A Key Recovery System Based on Password-Protected Secret Sharing in a Permissioned Blockchain

Gyeong-Jin Ra[1], Chang-Hyun Roh[1] and Im-Yeong Lee[1, *]

Abstract: In today's fourth industrial revolution, various blockchain technologies are being actively researched. A blockchain is a peer-to-peer data-sharing structure lacking central control. If a user wishes to access stored data, she/he must employ a private key to prove ownership of the data and create a transaction. If the private key is lost, blockchain data cannot be accessed. To solve such a problem, public blockchain users can recover the key using a wallet program. However, key recovery in a permissioned blockchain (PBC) has been but little studied. The PBC server is Honest-but-Curious (HBC), and should not be able to learn anything of the user; the server should simply recover and store the key. The server must also be resistant to malicious attacks. Therefore, key recovery in a PBC must satisfy various security requirements. Here, we present a password-protected secret sharing (PPSS) key recovery system, protected by a secure password from a malicious key storage server of a PBC. We describe existing key recovery schemes and our PPSS scheme.

Keywords: Password-protected secret sharing, key recovery, blockchain, wallet security.

1 Introduction

A blockchain ensures the integrity of stored data and service availability [Liu, Yu, Chen et al. (2017)]. The blockchain generates a private user key that is employed to sign data. After signature, the transaction is propagated to the blockchain network to inform all about ownership of the data [Nakamoto (2019)]. Thus, the private user key is very important because it proves ownership of data when the signature appears in the blockchain. Therefore, the key must be secure. In a public blockchain, key management is performed within a personal software wallet, but stored key information may be leaked by an attacker who accesses the user's wallet. Leakage compromises user data [He, Zeng, Zhang et al. (2018)]. However, a PBC uses a more reliable network and solves this problem by creating a separate key management authority.

Motivating Example: One method of key recovery is a secret key distribution/storage technique. Secret values are fragmented and the pieces hidden in other objects [Wazod,

[1] First Soonchunhyang University, Asan-si, 31538, Korea.

[*] Corresponding Author: Im-Yeong Lee. Email: imylee@sch.ac.kr.

Received: 30 April 2020; Accepted: 08 June 2020.

Das, Odelu et al. (2017)]. Secret key recovery is possible only by collecting at least k values from n secret variances [Harn, Xia, Hsu et al. (2020)]. If an attacker collects less than k values, the key cannot be recovered. However, it is difficult to apply the Shamir secret distribution technology to blockchain key recovery because there is no user authentication step, associated with a risk of spoofing attacks (by user mimics). Also, as the secret fragments are not encrypted, the secret is exposed.

Thus, we developed a PPSS method; key recovery is authenticated via secret distribution and employment of a user password. PPSS was first proposed by the Bagherzandi group [Bagherzandi, Jarecki, Saxena et al. (2011)]. However, if an attacker attempts authentication and key recovery using many random passwords, it may be that one is in fact the user's password.

In this paper, we propose a key recovery system in a permissioned blockchain following contributions and organization:

Contributions: To solve this problem, we add a data authentication value to the secret distribution value and encrypt the Ogata approach to confirm that the correct data have been received. The key that encrypts the secret fragment is itself encrypted with a knowledge-based symmetrical key that only the user knows. This ensures the confidentiality of secret fragmentation and user authentication. The key recovery server does not know the secret because the secret distribution value is encrypted and transmitted. In addition, users authenticate other users via their abilities to recover secrets. In addition, as the secret fragment bears a hash value, a forgery can be detected early even if the secret fragment is not completely recovered.

Organization: The structure of the paper is as follows. Section 2 describes relevant studies. Section 3 describes the PPSS and Ogata schemes that form the basis of the paper. Section 4 describes the security requirements of key recovery. Section 5 describes our new PPSS scheme that blocks malicious servers. Section 6 analyzes the security requirements of the new method and compares the calculations to those of existing PPSS schemes.

2 Related work

In this section, we research related work. First, a blockchain will be described, as will variations in blockchain environments caused by structure and network configurations. Second, the key recovery techniques of existing blockchains will be discussed, and the secret distribution technique of Shamir.

2.1 Blockchain

A blockchain was first proposed in 2009 by Satoshi Nakamoto. Bitcoin is not issued and managed by a central bank. In a peer-to-peer network, all participants form a network that uses a distributed database with the same ledger [Li, Zhang, Luo et al. (2019)]. A blockchain is a list of records that grows continuously (a type of distributed database) and cannot be manipulated by distributed node operators. A bitcoin blockchain collects bitcoin transaction histories, creates and records blocks, and periodically creates special blocks that connect with the front block, generating data in the form of chains. As the periodically generated blocks are connected in order of creation, it is difficult to forge data because it would be necessary to modulate all data [Zhang, Zhong, Wang et al. (2020)]. Recall that a blockchain lacks a management server; data are shared by all participants. It is very difficult to falsify data; it would be necessary to tamper with over 51% of the total computing power [Ismail, Matter, Walla (2019)]. Blockchain is a permission-free environment. Blockchain participants can be publicly configured without any restrictions. In a PBC, only a predetermined node can form a blockchain network; that node may provide a specific function agreed upon in advance. A PBC can be subdivided into public and private environments depending on the blockchain data to be accessed. Again, a user must keep the private key safe. As there are no trusted institutions in an unlicensed blockchain, users must often perform key recovery unaided. In a PBC, there are several ways by which a user can recover a key.

