Implementation of Access Control using Chained Weighted Secret Sharing for Directory Access

Bagas Sambega Rosyada
Informatics Engineering, School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
13522071@std.stei.itb.ac.id, bagassambega@gmail.com

***Abstract*—In organizational systems, administrators often have excessive privileges that can compromise user privacy, but operational requirements still demand certain administrative capabilities. This paper proposes a folder encryption system that balances administrative control with user privacy using weighted secret sharing and capability-based access control.**

The system works by splitting administrative keys to multiple administrators with different weights, so it is requiring authorized collaborations of administrators to meet weight thresholds before delegating access. Users receive delegated keys that must be combined with their personal keys to access encrypted data so that neither administrators nor users alone can decrypt the content. The proposed implementation divided into two cases: read-only access and enhanced operation by using capability access for write and delete. This approach eliminates single points of trust while maintaining necessary administrative functions.

***Index Terms*—access control, weighted secret sharing, capability-based access control, directory encryption, key delegation**

I. Introduction

In modern information systems, especially in a corporate or organizational environment, managing access rights to sensitive data is a must. Company usually enforce some rules to prevent unauthorized or unrelated users to access certain data or resources. In an organizational structure, usually a person's level or position can determine the access rights they have. The higher a person's position in the company's administration, the higher the access they have and the more capabilities they can perform. Usually higher positioned person acts as administrators who has more access and control over common users or lower positioned person.

However, in practice, administrator or higher level user often have overpermissive privileges and access that can increasing the risk of misuse and increasing chance of insider attacks. To mitigate these risks, various security mechanisms have been proposed, such as data encryption and the use of key management systems (KMS) to ensure that administrators cannot directly access sensitive data. However, operational requirements still require administrators to have certain capabilities, such as to perform recovery, maintenance, or grant temporary access permissions to users. The conflict between the principle of least privilege access and operational requirements shows the issue of how to provide administrative control without making administrators the single point of trust for sensitive data.

The other problem is company usually doesn't only store company or business-related data, but also user-privacy related data that shouldn't be exposed or accessed by anyone, even administrators themselves. Users are limited to access company or to access and modify their data to ensure company trust the user, but user privacy-related data shouldn't be able to accessed by other rather than user itself. Example of this case is personal account that used to authorize users to company system. Company can store the user's credential but only the user themselves should be able to access and modify them.

One of the mechanism that can be used by company is to split responsibilites over high security data to more than one person, so if one administrator want to misuse their capability over sensitive data, they can not doing this by themselves. This can be done by splitting access keys of single main key to access data to several administrators. But they need to also consider delegation access to user who wants to access or modify their own personal data, without making the administrator themselves be able to access the user's data.

To address these needs, we propose an approach based on shared and chained secret sharing. Shared secret sharing is used to distribute administrative keys to multiple administrators, utilizing a weighted secret sharing scheme to represent different levels of trust and capability for each administrator. Then to delegate access from the administrators to the users without making administrator have capabilites over user's data, we need to chaining the administrator rights to the user by composing a new key that can be used by user along with user personal key. In this paper, user data is limited to a single directory that related to user's private data.

This paper illustrate how capability based access control using chained and weighted shared secret can contruct access permissions in a modular and layered manner by following the principle of least privilege and without compromising system operational capabilities. By combining weighted secret sharing and key composition, the proposed scheme enables the establishment of a cryptographic chain of trust from administrators to users without providing direct access to data.

II. Theoretical Basis

A. Shamir Secret Sharing

Secret sharing is a cryptographic technique to dividing secrets to multiple parties so that combination of parties can reconstruct the original secret. Shamir [1] introduced a (k, n) threshold secret sharing scheme where a secret S is divided into n shares, and any k or more shares can reconstruct the secret, but fewer than k shares reveal no information about S.

Given a secret S and parameters (k, n) where $1 \leq k \leq n$, there exists a scheme to divide S into n shares such that:

1) Any k shares can efficiently reconstruct S
2) Any $k-1$ or fewer shares cannot gives information about S

This scheme is called threshold scheme. The scheme is based on polynomial interpolation over a finite field.

1) Share Generation: The algorithm for generating n number of shared keys is,

1) Choose a prime p where $p > \max(S, n)$ and in the finite field $\mathbb{Z}_p$
2) Create a random polynomial of degree $k - 1$:
$$f(x) = S + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1} \mod p \quad (1)$$
where $f(0) = S$ is the secret, and $a_1, a_2, \ldots, a_{k-1}$ are random coefficient numbers from $\mathbb{Z}_p$
3) Generate n shared keys by evaluating the polynomial at n distinct non-zero points. Each user i receives:
$$\text{key}_i = (x_i, y_i) = (x_i, f(x_i)) \mod p \quad (2)$$
for $i = 1, 2, \ldots, n$.
4) Each user i gets key_i.

