycle management. The architecture applies threshold cryptography to ensure that no individual cloud provider can independently reconstruct the complete key. The BRSCCS Benzene Ring serves as the architectural model for the proposed system. The benzene ring is used as a visual representation of distributed cryptographic trust. Concepts such as resonance and electron delocalization are used as architectural metaphors for explaining how cryptographic authority may be distributed across multiple cloud providers. The model is intended as a design pattern for threshold-based key management in heterogeneous multi-cloud environments rather than as a cryptographic proof. Soliman and Järveläinen (2024) noted that metaphors are crucial not only to advancing scientific fields but also to facilitating the development of knowledge and understanding in general.

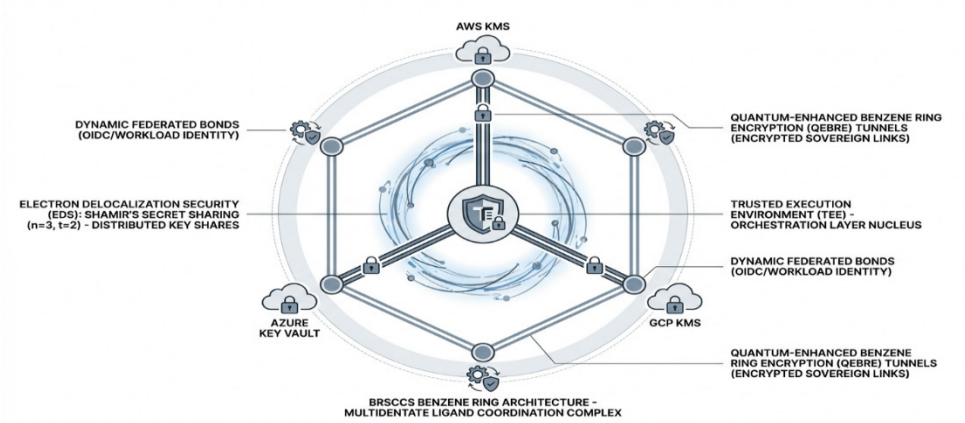



**Figure 1:** The figure illustrates the proposed BRSCCS architecture in which AWS KMS, azure key vault, and Google Cloud KMS function as independent trust domains surrounding a secure orchestration layer. Cryptographic key shares are distributed across cloud providers using a threshold-based model to prevent any single provider from controlling the complete key. The term "Electron Delocalization Security" should therefore be used as a metaphor for threshold-based key-share distribution.

Electron Delocalization Security is the conceptual model used in BRSCCS to represent distributed cryptographic trust. In benzene, π-electrons are delocalized across the ring rather than being confined to a single bond (LibreTexts, 2024). In the proposed architecture, cryptographic authority is similarly distributed across multiple cloud providers rather than being concentrated in a single KMS. The concept is intended as an architectural metaphor rather than a cryptographic proof. BRSCCS implements this concept using Shamir's Secret Sharing. A 256-bit AES master key is generated within the orchestration environment and divided into three shares under a 2-of-3 threshold policy. AWS KMS, Azure Key Vault, and Google Cloud KMS each store a separate share. Any two valid shares can reconstruct the master key. The threshold structure can be represented as follows: $n = 3$, shares with threshold: $t = 2$.

Thus:

$$Share_1 \rightarrow AWS\ KMS,\ Share_2 \rightarrow Azure\ Key\ Vault,\ and\ Share_3 \rightarrow Google\ Cloud\ KMS.$$

Any two valid shares can reconstruct the master key: $\{Share_i, Share_j\} \Rightarrow S$ Were, $i \neq j$. However, one share alone is insufficient: $\{Share_i\} \not\Rightarrow S$

## Threshold-Based Key Management Across Heterogeneous Multi-Cloud KMS Systems

Threshold-Based Key Management Across Heterogeneous Multi-Cloud KMS Systems forms the technical foundation of the BRSCCS architecture. A 256-bit AES master key is generated within the orchestration environment and divided into three shares using Shamir's Secret Sharing. AWS KMS, Azure Key Vault, and Google Cloud KMS each store one share under a

2-of-3 threshold policy. Under Shamir's construction, any two valid shares can reconstruct the secret, whereas a single share cannot recover the key (Shamir, 1979). According to Shamir (1979), mathematically, the master key is represented as a secret: $S \in \mathbb{F}_p$, where $\mathbb{F}_p$ is a finite field. For the proposed 2-of-3 model, the orchestrator constructs a polynomial of degree: $t-1 = 1$, so that: $f(x) = a_0 + a_1 x (\mathrm{mod} p)$, where: $a_0 = S$ and $a_1$ is selected uniformly at random from $F_P$. The three cloud providers receive distinct points on the polynomial:

$$Share_1 = (x_1, f(x_1)) \rightarrow AWS\ KMS$$
$$Share_2 = (x_2, f(x_2)) \rightarrow Azure\ Key\ Vault$$
$$Share_3 = (x_3, f(x_3)) \rightarrow Google\ Cloud\ KMS$$

