Key management mechanism with non-stored and user authentication-based key recovery*

Su Jin Shin[1], Siwan Noh[2], Kyung-Hyune Rhee[1], and Sang Uk Shin[1†]

[1] Pukyong National University, Republic of Korea
inuin9014@pukyong.ac.kr, {khrhee, shinsu}@pknu.ac.kr
[2] Dong-Eui University, Republic of Korea
nsw@deu.ac.kr

Abstract

Existing key management mechanisms stored the user's biometric information used for key generation or the generated private keys in external storage or within the device, leading to issues such as private key manipulation and extraction. To address these issues, this paper proposes a secure and reliable integrated key management framework. The proposed framework does not store the user's biometric information or the generated keys. Furthermore, to prevent unauthorized users from attempting attacks to recover key, the framework is designed to perform key recovery only after a smart contract-based user authentication process is completed. Finally, unlike previous studies that individually researched protocols for specific stages, the paper designs an integrated key management mechanism that encompasses protocols for the main stages of the key lifecycle, including key generation, regeneration, backup, and recovery.

1 Introduction

Secure management and operation of digital assets require careful private key management. Blockchain wallets can be categorized into custodial and non-custodial wallets. Custodial wallets entrust private key management to a trusted third party, presenting issues such as a single point of failure and concerns about private key security. Non-custodial wallets are managed directly by the user, posing a significant risk of private key loss due to user mistake. To address these issues, many researchers have explored managing private keys by splitting them into multiple fragments and encrypting them. This approach makes private key reconstruction difficult when fewer than the threshold number of fragments are collected. However, it also introduces the problem that malicious attackers who colluded beforehand and collected more than the threshold number of fragments could leak and manipulate the private key. To address this, key generation methods utilizing the user's biometric information, which provides uniqueness and immutability, has been researched [5]. Most studies on biometric-based key generation store the user's biometric information for convenience. However, this approach raises issues such as privacy protection and confidentiality of sensitive user data. Furthermore, existing studies have primarily addressed key management solely from the perspective of a specific protocol phase, such as key generation or backup. Key management encompasses the entire process, including not only key generation but also key backup, recovery, and reconstruction. Therefore, integrating individual protocols, each focused on a specific stage, into a single key management framework may lead to unexpected issues or incur significant costs in aspects like computational complexity.

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 33, December 16-18, 2025, Sapporo, Japan. © The copyright of this paper remains with the author(s).

†Corresponding Author

This paper proposes a key management mechanism encompassing the entire process from key generation to recovery. It performs key generation is based on the user's biometric information. At this time, the user's biometric information and the generated private key are neither stored on the device nor in external storage. As a mechanism based on a blockchain environment, key backup and recovery are performed through interactions with computation nodes randomly selected by the user. Furthermore, key recovery is performed only when a user's key loss event occurs, and a user authentication process is preceded before the recovery process is performed. The proposed mechanism ensures the following contributions:

- Non-stored: The proposal mechanism must not store the generated private key and the user's biometric information on the device or any external storage device.
- Event-based key recovery at an unspecified point in time: In the proposed mechanism, key recovery only proceeds when the user triggers a key loss event. At this point, the user authentication process utilizing the user's biometric information and a homomorphic commitment technique must be completed in advance via the smart contract.
- Decentralized key backup and recovery: The proposed mechanism requires users to randomly select computational nodes to participate in the process before performing key backup and recovery. Key backup and recovery must proceed through the execution of a distributed cryptographic protocol between the selected computation node and the user.

1.1 Related work

Cai et al. [5] proposed a biometric-based private key management framework, which generates key pairs using the LWE (Learning with Errors) algorithm and then generates a biometric key based on the user's biometric information. This biometric key is used to encrypt the user's private key. The encrypted private key is split using a CRT (Chinese Remainder Theorem)-based secret sharing technique and stored on the blockchain. Key recovery involves obtaining the user's new biometric information and using it to recovery the session key. The recovered session key is then used to decrypt the encrypted shares, which are recombined to recovery the private key. Daudén-Esmel et al. [8] proposed a fully decentralized multi-platform wallet based on blockchain and IPFS (InterPlanetary File System) technology. In Daudén-Esmel et al. [8], users receive a personal password and 24 mnemonic words. Additionally, Daudén-Esmel et al. [8] generates a session key to encrypt the user's key pair. This session key is encrypted using the root public key. All encrypted values are stored on IPFS. Key recovery requires the user's personal password and 24 mnemonic words. Chase et al. [6] proposed a custodial wallet-based key management framework that provides audit functions based on distributed trust. During the registration phase in Chase et al. [6], the user selects a one-time cryptographic key and encrypts their own key with it. The encrypted key is then split, and the split pieces are distributed to the guardians selected by the user. Guardians encrypt the received value using their own public key and stores it on the server. Key recovery proceeds after user authentication. Key recovery is performed by guardians receiving the encrypted value from the server and decrypting it. Wang et al. [13] proposed a key generation mechanism based on PUF (Physically Unclonable Function) and user biometric information. Wang et al. [13] generates private keys based on user biometric information using PUF to ensure user anonymity. During key generation, helper data is created, which is encrypted as a secret and stored in a database. When recovering the key, the secret is recovered using the polynomial $f(x)$ and Lagrange interpolation. Then, the recovered secret is used to decrypt the encrypted helper data. Subsequently, a fuzzy extractor is employed to recover the private key.

The proposed framework does not store keys or user biometric information to prevent issues such as key leakage or tampering by malicious attackers and protecting user privacy. It also performs key backup and recovery through a distributed encryption and decryption protocol. Based on these characteristics, this paper proposes a key management framework that provides greater security than existing research.

