Design and Implementation of a Trustless Digital Inheritance Protocol Using Hierarchical Verifiable Secret Sharing

Stefany Josefina Santono - 18223116
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: stefanyjoaefina22@gmail.com , 18223116@std.stei.itb.ac.id

***Abstract*—Digital assets such as cryptocurrency wallets, password manager vaults, cloud accounts, and encrypted documents are increasingly protected by strong cryptographic mechanisms. While these protections ensure confidentiality, they also create challenges for digital inheritance, as assets may become permanently inaccessible after the owner's death. Existing inheritance mechanisms rely heavily on trusted third parties or single individuals, introducing risks of premature disclosure and single points of failure. To address these limitations, a trustless digital inheritance protocol based on Hierarchical Verifiable Secret Sharing is designed and implemented. A two-layer composition of Shamir Secret Sharing enforces a conjunctive access structure requiring at least one notary and two heirs, while Feldman Verifiable Secret Sharing provides protection against forged shares. Experimental evaluation and security analysis indicate that the proposed approach offers secure and controlled secret recovery without relying on trusted third parties.**

Keywords—Digital Inheritance, Hierarchical Secret Sharing, Verifiable Secret Sharing, Shamir Secret Sharing, Feldman VSS, Cryptography.

I. Introduction

The rapid growth of digital technologies has led to the widespread adoption of digital assets, including cryptocurrency wallets, password manager vaults, cloud accounts, social media accounts, and encrypted personal documents. These assets often possess both economic and sentimental value, making their preservation and transfer increasingly important. However, unlike conventional assets, digital assets are protected by passwords, encryption, and platform-specific policies that may prevent lawful heirs from accessing them after the owner's death [1].

Legal issues surrounding digital inheritance have also become increasingly prominent. Questions regarding whether digital assets, including social media accounts, can be inherited and how ownership rights should be established have raised new challenges in inheritance law [2]. In Indonesia, the absence of specific regulations governing digital inheritance further complicates the implementation of inheritance mechanisms in practice [3]. Recent studies have shown that digital assets and income generated from digital platforms may constitute inheritable property under Indonesian civil law, yet technical barriers to accessing protected accounts remain unresolved [4].

Existing approaches to digital inheritance suffer from several limitations. Storing passwords in physical wills enables premature disclosure, entrusting credentials to a single individual introduces a single point of failure, and vendor-specific inheritance services rely heavily on trusted third parties. Consequently, a secure inheritance mechanism should prevent unilateral access while ensuring that authorized parties can recover the assets when necessary. These challenges highlight the need for a mechanism that distributes trust among multiple parties while maintaining confidentiality and preventing unauthorized access.

Based on these considerations, a digital inheritance protocol based on Hierarchical Verifiable Secret Sharing is designed. The protocol aims to provide secure and controlled secret recovery while minimizing reliance on trusted third parties. In addition, mechanisms for share verification and controlled activation are incorporated to ensure that inheritance claims are performed only under authorized conditions. The proposed work consists of the following aspects:

1) Designing a digital inheritance protocol with a conjunctive access structure requiring the participation of at least one notary and two heirs.
2) Incorporating verification and controlled activation mechanisms to prevent forged shares and premature inheritance claims.
3) Implementing the protocol and experimentally evaluating its performance under various adversarial scenarios.

The remainder of this paper is organized as follows. Section II presents the theoretical foundations underlying the proposed approach. Section III describes the proposed digital inheritance protocol, including the actors, state machine, and threat model. Section IV presents the system design and implementation of the protocol. Section V provides the experimental results and analysis. Finally, Section VI concludes the paper and discusses potential directions for future work.

II. Theoretical Foundations

A. Shamir Secret Sharing

Secret sharing is a cryptographic technique that enables a dealer to distribute a secret among multiple participants such that only authorized subsets of participants are able to reconstruct the original secret, whereas unauthorized subsets obtain no information about it. Secret sharing schemes constitute fundamental building blocks in various cryptographic applications, including threshold cryptography, secure multiparty computation, access control, and distributed storage systems.

Among various secret sharing schemes, Shamir Secret Sharing (SSS), introduced by Shamir in 1979, is one of the most widely adopted threshold schemes. Given a threshold parameter t and a total number of participants n, the scheme guarantees that any subset containing at least t shares can reconstruct the secret, whereas fewer than t shares reveal no information about it. This information-theoretic security makes the scheme independent of computational assumptions and suitable for long-term secret protection.

The scheme operates over a finite field $\mathbb{Z}_q$, where q is a sufficiently large prime number. Let $s \in \mathbb{Z}_q$ denote the secret to be protected. A random polynomial of degree $t-1$ is constructed as

$$f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \ (mod\ q),$$

where

$$a_0 = s$$

By evaluating the polynomial at n distinct points, a set of shares is generated according to

$$y_i = f(x_i)\ (mod\ q),$$

Each participant receives one share (x_i, y_i), while the polynomial itself is discarded after distribution. Figure 1 illustrates the overall process of secret distribution and reconstruction in a Shamir Secret Sharing scheme.

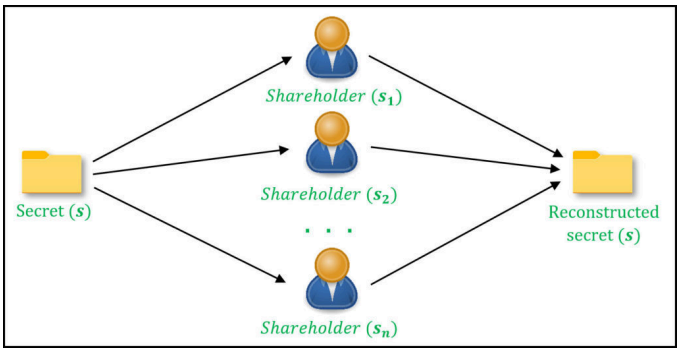

Fig. 1. Overview of secret distribution and reconstruction in Shamir Secret Sharing
(Source: GeeksforGeeks)

Since a polynomial of degree $t-1$ is uniquely determined by t distinct points, collecting at least t valid shares enables reconstruction of the original polynomial through Lagrange interpolation. Consequently, the secret can be recovered by

$$s = f(0)$$

As an example, consider a $(3, 5)$-threshold scheme. Five shares are distributed among five participants, and any combination of three or more shares is sufficient to recover the original secret. Consequently, Shamir Secret Sharing provides both perfect secrecy and fault tolerance, making it one of the most widely used primitives in modern distributed cryptographic systems.