The key can be reconstructed only when the orchestrator retrieves at least two valid shares. For example:

$$\{AWS\ Azure\} \rightarrow S,\ \{AWS, GCP\} \rightarrow S,\ \{Azure, GCP\} \rightarrow S$$
$$\text{but: } \{AWS\} \nvdash S,\ \{Azure\} \nvdash S, \text{ and } \{GCP\} \nvdash S$$

The architecture uses a 2-of-3 threshold structure in which AWS KMS, Azure Key Vault, and Google Cloud KMS each store a separate key share. No individual provider possesses sufficient information to reconstruct the complete master key. The orchestration layer coordinates the retrieval of shares, threshold reconstruction, policy enforcement, cryptographic operations, and lifecycle management across heterogeneous cloud environments. To reduce the risk associated with transient key reconstruction at the orchestration layer, key reconstruction occurs within a protected execution environment such as a Trusted Execution Environment (TEE). Figure 2 illustrates the threshold-based workflow used by the proposed architecture.

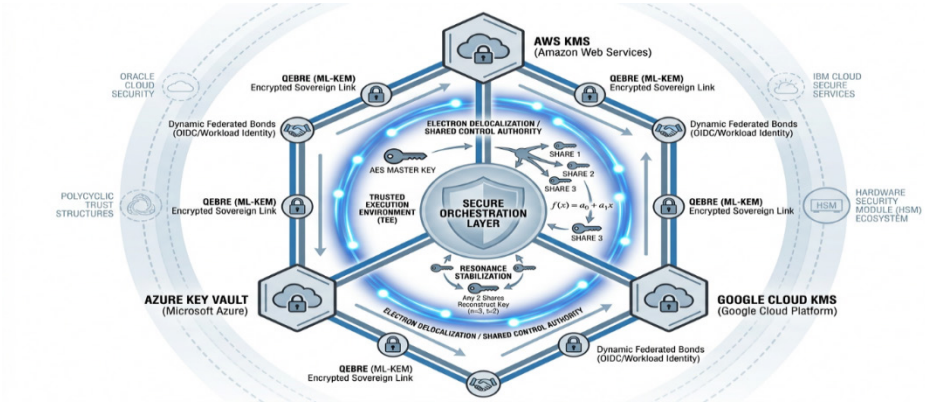



**Figure 2:** Threshold-based key management across heterogeneous multi-cloud KMS systems. The BRSCCS architecture, benzene ring model, electron delocalization security concept, resonance stabilization concept, and QBRE framework were developed by the author.

## CONCLUSION

This paper presents BRSCCS, a threshold-based multi-cloud key-management architecture designed to distribute cryptographic trust across heterogeneous cloud-native KMS platforms. The research was conducted using a Design Science Research methodology in which the BRSCCS architecture was developed as an artifact, demonstrated through an early prototype, and evaluated through controlled functional scenarios. Design Science Research supports the creation and evaluation of innovative artifacts as a means of developing knowledge and understanding about an identified problem and its solution (Hevner et al., 2004). The Python prototype implemented the core threshold workflow using simulated AWS KMS, Azure Key Vault, and Google Cloud KMS trust domains. The prototype demonstrated reconstruction from each tested two-share combination, continued decryption during a simulated single-provider outage, refusal of reconstruction when fewer than two shares remained available, key-share refresh without changing the master key, and separation between the specific old and refreshed share sets evaluated. The custom secret-sharing implementation was also cross-validated using PyCryptodome and exercised with 20 randomly generated 256-bit keys.

These findings establish functional feasibility for the evaluated threshold workflow within the local prototype. They do not demonstrate production-scale multi-cloud effectiveness. The current implementation does not integrate live cloud-provider APIs, identity federation, provider-native policy controls, or a Trusted Execution Environment. It also does not establish operational performance, scalability, cost, formal security, or resistance to adversarial attacks. Future work will replace the simulated providers with live cloud integrations, deploy the orchestration component within an appropriate confidential-computing environment, and evaluate security, performance, lifecycle management, and operational resilience under realistic multi-cloud conditions.

## REFERENCES

Alonso, J., Orue-Echevarría, L., Casola, V., Torre, A. I., Huarte, M., Osabe, E., & Lobo, J. L. (2023). Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review. J Cloud Comp 12, 6, https://doi.org/10.1186/s13677-022-00367-6.

Barker, E. (2020). Recommendation for Key Management: Part 1— General. NIST Special Publication 800-57 Part 1 Revision 5. National Institute of Standards and Technology, https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final.

Cachin, C. (2023). Multi-Party Threshold Cryptography. In book: Trends in Data Protection and Encryption Technologies (pp. 65–69), DOI: 10.1007/978-3-031-33386-6_13. pp. 65–69.

