Comparative Analysis and Implementation of Hierarchical Secret Sharing via REST API for Centralized Credential Security

Derick Amadeus Budiono - 18223090
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: derickamadeus708@gmail.com , 18223090@std.stei.itb.ac.id

***Abstract*—Securing centralized database credentials against insider threats, such as employee collusion, remains a critical challenge. Standard secret sharing schemes distribute trust uniformly, creating vulnerabilities where lower-level employees can bypass executive authority. This paper presents a comparative implementation of three hierarchical secret sharing architectures: Application-Layer Role-Based Access Control, Weighted Secret Sharing, and Tassa Birkhoff Interpolation, integrated within a hybrid REST API environment. The system utilizes a stateless cryptographic engine to prevent credential leakage and a stateful database for immutable hash-based audit logging. Experimental benchmarking evaluated computational latency, payload efficiency, and algorithmic resilience. Results indicate that while the application-layer approach maintains a constant payload size, it relies on bypassable routing logic. The weighted scheme enforces mathematical security but suffers from linear payload expansion and increased latency. The Tassa scheme emerges as the optimal solution, mitigating insider threats natively through matrix singularity properties while preserving a highly efficient, constant payload size per user. This study demonstrates that derivative-based cryptography combined with stateful auditing provides a scalable and robust framework for secure enterprise authentication.**

Keywords— Secret Sharing, Cryptography, REST API, Zero-Trust, Birkhoff Interpolation, Weighted Secret Sharing, Tassa's Secret Sharing, Python

I. INTRODUCTION

In modern organizational infrastructures, centralized databases storing highly sensitive credentials remain a critical point of failure. While external perimeter defenses have grown sophisticated, organizations are increasingly vulnerable to insider threats and privilege escalation. The traditional approach of relying on a single master password or static cryptographic keys violates the core principles of Zero-Trust Architecture. To distribute trust, Shamir's Secret Sharing (SSS) scheme is widely implemented to divide a master credential into multiple shares, requiring a predefined mathematical threshold to reconstruct the secret [1].

However, the standard implementation of SSS treats all shares with mathematical equality. In a corporate environment, this homogeneous distribution introduces a critical security flaw known as the "Employee Coup" scenario. For example, if the threshold is set to three, any three lower-level employees can bypass executive authority and reconstruct the database credential independently. To address this logical vulnerability, hierarchical access controls must be enforced, ensuring that higher-ranking entities (e.g., Executives and Managers) possess greater influence in the reconstruction process than standard employees.

This paper explores the issues, methods, and problems in developing practical applications of cryptography , specifically focusing on the topic of secret sharing schemes. The primary contribution of this research is the design, implementation, and experimental analysis of three distinct architectural approaches to enforce hierarchical secret sharing in a REST API environment.

The study implements and compares the following access structures:

- **Application-Layer RBAC** by utilizing standard Lagrange Interpolation paired with programmatic Role-Based Access Control to filter credentials before mathematical execution.
- **Weighted Secret Sharing (Multiple Shares)** works by providing multiple distinct polynomial coordinates to higher-ranking individuals, enforcing hierarchy purely through standard mathematical thresholds.
- **Tassa's Hierarchical Secret Sharing (Birkhoff Interpolation)** where hierarchy is determined by polynomial derivatives, effectively neutralizing malicious collusion without expanding the payload size.

By evaluating these three algorithms through direct experimentation , this paper provides a comprehensive analysis of data security, computational overhead, payload efficiency, and algorithmic resilience against internal tampering. The findings aim to serve as a medium for sharing research insights on deploying scalable, coup-resistant cryptographic protocols for secure database authentication.

II. LITERATURE REVIEW

A. Shamir's Secret Sharing

The secret sharing scheme was first introduced by Adi Shamir and George R. Blakley in 1979. This mechanism divides a piece of secret information into parts, known as shares, such that reconstruction can only be performed if a minimum threshold of k participants ($k \leq n$) pool their shares together [1]. Conversely, a subset of participants with fewer than k shares will not obtain any partial information about the secret data.