B. Hierarchical Secret Sharing and Access Structures

Traditional threshold secret sharing schemes treat all participants equally, meaning that any subset containing at least a predefined number of shares can reconstruct the secret. Although such schemes are effective in many applications, they are insufficient for scenarios involving participants with different roles and privileges. In practical systems, certain participants may possess higher authority or greater trust than others, requiring more flexible authorization policies than a single threshold can provide.

Hierarchical Secret Sharing (HSS) extends conventional secret sharing by organizing participants into multiple levels according to their authority or priority. In hierarchical schemes, higher-level participants typically have greater influence in the reconstruction process, while lower-level participants may require cooperation from higher levels to recover the secret. Such schemes are particularly useful in organizational structures, financial institutions, and digital inheritance applications where participants possess different responsibilities and trust levels.

The collection of authorized subsets of participants that are allowed to reconstruct the secret is called an access structure. Let

$$P = \{P_1, P_2, \cdots, P_n\}$$

denote the set of all participants. An access structure Γ is defined as

$$\Gamma \subseteq 2^P,$$

where each subset in Γ represents an authorized group of participants, any subset that does not belong to Γ is considered unauthorized and should obtain no information about the secret.

Compared with conventional threshold schemes, hierarchical secret sharing provides greater flexibility in modeling complex authorization policies and enables the distribution of trust among heterogeneous participants. These properties make hierarchical secret sharing particularly suitable for digital inheritance systems, where different entities, such as notaries and heirs, may possess distinct responsibilities and privileges. Consequently, hierarchical secret sharing serves as an essential foundation for the protocol proposed in this work.

C. *Verifiable Secret Sharing*

Traditional secret sharing schemes assume that the dealer behaves honestly during the share distribution process. However, a malicious or faulty dealer may distribute inconsistent shares, causing secret reconstruction to fail even when the required threshold is satisfied. To overcome this limitation, Verifiable Secret Sharing (VSS) extends conventional secret sharing by enabling participants to verify the correctness of their shares without revealing the secret itself .

Among various VSS schemes, Feldman Verifiable Secret Sharing, proposed by Feldman in 1987, is one of the most widely adopted approaches. The scheme introduces public commitments to the coefficients of the secret polynomial, allowing each participant to independently verify whether the received share is consistent with the original polynomial. Consequently, forged or corrupted shares can be detected before the reconstruction stage, thereby improving the integrity and reliability of the secret sharing process.

Let

$$f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$$

denote the polynomial used in Shamir Secret Sharing. Feldman VSS publishes commitments corresponding to each coefficient as

$$C_j = g^{aj}, j = 0, 1, \cdots, t - 1.$$

where g is a generator of a finite cyclic group.

Given a share (x_i, y_i), a participant i can verify the validity of the share by checking

$$g^{y_i} = \prod_{j=0}^{t-1} (C_j)^{x_i^j}$$

where y_i idenotes the share value and x_i represents the corresponding evaluation point. If the equality holds, the share is considered valid; otherwise, the participant can conclude that the share has been corrupted or maliciously generated.

Figure 2 illustrates the distribution and verification process in a verifiable secret sharing scheme. Participants receive shares from the dealer and perform verification before the reconstruction stage, ensuring that invalid shares are detected early and preventing failures during secret recovery.

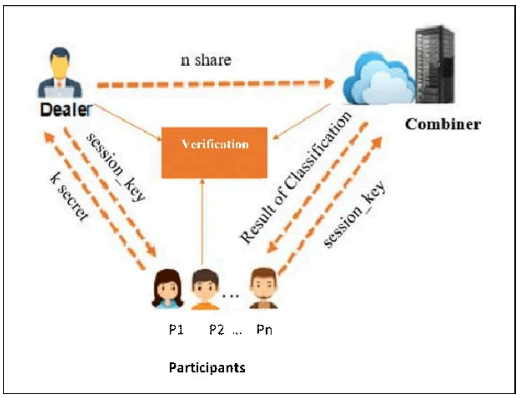

Fig. 2. Distribution and verification process in a verifiable secret sharing scheme
(Source: ResearchGate)

By enabling share validation without revealing the secret itself, Verifiable Secret Sharing improves the reliability and robustness of secret reconstruction. These properties are particularly important in distributed environments involving multiple parties, where trust cannot be entirely placed on a single participant. Consequently, Verifiable Secret Sharing serves as an essential component of the hierarchical digital inheritance protocol proposed in this work.

D. *Authenticated Encryption Using AES-256-GCM*

Authenticated Encryption with Associated Data (AEAD) provides both confidentiality and integrity protection for sensitive information. Among various authenticated encryption schemes, AES-256-GCM (Advanced Encryption Standard with 256-bit keys in Galois/Counter Mode) is widely adopted due to its efficiency and strong security guarantees. In addition to encrypting plaintext into ciphertext, AES-GCM generates an authentication tag that enables the receiver to verify the integrity and authenticity of the encrypted data.

AES-256 employs a 256-bit symmetric key and performs encryption using counter mode, while the Galois field multiplication mechanism generates an authentication tag for message authentication. Consequently, AES-256-GCM simultaneously provides confidentiality, integrity, and authenticity, making it suitable for protecting sensitive information in distributed systems.

Given a plaintext P, encryption key K, initialization vector IV, and additional authenticated data AAD, the encryption process produces a ciphertext C and an authentication tag T, which can be represented as

$$(C, T) = AES - GCMK(IV, P, AAD),$$

where C denotes the encrypted message and T is used to verify the integrity and authenticity of the ciphertext. During decryption, the authentication tag is validated before recovering the plaintext. Any modification to the ciphertext or associated data results in authentication failure, thereby preventing unauthorized tampering.