2.2 Keys and key recovery in a blockchain

Blockchain key data must be securely stored and managed, as described above. In the early days of Bitcoin, the key management program was termed a wallet that stored and used all private keys. Employing the wallet, a user signed a transaction; if it was blocked, the transaction owner was identified using a public key [Liu, Li, Wang et al. (2017). The wallet created, stored, used, recovered, and retired keys. In general, public blockchains are unreliable; many methods of self-recovery are employed.

The first advance in key recovery was a key encryption method. A key in the wallet was encrypted using a password known only to the user. However, the key may be leaked if it is recovered using the password. The second advance was to move the associated code to another medium for storage. This reconstituted the bits of an existing key by connecting words in different languages to words with designated bit lengths. The key can be easily recovered, but can be leaked if the other medium is stolen. A blockchain commonly focuses on user self-recovery. However, in an authorized blockchain, trusted permissions can be used for key recovery. Existing key recovery methods are described below.

2.3 Shamir's (k, n) secret sharing (SSS)

The best-known, existing, key recovery method is secret distribution. In a PBC, the SSS scheme is used to distribute secret values for key recovery by a trusted third-party institution. SSS satisfies two properties:

1) If a user gathers k-1 or less secret pieces, the secret is not recovered.
2) The user must collect more than k shared values to recover the secret.

The scheme is called the (k, n) threshold scheme; if $\mathrm{k} = \mathrm{n}$, all secret pieces are required to recover the secret. The secret data can be converted to numbers. The secret and shared areas are elements of a finite field Z^p; the dealer constructs a polynomial of degree $\mathrm{k} - 1$ to divide the secret value by n.

3 Preliminaries

PPSS encrypts and distributes secret sharing employing a password that only the user knows. We now describe the basic Bagherzandi PPSS scheme and the complementary Ogata scheme.

3.1 Bagherzandi et al. Scheme

In general, Shamir's secret sharing scheme works based on a trusted key storage server. In this environment, the user transmits the secret sharing value to the storage server in a state where the secret sharing value is revealed. However, this scheme has a disadvantage in that if the server attempts a collusion attack, it can recover the key without the user's consent [Bagherzandi, Jarecki, Saxena et al. (2011)]. Bagherzandi first proposed the concept of PPSS to solve this problem. PPSS is a technique that calculates and transmits a user's secret shared value to a key storage server using a password known only to the user. Because of this, the secret shared value is secured with a password, it is a technique to prevent collusion attacks between servers [Hasegawa, Isobe, Iwazaki et al. (2015)].

3.1.1 Parameters

The coefficients of the Bagherzandi et al. scheme are as follows.

- c, d: Password-protected value.
- S: A set of n servers, such as $(S_1, S_2, \cdots, S_n)$.
- p: Uesr passwords.
- $\mathrm{s}k$: User secrets.
- r_s, r_p, t_j: Random numbers.
- g: A generator of group G of prime order P.

3.1.2 Distribution phase

The key distribution steps follow. The user generates a private key, fragments it, and distributes it to the server.

Step 1. The user calculates a private key.

$$sk \in Z^p \quad (1)$$

Step 2. The user distributes the secret key value sk via SSS.

$$sk = (s_1, s_2, \cdots, s_n) \quad (2)$$

$$(c_s, d_s) = (g^{r_s}, y^{r_s} * s_i) \quad (3)$$

$$\left(c_p, d_p\right) = (g^{r_p}, y^{r_p} * g^p) \quad (4)$$

Step 3. Subsequent secret distribution.

$(c_s, d_s), \left(c_p, d_p\right)$ to all S_i. (5)

3.1.3 Reconstruction phase

Key recovery includes the following steps: The user calculates a random value for the key recovery request and sends it to the server. The server returns a key fragment via the user's random calculation. The user recovers the fragment of the private key.

Step 1. The user chooses a random value r_p' and calculates r_p'.

$$\left(c_p, d_p\right) = (g^{r_p}, y^{r_p} * g^p) \quad (6)$$

Step 2. The calculated value (c_p, d_p) is sent to all S_i.

Step 3. A total of k $S_j$servers choose a random value $t_j$and derive $\frac{c_p}{c_p{'}}$, $\frac{d_p}{d_p{'}}$ using $\left(c_{b_j}, d_{b_j}\right) =$ $(g^{(r_p-r_p{'})*t_j}, y^{(r_p-r_p{'})*t_j} * g^{\left(p-p^{'}\right)*t_j})$ and $\left(c_{b_j}, d_{b_j}\right)$, and send these to the user.