In the standard Shamir's Secret Sharing (SSS) algorithm, the secret is represented as the free coefficient (a_0) of a random polynomial function $P(x)$ of degree $k-1$ shown on (1)

$$\boldsymbol{P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}} \qquad (1)$$

where $a_0 = S$, and the remaining coefficients are randomly selected over a finite field $\mathbb{F}_q$, with q being a large prime number. Secret recovery in this standard scheme relies on the Lagrange interpolation approach. Lagrange interpolation reconstructs the polynomial from k unstructured, pure coordinate points efficiently [6]. However, this conventional scheme treats all shares with uniform mathematical weight, failing to accommodate organizational structures with hierarchical authority.

B. Weighted Secret Sharing (Multiple Shares)

To tighten security without relying on logical server validation, modifications can be made directly at the mathematical layer through Weighted Secret Sharing (WSS). In conventional weighted secret sharing schemes, each participant j is assigned an integer weight w_j . The threshold requirement shifts from a headcount (k) to a minimum accumulated weight (τW) that must be achieved to reconstruct the secret [2].

One practical method to implement WSS using the standard Shamir algorithm involves issuing multiple distinct linear shares (multiple coordinates) to users with higher authority. For instance, if the mathematical threshold is set to $k = 7$, an executive with a weight of 3 receives 3 unique (x, y) coordinates, a manager with a weight of 2 receives 2 points, and a standard employee receives 1 point. This approach purely utilizes standard Lagrange Interpolation for its recovery. The primary limitation of this multiple shares method is the linear expansion of payload size as user weights increase, rendering it highly inefficient for token management in large-scale REST API network transmissions. The master credential S is reconstructed by solving for the free coefficient using the formula shown on (2)

$$S = \sum_{m=1}^{TW} y_m \prod_{j \neq m, j \in C} \frac{-x_j}{x_m - x_j} \pmod{q} \qquad (2)$$

C. Tassa's Hierarchical Secret Sharing

To resolve the token size inefficiency in conventional weighted methods, Tamir Tassa (2004) proposed an ideal Hierarchical Threshold Secret Sharing scheme. This scheme ensures that each participant holds exactly one physical share that is equal in size to the original secret data, yet their access power is mathematically locked based on the degree of polynomial derivatives [3].

In Tassa's scheme, the secret remains hidden as the free coefficient of a polynomial $P(x)$ of degree $k-1$. The distinction lies in the share distribution: participants at the highest level (e.g., Executives) receive pure function values $P(u)$, whereas participants at lower levels (Managers and Employees) receive values from the j-th derivative of the polynomial, $P^{(j)}(u)$ [3]. Because higher-order derivatives eliminate initial coefficients and exponentially reduce variable exponents, lower-level shares carry significantly less mathematical information.

Due to the use of derivative values, secret recovery in Tassa's method can no longer be solved with standard Lagrange interpolation and must instead utilize Birkhoff interpolation [5]. Birkhoff interpolation deals with polynomial reconstruction from a lacunary (unstructured) set of data that combines function points and derivative values at various abscissas. A major challenge in Birkhoff interpolation is the risk of the linear equation system being ill-posed or singular (the matrix lacks a unique solution). Tassa addresses this by applying Pólya's condition and restricting the allocation of x-coordinate identities monotonically over a large prime field to ensure the reconstruction matrix is always regular and yields a non-zero determinant when the quorum is met.

III. DESIGN AND IMPLEMENTATION

A. System Architecture

The system implemented in this study adopts a hybrid client-server architecture based on a REST API using the FastAPI framework. This architecture is specifically designed to combine two main characteristics: a stateless cryptographic engine and a stateful security audit log.

Cryptographic computations run on stateless with the server's RAM to ensure neither the master credentials and the secret shares reside in the server's persistent storage. Conversely, accountability is achieved statefully through a relational SQLite database acting as a one-way audit log system based on cryptographic hashing.