2 Main cryptographic techniques

The framework proposed in this paper applies fuzzy extractors, DPRFs (Distributed Pseudo-Random Functions), commitments, and zero-knowledge proofs to achieve secure and reliable key management. The proposed framework utilizes the following cryptographic primitives:

- A cryptographically secure hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ (where λ is the security parameter) [2].
- An AEAD (Authenticated Encryption with Associated Data) technique with IND-CCA and INT-CTXT security, composed of $(AEADEnc, AEADDec)$ [4].
- Semantically secure asymmetric cryptosystem against chosen ciphertext attacks composed of $(AsymKGen, AsymEnc, AsymDec)$ [12].
- A secure digital signature scheme providing existential unforgeability under chosen message attacks (EUF-CMA), composed of polynomial-time algorithms $(SignKGen, SignVerify)$ [7].

2.1 Fuzzy extractor

The proposed framework employs a fuzzy extractor to ensure fault tolerance, enabling key recovery even when input data contains noise [13]. The fuzzy extractor consists of the following algorithm:

- $FE.Gen(W) = (R, P)$: This is a generation algorithm that takes data W as input and generates an l-bit random key R and public helper data P.
- $FE.Rec(W', P) = R$: This is a recovery algorithm that recovers a random key R from the corresponding public helper data P and an input W' containing some noise.

2.2 DPRF (Distributed Pseudo-Random Function)

The proposed framework employs DPRF, a cryptographic technique where each participant collaboratively computes a PRF (Pseudo-Random Function) [1]. This technique provides pseudo-randomness, in which the results are not disclosed to all participants except the evaluator, even if participants below the threshold t are compromised. It also ensures consistency, guaranteeing that the evaluation is independent of the participants. The technique consists of the following algorithms:

- $Setup(1^k, n, t) \rightarrow ((sk_1, ..., sk_n), pp)$: This algorithm takes as input the security parameter 1^n, the number of participants n, and the threshold t, and generates n secret keys $sk_1, ..., sk_n$ and a public parameter pp.

- $Eval(sk_i, x, pp) \rightarrow z_i$: This is an algorithm that generates pseudo-random shares for a given value. Participant i computes z_i, the i-th share of x, using sk_i, x, and pp as inputs.
- $Combine((i, z_i)_{i \in S}, pp) =: z / \perp$: This is an algorithm that generates z by combining the shares $\{z_i\}_{i \in S}$ from the participants of the set S.

2.3 Commitment scheme

The proposed framework employs this technique for integrity verification and user authentication. This cryptographic technique allows users to commit to their confidential data while hiding it from others, and later disclose the committed value. It guarantees the properties of hiding and binding [14].

The user authentication process in the proposed framework applies a Pedersen commitment scheme that provides homomorphic properties [11]. This technique enables arithmetic operations to be performed on committed messages without revealing the original message. Furthermore, this technique is defined over a multiplicative cyclic group $\mathbb{G}$ with a large prime order $p \in \mathbb{Z}$, where g and h are generators belonging to $\mathbb{G}$. Here, $\mathbb{G}, p, g, h$ are public parameters in the system. The prover computes the commitment value for a message $f \in \mathbb{Z}_p$ as follows:

$$Comm = Comm(f, r) = g^f h^r \ mod \ p$$

Here, r is a randomly selected blinding element from $\mathbb{Z}_p$, and $Comm(\cdot)$ denotes the Pedersen commitment operation using the public parameters, generators g and h. Assume that commitment values $Comm^{(0)}$ and $Comm^{(1)}$ for each message $f^{(0)}$ and $f^{(1)}$ were generated based on the same generators g and h. The Pedersen commitment scheme provides the following homomorphic properties for addition, subtraction, and multiplication operations on the commitment values of the original messages $f^{(0)}$ and $f^{(1)}$ (for details on these operations, refer to [11]):

- Addition: In the Pedersen commitment scheme, the commitment for $f^{(0)}+f^{(1)}$ is computed as follows.

  $$Comm(f^{(0)}, r^{(0)}) \oplus Comm(f^{(1)}, r^{(1)}) = Comm(f^{(0)} + f^{(1)}, r^{(0)} + r^{(1)})$$

  Here, $\oplus$ denotes addition, and the actual operation involves multiplying the two commitments.

- Subtraction: In the Pedersen commitment scheme, the commitment for $f^{(0)} - f^{(1)}$ is computed as follows

  $$Comm(f^{(0)}, r^{(0)}) \ominus Comm(f^{(1)}, r^{(1)}) = Comm(f^{(0)} - f^{(1)}, r^{(0)} - r^{(1)})$$

  Here, $\ominus$ denotes subtraction, and the actual operation multiplies the inverse element of $Comm^{(1)}$.

- Multiplication: In the Pedersen commitment scheme, the computation is performed as follows via an interactive protocol.

  $$Comm^{(0)} \otimes Comm^{(1)} \rightarrow Comm^{(0,1)}$$

  Here, $\otimes$ denotes the multiplicative homomorphic operator, and $Comm^{(0,1)}$ is computed as follows.

  $$Comm(f^{(0)} f^{(1)}, r^{(0,1)}) = g^{f^{(0)} f^{(1)}} \cdot h^{r^{(0,1)}} \ mod \ p$$

  At this point, $r^{(0)}$ and $r^{(1)}$ are blinding elements selected from $\mathbb{Z}_p$, and $r^{(0,1)}$ is computed as $r^{(0)} \otimes r^{(1)}$.

