A Threshold Cryptography Framework for Secure and Resilient Symmetric Key Management in Multi-Cloud Environments

Dewank Pant, Avijit Kumar, Shruti Lohani, Manan Wason
{dewankpant, avijitkumar2002, shrutilohani9, manan.wason}@gmail.com

***Abstract*—This paper addresses the critical security and availability risks inherent in centralized key management systems (KMS) for cloud data protection. The compromise or failure of a single KMS can lead to catastrophic data breaches or loss of access. We propose a decentralized framework to mitigate this single point of failure. Technology or Method: We introduce the Threshold Key Management System (TKMS), a novel framework that leverages threshold cryptography to manage symmetric encryption keys. The framework employs a (k, n)-threshold secret sharing scheme to shard a master symmetric key across n independent Key Management Nodes (KMNs), which are distributed across multiple cloud provider infrastructures. Key generation is performed collaboratively using a Distributed Key Generation (DKG) protocol, eliminating the need for a trusted dealer. Results: The security analysis demonstrates that TKMS guarantees key confidentiality as long as fewer than k KMNs are compromised. The system provides high availability, tolerating the failure of up to $n-k$ nodes. The projected performance evaluation indicates that the cryptographic overhead is manageable and offers a favorable trade-off for the significantly enhanced security and resilience. Conclusions: TKMS presents a robust and fault-tolerant alternative to traditional cloud KMS. By distributing trust across multiple administrative domains, it significantly raises the bar for attackers and protects against provider-level failures. Impact: This work provides a practical blueprint for building highly secure, resilient, and trust-minimized data protection services in the cloud, with direct applications in securing sensitive corporate data, personal information, and critical infrastructure backups.**

***Index Terms*—Cloud Security, Distributed Key Generation (DKG), Key Management, Secret Sharing, Symmetric Key Encryption, Threshold Cryptography.**

I. Introduction

THE paradigm of cloud computing has revolutionized how organizations and individuals store and process data, offering unprecedented scalability and accessibility.[1] This massive migration of data to public and private cloud infrastructures has, however, concentrated immense volumes of sensitive information, from corporate intellectual property to personal health records into the hands of a few Cloud Service Providers (CSPs). Consequently, the security of this data has become a paramount concern.[2] Encryption stands as the cornerstone of modern data protection strategies, rendering data unintelligible to unauthorized parties.[3] Yet, the efficacy of any encryption scheme is entirely contingent upon the security and management of its cryptographic keys. If a key is compromised, the confidentiality of the data it protects is nullified.[4]

This dependency places an enormous burden on the Key Management System (KMS), the entity responsible for the entire lifecycle of cryptographic keys, including their generation, storage, distribution, and revocation.[2] The dominant architecture for KMS in cloud environments is centralized, whether managed directly by the CSP or by the customer through Bring Your Own Key (BYOK) models.[5] This centralization, while convenient, represents a critical architectural vulnerability. A centralized KMS becomes a high-value target for adversaries and a single point of failure for the entire system. A catastrophic event, such as a sophisticated cyberattack that exploits complex vulnerabilities in the cloud stack [6], a malicious insider at the CSP, or a government-compelled disclosure, could compromise the KMS, leading to a widespread data breach or an irreversible loss of access to mission-critical data.[4]

This paper introduces a novel framework to dismantle this single point of failure by fundamentally re-architecting the trust model for cloud key management. Drawing inspiration from the concept of "key sharding" [7], we formalize this intuition using the established principles of threshold cryptography. The core idea is to move away from entrusting a single entity with a complete key and instead distribute this trust across multiple, independent parties. The security of the system then relies not on the impenetrability of a single fortress, but on the logistical difficulty an adversary faces in compromising a quorum of distributed, non-colluding entities. The fundamental problem being addressed is not merely technological but one of trust distribution. While conventional models necessitate absolute trust in a single CSP, our approach minimizes this trust by requiring attacker collusion across multiple, potentially competing, administrative domains, such as different CSPs. This architectural shift from a centralized to a distributed trust model is the principal innovation of our work.