Step 4. The user computes $\left(c_{b_j}, d_{b_j}\right) = (g^{(r_p-r_p{'})*\Sigma t_j}, y^{(r_p-r_p{'})*\Sigma t_j} * g^{(p-p')*\Sigma t_j})$, finds $\prod_{j=1}^{k} c_{b_j}$, $\prod_{j=1}^{k} d_{b_j}$, and sends these to S_j.

Step 5. S_j computes $(c_a, d_a) = (g^{r_s+(r_p-r_p{'})*\Sigma t_j}, y^{r_s+(r_p-r_p{'})*\Sigma t_j} * s * g^{(p-p')*\Sigma t_j})$ and sends (c_a, d_a) to the user.

Step 6. The user reconstructs ${c_a}^x$ and computes:

$$a_0 = d_a * {c_a}^{-1} = s * g^{(p-p')*\Sigma t_j} \tag{7}$$

Step 7. If $a_0 =$ s, the passwords can match and *s* is reconstructed.

However, an attacker can discover the password if two recovery attempts are made using values other than that of the password. To solve this problem, Ogata [Ogata (2013)] proposed the following PPSS scheme.

3.2 Ogata scheme

In the Bagherzandi scheme, if an attacker requests key recovery using a random value (not the user's password), the server generates data employing the non-password and returns a value to the attacker. If the attacker does this at least twice, the password can be inferred because the exponential operation employs the password. Ogata used a threshold of $\left(\frac{k+1}{2}, n\right)$ instead of (k, n) when performing SSS. Password leakage was prevented by adding the value of $\overline{[0(\iota)]}$ [Ogata (2013)]. The Ogata scheme prevents leakage of secrets via $\overline{[0(\iota)]}_j$. However, in terms of security, malicious server behavior remains possible.

Table 1: A comparison between secret sharing and PPSS

	Threshold	**Communication Amount**	**Prevention of illegal key recovery**	**Malicious behavior prevention**
SSS	(t, n)	2n	Not offered	Not offered
PPSS	(t, n)	4n	Offered	Not offered
Ogata	$((k+1)/2, n)$	2n	Offered	Not offered

3.2.1 Parameters

The Ogata's scheme parameters are:

- S: A set of n servers, such as $(S_1, S_2, \cdots, S_n)$.
- $\overline{[s]_\iota}$: The share of s for player S_i.
- $[s]_i$: A set of shares, such as $(\overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota})$.

- p: User passwords.
- s: User secrets.
- r_s, r_p, t_j: Random numbers.

3.2.2 Distribution phase

The key distribution steps follow. The user generates a private key, fragments it, and distributes it to the server.

<u>Step 1.</u> The user generates $\overline{[s]_\imath}$ using the $((k+1)/2, n)$ threshold scheme. $\overline{[P]_\imath}$ calculates this and sends it to all S_i.

3.2.3 Reconstruction phase

Key recovery proceeds as follows. A user calculates a random value for the key recovery request and sends it to the server. The server returns a key fragment via the user's random calculation. The user employs this to recover the original private key.

<u>Step 1.</u> The user employs $((k+1)/2, n)$ scheme to generate $\overline{[P]_\imath}$ and send it to all S_i servers.

<u>Step 2.</u> All servers S_i select a random number r(i) and then proceed with secret sharing using the $((k+1)/2, n)$ SSS scheme. And servers S_i distribute 0 using (k, n) SSS and send $\overline{[r(\imath)]_\jmath}$ and $\overline{[0(\imath)]_\jmath}$ to servers S_j $(j = 0, \cdots k-1)$.

<u>Step 3.</u> Server S_j proceeds with the following calculation.

$$\overline{[D]_\jmath} = \left(\overline{[P]_\jmath} - \overline{[P']_\jmath}\right)\sum_{j=0}^{n-1}\overline{[r(\imath)]_j} + \overline{[s]_j} + \sum_{j=0}^{n-1}\overline{[0(\imath)]_j} \tag{8}$$

<u>Step 4.</u> The server sends $\overline{[D]_\jmath}$ to the user.

Table 2: Components and notation of the model threat and security scenario

Ad	**HBC Adversary:** This is the agent playing the role of the HBC adversary, thus the PBC key recovery server, which includes participants.
U	**User:** A participant in the PBC who stores a private key in the key recovery server.
$m_1, \dots m_n$	**Message:** A sequence of one or more informational items sent as a single unit. We assume that a message is sent by a single sender.
$\overline{[s]_\imath}$	**Private Key:** The private user key value distributed to and stored in the storage server S_i.
$[s]_i$	**Secret distribution fragment:** A secret fragment that can recover a user's private key $(\overline{[as]_\imath}, \overline{[a_0]_\imath}, \cdots \overline{[a_{k-1}]_\imath})$.
$E([s]_i, h([s]_i))$	**PPSS ciphertext:** Ciphertext encrypted with the key of the user's knowledge base for secret fragment $[s]_i$
Verifiable $[S_i] \leftarrow r$	**Key verification value:** The server verifies a legitimate user's request by extracting the PPSS using the user parameter r.
$H(k)$	**Cryptographic hash:** An integrity check of the secret piece k left by the user.