In the proposed protocol, the symmetric key is derived from the reconstructed secret generated through the secret sharing mechanism. The derived key is subsequently used to

encrypt and decrypt digital assets using AES-256-GCM. By combining confidentiality and integrity protection within a single cryptographic primitive, AES-256-GCM provides secure storage and transmission of sensitive information throughout the inheritance process.

III. Proposed Digital Inheritance Protocol

A. Actors and Trust Assumptions

The proposed digital inheritance protocol involves four entities with distinct responsibilities and trust assumptions. Instead of relying on a single trusted party, the protocol distributes authority among multiple participants to ensure secure and controlled asset recovery.

- Owner: The owner initializes the protocol and generates the cryptographic materials required for protecting digital assets. During the setup phase, the owner encrypts the assets and distributes secret shares to the corresponding participants.
- Heirs: Heirs are designated beneficiaries who participate in the secret reconstruction process. They are only able to recover the assets when the predefined access structure and inheritance conditions are satisfied.
- Notaries: Notaries provide an additional authorization layer and prevent heirs from unilaterally reconstructing the secret. Their involvement ensures that inheritance claims require cooperation among multiple parties.
- Untrusted Coordinator: The coordinator maintains encrypted vaults, public commitments, audit logs, and protocol states. Although it manages protocol operations, the coordinator only has access to ciphertext and public metadata and therefore cannot recover the protected assets.

Under these assumptions, no individual participant or system component is capable of recovering the secret independently. Successful inheritance requires cooperation among authorized parties according to the proposed protocol.

B. State Machine Lifecycle

To prevent premature inheritance claims and provide controlled asset release, the proposed protocol incorporates a state machine mechanism governing the entire inheritance lifecycle. The protocol operates through four states, namely **ACTIVE**, **DORMANT**, **PENDING**, and **RELEASED**, as illustrated in Fig. 3.

Initially, the protocol remains in the **ACTIVE** state while the owner periodically performs check-ins to indicate continued activity. If no check-in is received within a predefined period, the protocol automatically transitions to the **DORMANT** state, indicating a potential loss of owner availability.

Once sufficient shares are collected and an inheritance claim is submitted, the protocol enters the **PENDING** state. During this phase, a veto window is activated, allowing the owner to reject unauthorized or premature inheritance requests. If a valid veto is received, the protocol returns to the **ACTIVE** state. Otherwise, after the veto period expires, the protocol transitions to the **RELEASED** state, enabling the authorized parties to reconstruct the secret and recover the encrypted assets.

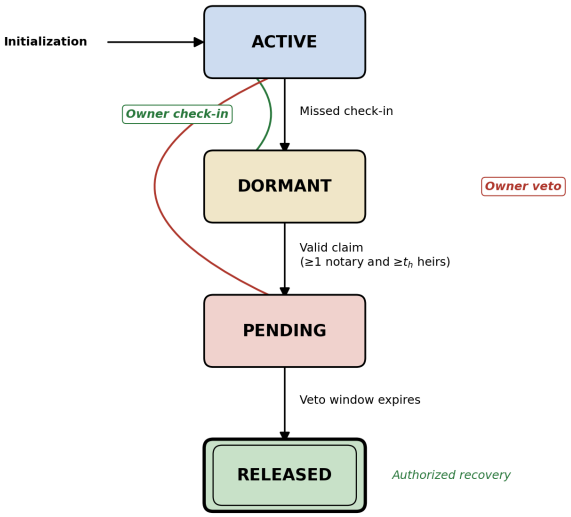

Fig. 3. State transition diagram of the proposed inheritance protocol

By separating the inheritance process into multiple states, the protocol provides additional protection against premature claims and enables controlled asset release while maintaining the trustless nature of the system.

C. Threat Model

The proposed protocol assumes a semi-honest environment in which participants generally follow the protocol specification but may attempt to gain unauthorized access to digital assets. The coordinator is considered untrusted and may be compromised, but only stores ciphertext, commitments, and public metadata. Under these assumptions, the protocol considers the following threat categories and corresponding security objectives:

- Confidentiality threats. Unauthorized parties, including a compromised coordinator, should not be able to reconstruct the secret without obtaining the required threshold shares from both the notary and heir groups.
- Integrity threats. Malicious participants may submit forged or modified shares during reconstruction. Feldman Verifiable Secret Sharing is used to detect invalid shares before they are accepted.
- Premature inheritance claims. Participants may attempt to initiate secret recovery while the owner is still active. The state machine and veto window mechanism prevent unauthorized or premature release of the inheritance.
- Participant collusion. A subset of heirs or notaries may cooperate to reconstruct the secret without satisfying the conjunctive access structure. Reconstruction therefore requires cooperation from both groups according to their respective thresholds.

- Availability threats. Participants may become unavailable or lose their shares. The threshold-based design provides fault tolerance by allowing reconstruction as long as sufficient valid shares remain available.
- Coordinator compromise. Even if the coordinator database is exposed, an adversary only obtains encrypted vault contents, public commitments, and audit logs. Without the required shares, the secret remains computationally infeasible to recover.

IV. System Design and Implementation

A. Overall System Workflow

The proposed digital inheritance protocol is implemented in Python using a modular architecture. The system is composed of several independent modules responsible for secret sharing, share verification, hierarchical composition, vault encryption, state management, and persistent storage. This modular design improves maintainability and facilitates testing of individual components.

The shamir.py and feldman.py modules provide the underlying cryptographic primitives, while hierarchy.py implements the proposed two-layer access structure. Digital assets are protected by the vault.py module using AES-256-GCM, whereas protocol.py manages the inheritance lifecycle through a state machine. Finally, db.py maintains persistent storage and records protocol events through a tamper-evident audit log.