Our contribution is the **Threshold Key Management System (TKMS)**, a comprehensive framework designed to securely manage symmetric encryption keys in a decentralized, fault-tolerant, and trust-minimized manner. The key features of TKMS are:

1) A (k, n)-threshold architecture, where a master symmetric key is split into n shares, and any k of these shares are sufficient to reconstruct the key.
2) The use of a **Distributed Key Generation (DKG)** protocol, which allows the key shares to be created

collaboratively by the nodes without a trusted dealer, meaning the master key is never held in its entirety by any single entity, not even at the moment of its creation.
3) The strategic deployment of key-share-holding nodes across heterogeneous environments (e.g., multiple CSPs and on-premise data centers) to provide resilience against provider-level failures and raise the bar for attackers.
4) The integration of advanced cryptographic mechanisms, including **Verifiable Secret Sharing (VSS)** and **Proactive Secret Sharing (PSS)**, to provide robustness against malicious participants and long-term security against persistent, mobile adversaries.

This paper is structured as follows. Section II provides a review of the necessary background concepts, including existing cloud key management paradigms and the cryptographic foundations of secret sharing and DKG. Section III presents the detailed architecture and protocols of the proposed TKMS framework. Section IV provides a rigorous security analysis of the framework and a comparative evaluation against traditional models. Section V outlines a plan for empirical performance evaluation. Finally, Section VI concludes the paper and discusses promising avenues for future research.

II. Background and Related Work

This section lays the groundwork for our proposed framework by reviewing the state of the art in cloud key management and introducing the foundational cryptographic primitives upon which our system is built.

A. Paradigms in Cloud Key Management

The management of cryptographic keys is a critical service in the cloud, and CSPs offer a spectrum of models to meet varying customer needs for security and control [4],.[5] These models can be broadly categorized as follows:

- **CSP-Managed Keys:** This is the most straightforward model, where the CSP assumes full responsibility for the entire key lifecycle. The customer simply enables encryption for a given service (e.g., cloud storage), and the CSP handles key generation, rotation, and storage transparently. While this model offers maximum convenience, it provides the customer with minimal control and visibility, requiring absolute trust in the CSP's security practices and personnel.[5]
- **Customer-Managed Keys (in Cloud KMS):** To afford customers greater control, CSPs offer more advanced KMS options. A popular model is Bring Your Own Key (BYOK), where the customer generates a key on-premise and securely imports it into the CSP's KMS. The customer retains control over the key's lifecycle policies, such as rotation schedules and access controls. However, while the customer manages the key's policies, the key material itself is still stored and used within the CSP's infrastructure, meaning trust is still placed in the CSP's implementation and operational security [4],.[5]
- **Hardware Security Modules (HSMs):** For the highest level of assurance, keys can be managed within FIPS 140-2 validated HSMs. These are specialized hardware devices designed to securely generate, store, and process cryptographic keys, with physical and logical protections against tampering.[5] CSPs offer HSM-backed KMS services, where customer keys are protected by an HSM. In some cases, customers can even provision dedicated HSMs in the cloud. While HSMs provide a strong root of trust, they still often exist within a single provider's ecosystem, representing a centralized point of failure, albeit a highly secured one.

A common thread across all these dominant paradigms is the reliance on a centralized point of trust or failure. Whether it is the CSP's software-based KMS or a dedicated HSM, the compromise or failure of that single logical entity can have catastrophic consequences. The TKMS framework is explicitly designed to overcome this fundamental limitation by distributing trust and eliminating any single point of failure.

B. Foundations of Secret Sharing

Secret sharing schemes are cryptographic protocols that allow a secret to be divided into multiple parts, called shares, which are distributed among a group of participants. The secret can only be reconstructed when a sufficient number of shares are combined [8],.[9] These schemes form the cryptographic bedrock of our proposed system.