In a blockchain-based proposal framework, incorporating interactive protocol-based multiplication operations into the user authentication process is somewhat challenging. Therefore, this paper applies the non-interactive Pedersen commitment scheme based on the Fiat-Shamir heuristic technique as proposed in [11].

2.4 Zero-knowledge proof

This technique enables proving the truth of a claim without disclosing additional information about that claim. Specifically, the prover wishes to demonstrate to the verifier only that they possess certain secret data, without revealing the secret data itself. This is used in the proposed framework when generating a user's authentication proof value during the user authentication process. The technique must satisfy Completeness, Soundness, Zero-knowledge, Succinctness, Proof of knowledge (POK) or Argument of knowledge (AOK) [3]. Furthermore, it consists of the following three algorithms:

- $GenKey(1^{\lambda}, C) \rightarrow (pk_z, vk_z)$: This takes the security parameter λ and the arithmetic circuit C as input to generate the proof key pk_z and the verification key vk_z.
- $GenProof(pk_z, x, w) \rightarrow \pi$: This algorithm takes pk_z, the public input data x, and the witness w as input to generate the proof π. The prover executes this algorithm.
- $VerProof(vk_z, x, \pi) \rightarrow 0/1$: This algorithm takes vk_z, x, and π as input and determines the validity of the proof. The verifier executes this algorithm.

The proposed framework employs the Groth16 [9] zk-SNARK algorithm among various zero-knowledge proof schemes.

3 The Proposed framework

The proposed framework in this paper is designed based on a consortium blockchain environment and consists of user U, user device Dev, and a set of computation nodes S. Here, S can consist of entities such as wallet platforms, cryptocurrency exchanges, and other user-owned devices excluding Dev. Before performing the private key backup and recovery process, U selects computation nodes cn from S to execute each process's computations.

3.1 Assumptions

This paper presents a key management framework based on blockchain platform supporting smart contracts. Smart contracts exhibit stateful ideal functionality, meaning they can expose their entire internal state to all participants, including attackers [10]. Therefore, the proposed framework inherits the security properties—such as integrity and tamper-resistance—of the underlying blockchain platform and smart contracts. Furthermore, in the proposed framework, Dev is assumed to be a secure trust entity. It is assumed that helper data generated during the key generation process and user-defined random values are securely stored within a wallet installed on Dev. Here, the wallet is assumed to be developed and distributed by a wallet provider, such as a cryptocurrency exchange, and is assumed to be secure. The security of the wallet itself is beyond the scope of this paper and is not discussed in detail.

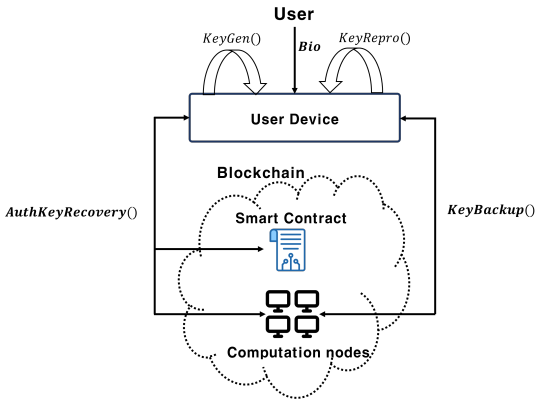

Figure 1: The overview of proposed framework

3.2 Detailed protocols

The proposed framework in this paper is a key management framework that includes protocols corresponding to the entire key lifecycle, from key generation to key recovery. The key recovery process in this framework assumes that it is performed when one of the multiple private keys owned by U is lost. If U loses its sole private key, it is assumed that U generates a new private key temporarily and requests the recovery of the lost private key. The proposed framework consists of key generation and backup, key regeneration, and key recovery protocols. The key recovery protocol proceeds only when a transaction containing U's key recovery request event occurs. Additionally, a user authentication process is performed before initiating key recovery. [Figure 1] shows an overview of the proposed framework.

In the key generation and backup protocol, U provides their biometric information Bio, and based on this, U's private key sk is generated. The helper data HD and the user-defined random values σ, γ, ρ generated during this process are securely stored within the wallet. Subsequently, the key backup protocol proceeds to prepare for key loss scenarios. Before key backup occurs, U selects a computation node set S_B from the entire node set S to perform the backup process. U interacts with S_B based on a distributed encryption and decryption protocol, encrypting HD and σ and publishing them to the blockchain. The key regeneration protocol is executed when U processes transactions for their digital assets. This protocol regenerates sk' based on U's biometric information Bio and the HD and σ securely stored within the wallet. It verifies whether the regenerated sk matches the sk' generated during the key generation protocol phase by using the commit value for the regenerated sk'. The key recovery protocol is initiated when U triggers a transaction on the blockchain containing an event indicating that they have lost their private key. At this stage, a smart contract verifies whether the U who generated the private key is the same entity as the U requesting recovery. Key recovery only proceeds if this user authentication passes. Upon passing verification, U selects a set of computation nodes S_R from S to compute the recovery process prior to key recovery. U then recovers their sk based on a distributed encryption and decryption protocol with S_R.

3.2.1 Key generation protocol

The key generation protocol begins when U provides Dev with their biometric information Bio. Dev generates σ using a PRNG and creates the feature vector $V^{(0)}$ using the pair-minutiae

feature scheme on the provided Bio. Here, $V^{(0)}$ consists of $V_1^{(0)}, ..., V_N^{(0)}$, where N is the length of the feature vector. Next, Dev uses a Pedersen commitment scheme to generate commit vectors $Comm^{(0)}$ and $Comm^{(0,0)}$ for $V^{(0)}$ (refer to [11] for the detailed computation process). Dev inputs σ and $V^{(0)}$ into the feature fusion module to generate feature templates F. It then uses the $FE.Gen()$ algorithm of the fuzzy extractor to generate the private key seed sk_{seed} and helper data HD. Finally, Dev generates sk using a cryptographically secure hash function.