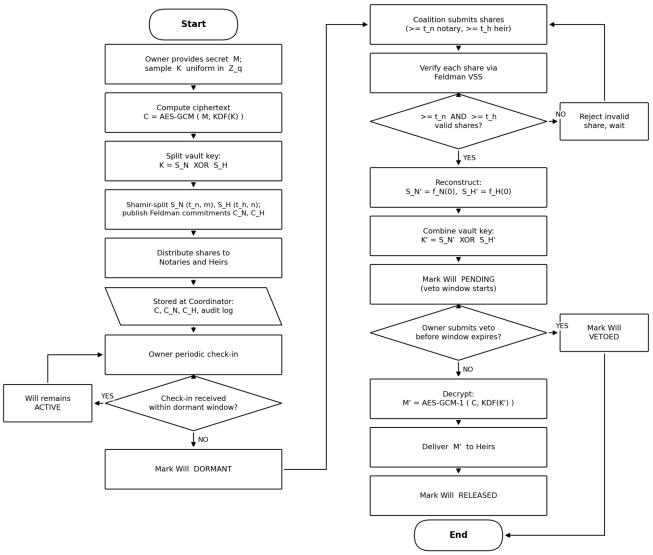

Fig. 4. Workflow and module interaction in TrustWill

B. Shamir Secret Sharing

Shamir Secret Sharing constitutes the foundation of the proposed protocol by providing mechanisms for share generation and secret reconstruction. The scheme transforms the secret into a random polynomial and produces multiple shares that can be distributed independently to different participants. Secret recovery is possible only when the number of available shares satisfies the predefined threshold.

```python
def reconstruct(shares: Sequence[Share], modulus: int) -> int:
    """Recover secret = f(0) from any subset of shares via Lagrange.

    Caller must ensure len(shares) >= threshold; this function will
    happily interpolate with fewer points (producing a wrong answer)
    or with extras (producing the right answer if all are valid).
    """
    if len(shares) < 1:
        raise ValueError("need at least 1 share")
    xs = [s.x for s in shares]
    if len(set(xs)) != len(xs):
        raise ValueError("duplicate share x-coordinates")
    if any(s.x == 0 for s in shares):
        raise ValueError("share x = 0 is invalid")
    return _lagrange_at_zero(shares, modulus)

```

Fig. 5. Share generation procedure

During share generation, random coefficients are selected to construct the polynomial, whose evaluations at distinct coordinates produce the corresponding shares. Each share represents a point on the polynomial and can be stored separately by different participants.

```python
def split(
    secret: int,
    threshold: int,
    n_shares: int,
    modulus: int,
    *,
    xs: Sequence[int] | None = None,
) -> tuple[Polynomial, list[Share]]:
    """Return (polynomial, shares) for a (t, n) sharing of `secret` mod q.

    Args:
        secret: integer in [0, modulus).
        threshold: minimum shares needed to reconstruct (t).
        n_shares: total shares to issue (n >= t).
        modulus: prime q.
        xs: optional explicit x-coordinates (must be non-zero, distinct,
            and < modulus). Defaults to 1..n_shares.
    """
    if not 1 <= threshold <= n_shares:
        raise ValueError(f"require 1 <= t <= n, got t={threshold}, n={n_shares}")
    if not 0 <= secret < modulus:
        raise ValueError("secret out of range [0, modulus)")

    if xs is None:
        xs_list = list(range(1, n_shares + 1))
    else:
        xs_list = list(xs)
        if len(xs_list) != n_shares:
            raise ValueError("len(xs) must equal n_shares")
        if any(x == 0 for x in xs_list):
            raise ValueError("share x-coordinate may not be 0 (reserved for secret)")
        if len(set(xs_list)) != len(xs_list):
            raise ValueError("duplicate x-coordinates")
        if any(not 0 < x < modulus for x in xs_list):
            raise ValueError("x-coordinate out of range")

    coeffs = [secret] + [secrets.randbelow(modulus) for _ in range(threshold - 1)]
    poly = Polynomial(coeffs=tuple(coeffs), modulus=modulus)
    shares = [Share(x=x, y=poly.evaluate(x)) for x in xs_list]
    return poly, shares
```

Fig. 6. Secret reconstruction procedure

Before reconstructing the secret, the implementation performs consistency checks to reject duplicate coordinates and invalid shares. Subsequently, the original secret is recovered through Lagrange interpolation at (x=0), ensuring that only valid shares contribute to the reconstruction process.

C. *Feldman Verifiable Secret Sharing*

To prevent malicious participants from distributing forged shares, the proposed protocol incorporates Feldman Verifiable Secret Sharing as an additional verification layer. Public commitments are generated from the polynomial coefficients, enabling participants to verify the consistency of their shares before reconstruction.

```python
def verify_share(share: Share, commitment: FeldmanCommitment) -> bool:
    """Return True iff share is consistent with the published commitment."""
    p = commitment.group.p
    q = commitment.group.q
    g = commitment.group.g

    if not 0 <= share.y < q:
        return False
    if share.x <= 0 or share.x >= q:
        return False

    lhs = pow(g, share.y, p)

    # rhs = prod C_j ^ (x^j)  mod p
    rhs = 1
    x_pow = 1  # x^0
    for c_j in commitment.coeffs_commit:
        rhs = (rhs * pow(c_j, x_pow, p)) % p
        x_pow = (x_pow * share.x) % q
    return lhs == rhs
```

Fig. 7. Share verification procedure

The verification process first validates the ranges of the share coordinates and computes both sides of the Feldman verification equation using the published commitments. A share is accepted only when the computed values are equal, indicating that the share is consistent with the original polynomial. Consequently, manipulated or corrupted shares can be detected before participating in the reconstruction process.

D. *Hierarchical Secret Sharing*

To enforce the conjunctive access structure, the proposed protocol adopts a two-layer hierarchical composition. Instead of directly distributing the vault secret, the secret is separated into two components corresponding to the notary layer and the heir layer. Each component is independently protected using Shamir Secret Sharing with different thresholds, preventing either group from recovering the secret on its own.

```python
s_n = secrets.randbelow(q)
s_h = secret ^ s_n
# If XOR overflows q (rare: top bit of q is ~always set, but be safe),
# resample. Most realistic groups have q ~ 2^|q| so this is uncommon.
while s_h >= q:
    s_n = secrets.randbelow(q)
    s_h = secret ^ s_n

poly_n, n_shares = split(s_n, notary_threshold, n_notaries, q)
poly_h, h_shares = split(s_h, heir_threshold, n_heirs, q)

c_n = commit(poly_n, group)
c_h = commit(poly_h, group)
```

Fig. 8. Hierarchical share generation

During share generation, a random value is assigned to the notary layer, while the corresponding heir component is obtained through an XOR operation with the original secret. Separate secret sharing procedures are then applied to both components, followed by the generation of Feldman commitments for each polynomial. During recovery, both groups independently reconstruct their respective components, which are subsequently combined through an XOR operation to recover the original vault key. Consequently, successful recovery requires cooperation between the notary and heir layers.