1) Shamir's Secret Sharing (SSS): Proposed by Adi Shamir in 1979, the (k, n)-threshold secret sharing scheme is a foundational and elegant method for distributing a secret [10],.[11] **The intuition behind it is simple: just as two points are needed to define a unique line, and three points to define a unique parabola, any k points are needed to define a unique polynomial of degree $k-1$.** It is based on this mathematical principle of polynomial interpolation: a unique polynomial of degree $k-1$ is defined by any k distinct points [10],.[12] To share a secret S, a random polynomial $q(x)$ of degree $k-1$ is constructed with the secret as the constant term:

$$q(x) = a_{k-1}x^{k-1} + \cdots + a_1 x + S \pmod{p} \qquad (1)$$

The coefficients $a_1, \ldots, a_{k-1}$ are chosen uniformly at random from Z_p. Shares are generated by evaluating $q(x)$ at n distinct points. To reconstruct the secret, any k shares are used to interpolate the polynomial $q(x)$, and the secret is recovered as $S = q(0)$ [12],.[13] SSS possesses *perfect secrecy*: knowledge of any $k-1$ or fewer shares reveals no information about S.[10]

2) Verifiable Secret Sharing (VSS): Shamir's scheme assumes an honest dealer. **However, a malicious dealer could intentionally distribute inconsistent shares to different participants, making it impossible to reconstruct the correct secret.** To counter this threat, Verifiable Secret Sharing (VSS) schemes, like Feldman's, address this by adding a layer of public verifiability.[15] The dealer broadcasts commitments to the polynomial coefficients, $C_j = g^{a_j}$. Each participant

P_i receiving a share (x_i, y_i) can verify its consistency by checking if the following equation holds:

$$g^{y_i} \equiv \prod_{j=0}^{k-1} (C_j)^{x_i^j} \pmod{p} \quad (2)$$

This ensures that a unique, well-defined secret can be reconstructed [15],.[16]

3) Proactive Secret Sharing (PSS): Static schemes like SSS and VSS are vulnerable to a *mobile adversary* who compromises participants over time. Proactive Secret Sharing (PSS) defends against this by introducing a periodic "share renewal" protocol.[17] During renewal, participants collaboratively update their shares without reconstructing the secret. This is done by adding shares of a secret '0' to their existing shares. This process re-randomizes the underlying polynomial while keeping the secret the same, rendering old, compromised shares useless for reconstruction with new shares [17],.[18]

C. Distributed Key Generation (DKG)

To create a fully decentralized system, the need for a trusted dealer must be eliminated. Distributed Key Generation (DKG) is a multi-party protocol that allows n participants to jointly generate a shared secret and its shares without any single party ever knowing the secret [19],.[20] Each participant acts as a dealer in their own VSS scheme, and the final secret is the sum of all individual secrets. The final share for each participant is the sum of the shares they received from all others. This ensures the key is "born secret" [21],.[22]

III. Proposed Framework: Threshold Key Management System (TKMS)

This section details the architecture and protocols of the Threshold Key Management System (TKMS).

TABLE I
Notation Summary

Symbol	Description
n	Total number of Key Management Nodes (KMNs).
k	Threshold number of KMNs required for an operation.
P_i	The i-th Key Management Node, for $i \in \{1, \ldots, n\}$.
K	The master symmetric key, shared among the KMNs.
s_i	The share of the key K held by node P_i.
p, q, g	Public parameters for the cryptographic scheme.
$f(x)$	The master secret polynomial of degree $k-1$.
C_j	The public commitment to the j-th coefficient of $f(x)$.
VK	The public verification key for the master secret, $VK = g^K$.

A. System Architecture and Threat Model

The TKMS architecture consists of three primary components: a Client, a set of n distributed Key Management Nodes (KMNs), and one or more untrusted Storage Providers. The KMNs are strategically deployed across heterogeneous environments (e.g., multiple CSPs) to maximize resilience against provider-level failures and collusion attacks.[23] The conceptual architecture is shown in Fig. 1.