In this process, Dev defines $meta_{key}$ as a combination of the identifier of sk, denoted as sk_{ID}, the time of key generation T_{sk}, and the description of the asset associated with sk, denoted as key_{des}. Algorithm 1 shows the key generation process.

Algorithm 1 Key generation

The input data is U's Bio, γ and the Dev processes the following:

1: $\sigma \leftarrow PRNG()$
2: Extracting minutiae feature from $Bio \rightarrow$ minutiae feature point set M
3: Generates feature vector $V^{(0)}$ based on M
4: $Comm^{(0)} \leftarrow [Comm(V_1^{(0)}, r_1^{(0)}), ..., Comm(V_N^{(0)}, r_N^{(0)})]$
5: $Comm^{(0,0)} \leftarrow [Comm(V_1^{(0)}V_1^{(0)}, r_1^{(0,0)}), ..., Comm(V_N^{(0)}V_N^{(0)}, r_N^{(0,0)})]$
6: $F \leftarrow \sigma \oplus V^{(0)}$
7: $(sk_{seed}, HD) \leftarrow FE.Gen(F)$
8: $sk \leftarrow H(sk_{seed}, \sigma)$
9: $sk_{ID} \leftarrow H(\sigma, HD, T_{sk})$
10: $Com\_sk \leftarrow Comm(sk, \gamma)$

3.2.2 Key backup protocol

Before performing the key backup, U randomly selects computation nodes cn_i, $i = 1, ..., l$, from S to handle the computation for the key backup process. Here, l is the number of selected cn_i. The key backup process proceeds based on the interaction between U and cn_i.

U generates α, the commitment value of HD and σ, and publishes it to the blockchain network within the transaction $\{Back_\alpha\}$. cn_i obtains α through $\{Back_\alpha\}$ published on the blockchain. Then, cn_i uses α as input data for the DPRF's $Eval()$ algorithm to generate the share z_i to be used for encryption and decryption. cn_i encrypts z_i to generate Ez_i, places it in transaction $\{Back_{Ez_i}Tx\}$, and publishes it to the blockchain network. U decrypts Ez_i contained in $\{Back_{Ez_i}Tx\}$ to obtain z_i. Then, the DPRF's Combine() algorithm generates a reconstruction value w with a threshold number or more of z_i as input data. U generates the session key tk based on w, encrypts HD and σ using tk to produce e. Finally, U publishes the transaction $\{Back_eTx\}$ containing e to the blockchain network. Algorithm 2 shows the operations in key backup.

Algorithm 2 Key backup

// **Performed by** U
$\alpha \leftarrow Comm((HD, \sigma), \rho)$
Sends $\{Back_\alpha\}$ to Blockchain

// **Performed by** $cn_i, i \in \{1, ..., l\}$
$z_i \leftarrow Eval((sk_i, u||\alpha), pp)$
$Ez_i \leftarrow AsymEnc(PK_i, z_i)$
Sends $\{Back_{Ez_i}Tx\}$ to Blockchain

// **Performed by** U
$z_i \leftarrow AsymDec(SK_u, Ez_i)$
$w/ \perp \leftarrow Combine(\{(i, z_i)\}_{i \in |l|}, pp)$
$tk \leftarrow H(w)$
$e \leftarrow AEADEnc(tk, ((HD, \sigma)||\rho||\gamma)$
Sends $\{Back_eTx\}$ to Blockchain

3.2.3 Key regeneration protocol

U provides Bio' to Dev, and Dev derives the feature vector $V^{(2)}$ from Bio'. Here, Bio' refers to newly entered biometric information that may contain slightly different information from the Bio provided by U during the key generation phase. Dev obtains σ and HD securely stored in the wallet. Then, Dev generates F'' using the feature fusion module with σ and $V^{(2)}$. Afterwards, Dev recovery sk'_{seed} using the fuzzy extractor's $FE.Rec()$. Dev generates sk' based on the recovered sk'_{seed}. To verify that sk' and sk are identical, Dev generates the commit value Com_sk'' for sk'. Dev then compares this Com_sk'' with the Com_sk stored in $\{Back_\alpha\}$. If they match, Dev determines that sk' was regenerated correctly and uses it for transaction processing. Algorithm 3 shows the key regeneration process.

Algorithm 3 Key regeneration

The input data is U's Bio' and the Dev processes the following:
Generates feature vector $V^{(2)}$ from Bio'
$F'' \leftarrow \sigma \oplus V^{(2)}$
$sk'_{seed} \leftarrow FE.Rec(F'', HD)$
$sk' \leftarrow H(sk'_{seed}, \sigma)$
$Com\_sk'' \leftarrow Comm(sk', \gamma)$
Compare Com_sk and Com_sk''

3.2.4 User authentication and key recovery protocol

U publishes a transaction $\{lostTx\}$ to the blockchain network. This transaction includes $meta_{key}$—information about the private key to be recovered—the recovery request message $'recovery'$, and a function call to a smart contract to perform the user authentication process. This triggers the smart contract to initiate a user authentication process, verifying that the user requesting recovery is actually the owner of the corresponding key.