E. *Vault Encryption*

Digital assets are protected using AES-256-GCM, which provides both confidentiality and integrity. Instead of directly distributing the vault contents, the proposed protocol only shares the secret used to derive the encryption key. Consequently, the size of the protected data does not affect the secret sharing process.

```python
def derive_aes_key(secret_int: int, modulus: int) -> bytes:
    """K = SHA-256(secret_int encoded as fixed-width big-endian)."""
    if not 0 <= secret_int < modulus:
        raise ValueError("secret_int out of range [0, modulus)")
    width = (modulus.bit_length() + 7) // 8
    raw = secret_int.to_bytes(width, "big")
    return hashlib.sha256(raw).digest()
```

Fig. 9. AES-256 key derivation from the reconstructed secret

The reconstructed secret is first converted into a fixed-width byte sequence and subsequently hashed using SHA-256 to derive a 256-bit encryption key. The resulting key is then utilized by AES-256-GCM to encrypt the vault contents, producing ciphertext and an authentication tag required for integrity verification. During recovery, the same key derivation procedure is applied to regenerate the encryption key and decrypt the protected assets.

F. *Protocol State Machine*

The operational lifecycle of the proposed inheritance protocol is governed by a state machine consisting of four primary states, namely ACTIVE, DORMANT, PENDING, and RELEASED. These states regulate the inheritance process and prevent premature access to protected assets. In addition, a VETOED state is maintained to handle rejected or canceled claims.

```python
class State(str, Enum):
    ACTIVE = "ACTIVE"
    DORMANT = "DORMANT"
    PENDING = "PENDING"
    RELEASED = "RELEASED"
    VETOED = "VETOED"
```

Fig. 10. Protocol state machine and lifecycle states

G. *Audit Log and Hash Chain*

The proposed protocol maintains a tamper-evident audit log to record important events throughout the inheritance lifecycle. Each entry stores the hash of the previous entry and computes a new SHA-256 digest from the current event and the preceding hash, thereby forming a hash chain. Any modification to an existing record breaks the chain consistency and can be detected during verification.

```python
def append_audit(self, will_id: str, day: int, message: str) -> None:
        row = self.conn.execute(
            "SELECT entry_hash FROM audit_log WHERE will_id = ? "
            "ORDER BY log_id DESC LIMIT 1",
            (will_id,),
        ).fetchone()
        prev_hash = row["entry_hash"] if row else ""

        payload = f"{will_id}|{day}|{message}|{prev_hash}".encode("utf-8")
        entry_hash = hashlib.sha256(payload).hexdigest()

        self.conn.execute(
            "INSERT INTO audit_log(will_id, day, message, prev_hash, entry_hash) "
            "VALUES(?,?,?,?,?)",
            (will_id, day, message, prev_hash, entry_hash),
        )
```

Fig. 11. Audit log generation and hash chaining

V. Experimental Results and Analysis

A. *Scalability of Hierarchical Access Structure*

To evaluate the scalability of the proposed hierarchical access structure, experiments were performed by varying the number of heirs and the reconstruction threshold while measuring both setup and recovery latency. These experiments aim to determine whether the protocol complexity is influenced primarily by the number of participants or by the reconstruction threshold itself.

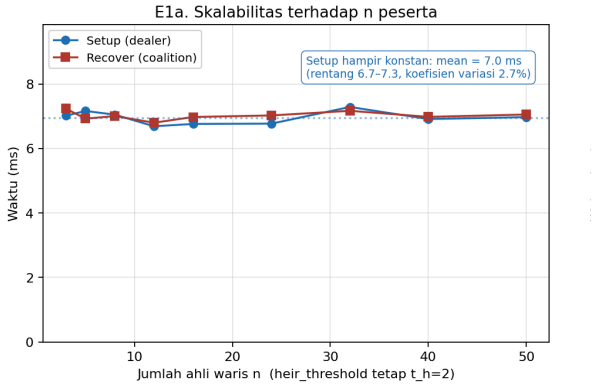

Fig. 12. Scalability with respect to the number of heirs

The results show that increasing the number of heirs from 3 to 50 produces only negligible variations in both setup and recovery latency, with an average execution time of approximately 7 ms and a coefficient of variation of only 2.7%. This observation indicates that the computational complexity of the protocol is not dominated by the total number of participants, but rather by the number of shares required for reconstruction.

Since the heir threshold is fixed at $t_h = 2$, the degree of the Shamir polynomial and the number of Feldman commitments remain unchanged. Consequently, increasing the number of heirs merely increases the number of distributed shares without affecting the amount of polynomial interpolation or modular exponentiation required during setup and recovery. Therefore, the protocol complexity is largely independent of the participant count when the threshold remains fixed.

This result is particularly important for practical digital inheritance scenarios, where the number of designated heirs may vary considerably. The nearly constant latency demonstrates that expanding the participant set does not impose a significant computational burden, thereby providing good scalability without sacrificing performance. More importantly, the results suggest that the protocol complexity is governed primarily by the threshold value rather than by the number of participants, allowing the system to support large beneficiary sets efficiently.

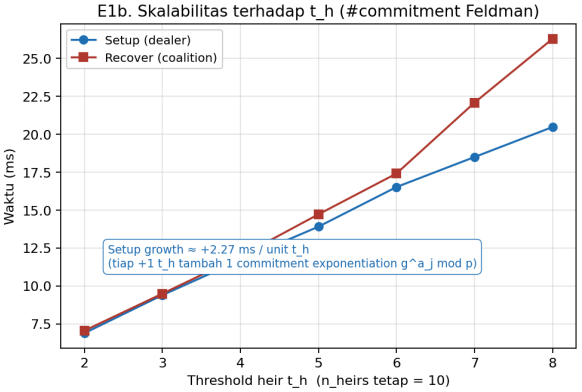

Fig. 13. Scalability with respect to the threshold value

Unlike the previous experiment, increasing the heir threshold causes both setup and recovery latency to grow approximately linearly. The setup time increases from about 7 ms at $t_h = 2$ to more than 20 ms at $t_h = 8$, while the recovery latency reaches approximately 26 ms. Experimental results indicate an average increase of about 2.27 ms per additional threshold unit.