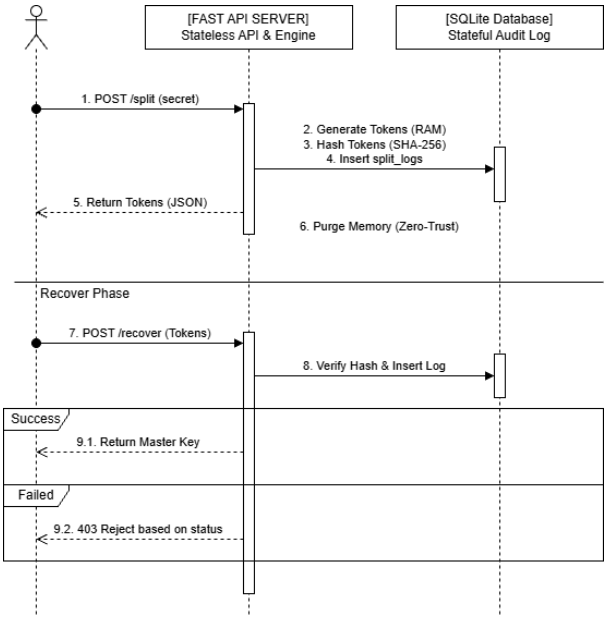

Fig 1. System Architecture Diagram

Component interactions within the system are divided into two main operational phases accessed via the HTTP protocol:

1. Token Generation Phase (Split Phase)

   The administrator or an entity acting as the Cryptographic Dealer sends an HTTP POST request to a server endpoint (e.g., /api/v1/secret/split-tassa). The submitted JSON payload contains the plaintext credential, the quorum threshold (k, and the hierarchical mapping of user roles/weights. The server processes the secret sharing, generates unique physical token shares, calculates the hash value of each token to record in the audit database, and returns the tokens to the client as a JSON array. Following the transmission of the HTTP 200 OK response, the server immediately purges all original credential variables and polynomial components from its internal memory.
2. Share Recovery Phase (Recover Phase)

   In the Share Recovery Phase, a qualified group of participants collaborates to reconstruct the centralized credential by submitting their distributed token shares via an HTTP POST request to the designated recovery endpoint (e.g., /api/v1/secret/recover-tassa). Upon receiving the payload, the server first performs a stateful structural security validation by calculating the SHA-256 hash of each submitted token and cross-referencing it with the immutable audit log database. If any token is unrecognized or forged, the system preemptively aborts the operation, returns an HTTP 403 Forbidden status, and records the anomaly to conserve computational resources and mitigate potential Denial-of-Service vectors. Conversely, if all tokens are cryptographically validated, the stateless mathematical engine executes the appropriate algebraic reconstruction algorithm to decode the master credential, successfully returning it to the authorized users without persistently storing any sensitive variables in the server's memory.

B. Cryptographic Schemes Implementation

The system implements and compares three pilars of hierarchical secret sharing schemes executed over a prime finite field $\mathbb{F}_q$ where $q = 2^{521} - 1$ to prevent large-scale brute-force attacks

1. Application-Layer RBAC Scheme (Standard Shamir)

   The first approach applies the traditional Shamir's secret sharing scheme combined with Role-Based access coded directly to the /recover phase on the API. During the reconstruction phase, the application intercepts the payload to explicitly count the number of incoming role IDs (Executive, Manager, Employee) before allowing the standard Lagrange Interpolation function to execute. The implementation of this application-layer interception is demonstrated in the following code snippet

```
submitted_ids = [vt["uid"] for vt in valid_tokens]
count_b = sum(1 for x in submitted_ids if 1 <= x <= 100)
count_m = sum(1 for x in submitted_ids if 101 <= x <= 300)
count_e = sum(1 for x in submitted_ids if x > 300)

authorized = False
error_msg = ""

if count_e > 0:
    if (count_b + count_m) >= 2: authorized = True
    else: error_msg = "Denied: Need at least 2 supervisors."
elif count_m > 0:
    if count_b >= 1: authorized = True
    else: error_msg = "Denied: Manager requires at least 1 Boss as backup."
elif count_b > 0:
    if count_b >= 2: authorized = True
    else: error_msg = "Denied: At least 2 Boss required."
else:
    error_msg = "Denied: Invalid token."
```

   The security system relies entirely on the integrity of the routing logic rather than cryptographic mathematics. Sadly, if an internal attacker successfully bypasses or spoofs the conditional validation block, the underlying recover_secret function will blindly process the shares and expose the master credential.