Step 5. The user computes s by collecting all the $\overline{[D]_J}$ strings. If the passwords match, recovery proceeds; if the passwords do not match, random values are displayed.

Tab. 1 compares the three schemes. In the SSS scheme, the secret is distributed by applying a threshold value, but a collusion attack by the storage servers would reveal the secret value. The shared value is password-protected in the PPSS scheme; attacks are prevented. However, as explained above, if an attacker uses a strange password, the password may be leaked. The Ogata scheme prevents password leakage by adjusting the threshold of the secret distribution value, but malicious servers remain problematic. Thus, our scheme explores whether the server is malicious by adding a parameter that verifies the value of the secret when the user first creates it.

4 Security models

In this section, we design a model for key recovery in a PBC. Section 4.1 describes security threats and adversaries. Subsection 4.1.1 defines the HBC adversary and adversarial participants (servers and users) in a PBC. Subsection 4.1.2 describes the security threats to a PBC posed by the key recovery scheme. We use the components and notation defined in Tab. 2. Section 4.2 analyzes the security requirements.

4.1 Security threat and adversary

4.1.1 The honest-but-curious adversary

For protocol analysis in general, the best-known adversary model is the Dolev-Yao (DY) model [Dong and Muller (2018)]. The model is one of the strongest possible adversaries in terms of capabilities. In the ideal case, security and privacy would be safe even against a DY adversary. However, in some cases, the DY adversary is too strong to be used in a realistic model of the system. The key recovery system must be protected from external DY attackers. However, legitimate participants in protocols (such as key recovery agencies) cannot realistically model DY antagonists. Various factors, including regulatory, auditing, and oversight concerns, as well as a desire to maintain a good reputation, limit the functionality of the key recovery server. However, although a DY model is not appropriate here, this does not necessarily mean that the key recovery server is not adversarial. We therefore model the agent as semi-honest [Paverd, Martin and Brown (2014)] or HBC, defined as follows:

Definition 1 (Honest-But-Curious Adversary). The HBC adversary is a legitimate participant who will not deviate from a defined communication protocol but attempts to learn all possible information from legitimately received messages. In a PBC, all participants are HBC. Therefore, all content can be collected and stored to calculate user information correctly according to the protocol.

Definition 2 (Adversarial capabilities). The PBC key recovery agency (a server) observes and collects secret key fragments and key data generated by users. Other participants can access the network to observe the communication process and acquire data.

4.1.2 Security threat

Tab. 2 describes objects and elements used to model threats to the key recovery system of a PBC.

Message sniffing

The PBC key recovery server, and participants, can learn user information because they are HBC. When a user sends a plaintext key or a secret key fragment to a key storage server, an attacker can observe and collect that value via eavesdropping. During key recovery, an attacker can obtain the key and the secret fragment values, and recover the key.

Threat Scenario 1 (Message sniffing): $U(\overline{[s]_\iota}) \rightarrow Ad \parallel U([s]_i \leftrightarrow \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}) \rightarrow Ad, Ad : recovery \left[\left([s]_i, \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}\right)\right] \leftrightarrow \overline{[s]_\iota}$

Message spoofing

The recovery server of the PBC employs user information in an HBC manner. An attacker tricks the user into also attacking. Participants in a malicious PBC can attack the user by tricking the user into acting as a key recovery server for a legitimate user or by obtaining a secret distributed value disguised by a user and sent to another key storage server.

Threat Scenario 2 (Message spoofing): $U(\overline{[s]_\iota}) \rightarrow Ad \| U([s]_i, \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}) = \rightarrow Ad, Ad : recovery \left[\left([s]_i, \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}\right)\right] \leftrightarrow \overline{[s]_\iota}$

Collusion of participants

The key recovery server of the PBC publicizes user information because the server is HBC. Key recovery servers collaborate to rebuild a user key. Participants in a malicious PBC can recover user keys by collecting an adequate number of secret key fragments.

Threat Scenario 3 (Collusion of participants): $U(\overline{[s]_\iota}) \rightarrow Ad \parallel U([s]_i, \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}) = \rightarrow Ad$, $Ad : recovery \left[\left([s]_i, \overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota}\right)\right] \leftrightarrow \overline{[s]_\iota}$

4.2 Security requirements

Key recovery schemes must be authenticated, and must prevent illegal key recovery and malicious behavior, the three basic elements of security are confidentiality, integrity, and availability [Abidin, Rúa and Preneel (2016)].