U provides Bio'' to Dev, and Dev derives $V^{(1)}$ based on the received Bio''. Then, Dev generates the Pedersen commitment vector $Comm^{(1)}$ using the technique described in Chapter

2 on $V^{(1)}$. Dev generates commit vectors $Comm^{(1,1)}$ and $Comm^{(0,1)}$ to compute the Euclidean distance d between $V^{(0)}$ and $V^{(1)}$ based on the homomorphic encryption property (see [11] for detailed computation). U generates a challenge value *chal* based on the values generated by Dev. U generates an auxiliary proof element *sub_aut* to prove the correctness of the homomorphic encryption based on *chal* (see [11] for detailed computation step). U computes the commit value c_d for d and the blinding factor r_d between biometric data using the homomorphic encryption property of the Pedersen commitment scheme. U generates a zero-knowledge proof π for c_d and creates an authentication proof Γ containing all values generated during this process. Finally, U distributes the transaction $\{UserVerifyTx\}$ to the blockchain network, which includes Γ and the user verification function call. Algorithm 4 shows the operations in user authentication.

Algorithm 4 User authentication

The input data is U's Bio'' and the Dev, U processes the following:

// **Performed by** Dev
Generates feature vector $V^{(1)}$ from Bio''
Generates $Comm^{(1)}$, $Comm^{(1,1)}$, $Comm^{(0,1)}$

// **Performed by** U
$chal \leftarrow H(Comm^{(0)}||Comm^{(1)}||Comm^{(0,0)}||Comm^{(1,1)}||Comm^{(0,1)}||U_{ID}||nonce)$
$r_d \leftarrow \sum_{i=1}^{N}(r_i^{(0,0)} + r_i^{(1,1)} - 2r_i^{(0,1)})$
$c_d \leftarrow \sum_{i=1}^{N}(Comm_i^{(0,0)} \oplus Comm_i^{(1,1)} \ominus 2Comm_i^{(0,1)})$
$\pi \leftarrow GenProof(pk_z, r_d, c_d, \epsilon)$
$\Gamma \leftarrow (id, nonce, Comm^{(1)}, Comm^{(0,0)}, Comm^{(1,1)}, Comm^{(0,1)}, sub\_aut, \pi)$

The smart contract executes the user verification function called by the user. First, the smart contract checks the *nonce* contained in Γ generated by U, verifying that it is greater than the value used in the user's last authentication process. If the *nonce* is valid, the smart contract generates $chal'$ using the values contained in Γ. Next, the smart contract verifies whether $Comm^{(0,0)}$, $Comm^{(1,1)}$, and $Comm^{(0,1)}$ are correct values based on *sub_aut*. Passing verification means the Pedersen commitment for $V^{(1)}$ was computed correctly. If verification fails, user authentication fails. If verification succeeds, the smart contract generates a Pedersen commit c_d' for d based on $Comm^{(0,0)}$, $Comm^{(1,1)}$, $Comm^{(0,1)}$. The smart contract performs a biometric similarity verification using the $VerProof()$ algorithm on c_d' and π contained in Γ. If the verification passes, it means the recently acquired biometric data is sufficiently similar to the biometric data acquired during key generation. That is, user authentication has succeeded. If the verification fails, it means the two biometric data sets do not match, so the key recovery process does not proceed. Algorithm 5 shows the user verification process.

Algorithm 5 User verification

The smart contract processes the following:

1: Compare *nonce*
2: $chal' \leftarrow H(Comm^{(0)}||Comm^{(1)}||Comm^{(0,0)}||Comm^{(1,1)}||Comm^{(0,1)}||U_{ID}||nonce)$
3: Verify commit values and sub_auth
4: $c'_d \leftarrow \sum_{i=1}^{N}(Comm_i^{(0,0)} \oplus Comm_i^{(1,1)} \ominus 2Comm_i^{(0,1)})$
5: Verify the proof π by $VerProof(vk_z, c'_d, \epsilon, \pi)$

If the verification passes, the smart contract publishes a transaction $\{ResultTx\}$ to the blockchain network, indicating that user authentication was successful.

U verifies this and randomly selects computation nodes cn_i, $i = 1, ..., l'$ from S to perform the key recovery process computation. Here, l' is the number of computation nodes randomly selected for the key recovery process. After confirming that $\{ResultTx\}$ has been distributed, cn_i verifies whether the information contained in $\{Back_eTx\}$, $\{lostTx\}$, and $\{UserVerifyTx\}$ matches. If all information matches, cn_i acquires the e contained in $\{Back_eTx\}$ and generates the share z_i using the DPRF $Eval()$ algorithm. Then, cn_i encrypts z_i to obtain Ez_i and includes it in the transaction $\{Rec_E\}$, publishing it to the blockchain network.

U decrypts Ez_i contained in $\{Rec_E\}$ to obtain z_i, then generates w using the $Combine()$ algorithm of the DPRF. Next, U generates tk using a secure hash function. Using the generated tk, U performs the AEAD algorithm to decrypt e. U generates a commitment value α' for the decrypted $((HD, \sigma)||\rho)$. Finally, U verifies whether α' matches an α contained in $\{Back_\alpha\}$. If the two values do not match, the entire process is repeated from the beginning. If they match, U provides Bio'' to Dev, and Dev extracts $V^{(3)}$ based on this. Dev generates F' using the feature fusion module, taking the decrypted σ and $V^{(3)}$ as inputs. Next, Dev generates sk_{seed} via the fuzzy extractor's $FE.Rec()$. Dev generates sk based on a cryptographically secure hash function and creates the corresponding commit value Com_sk'. Finally, Dev verifies whether the generated Com_sk' matches the Com_sk contained in $\{Back_\alpha\}$. If they match, it is determined that the key backup was performed correctly. Algorithm 6 shows the operations in key recovery.

