CipherGuard : A Threshold Cryptography Framework for Secure Blockchain Key Management

S.C.J. Bandara[1], D.M.G.S. Dissanayake[1], I.G.S.S.K. Karunadasa[2]*, L.R. Samarasinghe[1]

[1]Faculty of Engineering, Computing and the Environment, Kingston University, London, United Kingdom

[2]Postgraduate Institute of Science, University of Peradeniya, Sri Lanka

***Corresponding Author**

DOI: https://dx.doi.org/10.51244/IJRSI.2026.1313CS009

Received: 11 May 2026; Accepted: 16 May 2026; Published: 02 June 2026

ABSTRACT

The proliferation of blockchain technology has necessitated robust mechanisms for cryptographic key management. Traditional storage methods, often relying on single private keys or mnemonic phrases, present a single point of failure (SPOF) wherein loss or theft results in the irreversible loss of assets. This paper presents "CipherGuard," a web-based demonstration platform that implements Shamir's Secret Sharing (SSS) to mitigate these risks. By decomposing a secret into n shares such that any k shares can reconstruct it, CipherGuard demonstrates the information-theoretic security of threshold cryptography. This report critically analyzes the mathematical foundations of Lagrange Interpolation, details the TypeScript implementation of the algorithm, and evaluates its suitability against alternative control mechanisms like Multi-Signature (MultiSig) wallets. The study concludes that while SSS introduces implementation complexity, it offers superior privacy and off-chain recovery capabilities essential for institutional-grade blockchain custody.

Keywords - Shamir Secret Sharing, Threshold Cryptography, Blockchain Security, Key Management, Lagrange Interpolation, React.js

INTRODUCTION

The immutable nature of blockchain ledgers essentially shifts the security paradigm from centralized authority to individual responsibility. In this ecosystem, the private key is the sole arbiter of ownership [1]. Consequently, the management of these keys has become the most critical security challenge in the digital asset space. NIST Report IR 8202 highlights that while blockchain protocols themselves are tamper-resistant, the endpoints specifically key storage remain vulnerable [2].Historical precedence, such as the Mt. Gox exchange hack and countless individual wallet losses, demonstrates the fragility of storing private keys in a single location [3]. If a key is stored on a single device, that device becomes a Single Point of Failure (SPOF). Conversely, creating full backups of the key increases the attack surface; if any one backup is compromised, the entire asset is lost. To address this dichotomy between availability and confidentiality, threshold cryptography offers a mathematically proven solution. Specifically, Shamir's Secret Sharing (SSS), introduced by Adi Shamir in 1979, allows a secret to be divided into parts, or "shares," distributed among different participants or devices [4].This project, CipherGuard, aims to demystify this cryptographic primitive. By developing an interactive visualization tool using React and TypeScript, this project explores the practical implementation of (k,n)-threshold schemes. The report is structured as follows: Section II reviews the mathematical foundations of polynomial interpolation and finite fields. Section III details the CipherGuard system architecture and implementation. Section IV provides a critical analysis of the solution compared to industry alternatives, and Section V discusses configuration and management in a real-world context.

MATHEMATICAL FOUNDATIONS

The core security of the CipherGuard system relies on the information-theoretic security properties of polynomial interpolation. Unlike computational security (Ex: RSA), which relies on the difficulty of factoring large numbers, SSS is secure because an attacker with fewer than *k* shares lacks sufficient information to narrow down the search space for the secret [5].

A. Polynomial Interpolation

The fundamental theorem of algebra states that a unique polynomial of degree *k*-1 is determined by *k* points. Geometrically, two points define a line (degree 1), three points define a parabola (degree 2), and so on. Shamir's scheme constructs a polynomial P(*x*) of degree *k*-1:

$$P_{(x)} = a_{k_{-1}}x^{k-1} + \cdots + a_1 x + a_0 \qquad (1)$$

Where:

- $a_0 = S$ (The secret to be shared).
- $a_1,\ldots,a_{k-1}$ are random coefficients chosen from the field.
- *k* is the threshold number of shares required to reconstruct *S.*

The "shares" distributed to users are coordinate pairs(*x, y*), where *x* is the distinct share index (Ex: 1, 2, 3...) and $y = P(x)$ [6].