1) **Confidentiality:** Transmitted information must be protected against eavesdropping or surveillance.

2) **Integrity:** Transmitted information should not be forged or altered by third parties.

3) **Availability:** The key should be recoverable whenever the user wishes.

4) **Authentication:** It is necessary to confirm that the recovery request message is from the real user.

5) **Spoof attack prevention:** Third parties cannot masquerade as users or service providers.

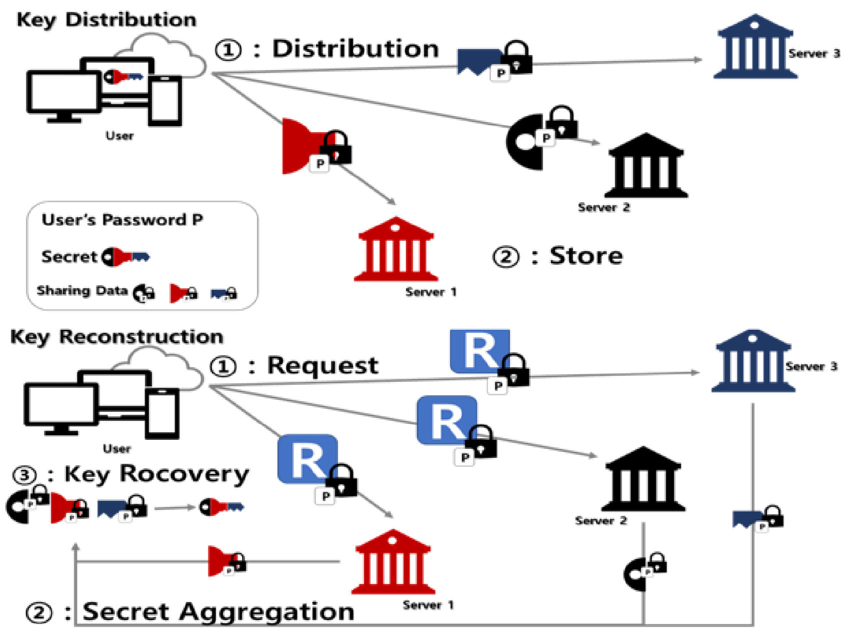


Figure 1: A model of our scheme

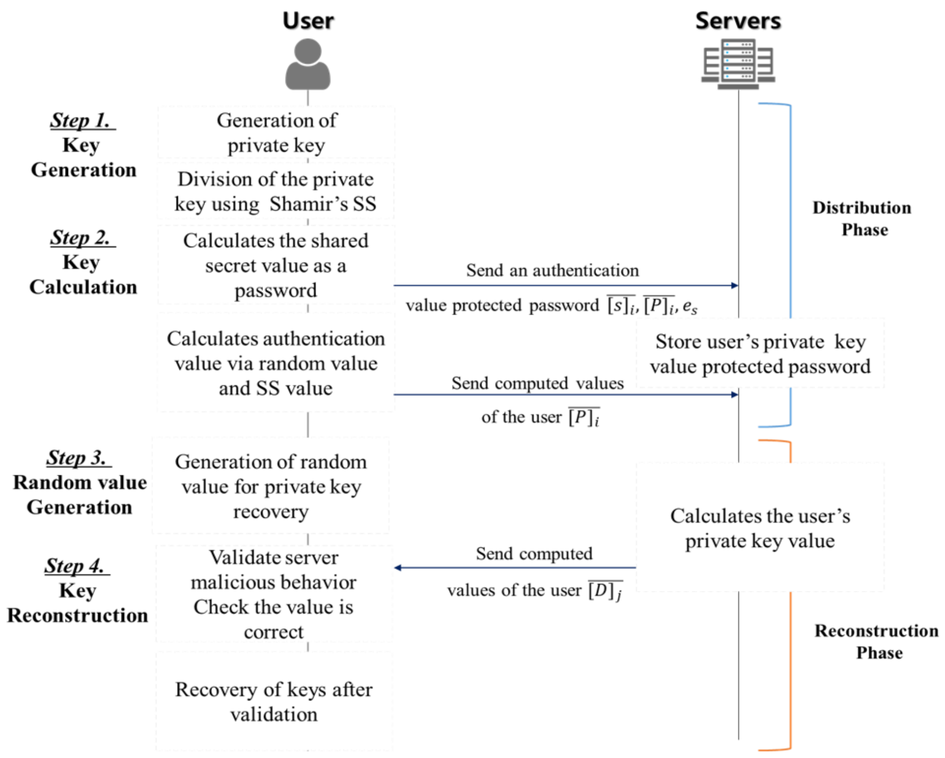

User
Servers
Step 1.
Key Generation
Generation of private key
Division of the private key using Shamir's SS
Step 2.
Key Calculation
Calculates the shared secret value as a password
Send an authentication value protected password $\overline{[s]}_i, \overline{[P]}_i, e_s$
Distribution Phase
Store user's private key value protected password
Calculates authentication value via random value and SS value
Send computed values of the user $\overline{[P]}_i$
Step 3.
Random value Generation
Generation of random value for private key recovery
Calculates the user's private key value
Step 4.
Key Reconstruction
Validate server malicious behavior Check the value is correct
Send computed values of the user $\overline{[D]}_j$
Reconstruction Phase
Recovery of keys after validation

Figure 2: The scenario of our scheme

6) **Prevention of illegal key recovery:** It must be impossible for servers with fragmented secret information to collaborate to recover keys.