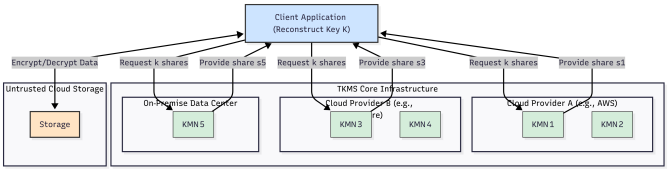

Fig. 1. Conceptual architecture of the TKMS, showing the interaction between the Client, distributed Key Management Nodes (KMNs) across multiple cloud providers, and the untrusted Storage Provider.

The **Threat Model** assumes a powerful adversary who can compromise up to $k-1$ KMNs, eavesdrop on all network communications, and fully compromise the storage provider. The adversary's goal is to recover the master key K or decrypt ciphertext without it.

B. Protocol 1: Distributed Symmetric Key Generation and Sharing

This one-time setup protocol uses DKG principles for the KMNs to collaboratively generate the master key K and their shares.

1) **Initialization:** All n KMNs agree on public parameters (p, q, g).
2) **Individual VSS Execution:** Each KMN P_i acts as a dealer in Feldman's VSS [15], generating a random polynomial $f_i(x)$, broadcasting commitments $C_{i,j}$, and secretly sending shares s_{ij} to other nodes P_j.
3) **Share Verification:** Each KMN P_j verifies its received shares s_{ij} using the broadcasted commitments. If verification fails, a complaint is broadcast.
4) **Final Share and Public Key Computation:** Each qualified KMN P_i computes its final private share $s_i = \sum s_{ji}$. The master key K is the sum of the constant terms of all polynomials, but is never explicitly computed. The master public verification key $VK = g^K$ is computed from the public commitments.

C. Protocol 2: Secure Key Reconstruction and Usage

This protocol outlines how a client uses TKMS for an encryption or decryption operation.

1) **Client Request:** The client sends a signed, timestamped request for key reconstruction to at least k KMNs.
2) **Share Provision:** The k contacted KMNs respond with their private shares s_i.
3) **Client-Side Verification and Reconstruction:** The client verifies each share using the master public commitments. With k valid shares, it uses Lagrange interpolation to reconstruct the key K in local memory.
4) **Cryptographic Operation:** The client uses K with a standard cipher like AES-256 GCM [24], [25] to encrypt or decrypt the file.
5) **Secure Erasure:** Immediately after the operation, the client securely wipes the key K and all shares s_i from its memory. The security of this transient phase is also contingent on the client application itself being hardened

against endpoint vulnerabilities, such as injection attacks, which could otherwise compromise the key during its brief existence in memory [26],.[27]

D. Security and Robustness Mechanisms: Integrating PSS

To ensure long-term security against a mobile adversary, TKMS incorporates a PSS mechanism. At regular intervals, KMNs execute a Share Renewal Protocol. Each KMN generates shares of a "zero-polynomial" (a polynomial with a constant term of zero) and distributes them. Each node adds the received "zero-shares" to its existing master key share. This re-randomizes the shares and the underlying polynomial without changing the master key K, thus invalidating any previously compromised shares.[17]

IV. Security Analysis

A. Confidentiality and Integrity

- **Key Confidentiality:** Protected by the (k, n)-threshold scheme. An adversary with fewer than k shares gains no information about the key K.[10] DKG ensures the key is never concentrated in one place [19], and PSS strengthens long-term confidentiality.[17]
- **Data and Share Integrity:** Data integrity is provided by the chosen cipher (e.g., AES-GCM). The integrity of the key management process is guaranteed by VSS, as clients can verify each share before reconstruction, preventing the use of a corrupted or incorrect key [14],.[15]

B. Availability and Fault Tolerance

The TKMS framework is inherently fault-tolerant. The system can withstand the failure of up to $n-k$ KMNs and remain operational.[28] In a multi-cloud deployment, this provides resilience against large-scale outages affecting an entire cloud provider, a level of business continuity that is difficult to achieve with centralized architectures.[23]

C. Comparative Analysis

The TKMS framework is inherently fault-tolerant. In a multi-cloud deployment, this provides resilience against large-scale outages affecting an entire cloud provider, a level of business continuity that is difficult to achieve with centralized architectures.[23]