Algorithm 6 Key recovery

// **Performed by** $cn_i, i \in \{1, ..., l'\}$
Get e through $\{Back_eTx\}$
$z_i \leftarrow Eval(sk_i, u||\alpha, pp)$
$Ez_i \leftarrow AsymEnc(PK_u, z_i)$
Sends $\{Back_{Ez_i}Tx\}$ to U

// **Performed by** U
$z_i \leftarrow AsymDec(SK_u, Ez_i)$
$w/ \perp \leftarrow Combine(\{(i, z_i)\}_{i \in |l'|}, pp)$
$tk \leftarrow H(w)$
$((HD, \sigma)||\rho||\gamma) \leftarrow AEADDec(tk, e)$
$\alpha' \leftarrow Comm((HD, \sigma), \rho)$
Compare α and α'

// **Performed by** Dev
if $\alpha == \alpha'$ **then**
    Get Bio'' through U
    Generates feature vector V from Bio''
    $F' \leftarrow \sigma \oplus V$
    $sk_{seed} \leftarrow FE.Rec(F', HD)$
    $sk \leftarrow H(sk_{seed}, \sigma)$
    $Com\_sk' \leftarrow Comm(sk, \gamma)$
    Compare Com_sk and Com_sk'
end if

4 Analysis

4.1 Comparative analysis with existing research

This section compares and analyzes relevant research and proposed frameworks based on the requirements mentioned in the introduction. Table 1 shows the comparative analysis with existing research.

Table 1: Comparative analysis with existing research

	Confidentiality	**Integrity**	**Non-stored**	**Resistance to collusion attack**	**User authentication**	**Event-based key recovery**
Cai et al. [5]	O	O	X	△	X	X
Daudén-Esmel et al. [8]	O	O	X	X	X	X
Chase et al. [6]	O	O	X	△	△	X
Wang et al. [13]	O	O	O	△	X	X
Proposed framework	O	O	O	O	O	O

O: satisfied, *X*: not satisfied, △: partially satisfied.

Daudén-Esmel et al. [8] generate a session key based on an initial seed value, then encrypt the user's key pair using this session key and store it on IPFS. This approach may lead to problems such as theft or manipulation by malicious attackers. Wang et al. [13] do not store the user's biometric information or private key, but it uses a PUF during key generation. That is, it has high device dependency, meaning key recovery is difficult if the user loses their device. Unlike these existing studies, the proposed framework does not store either the key material or the user's biometric information in any internal or external storage on the device. Through this approach, the proposed framework ensures the confidentiality of the key.

Chase et al. [6] contains a component called a guardian that holds a share of the encrypted key. While their paper provides security against collusion attacks by guardians below the threshold, it does not provide security against the worst-case scenario where guardians above the threshold attempt a collusion attack. Similarly, Cai et al. [5] also does not provide security against nodes above the threshold performing a collusion attack. Wang et al. [13] guarantee security against polynomial attacks, but in the worst-case scenario where a malicious attacker succeeds in a brute-force attack or acquires more than the threshold number of (x, y) pairs, there remains a possibility that the secret s could be obtained. That is, existing studies provide only partial resistance to collusion attacks. While the probability is low, they cannot guarantee security in the worst-case scenario where more than the threshold number of nodes attempt a collusion attack. The proposed framework ensures resistance to collusion attacks by randomly selecting computation nodes that actually perform the key backup or recovery computation. In the worst-case scenario, a number of nodes above the threshold among the randomly selected computation nodes could carry out a collusion attack. However, without the user's biometric information, malicious nodes cannot infer the private key based solely on the helper data and random values they obtain.

Previous studies have been configured to enable key recovery at any arbitrary point in time. Therefore, this means an attacker could attempt attacks by performing key backup and recovery at any time they choose. To counter such attempts, applying research such as [17] and [18], which enable decryption at a specific future point in time, to key recovery would allow configuration for decryption at a specific point in time, such as one associated with a timestamp or a chain. However, since each user's blockchain environment, chain generation speed, and timing differ, decryption may not proceed normally at the user's desired time. Furthermore, using this approach means key recovery occurs at a specific time pre-set by the user, not when the user's private key is actually lost. Finally, most existing studies do not consider the timing of key recovery. In such cases, a malicious attacker could indiscriminately request and attempt to recovery the user's private key. Therefore, the proposed framework is designed such that key recovery only proceeds when the user generates an event regarding their private key loss in the form of a transaction and calls the key recovery smart contract. Since a malicious attacker could impersonate a legitimate user to request key recovery, the proposed framework requires prior user authentication. User authentication in the proposed framework is based on a homomorphic commitment value for the user's non-stored biometric information.

Finally, previous studies have focused only on specific protocols that correspond to certain stages of the key lifecycle. Integrating individually designed protocols for each stage into a single key lifecycle may introduce new problems that have not been previously encountered. However, this paper presents an integrated design that enables protocols for each stage to naturally interoperate within a single framework encompassing the entire key lifecycle.