2) Secret Reconstruction: When k or more users use their keys to reconstruct/recreate the secret, they combine their keys using Lagrange interpolation [2, 3].

1) Each user have (x_i, y_i).
2) Apply Lagrange interpolation to find $f(0)$, which equals the secret S:
$$S = f(0) = \sum_{i=1}^{k} y_i \cdot L_i(0) \mod p \quad (3)$$

B. Weighted Secret Sharing

In standard Shamir secret sharing, all participants hold equal power/position. Weighted secret sharing gives asymmetric access control where some users are more trusted than others [4]. Weighted secret sharing assigning different weights to participants. More trusted or higher access user will have bigger weight than others.

To create a shared weighted secrets can be done by these steps,

1) Define threshold k and assign each user i weight w_i, where $\sum w_i = n$ and total threshold is k
2) Create a random polynomial of degree $k - 1$:
$$f(x) = S + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1} \mod p \quad (4)$$
3) For each user i with weight w_i, generate w_i distinct shares:
$$key_{i,j} = (x_{i,j}, f(x_{i,j})) \quad \text{for } j = 1, 2, \ldots, w_i \quad (5)$$
4) Distribute all w_i shares to user i

To reconstruct the original master key, users combine their shares together, so as long as the total weight of contributing users equal or exceeds k, they can reconstruct S using Lagrange interpolation.

C. Lagrange Polynomial Interpolation

Lagrange polynomial interpolation [2, 5] is a mathematical method to construct a polynomial that passes through a given set of points. This formula is used in Shamir's secret sharing scheme, because it makes the reconstruction of the secret polynomial from an enough number of shares possible.

Given k distinct points $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$, there exists a unique polynomial $P(x)$ of degree at most $k - 1$ that passes through all these points [2]. Lagrange interpolation formula constructs this polynomial with:

$$P(x) = \sum_{i=1}^{k} y_i \cdot L_i(x) \quad (6)$$

where $L_i(x)$ is the Lagrange basis polynomial defined as:

$$L_i(x) = \prod_{j=1, j \neq i}^{k} \frac{x - x_j}{x_i - x_j} \quad (7)$$

In usage with secret sharing, to reconstruct the secret $S = f(0)$ from k shares, we evaluate the Lagrange interpolation at $x = 0$:

$$S = P(0) = \sum_{i=1}^{k} y_i \cdot L_i(0) = \sum_{i=1}^{k} y_i \prod_{j=1, j \neq i}^{k} \frac{-x_j}{x_i - x_j} \quad (8)$$

This formula makes the participants can reconstruct the secret directly from their shares without need of computing the polynomial coefficients.

D. Access Control

Access control is a security mechanism that regulates who or what can view or use resources in a computing environment. It is a fundamental concept in security that ensures that only authorized entities can access specific resources, preventing unauthorized access and protecting sensitive information.

Access control systems typically consist of three main components:

- Subject: The entity requesting access (e.g., user, process, or system)
- Object: The resource being accessed (e.g., file, database, or service)
- Access Rights: The permissions that define what operations the subject can perform on the object (e.g., read, write, delete)

1) Role-Based Access Control (RBAC): Role-Based Access Control [6] is an access control model where permissions are assigned to roles rather than to individual users. Users are then assigned to appropriate roles based on their responsibilities and qualifications.

Usually in RBAC system, there are two primary roles:

- Administrator: Users with elevated privileges who can do all operations such as read, write, and delete
- Regular User: Users with limited privileges, usually restricted to read-only operations

How RBAC works can be seen in illustration 1.

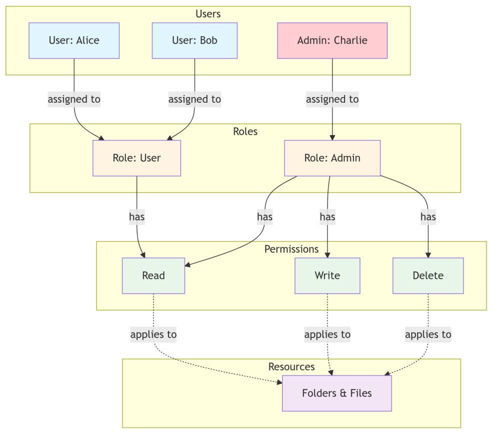

Figure 1: Illustration of RBAC

2) Capability-Based Access Control (CapBAC): Capability-Based Access Control [7, 8] is an access control model where access rights are granted through cryptographic capabilities (tokens) rather than identity checks. A capability is a token that gives the bearer permission to perform specific operations on specific resources.

In CapBAC, each capability token represents a specific permission. The CRUD (Create, Read, Update, Delete) operations are common examples:

- create: Allows the bearer to create new files or folders
- read: Allows the bearer to read existing files
- update Allows the bearer to modify existing files
- delete: Allows the bearer to delete files or folders

How CapBAC works can be seen in illustration 2.