2. Weighted Secret Sharing Scheme (Multiple Shares)

   The second approaches shifts the enforcement of hierarchical policies from the application level directly to

the algebraic level. The system assigns the varying number of coordinate shares to participants based on their predefined authoritative weight. For example, to satisfy a quorum threshold $K = 7$, an executive with weight of 3 receives 3 distinct (x, y) coordinate pairs, a manager with weight of 2 receive two, and a standard employee receive only 1 key.

During the recovery phase, the application layer is entirely stripped of hierarchical **if-else** validations. The server simply aggregates all submitted coordinates and feeds them directly into the Lagrange Interpolation engine. The implementation of this mathematically enforced recovery is demonstrated in the following code snippet:

```
# 1. Aggregate the submitted multiple shares
list_of_user_shares = []
valid_tokens = []

for uid_str, shares_list in req.submitted_shares.items():
    uid = int(uid_str)
    user_tuples = []

    for s in shares_list:
        # 1a. Security Validation (Hash checking)
        t_hash = hash_token(s.y)
        if not verify_token_hash(t_hash):
            raise HTTPException(status_code=403, detail="Unrecognized token")

        valid_tokens.append({"uid": uid, "hash": t_hash})
        user_tuples.append((s.x, int(s.y)))

    list_of_user_shares.append(user_tuples)

# 2. Pure Mathematical Reconstruction
try:
    # Hierarchy is strictly enforced by the algebraic threshold k
    recovered_secret = recover_secret_weighted(list_of_user_shares)
except ValueError as math_err:
    # Rejection occurs natively at the mathematical layer
    raise ValueError(math_err)
```

This architecture ensures that the hierarchy is mathematically guaranteed. If a group of employees attempts a coup and tries to recover the credential without sufficient executive weight, the total number of unique coordinates provided will naturally fall short of the k threshold. Consequently, the interpolation engine will inherently fail to resolve the polynomial, triggering a native mathematical exception.

3. Tassa's Secret Sharing

The third approach resolves the severe payload expansion drawback of the Weighted scheme by implementing Tassa's Hierarchical Secret Sharing. In this architecture, the system maintains a strict $O(1)$ payload size per user. Regardless of authoritative rank, every participant receives only one physical token. The hierarchical authority is determined not by the quantity of shares, but by the calculus derivative level of the polynomial evaluation

- Executive Level (Level 0): Evaluated at the pure function $P(x)$
- Manager Level (Level 1): Evaluated at the first derivative $P'(x)$
- Employee Level (Level 2): Evaluated at the second derivative $P''(x)$

During the recovery phase, the submitted payload consists of an unstructured data point with a mixture of standard coordinates and derivative values. Consequently, standard Lagrange Interpolation cannot be used. The API constructs an augmented coefficient matrix based on the participants derivative levels and processes it using Birkhoff Interpolation [3]. Similar to the Weighted approach, the application layer is entirely stripped of RBAC logic. The implementation of this matrix-driven recovery is demonstrated in the following code snippet:

```
# 1. Prepare structured input for the Birkhoff Matrix
formatted_input = []

for s in req.submitted_shares:
    # 1a. Database Audit & Security Validation
    t_hash = hash_token(s.y)
    if not verify_token_hash(t_hash):
        raise HTTPException(status_code=403, detail="Unrecognized token")

    # 1b. Map the derivative level directly to the payload
    # No if-else authority counting is performed here
    formatted_input.append({
        "uid": s.uid,
        "level": s.level,
        "y": int(s.y)
    })

# 2. Determine the matrix dimension dynamically based on input size
k_dim = len(formatted_input)

# 3. Pure Mathematical Reconstruction (Gaussian Elimination)
try:
    # Hierarchy is strictly enforced by matrix singularity properties
    recovered_secret = recover_secret_tassa(k_dim, formatted_input)
```

```
except ValueError as math_err:
    # Rejection occurs natively if the matrix
is singular
    raise ValueError(math_err)
```