B. Lagrange Interpolation

To reconstruct the secret *S* (which is *P(0)*), we do not need to solve the system of linear equations for all coefficients. Instead, we use the Lagrange Interpolation formula. Given a subset of *k* shares, denoted as pairs (x_0, y_0), (x_{k-1}, y_{k-1}), the polynomial *P(x)* can be expressed as the linear combination:

$$\mathrm{p}_{(\mathrm{x})} = \sum_{\mathrm{j}=0}^{\mathrm{k}-1} \mathrm{y_j L_j(x)} \qquad (2)$$

Where $L_j(x)$ are the Lagrange basis polynomials defined as:

$$L_j(x) = \prod_{\substack{i=0 \\ i\neq j}}^{k-1} \frac{x - x_i}{x_j - x_i} \qquad (3)$$

Since we are only interested in the secret $S = P(0)$, we substitute $x=0$ into the equation. This simplifies the basis calculation to:

$$S = \sum_{j=0}^{k-1} y_j \left(\prod_{\substack{i=0 \\ i\neq j}}^{k-1} \frac{-x_i}{x_j - x_i} \right) \qquad (4)$$

This formula is implemented directly in the CipherGuard shamirLogic.ts module to demonstrate the reconstruction process [7].

C. Finite Fields vs. Real Numbers

While CipherGuard utilizes real numbers for the purpose of visual demonstration (allowing the user to see the curve on a chart), a secure production implementation must operate over a Finite Field, typically Galois Field *GF(p)* where *p* is a large prime [8].

Using real numbers introduces two critical vulnerabilities:

1. Precision Errors: Floating-point arithmetic is inexact, leading to reconstruction failures for large secrets.
2. Side-Channel Information: In the real number system, share values might reveal the magnitude of the coefficients, leaking information about the secret (Ex: graphical proximity).

In *GF(p),* all arithmetic operations (addition, multiplication) are performed modulo p. This ensures that the result is always an integer within the range *[0, p-1]* and that the distribution of points is perfectly uniform, providing zero knowledge about the secret to any coalition of size $< k$ [9].

SYSTEM ARCHITECTURE:CIPHERGUARD

CipherGuard was designed as a client-side Single Page Application (SPA) to ensure that cryptographic operations occur locally on the user's device, adhering to the "trustless" principle of blockchain applications.

D. Technology Stack

The application is built using the following modern web technologies:

- React 19: For building a reactive component-based user interface.
- TypeScript: To ensure type safety, particularly for the data structures representing Points and Shares (defined in types.ts).
- Vite: As a high-performance build tool and development server.
- Recharts: For the D3-based visualization of the polynomial curve.
- Tailwind CSS: For rapid, responsive styling adhering to modern UI/UX principles.

The project structure separates concerns into three distinct layers:

1. Presentation Layer: App.tsx and components like ConfigurationPanel and Visualizer.
2. Logic Layer: services/shamirLogic.ts containing the pure mathematical functions.
3. Service Layer: services/geminiService.ts for generating AI-assisted technical reports.

E. Core Logic implementation

The mathematical core is encapsulated in shamirLogic.ts. The implementation avoids external cryptographic libraries to demonstrate the raw algorithmic logic required for the project. The generateShares function accepts the secret S, total shares n, and threshold k. It initializes the coefficient array with the secret as the first element: This direct implementation of the Horner's method or direct power summation highlights the computational overhead increases linearly with the threshold k.
The reconstruction phase, which is critical for recovering the private key, is handled by the reconstructSecret function. It implements the Lagrange basis calculation described in Section

The code iterates through the selected subset of shares. For each share (x_j, y_j), it computes the Lagrange term by multiplying the basis fractions. In a standard cryptographic library, this division would be a modular inverse operation over a finite field. However, CipherGuard simplifies this to floating-point division for visual demonstration, rounding the final result to correct for IEEE 754 precision errors. This implementation choice serves the educational objective of the tool but highlights a constraint: production systems must strictly avoid floating-point arithmetic to prevent non-deterministic behavior across different hardware architectures [10].