E. Directory/Folder Encryption

Folder encryption [9] is a security technique that protects directories and their contents by encrypting files with cryptographic keys.

Conventional folder encryption systems like BitLocker, VeraCrypt, or LUKS typically use symmetric encryption algorithms (e.g., AES-256) where a single master key locks and unlocks the entire folder.

But even though these systems use strong encryption, they have limitations of binary access, where users either have complete access to all files or no access at all.

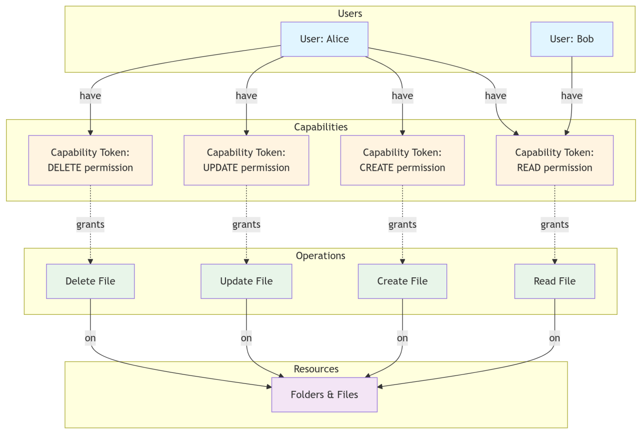

Figure 2: Illustration of CapBAC

F. Digital Signatures and Authentication

Digital signatures are cryptographic mechanisms that provide authentication and integrity for digital messages or documents [10].

1) Signature Creation and Verification: A digital signature scheme [11] consists of three algorithms:

- Key Generation: Creates a key pair (sk, pk) where sk is the private signing key and pk is the public verification key
- Signing: Given a message m and private key sk, produces a signature $\sigma = \text{Sign}_{sk}(m)$
- Verification: Given a message m, signature σ, and public key pk, returns true or false: $\text{Verify}_{pk}(m, \sigma) \in \{\text{true}, \text{false}\}$

2) Message Authentication Codes (MAC): In symmetric-key settings, Message Authentication Codes [12, 13] is giving similar functionality as digital signatures. A MAC is computed using a shared secret key:

$$\tau = \text{MAC}_K(m) \tag{9}$$

where K is the shared secret key, m is the message, and τ is the authentication tag.

Verification checks if:

$$\text{Verify}_K(m, \tau) = (\tau \stackrel{?}{=} \text{MAC}_K(m)) \tag{10}$$

If the verification succeeds, it proves that:

- The message has not been tampered with (integrity)
- The sender have the secret key K (authentication)

III. Proposed Methods

The implementation of proposed method divided into two cases, one where administrators only giving their base keys for providing user ability to read and access their data, and the other case where administrators also giving additional capabilities (such as create and delete data) to the user. The illustration of how the algorithm works can be seen on Figure 3.