This architecture represents the optimal solution for a stateless REST API environment. It enforces strict Zero-Trust security because the hierarchy is locked by the fundamental laws of linear algebra. If a group of low-level employees attempts a coup without an executive token, their second-derivative inputs will generate linearly dependent rows within the Birkhoff matrix. When the Gaussian Elimination algorithm runs, it will encounter a pivot of zero, rendering the matrix singular and mathematically rejecting the unauthorized access attempt without relying on application-layer logic. Simultaneously, it prevents side-channel attacks by ensuring all network packets remain uniform in size.

C. Data Flow and Audit Database Design

To separate operational responsibilities and improve log reading efficiency, the SQLite audit database is designed using the concept of table isolation. There are two independent tables documenting the lifecycle of the secret shares within a single credential_audit.db file

```
CREATE TABLE split_logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    uid INTEGER NOT NULL,
    role_level INTEGER NOT NULL,
    token_hash TEXT NOT NULL,
    algorithm TEXT NOT NULL,
    status TEXT NOT NULL,
    created_at          TIMESTAMP          DEFAULT
CURRENT_TIMESTAMP
);

CREATE TABLE recover_logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    uid INTEGER NOT NULL,
    role_level INTEGER NOT NULL,
    token_hash TEXT NOT NULL,
    algorithm TEXT NOT NULL,
    status TEXT NOT NULL,
    created_at          TIMESTAMP          DEFAULT
CURRENT_TIMESTAMP
);
```

The integrity of the data in this hybrid architecture is guaranteed through the SHA-256 one-way function implemented in the API's security module [4]. Before the token value y is stored in the token_hash column, the system performs a string transformation on the massive integer and converts it into a 64-character hexadecimal hash using the following function :

$$\boldsymbol{H(y) = \text{SHA-256}(y)} \quad (3)$$

Under this scheme, the database server has zero knowledge regarding the actual mathematical values of the secret shares, thus eliminating the risk of token leakage due to SQL Injection or the physical theft of the database file by an internal malicious actor.

id	uid	role_level	token_hash	algorithm	status	created_at
1	5	0	88ef74d60e321530c29518ffdaecbdcde0ad918406e048cec37289a00bcea949	TASSA	SUCCESS	2026-06-16 09:39:47
2	15	1	70aabca31ab5338c466a2ed5d862688e422ca074bf463889108 8d2eec09ad9ac	TASSA	SUCCESS	2026-06-16 09:39:47
3	40	2	9e8d9e56bbb74c25cb61c045c9db87a377a61e198b7ddf0fee63eb0130da0d31	TASSA	SUCCESS	2026-06-16 09:39:47
4	41	2	9e8d9e56bbb74c25cb61c045c9db87a377a61e198b7ddf0fee63eb0130da0d31	TASSA	SUCCESS	2026-06-16 09:39:47

Fig 2. Encrypted hash token on the database

IV. RESULT

A. Environment and Evaluation Procedure

The experimental evaluation was conducted on a localized machine to ensure a controlled testing environment, thereby eliminating external network latency interference during the benchmarking process. The hardware configuration consists of a single Personal Computer (ASUS TUF Gaming) equipped with an 11th Gen Intel® Core™ i7-11800H processor operating at 2.30 GHz, accompanied by 16.0 GB of Random Access Memory (RAM), running on a 64-bit Windows operating system.

The software environment was built utilizing Python 3 as the primary programming language. The REST API architecture was developed using the FastAPI framework and deployed via the Uvicorn ASGI server. The stateful audit logging mechanism was implemented natively using SQLite3, and cryptographic hashing utilized the standard hashlib library.