These security requirements are basic. The key storage server is HCB, and a legitimate participant, but must not be able to learn or exploit other information. For an HCB server, it is important to prevent malicious operation. Objects with fragmented shared keys must not be able to give false responses or tamper with data.

The key recovery protocol for an HBC PBC is a synchronous distributed algorithm over n nodes, with up to f=k−1 malicious or faulty nodes, where n=2f+1. The user provides initial inputs to all nodes. Note that verification is performed over some random prime field vp and not over the RSA modulus N (vp>N). The detailed protocols are introduced in Section x and the security requirements are analyzed using the security model of Section 6.

5 Our scheme

The plan is that for the blockchain of Fig. 1. The method features a user and a key storage server; stored secret data are password-protected and the key storage server is unknown. The method checks whether the key storage server is performing malicious actions when recovering the key; the password does not leak during this action.

The scheme features distribution and reconstruction steps (Fig. 2). In the key distribution step, the user's secret value is fragmented employing the Shamir method and exponentiation performed employing the user's password. A confirmation value is calculated using the divided secret value and other coefficients, and distributed to all servers. In the reconfiguration step, an arbitrary value is determined and a cryptographically calculated value transmitted to the server. Then, the returned value is collected and calculated to recover the password. After recovering the password, the confirmation value is recovered using the password and compared to the user's secret value. This detects malicious behavior.

5.1 Parameters

The parameters assume that there are several key storage servers. The components are the key recovery users and key storage servers. Secret distribution and key recovery are performed using the PPSS equations and the user's password.

- *: Participating object (User performing key recovery: User, Key store server: Server).
- n: The number of servers.
- k: The threshold required to restore the secret.
- Z^P: Finite filed on prime numbers.
- S: A set of n servers, such as $(\mathrm{S}_1, \mathrm{S}_2, \cdots, \mathrm{S}_{n-1})$.
- $\overline{[s]_\iota}$: A share of s for player S_i.
- $[s]_i$: A set of shares, such as $(\overline{[as]_\iota}, \overline{[a_0]_\iota}, \cdots \overline{[a_{k-1}]_\iota})$.
- p: Uesr passwords.
- s: User secrets.

- r_s, r_p, t_j: Random numbers.
- g: A generator of group G of prime order P.

5.2 Distribution phase

The key distribution phase is divided into a key generation step that uses the SSS to divide the keys, and a password calculation step that protects the secret shared values. In the key generation step, the user generates a private key and devises a secret value using the $((\mathrm{k}+1)/2, \mathrm{n})$ SSS. In the key calculation step, password-protected data are generated by calculating the secret shared value created in the key generation step using a password that only the user knows. After generating the data, the value is sent to the server.

5.2.1 Key generation step

The user first generates a private key and divides it into secret fragments using the SSS.

Step 1. The user chooses private keys.

$$sk \in Z^p \tag{9}$$

Step 2. The user divides the private key value sk using SSS.

$$sk = (s_1, s_2, \cdots, s_n) \tag{10}$$

5.2.2 Key calculation step

The key calculation step involves calculating a password for a random value selected by a user and the user's secret shared value. The node is verified as non-malicious and the calculated value sent to the servers.

Step 1. The user chooses a random value r_p and the password p and calculates.

$$\overline{[P]_\iota} = (c_p, d_p) = (g^{r_p}, y^{r_p} * g^p) \tag{11}$$

Step 2. The user selects a random value r_s and encrypts the private key value s using $((k+1)/2, n)$ SSS.

$$\overline{[s]_\iota} = (c_s, d_s) = (g^{r_s}, y^{r_s} * s_i) \tag{12}$$

Step 3. An e_s value is generated to identify the secret sharing value s.

$$e_s = (y^p * s) \tag{13}$$

Step 4. The user sends $\overline{[s]_\iota}, \overline{[P]_\iota}, e_s$ to all servers S_i.

Step 5. All servers S_i store $\overline{[s]_\iota}, \overline{[P]_\iota}, e_s$

5.3 Reconstruction phase

The key reconstruction phase begins with the user selecting a random value and performing the password-protected calculation. The user sends the calculated value to the key storage server and recovers the secret s via an operation requiring the password-protected secret shared value transmitted in the distribution phase.

5.3.1 Random value generation step

During random value generation, the user selects a random value, applies the password, and transmits the result to the server.