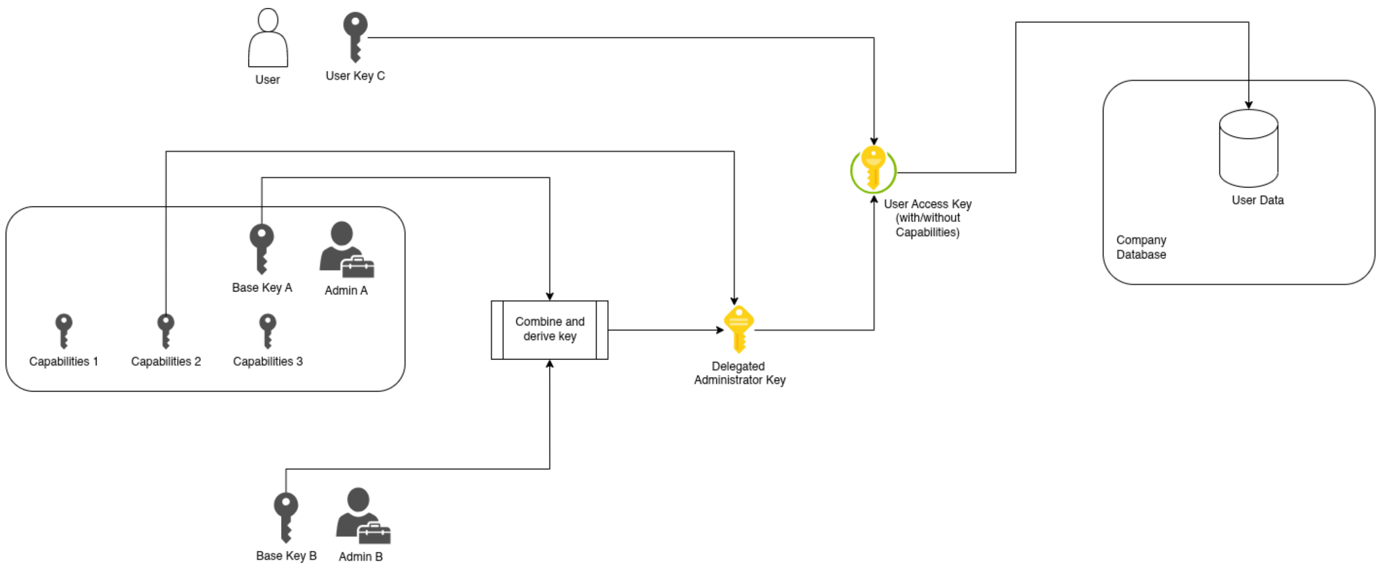

Figure 3: Proposed algorithm show the key distribution and delegation process from administrators and users

A. Actors and Properties Definitions

1) Actors and Keys: We first define the system actors and their properties. The system consists of administrators who control access rights and users who need to access their encrypted data.

1) Let there be a set of administrators of A,

$$\mathcal{A} = \{A_1, A_2, \ldots, A_n\},$$

and a user U.

2) Each administrator A_i has:
   - a base secret key share (used for basic access delegation)
   $$s_i \in \mathbb{F}_p,$$
   - a weight (determining the administrator's influence in decision making for delegated access)
   $$w_i \in \mathbb{N},$$
   - and a set of capability keys (additional permissions beyond read access)
   $$\{K_i^{op} \in \mathbb{F}_p \mid op \in \mathcal{O}\},$$
   where
   $$\mathcal{O} = \{\mathsf{read}, \mathsf{write}, \mathsf{delete}\}.$$

3) The user U have a private user key (which will be combined with delegated administrator keys to access data)

$$K_U \in \mathbb{F}_p.$$

All computations are performed over a finite field $\mathbb{F}_p$ for a large prime p. We assume these variables available or provided:

- Data of the user D_U
- A key derivation function $\mathsf{KDF}(\cdot)$.
- A hash function $H(\cdot)$.
- A symmetric encryption scheme $(\mathsf{Enc}_K, \mathsf{Dec}_K)$.
- An authenticated encryption scheme $(\mathsf{EncAuth}_K, \mathsf{DecAuth}_K)$, where decryption outputs $\perp$ if verification fails. This authenticated encryption scheme acts to provide signature for every additional capabilites performed by users. This signature acts to verified if the users actually have and can do such capabilities.

2) Threshold Definitions:

1) Administrator Threshold
   - Let W to be the minimum total weight required to authorize delegation. This to make sure that no single administrator can single-handedly grant access. A subset $S \subseteq \mathcal{A}$ is authorized if
   $$\sum_{A_i \in S} w_i \geq W.$$
   - Let
   $$t = \min\{|S| : \sum_{A_i \in S} w_i \geq W\}$$
   be the minimum effective threshold to be used in Shamir secret sharing. This represents the minimum number of administrators needed to meet the weight requirement.

2) User Access Thresholds

   We define:
   - $W_{\mathsf{read}} = W$: threshold to delegate read access,
   - $W_{\mathsf{cap}} \geq W$: threshold to delegate additional capabilities (e.g., write or delete).

   This makes a collaboration S can:
   - grant read access if $\sum_{A_i \in S} w_i \geq W_{\mathsf{read}}$,
   - grant capability access if $\sum_{A_i \in S} w_i \geq W_{\mathsf{cap}}$.

B. Schemes Algorithm

1) Shamir Secret Sharing Setup: Before the system operates, administrators must perform an initial setup to distribute secret shares. We first define how administrators get their keys and how they can reconstruct a key for enabling them to construct a delegation key.

1) Administrators collaborated define a master secret (which will be split between them)

$$s \in \mathbb{F}_p,$$

and construct a random polynomial of degree $t-1$:

$$f(x) = s + a_1 x + \cdots + a_{t-1}x^{t-1}.$$

2) Each administrator A_i receives a unique share by evaluating the polynomial at their identifier

$$s_i = f(i).$$

3) When a sufficient collaboration S forms, they can reconstructs the master secret

$$s = \mathsf{Rec}(\{(i, s_i)\}_{A_i \in S})$$

using Lagrange interpolation.

2) Case 1: Delegation Without Capability Keys (Read-Only): In this scenario, administrators grant only basic read access to the user without any additional operational capabilities.