4.2 Security analysis

Lemma 4.1. *(Non-stored) The proposed framework does not store the user's biometric information and the private key in Dev's internal or external storage to ensure non-storage.*

Proof. If a malicious attacker can obtain the user's Bio, they can successfully perform a sk reconstruction attack. However, the proposed framework provides non-stored by not storing Bio in either internal within Dev or external storage, preventing malicious attackers from acquiring Bio. Furthermore, it provides non-stored by not storing sk, shares of sk, or their encrypted values. □

Lemma 4.2. *(Resistance to collusion attacks) Assuming that Dev is secure and the cryptographic primitives applied in the proposed framework are secure, the proposed framework provides resistance against collusion attacks between a malicious attacker $\mathcal{A}$ and computation nodes cns.*

Proof. In the proposed framework, prior to performing the key backup or recovery process, U randomly selects a node cn from S to actually execute the computations for that process.

Even if $\mathcal{A}$ colluded with fewer than the threshold number of cn during the key recovery process, the security of the DPRF prevents the generation of w based on z_i. That is, $\mathcal{A}$ cannot generate the tk required to decrypt e. If $\mathcal{A}$ colludes with cn above the threshold during the key recovery process, $\mathcal{A}$ can collect z_i above the threshold. $\mathcal{A}$ can then generate tk to decrypt e and subsequently obtain HD and σ. However, sk reconstruction requires not only HD and σ but also Bio. Since the proposed framework provides non-stored property, $\mathcal{A}$ cannot obtain Bio. Therefore, $\mathcal{A}$ cannot reconstruct sk.

Consequently, the success probability of a collusion attack between $\mathcal{A}$ and cn in the proposed framework is negligible. □

Lemma 4.3. *(Confidentiality) Assuming that Dev and the applied cryptographic primitives are secure, the proposed framework guarantees confidentiality for sk.*

Proof. A malicious attacker $\mathcal{A}$ may attempt to infer sk from sk_{ID} contained within $\{Back_{\alpha}\}$. That is, A may attempt to obtain HD and σ via sk_{ID} for this purpose. However, since sk_{ID} is generated based on a cryptographically secure hash function, the probability that $\mathcal{A}$ can extract the desired information from sk_{ID} is negligible.

Additionally, $\mathcal{A}$ may attempt to infer sk by acquiring HD and σ through α or Com_{sk} contained in $\{Back_{\alpha}\}$. Here, commit values such as α and Com_{sk} are generated based on a cryptographically secure Pedersen commitment scheme. Therefore, the probability of successfully reconstructing sk or inferring related information through α or Com_{sk} is negligible. For this reason, the confidentiality of sk is guaranteed in the proposed framework. $\mathcal{A}$ can attempt to obtain Bio by acquiring $Comm^{(0)}$ and $Comm^{(0,0)}$ through $\{Back_eTx\}$. However, since $Comm^{(0)}$ and $Comm^{(0,0)}$ are also values generated by the Pedersen commitment scheme, Bio cannot be obtained through them. Therefore, the confidentiality of sk is guaranteed.

A may attempt to infer sk by obtaining HD and σ based on the Ez_i contained in $\{Back_{Ez_i}Tx\}$. However, if the public-key cryptographic algorithm used is secure, $\mathcal{A}$ cannot decrypt Ez_i, thus ensuring the confidentiality of sk. Furthermore, $\mathcal{A}$ may attempt to decrypt the e contained in $\{Back_eTx\}$ to obtain HD and σ, and use this to infer sk. However, since e is a value generated using an AEAD encryption algorithm, only the legitimate user can decrypt it. Thus, if the AEAD algorithm is secure, the confidentiality of sk is also guaranteed.

Finally, $\mathcal{A}$ may attempt to obtain Bio by capturing HD, which was acquired through decryption during the key recovery process. HD is a value generated via a fuzzy extractor and

inherits its security properties of the fuzzy extractor. Thus, $\mathcal{A}$ cannot derive Bio from HD. Thus, in the proposed framework, the confidentiality of sk is guaranteed by non-storing properties, resistance to collusion attacks, and the security of the applied cryptographic primitives (hash function, commitment scheme, public-key cryptography scheme, AEAD scheme, fuzzy extractor technique). □

Lemma 4.4. *(Integrity) Assuming the security of Dev, the applied cryptographic primitives, and the blockchain, the proposed framework guarantees the integrity of sk.*

Proof. A malicious attacker $\mathcal{A}$ could be assumed to have tampered with the HD and σ retrieved from the wallet during the key regeneration phase. However, the user verifies whether the regenerated sk is identical to the one generated during the key generation protocol by using the commit value. Therefore, if the applied cryptographic primitives are secure, the integrity of sk is guaranteed. The values required during the key recovery process are obtained through transactions published on the blockchain network. That is, due to the integrity and immutability properties of the underlying blockchain, it is difficult for A to manipulate these values. Furthermore, the validity of the recovered sk during the key recovery phase is verified using the commit value, thus ensuring the integrity of sk.

Therefore, if Dev is secure and the applied cryptographic primitives and blockchain are secure, the proposed framework guarantees the integrity of sk. □

Lemma 4.5. *(User authentication) Assuming the applied cryptographic primitives, blockchain, and smart contracts are secure, the proposed framework provides secure key recovery through the user authentication process.*

Proof. During the user authentication process, the smart contract acts as a verifier. At this point, the security of the smart contract inherits the security of the blockchain and the smart contract applied to the proposed framework. Therefore, it is reasonable to assume that the smart contract is honest and secure, and does not collude with a malicious attacker $\mathcal{A}$.

$\mathcal{A}$ can impersonate a legitimate user to request key recovery. To this end, $\mathcal{A}$ may attempt to impersonate a legitimate user by utilizing values contained within Γ that were used by the user during a previous authentication process. However, Γ includes a *nonce* value, and the smart contract verifies this value, thereby preventing replay attacks. Additionally, $\mathcal{A}$ may attempt a user forgery attack by calculating π using values contained in a previously used Γ. However, due to the security of the applied zero-knowledge proof technique, it is difficult for $\mathcal{A}$ to forge and compute π.