F. Visualization Engine

To bridge the gap between abstract algebra and user understanding, the application features a real-time Visualizer component. This component, located in components/Visualizer.tsx, utilizes the Recharts library to render the polynomial curve dynamically.

The function generateCurvePoints in shamirLogic.ts creates a dense set of coordinate pairs (interval 0.2) to approximate a smooth curve.

$$y = \sum_{i=0}^{k-1} c_i x^i \qquad (5)$$

This continuous rendering allows users to visually verify that:The curve passes exactly through the specific "share" points.The curve intersects the Y-axis ($x=0$) exactly at the secret value.Changing the secret shifts the entire curve vertically, while changing other coefficients alters the curvature.

This continuous rendering allows users to visually verify that:

1. The curve passes exactly through the specific "share" points.
2. The curve intersects the Y-axis ($x=0$) exactly at the secret value.
3. Changing the secret shifts the entire curve vertically, while changing other coefficients alters the curvature.

Fig. 4. CipherGuard Visualization Dashboard. The red node represents the secret (S), while blue nodes represent distributed shares *(P(x))*.

G. AI –Assisted Reporting

A novel feature of the CipherGuard architecture is the integration of Generative AI for technical documentation. The geminiService.ts module interfaces with the Google Gemini API. It constructs a prompt containing the current cryptographic configuration (Secret, N, K) and requests a "Critical Analysis" section. This demonstrates the potential for "Self-Documenting Security Architectures," where the system not only performs cryptographic operations but also explains its own security posture to the operator in real-time [11].

APPLICATION SCENARIO : DECENTRALIZED KEY CUSTODY

The primary application domain selected for this project is Blockchain Key Management, specifically for institutional custody of digital assets.

H. The Challenge : private Key Fragility

In the context of cryptocurrencies like Bitcoin or Ethereum, possession of the private key equates to ownership of the asset. Institutional investors face a unique dilemma:

- Hot Wallets: Keys connected to the internet are vulnerable to remote exfiltration.
- Cold Wallets: Keys stored offline in a physical safe are vulnerable to physical theft, fire, or loss.
- Bus Factor: If a single authorized signatory holds the key, their incapacitation results in total asset loss.

Traditional solutions involve "Multi-Signature" (MultiSig) wallets, which require transactions to be signed by *M* of *N* keys on-chain. However, MultiSig has distinct disadvantages: it is protocol-specific (not all blockchains support it), it reveals the access structure on the public ledger (privacy leak), and it incurs higher transaction fees [12].

I. The Solution: Threshold Key Management

CipherGuard proposes an off-chain solution using Shamir's Secret Sharing. Instead of a single private key stored in a hardware security module (HSM), the key is generated and immediately split into *n* shares.

Operational Workflow:

1. Initialization: The "Dealer" (the entity setting up the wallet) generates a private key (Secret *S*).

2. Distribution: The Dealer configures the scheme with *n=5* and *k=3*.

- Share 1 is stored in a bank vault.
- Share 2 is held by the CEO.
- Share 3 is held by the CFO.
- Share 4 is stored in a secure cloud enclave.
- Share 5 is held by legal counsel.

3. Destruction: The original Secret *S* is securely deleted from memory.

4. Reconstruction: To sign a transaction, any 3 of the 5 entities must bring their shares to an air-gapped machine. The secret is temporarily reconstructed in RAM, the transaction is signed, and the secret is immediately wiped.

J. Configuration and Management

The CipherGuard ConfigurationPanel allows the user to simulate this setup. The user inputs a "Secret Value" (simulating the private key integer). Sliders allow the adjustment of the Threshold (*K*) and Total Shares (*N*).