1) A collaboration of administrators $S \subseteq \mathcal{A}$ must satisfy the weight threshold

$$\sum_{A_i \in S} w_i \geq W_{\mathsf{read}}.$$

2) The administrators reconstruct the master secret s using Lagrange interpolation and derive a delegated read key

$$K_D^{\mathsf{read}} = \mathsf{KDF}(s).$$

Since no other capabilities are granted, other operation keys are remain undefined:

$$K_D^{\mathsf{write}} = \perp, \qquad K_D^{\mathsf{delete}} = \perp.$$

3) The delegated key K_D^{read} is sent to the user.
4) The user combines their personal key with the delegated key to derive the finalized access key

$$K_U^{\mathsf{read}} = \mathsf{KDF}(K_U \parallel K_D^{\mathsf{read}}).$$

5) Read encryption:

$$C = \mathsf{Enc}_{K_U^{\mathsf{read}}}(D_U), \quad D_U = \mathsf{Dec}_{K_U^{\mathsf{read}}}(C).$$

3) Case 2: Delegation With Capability Keys: In this scenario, administrators can grant additional operational capabilities such as write or delete permissions to users for their data.

1) A collaboration of administrators $S \subseteq \mathcal{A}$ must satisfy the capability threshold

$$\sum_{A_i \in S} w_i \geq W_{\mathsf{cap}}.$$

The administrators reconstruct the master secret s as before.
2) We define a composition function to make sure the administrator keys not exposed and we can include the capabilites keys.

$$\Phi(x, y) = \mathsf{KDF}(H(x \parallel y)).$$

where $x = s$ is the constructed keys from administrators earlier and $y = K_j^{op}$ is the capability key.
The delegated keys are derived separately for each operation:

$$K_D^{\mathsf{read}} = \mathsf{KDF}(s),$$

$$K_D^{op} = \Phi(s, K_j^{op}), \quad op \in \mathcal{C},$$

where $\mathcal{C} \subseteq \{\mathsf{write}, \mathsf{delete}\}$. For others, $K_D^{op} = \perp$.
3) Delegated keys are sent to the user.
4) User derives for each operation:

$$K_U^{op} = \begin{cases} \mathsf{KDF}(K_U \parallel K_D^{op}), & K_D^{op} \neq \perp, \\ \perp, & \text{otherwise.} \end{cases}$$

5) Operations:
   - Read:

$$C = \mathsf{Enc}_{K_U^{\mathsf{read}}}(D_U), \quad D_U = \mathsf{Dec}_{K_U^{\mathsf{read}}}(C).$$

   - Write:

$$(C', \tau) = \mathsf{EncAuth}_{K_U^{\mathsf{write}}}(D'_U).$$

   - Delete:

$$(\mathsf{cmd}, \tau) = \mathsf{EncAuth}_{K_U^{\mathsf{delete}}}(\text{“delete”} \parallel \mathsf{id}).$$

   where τ is sort of hashed keys or signature of the user doing write/delete operations.

The server accepts operation op if and only if

$$\mathsf{DecAuth}_{K_U^{op}}(\cdot) \neq \perp.$$

C. Example Case

We now showing examples of both cases with numerical values. For sake of simplifying the illustration of proposed algorithm above, we will use simplified mathematical operations especially for key derivation function and also use simple numbers.

Let $p = 17$. For this illustration, the system has three administrators each with their own weight:

$$A_1, A_2, A_3, \quad w_1 = 2,\ w_2 = 1,\ w_3 = 1.$$

Now we set thresholds:

$$W_{\text{read}} = 3, \quad W_{\text{cap}} = 3.$$

So that any collaboration from admisitrators that have total weight minimum 3 is authorized.

We separately define user key:

$$K_U = 7.$$

Next we will use Shamir secret sharing to provide administrators their own base keys.

For example let master secret to:

$$s = 5.$$

Define the function of polynomial (of degree 1 for simplicity):

$$f(x) = 5 + 3x \pmod{17}.$$

By providing $x_i = i$, shares for each $administarator_i$:

$$s_1 = 8,\ s_2 = 11,\ s_3 = 14.$$

Collaboration $S = \{A_1, A_2\}$ has total weight $2 + 1 = 3$ meets the threshold and reconstructs $s = 5$.

For simplicity in this example, we assume the function to delegate the keys to user (KDF) and function to create access key ($H()$):

$$z = H(x \| y) = x + y \pmod{17}, \quad \mathsf{KDF}(z) = z.$$

For sake of simplicity, we implement a simple authenticated encryption scheme for write operations that generates both the encrypted content and a signature for verification:

$$\mathsf{EncAuth}_k(m) = (m + k \bmod 17,\ k + (m + k) \bmod 17).$$

1) Case 1: Read-Only Access: The administrators generate a delegated read key and the user derives their access key. From the steps above, we already know that $s = 5$ and using:

$$K_D^{\text{read}} = 5, \quad K_U^{\text{read}} = 7 + 5 = 12.$$

Suppose the user wants to encrypt their data $D_U = 9$:

$$C = 9 + 12 = 21 \pmod{17} \equiv 4,$$

The encryption process resulting data to be value of 9. Then for the decryption process:

$$(4 - 12) \pmod{17} \equiv 9.$$

The result shows that encryption and then decryption matched.