Furthermore, when compared to other systems that use secret sharing, such as HashiCorp Vault's Shamir seal, TKMS offers distinct security advantages. While such systems effectively shard a root key, they typically rely on a trusted dealer model where the key is generated in one place and then split. This still presents a single point of compromise at the moment of creation. TKMS fundamentally eliminates this risk by using a Distributed Key Generation (DKG) protocol, where the key is "born secret" and never exists in its entirety on any single machine. Additionally, the integration of Proactive Secret Sharing (PSS) provides long-term security against persistent, mobile adversaries, a feature not commonly found in standard enterprise secret management tools.

Table II compares TKMS with traditional key management architectures, highlighting its superior trust model and fault tolerance at the cost of increased implementation complexity.

V. Projected Performance Evaluation

This section outlines a plan for assessing the performance of the TKMS framework.

A. Experimental Setup

KMNs would be deployed as virtual machines (e.g., **'t2.micro' instances on AWS, 'B1s' on Azure, and 'e2-micro' on GCP**) across multiple distinct CSPs to simulate a realistic geo-distributed environment. The protocols would be implemented in **Go, utilizing its native concurrency features and cryptographic libraries such as 'crypto/elliptic' for the underlying group operations. The cryptographic scheme would be instantiated over a standard elliptic curve group like 'P-256' for the public parameters (p, q, g) to ensure both security and performance.** Experiments would vary system parameters n (total nodes) and k (threshold), as well as the size of the data being processed.

B. Performance Metrics

1) **DKG Latency:** Wall-clock time to complete the initial key generation protocol.
2) **Reconstruction Latency:** Wall-clock time from client request to successful key reconstruction in memory. This is the most critical user-facing metric.
3) **Encryption/Decryption Throughput:** Rate (in MB/s) at which a file is processed, including reconstruction latency.

C. Expected Outcomes and Discussion

We project that DKG latency will scale polynomially with n, while reconstruction latency will be dominated by network RTT and scale linearly with k. Throughput for small files will be impacted by the reconstruction overhead, but for large files, this fixed cost will be amortized, and throughput will approach the native speed of the underlying symmetric cipher. These projected outcomes are illustrated in Fig. 2, Fig. 3, and Fig. 4.

VI. Conclusion and Future Work

This paper introduced the Threshold Key Management System (TKMS), a novel framework that leverages threshold cryptography to create a decentralized, fault-tolerant, and trust-minimized environment for managing symmetric encryption keys. By integrating DKG, VSS, and PSS, TKMS provides a complete, end-to-end secure lifecycle for a shared key, offering robust protection against insider threats and provider-level failures.

The primary limitation of TKMS is its increased operational complexity compared to turnkey KMS solutions. **This complexity manifests in several areas. First, the framework's performance, particularly during the interactive DKG and reconstruction phases, is sensitive to network latency and**

TABLE II
COMPARATIVE ANALYSIS OF KEY MANAGEMENT ARCHITECTURES

Feature	CSP-Managed KMS	BYOK with Cloud HSM	Proposed TKMS ((k, n)-threshold)
Trust Model	Single CSP is fully trusted.	Trust in CSP for HSM infra, customer for key.	Trust distributed across k of n parties.
Fault Tolerance	Reliant on single CSP's SLA.	Reliant on single CSP's SLA for HSM service.	Tolerates up to $n-k$ node/provider failures.
Insider Threat (CSP)	High risk. A malicious/compelled CSP has access.	Lower risk, but CSP controls HSM environment.	Very high resistance. Requires collusion across providers.
Key Compromise Impact	Total loss of all keys managed by the service.	Total loss of imported keys if HSM is broken.	No key loss if $< k$ shares are compromised.
Implementation Complexity	Low (fully managed service).	Medium (requires key generation and import).	High (requires deploying and managing KMNs).
Operational Flexibility	Limited to CSP's offerings.	Moderate, customer controls key lifecycle.	High, fully customizable policies.