The evaluation procedure was structured into two primary testing scenarios :

1. Computational Performance Testing**:** A Python-based automation script was utilized to transmit 50 iterative HTTP POST requests to each algorithm's generation endpoint (/split). The execution time was precisely measured in milliseconds to evaluate the computational overhead of the cryptographic mathematical engines.
2. Security and Threat Simulation: The recovery endpoints (/recover) were subjected to simulated insider threat scenarios. This included injecting forged tokens to test the SQLite database verification layer and submitting incomplete hierarchical combinations (e.g., standard employees attempting to recover the secret without executive tokens) to evaluate the native mathematical rejection capabilities of the Birkhoff matrix solver.

B. Computational Latency Analysis

The results of the experimental benchmarking are summarized in Table I.

TABLE I

LATENCY EVALUATION OF SECRET SPLITTING PHASE (50 ITERATIONS)

Architectural Method	*Average (ms)*	*Minimum (ms)*	*Maximum (ms)*
Application-Layer RBAC	50.985	40.381	56.177
Weighted Multiple Shares	88.047	81.818	100.154
Tassa's Birkhoff HSS	51.237	47.905	62.343

The empirical data presents a highly consistent computational overhead, accurately reflecting the impact of the newly integrated hybrid stateless-stateful architecture. Both the Application-Layer RBAC and Tassa's Birkhoff schemes demonstrate nearly identical latency profiles, averaging approximately 51 ms. This proves that the complex calculus differentiation required for Tassa's scheme introduces negligible computational friction compared to standard polynomial generation. The baseline latency of ~50 ms is strictly attributed to the disk I/O operations of the SQLite database and the cryptographic execution of the SHA-256 one-way hashing function required to generate the audit logs.

Conversely, the Weighted Secret Sharing scheme exhibits a significantly higher computational burden, averaging 88.047 ms. This substantial 72% increase in average latency directly correlates with the scheme's structural inefficiency: generating multiple physical shares per participant based on their authoritative weight. Because each generated share must be individually hashed and inserted into the SQLite database sequentially, the system suffers from an $O(W)$ linear processing expansion.

This empirical evidence firmly establishes Tassa's Birkhoff scheme as the superior and most optimal solution. It enforces a strict mathematical hierarchy without inflating the database I/O operations, maintaining a consistent $O(1)$ computational efficiency alongside its optimal payload size.

C. Secret Sharing Results Analysis

The analysis is based on the JSON responses produced by the REST API, simulating an organizational distribution to Executive, Manager, and standard Employee entities.

The standard Shamir's Secret Sharing approach yields a strictly uniform payload structure. Each participant, regardless of their hierarchical role, receives exactly one pair of (x, y)coordinates.

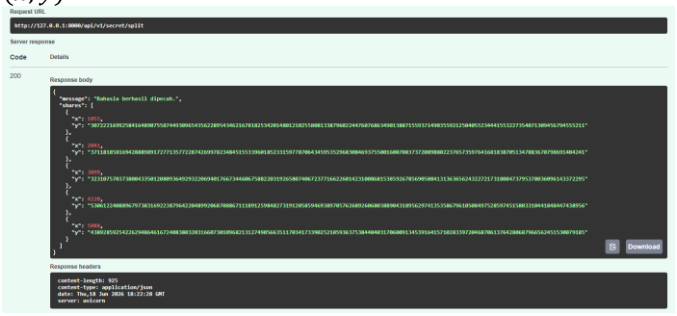

Fig 3. JSON payload of the RBAC split phase.

As illustrated in the JSON response, the array demonstrates an optimal $O(1)$ space complexity per user. The stringified (y) value represents the massive integer mapped to the finite field $\mathbb{F}_q$. However, the token itself contains no mathematical indicator of authority, empirically confirming that the hierarchy is enforced purely through application-level validation rather than cryptographic integration.

The weighted scheme exhibits a drastic structural variance. To enforce a mathematical hierarchy, the payload must undergo linear expansion corresponding to the participant's assigned weight.

Fig 4. JSON payload of the Weighted scheme demonstrating linear expansion.