This trend can be explained by the computational complexity of Feldman Verifiable Secret Sharing. Increasing the threshold raises the degree of the polynomial, which directly increases the number of coefficients that must be committed. Each additional coefficient requires one extra modular exponentiation during commitment generation, while share verification involves evaluating additional commitment terms. Consequently, the overall computational cost grows approximately linearly with the threshold value.

Interestingly, the recovery phase consistently exhibits higher latency than the setup phase. This difference arises because reconstruction requires both share verification and Lagrange interpolation, whereas setup primarily consists of polynomial generation and commitment computation. Nevertheless, even at $t_h = 8$, the measured latency remains below 30 ms, indicating that stronger access requirements can be achieved without causing excessive performance degradation.

From a security perspective, increasing the threshold improves resistance against collusion and unauthorized recovery, but at the cost of additional computational overhead. Therefore, the threshold parameter represents a tradeoff between security and efficiency. The approximately linear growth observed in the experiment suggests that stronger security guarantees can be obtained in a predictable manner without causing exponential performance degradation.

Taken together, these results reveal that the proposed hierarchical structure scales favorably with respect to the number of participants while exhibiting a predictable dependence on the threshold parameter. Consequently, the protocol is more sensitive to the security level represented by the threshold than to the total size of the participant set, making it suitable for digital inheritance scenarios involving many beneficiaries with moderate reconstruction requirements.

B. Feldman VSS Overhead

To quantify the cost of introducing verifiability, experiments compare Feldman Verifiable Secret Sharing with plain Shamir Secret Sharing in terms of computational latency and public metadata storage. Different threshold values and security parameters are considered to evaluate the tradeoff between additional overhead and improved integrity protection.

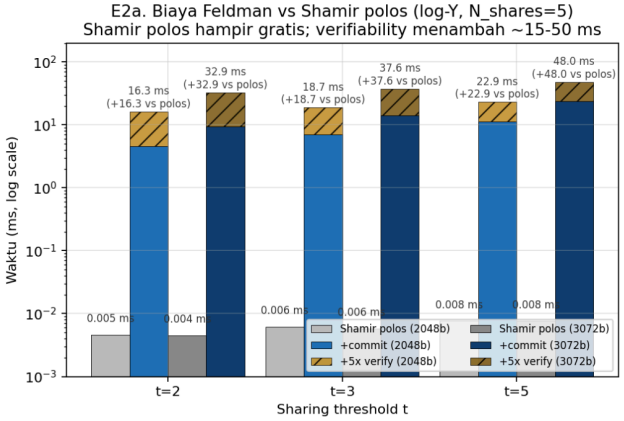

Fig. 14. Computational overhead of Feldman VSS compared with plain Shamir Secret Sharing

The results show that plain Shamir Secret Sharing incurs almost negligible computational cost, remaining below 0.01 ms across all threshold values. In contrast, the introduction of Feldman commitments increases the setup latency to approximately 16–48 ms, while the verification of five shares contributes an additional overhead of roughly 15–23 ms. This overhead originates primarily from modular exponentiation operations required to compute public commitments and to verify share consistency.

Furthermore, the increase in execution time is approximately proportional to the threshold value. Since a polynomial of degree $t - 1$contains t coefficients, increasing the threshold requires generating additional commitments $C_j = g^{aj} \bmod p$, thereby increasing the number of exponentiation operations. This explains why the 3072-bit group exhibits higher latency than the 2048-bit group, as larger moduli require more expensive arithmetic operations. Despite this increase, the total overhead remains below 50 ms, indicating that the cost of adding verifiability is moderate compared with the security benefits obtained.

From a practical perspective, these results reveal that the dominant cost of Feldman VSS originates not from Shamir secret sharing itself, but from the commitment and verification stages. Consequently, the protocol exchanges a small amount of additional latency for the ability to detect forged shares before reconstruction, thereby preventing silent failures that may occur in plain Shamir Secret Sharing.

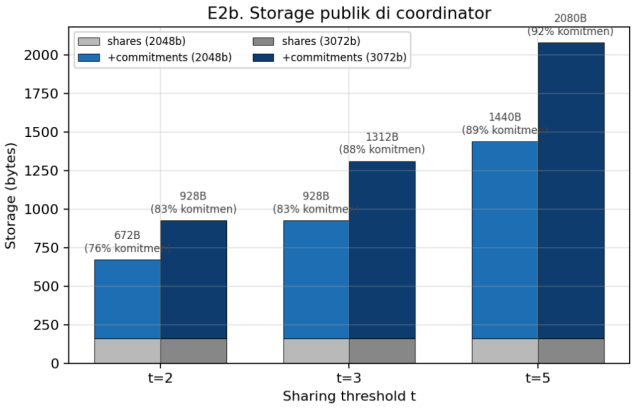

Fig. 15. Public metadata storage overhead introduced by Feldman commitments

The results indicate that the public metadata stored by the coordinator is dominated by Feldman commitments. Depending on the threshold and security parameter, commitments account for approximately 76–92% of the total storage requirement. For example, under the 3072-bit parameter and $t = 5$, the total storage reaches approximately 2080 bytes, of which more than 90% consists of commitment values.

This behavior can be explained by the fact that each additional threshold level introduces one more commitment element, whereas the size of individual shares remains constant. Consequently, storage growth is mainly determined by the number and size of commitment elements rather than

by the secret shares themselves. Larger security parameters also increase the size of each commitment, which explains the higher storage requirements observed for the 3072-bit group.

Although Feldman commitments dominate the relative storage overhead, the absolute storage requirement remains very small, amounting to only a few kilobytes even under the strongest security settings. In practical deployments, this overhead is negligible compared with the size of typical digital assets such as documents, photographs, or encrypted backups. Therefore, public verifiability can be achieved without imposing a significant storage burden on the coordinator.