$\mathcal{A}$ may attempt to perform a spoofing attack by forging information such as $Comm^{(0,0)}$, $Comm^{(1,1)}$, and $Comm^{(0,1)}$. However, since these values are commit values generated based on Bio, security against forgery attacks is guaranteed by the security of the applied Pedersen commitment scheme. $\mathcal{A}$ may attempt to generate $Comm$ based on forged Bio. It can then transmit Γ, containing this $Comm$, to the smart contract. However, this results in a mismatch in "$Comm$" during the verification process of the smart contract, and thus fails to pass the user authentication procedure. Finally, $\mathcal{A}$ may attempt to pass the user authentication process by acquiring Bio through $Comm^{(0)}$ and $Comm^{(0,0)}$ contained in $\{Back_eTx\}$. However, since $Comm^{(0)}$ and $Comm^{(0,0)}$ are values generated based on the Pedersen commitment scheme, they ensure the confidentiality of Bio. Therefore, $\mathcal{A}$ cannot obtain Bio using $Comm^{(0)}$ and $Comm^{(0,0)}$. Thus, in the proposed framework, the probability that $\mathcal{A}$ succeeds in passing the user authentication process is negligible. □

Lemma 4.6. *(Event-based key recovery) Assuming the blockchain and smart contracts applied to the proposed framework are secure, the proposed framework ensures that the key recovery process is performed only in response to key recovery request events initiated by the user.*

Proof. A user requests key recovery by distributing $\{lostTx\}$ to the blockchain network, signifying an event indicating the loss of their private key. At this stage, a malicious attacker $\mathcal{A}$ can exploit transactions published during the key backup process to impersonate a legitimate user and request key recovery. However, Lemma 4.5 ensures the proposed framework's security against forgery attacks. During the key recovery process, *cn* verifies $\{ResultTx\}$ and $\{lostTx\}$ before performing the key recovery. The security against collusion attacks by malicious *cn*s attempting key recovery is guaranteed by Lemma 4.2. Therefore, the proposed framework ensures that the key recovery process is performed only upon a key recovery request from an authenticated user. □

Theorem 4.7. *If the security of the underlying blockchain and smart contracts, along with the applied cryptographic primitives, is guaranteed, then the proposed framework satisfies non-stored, resistance to collusion attacks, confidentiality, integrity, user authentication, and event-based key recovery.*

Proof. It is guaranteed by Lemmas 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6. □

5 Conclusion

This paper proposed a key management mechanism that encompasses protocols for each stage of the key lifecycle. The framework provides non-stored by not storing the user's biometric information used for private key generation or the private key itself. It also provides resistance to collusion attacks by randomly selecting nodes to perform computations during the key backup and recovery process. Furthermore, key recovery is performed only at the user's request and is carried out solely by an authenticated user through a smart contract prior to execution. It is an integrated key management framework that differs from existing research, providing decentralized key backup and recovery, confidentiality, and integrity as well.

6 Acknowledgments

- This work was supported by the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT)(IITP-2025-RS-2020-II201797)

- This research was supported as a 'Technology Commercialization Collaboration Platform Construction' project of the INNOPOLIS FOUNDATION (Project Number: 1711202494).

References

[1] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. Dise: distributed symmetric-key encryption. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1993–2010, 2018.

[2] Saif Al-Kuwari, James H Davenport, and Russell J Bradford. Cryptographic hash functions: Recent design trends and security notions. *Cryptology ePrint Archive*, 2011.

[3] Rakesh Arora, Han Du, Raza Ali Kazmi, et al. Privacy-enhancing technologies for cbdc solutions. Technical report, Bank of Canada Staff Discussion Paper, 2025.

[4] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 531–545. Springer, 2000.

[5] Hao Cai, Han Li, Jianlong Xu, Linfeng Li, and Yue Zhang. Bpkem: A biometric-based private key encryption and management framework for blockchain. *Plos one*, 19:e0286087, 2024.

[6] Melissa Chase, Hannah Davis, Esha Ghosh, and Kim Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.

[7] Jason Chia, Ji-Jian Chin, and Sook-Chin Yip. Digital signature schemes with strong existential unforgeability. *F1000Research*, 10:931, 2021.

[8] Cristòfol Daudén-Esmel, Jordi Castellà-Roca, Alexandre Viejo, and Ignacio Miguel-Rodríguez. Multi-platform wallet for privacy protection and key recovery in decentralized applications. *Blockchain: Research and Applications*, 6(1):100243, 2025.

[9] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

[10] Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Qishi Wu. Fair peer-to-peer content delivery via blockchain. In *European Symposium on Research in Computer Security*, pages 348–369. Springer, 2021.

[11] Junhao Lai, Taotao Wang, Shengli Zhang, Qing Yang, and Soung Chang Liew. Biozero: An efficient and privacy-preserving decentralized biometric authentication protocol on open blockchain. *arXiv preprint arXiv:2409.17509*, 2024.

[12] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.

[13] Yazhou Wang, Bing Li, Yan Zhang, Jiaxin Wu, Guozhu Liu, Yuqi Li, and Zhen Mao. A novel blockchain's private key generation mechanism based on facial biometrics and physical unclonable function. *Journal of Information Security and Applications*, 78:103610, 2023.

[14] Xiaohui Yang and Wenjie Li. A zero-knowledge-proof-based digital identity management scheme in blockchain. *Computers & Security*, 99:102050, 2020.