The generated output highlights the payload bloat inherent to this architecture. An executive assigned a weight of 3 receives a JSON array containing three distinct cryptographic coordinates. This $O(W)$ payload expansion significantly increases network bandwidth consumption during transmission. Furthermore, it creates a vulnerability to side-channel network analysis, as packet sizes visibly differ between corporate ranks, allowing eavesdroppers to deduce the organizational structure by measuring packet lengths.

The output from Tassa's scheme demonstrates a perfect equilibrium between payload efficiency and mathematical security.

Fig 5. JSON payload of Tassa's Birkhoff scheme featuring derivative levels.

As shown in the payload structure, the generated JSON array returns to the optimal $O(1)$ size per user, identical to the standard RBAC method. However, the token structure introduces a level parameter, representing the polynomial derivative tier (0 for the pure function, 1 for the first derivative, and 2 for the second derivative). The integer y represents the evaluated result of that specific calculus derivative. This

structural analysis confirms that Tassa's architecture successfully embeds the hierarchical authority directly within the mathematical token without inflating the data transmission size.

TABLE II

Token Generated In 4 user split

Architectural Method	*Total Token*	*Avg Token Length*
Application-Layer RBAC	4	157
Weighted Multiple Shares	7	157
Tassa's Birkhoff HSS	4	157

D. Auditability and Threat Mitigation Analysis

The integration of a stateful SQLite audit database successfully elevated the system's security posture without compromising the stateless nature of the cryptographic engine. The system's resilience was evaluated using an automated penetration testing script designed to simulate two distinct threat vectors: external token forgery and internal employee collusion. This test is conducted by using script named **test_threats.py** (file can be seen on github)

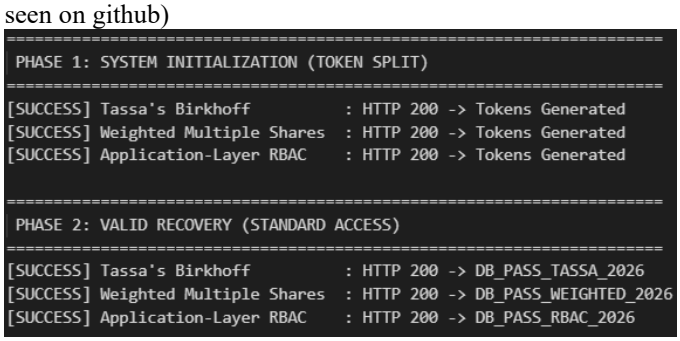

Fig 6. Happy path threat testing

The first simulation tested the system's capability to detect manipulated or completely fabricated tokens. An otherwise valid recovery payload was intercepted, and a single character within the y integer string of the executive's token was altered.

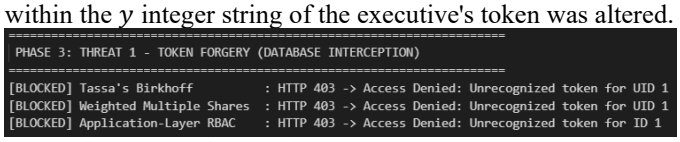

Fig 7. Token Forgery Threat Testing

When the forged payload was submitted to the /recover endpoint, the system's security module immediately computed the SHA-256 hash of the incoming tokens and cross-referenced them with the split_logs table. Because the altered token's hash did not exist in the database, the API preemptively rejected the request with an HTTP 403 Forbidden status. This architectural design successfully prevented the server from feeding malicious data into the intensive Lagrange or Birkhoff mathematical engines, thereby conserving CPU resources and mitigating potential Denial-of-Service (DoS) vectors.

The second simulation evaluated the system's structural integrity against an internal coup, where a group of valid lower-level employees attempted to pool their tokens to reconstruct the credential without the required executive authority.

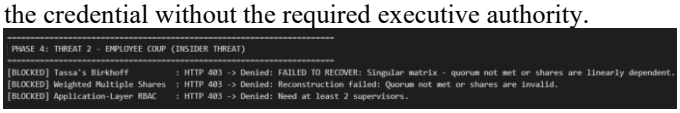

Fig 8. Employees coup Threat Testing

Because the hierarchical combination was incomplete (missing the pure function token from an executive), the resulting matrix contained linearly dependent rows. During the Gaussian Elimination sequence, the algorithm encountered a zero-pivot, mathematically proving that the matrix was singular. The system natively threw a mathematical exception and aborted the recovery, ensuring that Zero-Trust policies were enforced by the immutable laws of linear algebra rather than programmable application logic.