Chandramouli, R., Iorga, M., & Chokhani, S. (2013). Cryptographic Key Management Issues & Challenges in Cloud Services. NIST, http://dx.doi.org/10.6028/NIST.IR.7956. pp. 1–36.

Desmedt, Y. (2025). Secret Sharing and Shamir Threshold Scheme. In: Jajodia, S., Samarati, P., Yung, M. (eds) Encyclopedia of Cryptography, Security and Privacy. Springer, Cham, pp. 2212–2216. https://doi.org/10.1007/978-3-030-71522-9_1822.

ESMA. (2021). Financial stability: Cloud Outsourcing and financial stability risks. ESMA Report on Trends, Risks and Vulnerabilities, Contact: alexander.harris@esma.europa.eu, pp. 1–9.

Hakatemia. (2023). Key management in cloud services: AWS, GCP and Azure. Retrieved from https://www.hakatemia.fi/en/courses/cryptography/key-management-in-cloud-services-aws-gcp-and-azure.

Herzberg, A., Jarecki, S., Krawczyk, H., & Yung, M. (1995). Proactive secret sharing or: How to cope with perpetual leakage. In Advances in Cryptology — CRYPTO '95, pp. 339–352.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. JSTOR. Management Information Systems Research Center, University of Minnesota, pp. 75–105. https://doi.org/10.2307/25148625.

Honzik, T. (2024, 02 13). Multisig< Shamir's secret sharing, & MPC compared. Retrieved from Unchained: https://www.unchained.com/blog/mpc-vs-multisig-vs-sss

Huang, X., & Chen, R. (2018). A Survey of Key Management Services in Cloud. Conference: 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), DOI: 10.1109/ICSESS.2018.8663805.

Karanjai, R., Xu, L., Chen, L., Diallo, N., & Shi, W. (2024). Decentralized FaaS over Multi-Clouds with Blockchain-based Management for Supporting Emerging Applications. The 39th ACM/SIGAPP Symposium On Applied Computing. Software Engineering (cs.SE), https://doi.org/10.48550/arXiv.2404.08151.

Krishnan, V. V. (2024). Key to the Cloud: A Review of Amazon Web Services, Azure, and Google Cloud Platform Key Management Systems. International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET), https://www.ijirset.com/upload/2024/august/138_Key.pdf.

Le Mouël, F., Godon, M., Brien, R., Beurier, E., Boulahia-Cuppens, N., & Cuppens, F. (2024). Trustless distributed symmetric-key encryption. arXiv., https://arxiv.org/abs/2408.16137.

National Institute of Standards and Technology. (2026, April 23). *Multi-Party Threshold Cryptography (MPTC) Project*. U.S. Department of Commerce. https://csrc.nist.gov/projects/threshold-cryptography

NSA/CISA. (2024). Use Secure Cloud Key Management Practices. National Security Agency (NSA) & Cybersecurity and Infrastructure Security Agency (CISA), https://media.defense.gov/2024/Mar/07/2003407858/-1/-1/0/CSI-CloudTop10-Key-Management.

Opara-Martins, J., Sahandi, R., & Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. J Cloud Comp 5, 4, https://doi.org/10.1186/s13677-016-0054-z.

Open Policy Agent. (n.d.). Open Policy Agent documentation. Retrieved from https://www.openpolicyagent.org/docs?utm_source=chatgpt.com

Peffers, K., Tuunanen, T., Tuunanen, M. A., & Chatterjee, s. (2007). A design science research methodology for information systems research. Journal of Management Information Systems 24(3), pp. 45–77. Online ISSN: 1557-928X.

Rai, A. K. (2026, Feb. 28). AWS KMS vs Azure Key Vault vs Google Cloud KMS: Complete Comparison. Retrieved from QCecuring: https://www.qcecuring.com/blog/aws-kms-vs-azure-key-vault-vs-gcp-kms

Scott, H., Guilliver, J., Nadler, H., & Ondrejko, J. (2023). Cloud Adoption in the Financial Sector and Concentration Risk. Amazon Web Services, Program on International Financial Systems, pp. 1–32.

Shamir, A. (1979). How to share a secret. Communications of the ACM, 22(11), pp. 612–613. https://dl.acm.org/doi/10.1145/359168.359176.

Soliman, W., & Järveläinen, J. (2024). Reconceptualizing the Human in the Loop: A Problematization of Taken-for-Granted Metaphors in Cybersecurity Research. ECIS 2024 Proceedings. 5. https://aisel.aisnet.org/ecis2024/track02_general/track02_general/5.

Zatlavi, L. (2024, Jun 25). Understanding Customer Managed Encryption Keys (CMKs) in AWS, Azure, and GCP: A Comparative Insight. Retrieved from tenable: https://www.tenable.com/blog/understanding-customer-managed-encryption-keys-cmks-in-aws-azure-and-gcp-a-comparative-insight?utm_source=chatgpt.com