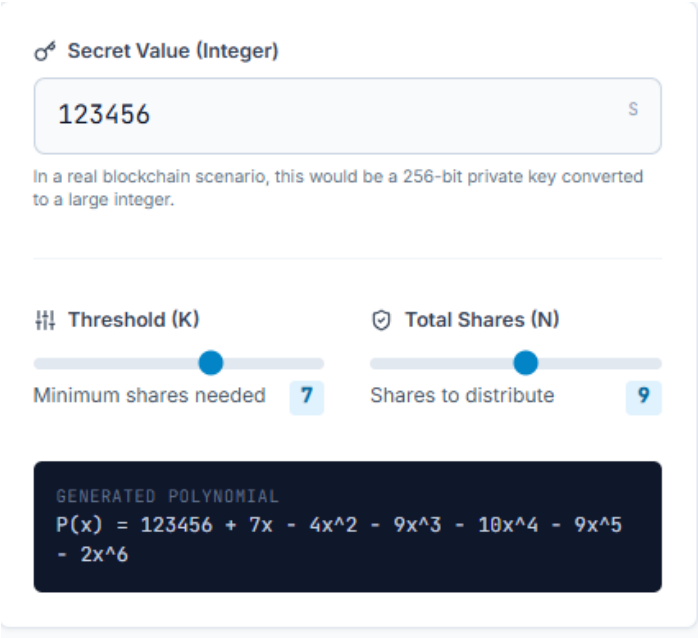

Fig. 5. Configuration interface allowing dynamic adjustment of redundancy parameters.

A critical management aspect implemented in the code is the validation logic:

This ensures the mathematical constraint $k \leqslant n$ is always respected, preventing invalid configurations that would lead to irrecoverable secrets.

CRITICAL ANALYSIS

This section evaluates the proposed solution against alternative controls and justifies the selection of Shamir's Secret Sharing.

K. Suitability Justification

The selection of SSS over AES encryption or simple backups is justified by the requirement for fault tolerance combined with secrecy.

- Redundancy without Replication: If we simply backed up the key to 5 locations, we would increase the risk of theft by a factor of 5 (any one location compromising the key). With SSS *(3, 5),* an attacker must compromise 3 distinct locations to recover the key.

- Resilience: The organization can lose up to *n-k* (Ex: 5-3=2) shares and still recover the funds. This addresses the "Bus Factor" and physical disaster recovery scenarios [13].

L. Comparison with Multi-Signature (MultiSig)

While MultiSig is the industry standard for on-chain coordination, SSS offers distinct advantages for Key Management Systems (KMS):

Table 1: Comparative analysis of MultiSig vs. SSS for Blockchain Custody.

Feature	Multi-Signature (MultiSig)	Shamir's Secret Sharing (SSS)
Visibility	Access structure is visible on the blockchain (Ex: "2 of 3" is public).	Totally opaque; the blockchain only sees one standard signature.
Cost	Higher fees (multiple signatures increase transaction size).	Standard fee (only one reconstructed signature is submitted).
Compatibility	Limited to chains that support specific MultiSig scripts (Ex: Bitcoin Script, EVM).	Universal; works with any private key (ECDSA, Ed25519, etc.).
Flexibility	Hard to change access structure once deployed.	Easy to "refresh" shares without moving funds (Proactive Secret Sharing).

As demonstrated in TABLE I, SSS provides a "protocol-agnostic" layer of security. This makes CipherGuard highly suitable for an exchange or custodian holding varying assets (Bitcoin, Monero, Solana) that need a unified security governance policy.

SECURITY CONSIDERATIONS & LIMITATIONS

While CipherGuard successfully demonstrates the mechanics of threshold cryptography, a critical academic evaluation must acknowledge the gap between this educational demonstration and a production-grade secure system.

M. The Floating-Point Vulnerability

As noted in the Code Analysis (Section III), CipherGuard utilizes standard JavaScript number types (IEEE 754 floating-point) for polynomial generation. In a rigorous cryptographic context, this is a severe vulnerability.

- Precision Loss: For large secrets (Ex: a 256-bit private key), floating-point arithmetic lacks the precision to reconstruct the secret perfectly.
- Side-Channel leakage: The time taken to multiply floating-point numbers can vary based on the operands, potentially exposing the system to timing attacks.
- Mitigation: A production iteration of CipherGuard would require a BigInt library implementing arithmetic over a finite field *GF(p),* ensuring constant-time operations and exact precision [14].