Step 1. The user calculates a $((k+1)/2, n)$ SSS using the random value r_p and the known p'.

$$\overline{[P]_\iota} = (c_p, d_p) = (g^{r_p}, y^{r_p} * g^{p'}) \tag{14}$$

Step 2. The user sends $\overline{[P]_\iota}$ to all servers S_i.

5.3.2 Key reconstruction step

The key reconstruction step restores the value calculated by the password sent by the user to the password-protected value stored by the user, and returns the result to the user. The user gathers the data, reconstructs the password, and verifies whether the node is malicious by calculating a verification value.

Step 1. A total of k server S_j perform the following calculation.

$$\overline{[D]_J} = \left(\overline{[P]_J} - \overline{[P']_J}\right) \textstyle\sum_{j=0}^{n-1} \overline{[r(\iota)]}_j + \overline{[s]}_j + \sum_{j=0}^{n-1} \overline{[0(\iota)]}_j \tag{15}$$

Table 3: An security comparison of two existing schemes and our new scheme

	Bagherzandi	Ogata	Our scheme
Confidentiality	Secret Sharing + PPSS	Secret Sharing + PPSS	Secret Sharing + PPSS
Integrity	PPSS	PPSS	PPSS
Availability	Threshold Recovery	Threshold Recovery	Threshold Recovery
Authentication	PPSS	PPSS	PPSS
Prevent spoofing attacks	PPSS	PPSS	Validity Operation
Prevention of illegal key recovery	Not offered	Validity value $\overline{[0(\iota)]}$	Validity Operation
Preventing malicious behavior	Not offered	Not offered	Malicious nodes can be identified via e_s, the secret sharing verification value

Step 2. The user computes s by collecting all the $\overline{[D]_J}$ strings.

Step 3. The user recovers $\overline{[D]_J}$ and computes:

$$a_0 = \overline{[D]_J} * {c_a}^{-1} = (s * g^{(p-p')*\sum t_j}) \tag{16}$$

Step 4. If the value calculated in (16) is $a_0 = s$, password can match and s is reconstructed.

Step 5. The user verifies that $s = (e_s)^{-p}$ and treats s as normal if so. If there is no match, the node is malicious.

6 Analysis

6.1 Security

We analyze existing schemes and our new scheme in terms of the key recovery algorithm for the PBC-blockchain-based HBC security model of Section 4.2 (Tab. 3).

6.1.1 Security requirements

· **Confidentiality**: Confidentiality is guaranteed when performing the steps of rounds 1 and 2 of the key recovery algorithm for PBC-based HBC security (the Bagherzandi and Ogata models). The scheme determines a random value r_p', and encrypts this using $(c_p, d_p) = (g^{r_p}, y^{r_p} * g^{p})$ as the password for the key, then sending the ciphertext (c_p, d_p) to all S_i, ensuring confidentiality. Our method ensures the confidentiality of transmitted data by encrypting the private key with a PPSS-based password:

From Eq. (12): $\overline{[s]_\iota} = (c_s, d_s) = (g^{r_s}, y^{r_s} * s_i)$

· **Integrity:** Integrity is guaranteed when performing the steps of rounds 1 and 2 of the key recovery algorithm for PBC-based HBC security (the Bagherzandi and Ogata models). The user (c_p, d_p) are sent to all S_i and stored. A total of k servers S_j calculate $(c_{b_j}, d_{b_j}) = (g^{(r_p - r_{p'}) * t_j}, y^{(r_p - r_{p'}) * t_j} * g^{(p-p') * t_j})$ using $\frac{c_p}{c_{p'}}, \frac{d_p}{d_{p'}}$ and send (c_{b_j}, d_{b_j}) to the user to ensure integrity. The user calculates a $((k+1)/2, n)$ SSS using a random r_p and the known p'. The k servers S_j calculate $\overline{[P]_\iota} = (c_p, d_p) = (g^{r_p}, y^{r_p} * g^{p'})$ Finally, the k servers return the same $\overline{[D]_J} = (\overline{[P]_J} - \overline{[P']_J}) \sum_{j=0}^{n-1} \overline{[r(\iota)]_j} + \overline{[s]_j} + \sum_{j=0}^{n-1} \overline{[0(\iota)]_j}$ to ensure integrity.

· **Availability:** The secret distribution is sent to n servers during key recovery, and can thus be completely recovered even if some servers are not functional. If 3n+1≤S (where n is the number of malicious or inoperable servers and S the total number of servers), Byzantine fault tolerance indicates that the user obtains a key recovery fragment from the key recovery server. Thus, the key can be recovered normally. Integrity is guaranteed when performing the steps of round 4 of the key recovery algorithm for a PBC-based HBC security model. In the Bagherzandi and Ogata schemes, a user selects a random value r_s and encrypts the secret key value s using $((k+1)/2, n)$ SS ($\overline{[s]_\iota} = (c_s, d_s) = (g^{r_s}, y^{r_s} * s_i)$) to ensure availability. The user calculates a $((k+1)/2, n)$ SSS using the random r_p and the known p'. $\overline{[P]_\iota} = (c_p, d_p) = (g^{r_p}, y^{r_p} * g^{p'})$. This ensures availability.