2) Case 2: Delegation With Write Capability: In this case, administrator A_1 has write capability and provides their capability key to grant write access to the user:

$$K_1^{\text{write}} = 6.$$

Then the administrator delegate the keys to user,

$$K_D^{\text{read}} = 5, \quad K_D^{\text{write}} = 5 + 6 = 11.$$

After receiving the delegated admin keys, the user construct access keys using same function as in the case 1,

$$K_U^{\text{read}} = 12, \quad K_U^{\text{write}} = 7 + 11 = 18 \equiv 1.$$

Now suppose the user wants to write new data $D'_U = 10$ to the system.

With correct write capability key, the authenticated encryption succeeds:

$$C' = 10 + 1 = 11, \quad \tau = 1 + 11 = 12.$$

The server verifies $1 + 11 = 12$ and accepts the operation.

However, if the user tries to write using only the read key or other manipulated key:

$$C'' = 10 + 12 = 22 \equiv 5, \quad \tau'' = 12 + 5 = 17 \equiv 0.$$

Server checks $1 + 5 = 6 \neq 0$, so

$$\mathsf{DecAuth}_1(C'', \tau'') = \perp.$$

So that without the write capability key, the user cannot generate a ciphertext that passes verification too.

IV. Implementation and Evaluation

A. Implementation Overview

The proposed method is implemented using Python using `hashlib`, `secrets`, and `tempfile` libraries. The implementation consists of three main components:

1) Secret sharing module: Implements Shamir Secret Sharing for distributing administrator base keys using weighted thresholds.
2) Key derivation module: Key derivation chain.
3) Encryption module: Implements both Case 1 (read-only delegation) and Case 2 (capability-enhanced delegation) using AES-GCM for authenticated encryption and MAC-based signatures for capability verification.

B. Folder Encryption Algorithms

The main algorithms' snippets can be seen below. Full implementation can be seen in repository of: Github.

1) Read Encryption

```
def encrypt_for_read(folder_path,
    user_keys):
    # Archive folder to bytes
    archive =
        tar_compress(folder_path)

    # Derive AES key from K_U^read
    aes_key =
        SHA256(K_U_read.to_bytes(32))

    # Generate random nonce
    nonce = random_bytes(12)

    # Encrypt with AES-GCM
    ciphertext =
        AES_GCM_Encrypt(aes_key,
        nonce, archive)

    return
        EncryptedDirectory(nonce,
        ciphertext)
```

Listing 1: Read Encryption Algorithm

This code uses the user's derived read key K_U^{read} then converts it to AES-256 key with SHA-256, and then encrypts the folder archive using AES-GCM mode.

2) Read Decryption

```
def decrypt_for_read(encrypted,
    user_keys, output_path):
    # Derive AES key from K_U^read
  aes_key =
      SHA256(K_U_read.to_bytes(32))

    # Decrypt using AES-GCM
    try:
      archive =
          AES_GCM_Decrypt(aes_key,
          encrypted.nonce,
          encrypted.ciphertext)
    except AuthenticationError:
        return False  # Wrong key

    # Extract archive
    tar_extract(archive,
        output_path)
    return True
```

Listing 2: Read Decryption Algorithm

3) Write Encryption

```
def encrypt_for_write(folder_path,
    user_keys):
    if K_U_write is None:
        return None  # No write
            capability

    # Archive then encrypt
    archive =
        tar_compress(folder_path)
  aes_key =
      SHA256(K_U_write.to_bytes(32))
    nonce = random_bytes(12)
    ciphertext =
        AES_GCM_Encrypt(aes_key,
        nonce, archive)

    # Create authentication tag
    # tau = H(K_U^write ||
        ciphertext)
    tag_input =
        K_U_write.to_bytes(32) +
        ciphertext
    auth_tag =
        SHA256(tag_input)[:16]

    return
        EncryptedDirectory(nonce,
        ciphertext, auth_tag)
```

Listing 3: Write Encryption with Authentication

The authentication tag τ implements the signature just like in section 3.2 for giving proof that the user have the write capability key.

4) Write Verification

The server verifies write capability before accepting the operation:

```
def verify_and_decrypt_write(
    encrypted, user_keys,
        output_path):
    if K_U_write is None:
        return False

    # Verify authentication tag
    expected_tag = SHA256(
        K_U_write.to_bytes(32) +
        encrypted.ciphertext)[:16]

    if encrypted.auth_tag !=
        expected_tag:
        return False  #
            Verification failed

    # Decrypt
  aes_key =
      SHA256(K_U_write.to_bytes(32))
    archive =
        AES_GCM_Decrypt(aes_key,
        encrypted.nonce,
        encrypted.ciphertext)
    tar_extract(archive,
        output_path)
    return True
```

Listing 4: Write Capability Verification

This verification making sure that only users with valid write capability keys can do write operations since the authentication tag cannot be modified without having K_U^{write}.