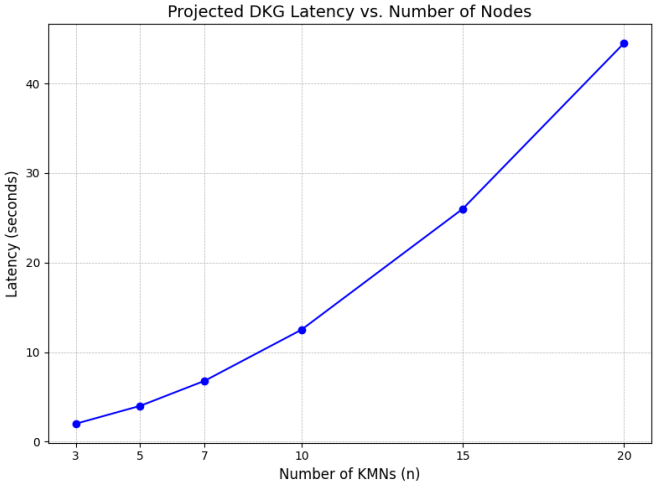

Fig. 2. Projected DKG Latency as a function of the number of KMNs (n).

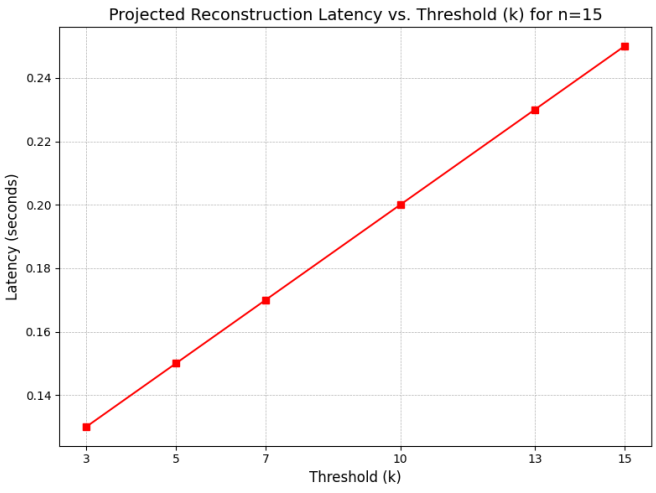

Fig. 3. Projected Reconstruction Latency as a function of the threshold (k) for a fixed n.

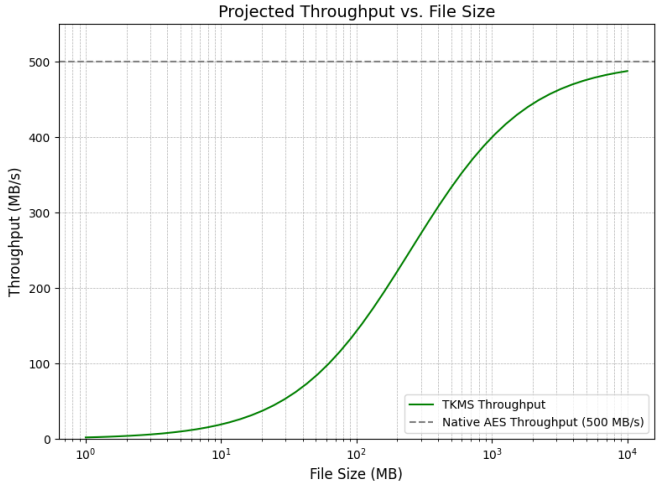

Fig. 4. Projected Encryption/Decryption Throughput as a function of file size, showing the amortization of reconstruction overhead.

stability; geo-distributed KMNs may experience delays that impact user-facing operations. Second, while resilient to node failure, the framework currently lacks a defined protocol for catastrophic key recovery (e.g., if a client permanently loses credentials or more than $n-k$ nodes are destroyed), which is a critical operational consideration. Finally, integrating TKMS into existing applications requires a client-side SDK or proxy layer to translate calls from standard cloud KMS APIs, posing an adoption hurdle.