E. Stateful Audit Log Verification

Every simulated interaction, both successful and malicious, was chronologically recorded in the SQLite database to ensure strict accountability. An examination of the recover_logs table confirms that the system accurately classifies and records security anomalies. Forgery attempts are permanently stamped with a FAILED_INVALID_TOKEN status, providing security administrators with the precise User ID (UID) whose token was compromised. Meanwhile, the failed collusion attempts in Tassa's scheme are recorded as FAILED_MATRIX_REJECTED.

V. CONCLUSION

This study successfully implemented and evaluated three distinct hierarchical secret sharing architectures within a hybrid stateless-stateful REST API environment. The integration of a stateless cryptographic mathematical engine with a stateful SQLite audit database proved to be a highly effective strategy for securing centralized database credentials. By utilizing SHA-256 one-way hashing, the system guarantees an immutable audit trail for forensic analysis without ever exposing the original mathematical values of the physical shares to the server's persistent storage.

Through empirical benchmarking and automated threat simulation, the comparative analysis exposed critical vulnerabilities and inefficiencies in conventional approaches. The Application-Layer RBAC scheme, while maintaining an optimal $O(1)$ payload size, inherently violates Zero-Trust principles by relying on bypassable programmatic routing logic rather than cryptographic algorithms. Conversely, the Weighted Secret Sharing scheme enforces strict mathematical security but suffers from an $O(W)$ linear payload expansion. This expansion significantly increases bandwidth consumption and token generated, exposes the system to side-channel analysis, and increases computational latency by over 72% due to sequential database I/O operations.

Furthermore, Tassa's Hierarchical Secret Sharing scheme, utilizing Birkhoff Interpolation, definitively emerged as the most optimal and secure architecture for enterprise API deployment. It successfully mitigates the employee coup scenario through the unalterable properties of matrix singularity, decisively rejecting unauthorized access without relying on application-level validation, while maintaining the number of tokens generated with an ideal $O(1)$ payload size. Ultimately, this research demonstrates that coupling derivative-based cryptographic algorithms with stateful hash auditing yields a scalable, coup-resistant, and high-performance authentication framework for modern organizational infrastructures.

VI. Appendix

Source code repository :
https://github.com/DerickAmadeus/18223090-Makalah-II4021-Kriptografi

YouTube Video Link :
https://youtu.be/BlxlqlARuTM?si=UbGq_FEP0J6xDpKA

Acknowledgment

The author would like to express his gratitude to Almighty God for all the blessings and success in completing this course. Besides, that the author wishes to extend his deepest appreciation to Dr. Ir. Rinaldi, M.T**.** as the lecturer of the II4021 Cryptography course, for the invaluable guidance, comprehensive feedback, and continuous support throughout the semester.

Finally, the author is highly grateful to his family, colleagues and peers for their support, constructive discussions, technical insights, and encouragement during the courses.

References

[1] Rinaldi M. "Skema Pembagian Data Rahasia". Available : https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2025-2026/36-Skema-Pembagian-Data-Rahasia-2026.pdf

[2] F. Benhamouda, S. Halevi, and L. Stambler, "Weighted Secret Sharing from Wiretap Channels," Feb. 2023. [Online].

[3] T. Tassa, "Hierarchical Threshold Secret Sharing", 2004, 473–490

[4] Rinaldi M. "Beberapa Fungsi Hash". Available : https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2025-2026/27-Beberapa-fungsi-hash-2026.pdf

[5] G. G. Lorentz and K. L. Zeller "SIAM Journal on Numerical Analysis", Vol. 8, No. 1 (Mar., 1971), pp. 43-48

[6] Wolfram MathWorld "Lagrange Interpolating Polynomial". Available : https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Derick Amadeus Budiono 18223090