C. Test Cases and Evaluation

We are doing testing to validate the correctness of the implementation. The tests verify the validity of authorized operations and the rejection of unauthorized attempts.

1) Test Configuration: The test environment uses these global parameters as shown in table I.

2) Test Case: We are using five test scenarios. The results can be seen in table II.

1) Test 1: Read-Only Delegation Success: Verifies that authorized collaboration of administrators with correct

Parameter	Value
Prime p	256-bit prime
Threshold W	3
Admin weights	$w_1 = 2, w_2 = 1, w_3 = 1$
User key K_U	12345678901234567890
Master secret s	98765432109876543210

Table I: Test Configuration Parameters

keys can successfully delegate read access and users can decrypt folders.

2) Test 2: Manipulated Delegation Key Rejection: Validates that tampering with the delegated key K_D^{read} prevents decryption because of key derivation doesn't match.
3) Test 3: Wrong User Key Rejection: Confirms that users with incorrect personal keys K_U cannot decrypt.
4) Test 4: Write Capability Success: Demonstrates that users with valid write capability keys can do write operations.
5) Test 5: Unauthorized Write Rejection: Tests two scenarios:
   - Test 5a: Fake write keys are rejected because there is no capability key
   - Test 5b: Tampered authentication tags verification fail

Test	Result	Key Insight
1. Read success	✓	Correct K_U + correct K_D = valid K_U^{read}
2. Tamper K_D	×	$K'_D \neq K_D \Rightarrow K'^{\text{read}}_U \neq K_U^{\text{read}}$
3. Wrong K_U	×	$K'_U \neq K_U \Rightarrow K'^{\text{read}}_U \neq K_U^{\text{read}}$
4. Write success	✓	Valid capability key produces valid auth tag
5a. Fake K_D^{write}	×	Server has no write key to verify
5b. Tamper τ	×	$\tau' \neq H(K_U^{\text{write}} \parallel C')$

Table II: Summary of Test Results

3) Test Results Summary: Detailed mathematical explanations of each test case, including step-by-step computations and security proofs, are provided in Appendix A.

V. Conclusion and Suggestions

This paper shows that folder encryption can be done by combines weighted secret sharing with capability-based access control. The proposed system implements two delegation cases: read-only access where administrators doing collaborations to create delegated keys, and capability access to write and delete operations through capability keys. By creating delegated keys with user personal keys, the system make sure that not only administrators alone or users alone can access the encrypted data so it can eliminating single points of trust.

However there are many limitations of this implementation and concepts, and can be fixed or enhanced by the next works.

- The complexity of providing multiple access keys for multiple capabilities can be fixed by creating more sophisticated sharing secret mechanism
- Develop capability revocation and key rotation protocols to prevent overused or key stolen
- Finding more robust mathematical operation to make sure capabilities can be verified and not bruteforced

Acknowledgment

The writer would like to thank the IF4020 Cryptography lecturer, Dr. Ir. Rinaldi Munir, M.T., for his teaching and supporting for students in making contributions through innovative papers. The author also expresses his gratitude for all the knowledge, experience and benefits that have been given and taught so that he can provide understanding, skills and knowledge that will be useful in the future.

Declaration

I hereby declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and is not plagiarism.

Bandung, December 27, 2025

[illegible]

Bagas Sambega Rosyada

References

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. DOI: 10.1145/359168.359176.

[2] J.-L. Lagrange, *Leçons sur le calcul des fonctions*. Paris: Courcier, 1806.

[3] J.-P. Berrut and L. N. Trefethen, "Barycentric lagrange interpolation," *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004. DOI: 10.1137/S0036144502417715.

[4] S. Iftene, "General secret sharing based on the chinese remainder theorem with applications in e-voting," in *Electronic Notes in Theoretical Computer Science*, vol. 186, Elsevier, 2007, pp. 67–84. DOI: 10.1016/j.entcs.2006.12.043.

[5] Wikipedia. "Lagrange polynomial." Accessed: 2025-12-25. [Online]. Available: https://en.wikipedia.org/wiki/Lagrange_polynomial.

[6] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," in *Computer*, vol. 29, IEEE, 1996, pp. 38–47. DOI: 10.1109/2.485845.

[7] J. B. Dennis and E. C. Van Horn, "Programming semantics for multiprogrammed computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, Mar. 1966. DOI: 10.1145/365230.365252.