Future work should focus on addressing these practical challenges. Automating the deployment, monitoring, and lifecycle management of KMNs using modern infrastructure-as-code tools would significantly lower the operational barrier. For future research, a compelling direction is to design and evaluate adaptive share renewal policies for PSS, where the frequency of updates is based on threat intelligence or node behavior. Another critical avenue is to extend the framework to support **threshold asymmetric cryptography**.[29] Protocols for threshold digital signatures, such as Threshold ECDSA, allow the signing operation itself to be a distributed protocol without ever reconstructing the private key [30],.[31] This would eliminate the transient reconstruction of the key on the client, representing the gold standard for distributed trust systems.

Such advancements are critical for enabling a wider range of high-security applications, especially as modern systems increasingly integrate complex components like Large Language Model (LLM) agents, which introduce novel attack surfaces and security challenges.[32]

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments (GCE) Workshop*, 2008, pp. 1-10.

[2] M. Armbrust et al., "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. Pearson, 2017.

[4] A. Buchade and R. Ingle, "Key Management for Cloud Data Storage: Methods and Comparisons," in *International Conference on Electronic Systems, Signal Processing and Computing Technologies*, 2014.

[5] National Security Agency & Cybersecurity and Infrastructure Security Agency, "Use Secure Cloud Key Management Practices," CSI-CloudTop10-Key-Management, Mar. 2024.

[6] A. Boldyreva, V. Goyal, and V. Kumar, "Identity-based encryption with outsourced revocation in cloud computing," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2011, pp. 361-370.

[7] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. AFIPS National Computer Conference*, vol. 48, 1979, pp. 313-317.

[8] C. A. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 208-210, Mar. 1983.

[9] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, Nov. 1979.

[10] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1997.

[11] C. L. Liu, *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[13] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proc. 26th IEEE Symp. Found. Comput. Sci.*, 1985, pp. 383-395.

[14] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. 28th IEEE Symp. Found. Comput. Sci.*, 1987, pp. 427-437.

[15] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proc. 21st ACM Symp. Theory Comput.*, 1989, pp. 73-85.

[16] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proc. 10th ACM Symp. Princ. Distrib. Comput.*, 1991, pp. 51-59.

[17] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Advances in Cryptology—CRYPTO '95*, 1995, pp. 339-352.

[18] T. Pedersen, "A threshold cryptosystem without a trusted party," in *Advances in Cryptology—EUROCRYPT '91*, 1991, pp. 522-526.

[19] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Advances in Cryptology—EUROCRYPT '99*, 1999, pp. 295-310.

[20] I. Goldberg, D. Wagner, and E. Brewer, "A secure environment for untrusted helper applications," in *Proc. 6th USENIX Security Symp.*, 1996.

[21] A. Kate and I. Goldberg, "Distributed key generation for the internet," in *Proc. 14th Int. Conf. Financial Cryptography and Data Security*, 2010, pp. 331-346.

[22] K. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2009, pp. 187-198.

[23] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, Nov. 2007.

[24] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS PUB 197, Nov. 2001.

[25] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Advances in Cryptology—CRYPTO '89*, 1990, pp. 307-315.

[26] Y. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449-457, 1994.

[27] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 1179-1194.

[28] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology—EUROCRYPT 2000*, 2000, pp. 207-220.

[29] D. Pant, "Server-Side Request Forgery on LaTeX Editor Leading to Docker Bypass and Total Server Compromise," Zenodo, 2022. [Online]. Available: https://doi.org/10.5281/zenodo.16301382

[30] D. Pant, "Independent Security Research Vulnerability Disclosure Report: CVE-2017-16568," Zenodo, 2017. [Online]. Available: https://doi.org/10.5281/zenodo.15111221

[31] D. Pant, "Independent Security Research Vulnerability Disclosure Report: CVE-2017-16567," Zenodo, 2017. [Online]. Available: https://doi.org/10.5281/zenodo.15111380

[32] A. Joshi, D. Pant, and I. Kumar, "DILLMA: Damn Insecure LLM Agent (1.0.0)," Zenodo, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15232655