Taken together, these results demonstrate that the price of verifiability is primarily paid in modular exponentiation and commitment storage. Nevertheless, the resulting overhead remains modest in absolute terms, making Feldman VSS a practical mechanism for enhancing the integrity and reliability of secret reconstruction.

C. *Adversarial Detection Capability*

To investigate the robustness of the proposed scheme against malicious participants, forged shares were intentionally injected into the reconstruction coalition and the behavior of Feldman Verifiable Secret Sharing was compared with that of plain Shamir Secret Sharing. The objective is to determine whether the verification mechanism can reliably detect manipulated shares before reconstruction. The number of manipulated shares, denoted by kkk, was varied from zero to four while maintaining a coalition size of $t = 5$ and $n = 8$.

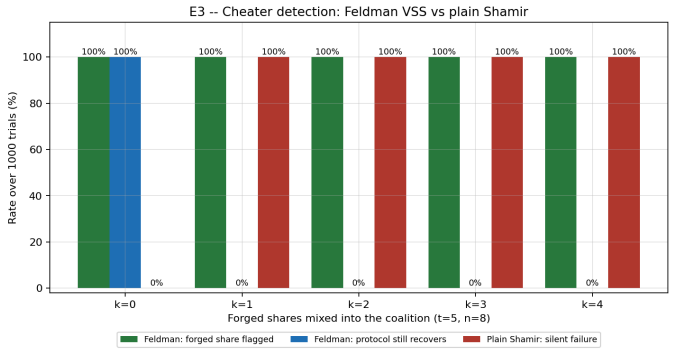

Fig. 16. Detection capability against forged shares

The results demonstrate that Feldman VSS consistently detects all manipulated shares with a detection rate of 100% across all scenarios. When no forged shares are present ($k = 0$), the protocol successfully reconstructs the secret, resulting in a 100% recovery rate. However, once one or more forged shares are introduced, the verification mechanism immediately identifies the inconsistency and rejects the reconstruction process. Consequently, invalid shares are prevented from influencing the recovered secret.

In contrast, plain Shamir Secret Sharing exhibits fundamentally different behavior. Because no share verification mechanism exists, forged shares cannot be distinguished from legitimate ones. As a result, reconstruction proceeds normally even when corrupted shares are present, leading to silent failures in which an incorrect secret is produced without any indication of tampering. The probability of such silent failures reaches 100% when at least one forged share is included in the coalition.

These results highlight an important difference between the two approaches. Plain Shamir Secret Sharing prioritizes confidentiality but provides no guarantees regarding share integrity, whereas Feldman VSS augments the scheme with authenticity and consistency checks through public commitments. Therefore, the additional computational overhead introduced by Feldman VSS directly translates into a substantial security advantage by transforming undetectable reconstruction errors into explicitly detectable events.

From a practical perspective, this property is particularly important in digital inheritance applications, where participants may accidentally submit corrupted shares or intentionally provide malicious inputs. By detecting manipulated shares before reconstruction, the proposed protocol avoids releasing incorrect secrets and preserves the integrity of the inheritance process. Consequently, the experiment demonstrates that verifiability is not merely an additional feature, but a fundamental requirement for achieving reliable secret recovery in adversarial environments.

D. *End-to-End Latency*

To assess the practical performance of the proposed protocol, end-to-end latency was measured under different vault sizes. The execution time was decomposed into individual phases in order to identify the dominant performance bottlenecks and to analyze how vault size affects the overall latency. The total execution time was decomposed into individual phases, including setup, share verification, secret reconstruction, and AES-256-GCM decryption. In addition, the impact of vault size on the overall latency was analyzed to identify the dominant performance bottlenecks.

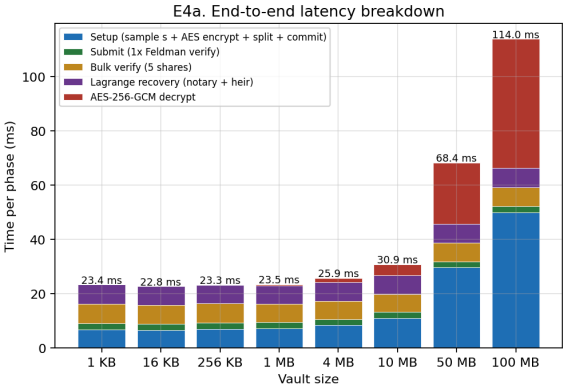

Fig. 17. End-to-end latency breakdown

The results indicate that the setup phase constitutes the largest constant component of the protocol, accounting for secret sampling, AES encryption, hierarchical splitting, and Feldman commitment generation. Share verification and Lagrange reconstruction contribute relatively small and nearly

constant overheads. For vault sizes ranging from 1 KB to 1 MB, the total latency remains almost unchanged at approximately 23 ms, demonstrating that the performance of the secret-sharing protocol is largely independent of the amount of protected data.

However, as the vault size increases, AES-256-GCM decryption gradually becomes the dominant component. At 50 MB, the total latency reaches approximately 68 ms, while at 100 MB it increases to 114 ms. This growth is primarily caused by the linear dependence of symmetric encryption and decryption on the amount of processed data. In contrast, the computational cost associated with secret sharing and Feldman verification remains nearly constant because these operations depend only on the threshold parameters rather than the vault size itself.

Therefore, the experimental results reveal that the bottleneck of the protocol shifts from secret-sharing operations to symmetric cryptographic processing when protecting large digital assets. For small and medium-sized vaults, the overhead introduced by the inheritance protocol itself dominates the total execution time, whereas for large vaults the data-dependent cost of AES-256-GCM becomes increasingly significant.