[8] H. M. Levy, *Capability-Based Computer Systems*. Bedford, MA: Digital Press, 1984, ISBN: 0932376223.
[9] J. Reardon, D. Basin, and S. Capkun, "Sok: Secure data deletion," *IEEE Symposium on Security and Privacy*, pp. 301–315, 2013. DOI: 10.1109/SP.2013.28.
[10] R. Munir, *Salindia Perkuliahan IF4020 Kriptografi*. 2025.
[11] W. Diffie and M. E. Hellman, "New directions in cryptography," in *IEEE Transactions on Information Theory*, vol. 22, Nov. 1976, pp. 644–654. DOI: 10.1109/TIT.1976.1055638.
[12] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology — CRYPTO'96*, pp. 1–15, 1996. DOI: 10.1007/3-540-68697-5_1.
[13] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," *RFC 2104*, Feb. 1997. DOI: 10.17487/RFC2104.

APPENDIX

This appendix gives proofs for each test case described in Chapter 4.

A. *Test 1: Read-Only Delegation Success*

This test validates the valid operation when all parties use valid keys.

Step	Operation	Value
1	Coalition $S = \{A_1, A_2\}$	$\sum w_i = 2 + 1 = 3 \geq W$
2	Reconstruct s	$s = \text{Rec}(\{s_1, s_2\})$
3	$K_D^{\text{read}} = \text{KDF}(s)$	SHA256-based derivation
4	$K_U^{\text{read}} = \text{KDF}(K_U \parallel K_D^{\text{read}})$	Composed key
5	Encrypt	$C = \text{AES-GCM}_{K_U^{\text{read}}}(\text{folder})$
6	Decrypt	Success

Table III: Test 1: Successful Read-Only Delegation

B. *Test 2: Manipulated Delegation Key Fails*

This test demonstrates that tampering with the delegated key prevents decryption fails.

Step	Operation	Value
1	Valid delegation	$K_D^{\text{read}} = k$
2	Encrypt with valid key	$C = \text{AES-GCM}_{K_U^{\text{read}}}(\text{folder})$
3	Attacker modifies	$K_D'^{\text{read}} = k + 1$
4	Wrong derived key	$K_U'^{\text{read}} \neq K_U^{\text{read}}$
5	Decryption attempt	Fails $\times$

Table IV: Test 2: Manipulated Delegation Key Rejection

C. *Test 3: Wrong User Key Fails*

This test shows that possession of the valid delegated key is failed without the correct user key.

Step	Operation	Value
1	Original user encrypts	$K_U = 12345678901234567890$
2	Attacker has different key	$K'_U = K_U + 999999$
3	Same delegation key	K_D^{read} (identical)
4	Attacker's derived key	$K_U'^{\text{read}} \neq K_U^{\text{read}}$
5	Decryption attempt	Fails $\times$

Table V: Test 3: Wrong User Key Rejection

Step	Operation	Value
1	Coalition weight	$\sum w_i = 3 \geq W_{\text{cap}}$
2	Derive write key	$K_D^{\text{write}} = \Phi(s, K_1^{\text{write}})$
3	User compose	$K_U^{\text{write}} = \text{KDF}(K_U \parallel K_D^{\text{write}})$
4	Encrypt with auth	$(C', \tau) = \text{EncAuth}_{K_U^{\text{write}}}(\text{folder})$
5	Verify tag	$\tau = \tau_{\text{expected}}$
6	Decrypt	Success

Table VI: Test 4: Successful Write Capability Delegation

D. *Test 4: Write Capability Success*

This test validates capability delegation when valid write capability is given.

E. *Test 5a: Fake Write Key Rejected*

This test shows that users cannot manipulate write capabilities.

Step	Operation	Value
1	Read-only delegation	$K_D^{\text{write}} = \perp$
2	Attacker creates fake	$K_D'^{\text{write}} = K_D^{\text{read}} + 12345$
3	Fake user key	$K_U'^{\text{write}}$ derived from fake
4	Attacker encrypts	(C', τ') with fake key
5	Server checks	Server has $K_U^{\text{write}} = \perp$
6	Verification	Fails $\times$

Table VII: Test 5a: Fake Write Key Rejection

F. *Test 5b: Tampered Auth Tag Rejected*

This test shows that authentication tags cannot be forged or modified.

Step	Operation	Value
1	Valid write encryption	$\tau = H(K_U^{\text{write}} \parallel C')$
2	Attacker tampers	$\tau' = \tau \oplus \texttt{0xFF...}$
3	Server verifies	$\tau_{\text{expected}} = H(K_U^{\text{write}} \parallel C')$
4	Comparison	$\tau' \neq \tau_{\text{expected}}$
5	Result	Rejected $\times$

Table VIII: Test 5b: Tampered Authentication Tag Rejection

G. *Source Code*

Source code for this program can be seen in here: GitHub Repository