N. The Trusted Dealer Problem

The standard SSS scheme implemented here assumes a "Trusted Dealer" a single entity that generates the secret and distributes the shares. In a decentralized ethos, this Dealer is a central point of failure. If the Dealer is malicious or compromised during the setup phase, they retain a copy of the secret.

- Future Work: This limitation can be addressed by implementing Verifiable Secret Sharing (VSS) or Distributed Key Generation (DKG) protocols (Ex: Feldman's Scheme), where the secret is generated cooperatively by the participants without ever existing on a single machine. The application has also extended into a real world non IT or technical user friendly application called Cipherguard DAO: which is primarily based on Shamir's Secret Share and has been implemented and has continued into an extensive research based application as we speak. [15].

CONCLUSION

This project set out to address the critical vulnerability of single-point storage in blockchain architectures. By implementing Shamir's Secret Sharing in the CipherGuard platform, we have demonstrated a viable technical control that decouples availability from centralization.

The development process highlighted the elegance of Lagrange Interpolation as a tool for ensuring data resilience. The operational comparison revealed that while Multi-Signature wallets are currently more prevalent due to native protocol support, Threshold Cryptography offers a superior, protocol-agnostic layer of privacy and security.

For institutional custody, the ability to distribute trust across multiple legal entities (CEO, Bank, Cloud) without exposing the access structure on the public ledger is invaluable. While the current TypeScript implementation serves as a robust educational visualization, transitioning to a production environment would necessitate the adoption of finite field arithmetic and potentially a move to Distributed Key Generation to remove the trusted dealer.

Ultimately, CipherGuard serves as a proof-of-concept for the future of digital asset management: a future where security is derived not from the strength of a single vault, but from the mathematical impossibility of breaking a distributed consensus.

REFERENCES

[1] S. Nakamoto, " "Bitcoin: A Peer-to-Peer Electronic Cash System," Cryptography Mailing List," 2008.

[2] D. Yaga, P. Mell, N. Roby and K. Scarfone, "Blockchain Technology Overview," National Institute of Standards and Technology, NIST Interagency/Internal Report (NISTIR) 8202. doi: 10.6028/NIST.IR.8202, Gaithersburg, MD, 2018.

[3] K. Kassev and L. Nenov, "Security Analysis of Shamir's Secret Sharing," 2022," in *Seventh Junior Conference on Lighting (Lighting), Sozopol*, 2022.

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

[5] A. Beimel, ""Secret-Sharing Schemes: A Survey," in Coding and Cryptology," 2011, pp. 11 - 46.

[6] D. R. Stinson, Cryptography: Theory and Practice, Boca Raton: CRC Press, 2006.

[7] G. Dantzig, "Linear Programming and Extensions", Princeton University Press, 1963.

[8] National Institute of Standards and Technology, "Recommendation for Key Management, Part 1: General," NIST Special Publication, 2020.

[9] H. Krawczyk, ""Secret Sharing Made Short," in Advances in Cryptology — CRYPTO '93," in *D. R. Stinson, Ed. Springer Berlin Heidelberg*, 1994.

[10] IEEE, ""IEEE Standard for Floating-Point Arithmetic," 754-2019 (Revision of IEEE Std 754-2008)," IEEE Std, 2019.

[11] M. Wason, D. Pant and A. Kumar, ""A Threshold Cryptography Framework for Secure and Resilient Symmetric Key Management in Multi-Cloud Environments," in 2024," in *International Conference on Green Informatics (ICGI)*, 2024.

[12] A. Narayanan, J. Bonneau, E. Felten, A. Miller and S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton, NJ, Princeton University Press, 2016.

[13] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, Los Angeles, CA, USA, 1987.

[14] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer, 1999.

[15] Y. Lindell, "Secure Multiparty Computation (MPC)," *Communications of the ACM,* vol. 64, no. 1, pp. 86 - 96, 2020.