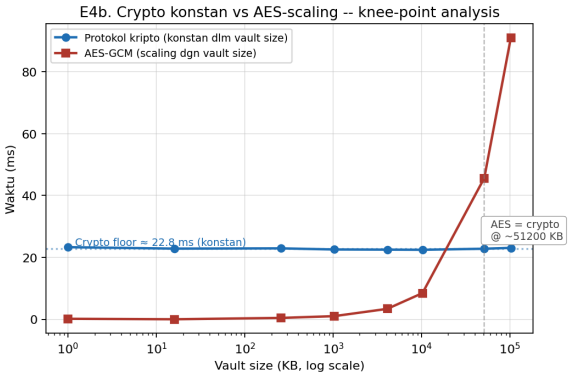

Fig. 18. Knee-point analysis of protocol latency

The second experiment separates the constant overhead introduced by the inheritance protocol from the latency associated with AES-256-GCM. The results show that the cryptographic protocol contributes a nearly constant baseline latency of approximately 22.8 ms regardless of vault size. This baseline consists mainly of secret sharing, Feldman commitments, verification, and reconstruction operations, all of which depend on the access structure rather than the size of the encrypted asset.

By contrast, the AES-256-GCM component scales approximately linearly with vault size. For small vaults, its contribution is negligible compared with the constant protocol overhead. However, as the vault size increases, the AES latency eventually intersects the protocol baseline, producing a knee point around 50 MB. Beyond this point, symmetric encryption and decryption dominate the overall execution time.

This observation has an important implication for system scalability. The performance limitation of the proposed protocol is determined not by Shamir Secret Sharing or Feldman VSS, but by the throughput of symmetric encryption. Consequently, the inheritance mechanism itself introduces a fixed and predictable latency, while the processing of large digital assets remains bounded primarily by AES performance. This behavior demonstrates that the proposed protocol scales favorably and that its cryptographic overhead remains modest even for large vault sizes.

VI. Conclusion

This paper presented the design and implementation of a trustless digital inheritance protocol based on Hierarchical Verifiable Secret Sharing. The proposed approach combines a two-layer Shamir Secret Sharing structure with Feldman Verifiable Secret Sharing to enforce a conjunctive access policy requiring cooperation between notaries and heirs while simultaneously providing protection against forged shares. In addition, a state-machine mechanism and audit logging facilities are incorporated to support controlled inheritance claims and improve accountability throughout the inheritance lifecycle.

Experimental results demonstrate that the proposed hierarchical access structure scales efficiently with respect to the number of participants and exhibits approximately linear growth with the reconstruction threshold. Although Feldman VSS introduces additional computational and storage overhead, the cost remains moderate and provides substantial security benefits by enabling share verification before reconstruction. Adversarial experiments further show that manipulated shares are detected with a 100% success rate, whereas plain Shamir Secret Sharing suffers from silent failures due to the absence of integrity verification. End-to-end latency measurements indicate that the inheritance protocol contributes a nearly constant cryptographic overhead, while the dominant bottleneck for large vault sizes originates from AES-256-GCM rather than from the secret-sharing mechanism itself.

Overall, the results indicate that Hierarchical Verifiable Secret Sharing provides a practical and secure foundation for digital inheritance systems by eliminating single points of failure, supporting controlled asset recovery, and maintaining modest performance overhead. The proposed protocol demonstrates that confidentiality, integrity, availability, and controlled access can be achieved simultaneously without relying on a fully trusted third party, making the approach suitable for real-world digital inheritance scenarios involving multiple stakeholders.

Future work may investigate decentralized coordinators, distributed or blockchain-based audit logs, proactive secret sharing, and stronger authentication mechanisms to further enhance resilience and trustworthiness. In addition, integrating the protocol with real-world digital asset platforms and conducting evaluations under larger-scale deployments would

provide valuable insights into its practicality and long-term applicability.

VII. Appendix

Source Code

https://github.com/StefanyJosefina/TrustWill

Video Link at Youtube

https://youtu.be/OKX5BWG-vZA

Acknowledgment

The author expresses sincere gratitude to all parties who have contributed to the completion of this paper, especially:

1. The Almighty God, for His blessings, guidance, and grace throughout the process of conducting this work.
2. The author's parents and family, for their continuous moral support, encouragement, and prayers.
3. Friends and colleagues who have provided support, discussions, and motivation during the completion of this paper.
4. Dr. Ir. Rinaldi Munir, M.T., as the lecturer of the II4021 Cryptography course at Institut Teknologi Bandung, for his valuable lectures, guidance, and insights that greatly contributed to the development of the ideas and implementation presented in this paper.

The author deeply appreciates all the assistance, encouragement, and kindness received from these individuals and groups, without which the completion of this paper would not have been possible.

References

[1] Hukumonline, "Kendala Bagi Waris Akun/Aset Digital saat Pemilik Wafat," Mar. 16, 2026. [Online]. Available: https://www.hukumonline.com/stories/article/lt69b89e7dae650/kendala-bagi-waris-akun-aset-digital-saat-pemilik-wafat/

[2] Mahkamah Agung Republik Indonesia, "Sengketa Warisan Digital: Apakah Akun Medsos Bisa Diwariskan?," Jul. 29, 2025. [Online]. Available: https://marinews.mahkamahagung.go.id/artikel/sengketa-warisan-digital-apakah-akun-medsos-bisa-diwariskan-0rA

[3] Mahkamah Konstitusi Republik Indonesia, "Pewarisan Aset Digital Sulit Diterapkan," Jun. 26, 2025. [Online]. Available: https://www.mkri.id/berita/pewarisan-aset-digital-sulit-diterapkan-23428

[4] N. Kaylani, "Implikasi Harta Warisan Aset Digital Pendapatan Konten TikTok dalam Perspektif Hukum Perdata Indonesia," Undergraduate Thesis, Universitas Gadjah Mada, 2025. [Online]. Available: https://etd.repository.ugm.ac.id/penelitian/detail/253507

[5] A. Beimel, "Secret-Sharing Schemes: A Survey," in Coding and Cryptology, Y. Chee, Ed. Berlin, Germany: Springer, 2011, pp. 11–46. [Online]. Available: https://www.cs.bgu.ac.il/~beimel/Papers/Survey.pdf

[6] I. Alam, A. S. Alali, S. Ali, and M. S. M. Asri, "A Verifiable Multi-Secret Sharing Scheme for Hierarchical Access Structure," Axioms, vol. 13, no. 8, Art. no. 515, Aug. 2024. [Online]. Available: https://www.mdpi.com/2075-1680/13/8/515

[7] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, Nov. 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

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

Stefany Josefina Santono 18223116