From Eqs. (12) and (14): (t, n) , $((k+1)/2, n)$ Secret Sharing.

· **Authentication:** Authentication is guaranteed when performing the steps of round 3 of the key recovery algorithm for a PBC-based HBC security model. In the Bagherzandi and Ogata schemes, the user recovers ($\overline{[D]_J}$ and computes $a_0 = \overline{[D]_J} * {c_a}^{-1} = (s * g^{(p-p') * \sum t_j})$. If $a_0 = s$, the passwords match and s is reconstructed. This affords authentication.

· **Preventing malicious behavior:** Malicious server operation is prevented because the user employs 'e-s'.

6.1.2 Threat and mitigation

Mitigation of sniffing

The Bagherzandi, Ogata, and our schemes feature an HBC adversary. The user key is $\overline{[s]_\iota} = (c_s, d_s) = (g^{r_s}, y^{r_s} * s_i)$. This is an encrypted knowledge-based parameter that is difficult to eavesdrop or collect. Acquisition/decryption without a key is very difficult because of the Diffie-Hellman problem (DHP). In some forms of cryptography, the DHP is hard (Description 1).

Description 1. The Diffie-Hellman problem can be stated informally as follows: Given an element g and the values of g^x and g^y, what is the value of g^{xy}? Formally, g is a generator of some group (typically the multiplicative group of a finite field or an elliptic curve group) and x and y are randomly chosen integers. For example, in Diffie–Hellman key exchange, an eavesdropper observes g^x and g^y that are exchanged as part of the protocol, and the two parties both compute the shared key g^{xy}. A rapid means of solving the DHP would allow an eavesdropper to violate the privacy of the Diffie–Hellman key exchange and many of its variants, including ElGamal encryption.

Table 4: An efficiency comparison of two existing schemes and our new scheme

	Bagherzandi	**Ogata**	**Our scheme**
Communication level	4nW	2nW	2nW
Computational complexity	Distribution: (S+6E) Recovery: S+(6k+7) E	Distribution: 2S Recovery: (n+3k) S	Distribution: 2S+k Recovery: (n+4k) S

Mitigation of spoofing

The Bagherzandi server computes $\left(c_{b_j}, d_{b_j}\right) = (g^{(r_p - r_{p'}) * t_j}, y^{(r_p - r_{p'}) * t_j} * g^{(p-p') * t_j})$ when the server returns the user key distribution fragment $\overline{[s]_\iota} = (c_s, d_s) = (g^{r_s}, y^{r_s} * s_i)$ to the user. An HBC adversary requests a key but cannot infer $\left(c_{c_j}, d_{c_j}\right) = (g^{(r_p - r_{p'}) * t_j}, y^{(r_p - r_{p'}) * t_j} * g^{(p-p') * t_j})$. The servers of the Ogata scheme and our new scheme choose a random value r(i) and create $\overline{[r(\iota)]_J}$, $\overline{[0(\iota)]_J}$ to be shared by all servers ($j = 0, \cdots k-1$). The servers return the user key distribution to the user: $\overline{[D]_J} = \left(\overline{[P]_J} - \overline{[P']_J}\right) \sum_{j=0}^{n-1} \overline{[r(\iota)]_j} + \overline{[s]_j} + \sum_{j=0}^{n-1} \overline{[0(\iota)]_j}$. An HBC adversary cannot be inferred from key fragments and ciphertexts.

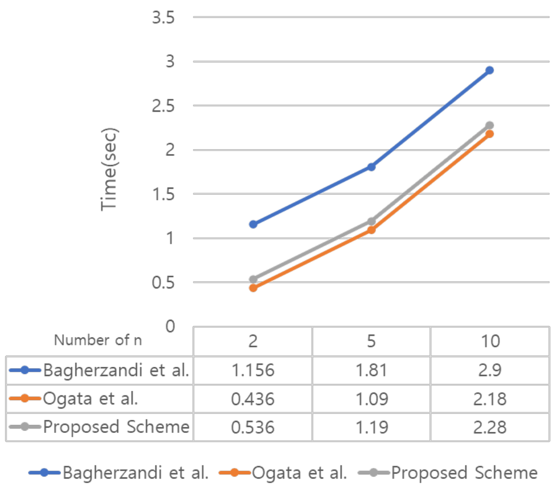
Mitigation of collusion

The S_j servers of the Bagherzandi and Ogata schemes calculate $\overline{[D]_J} = \left(\overline{[P]_J} - \overline{[P']_J}\right) \sum_{j=0}^{n-1} \overline{[r(\iota)]_j} + \overline{[s]_j} + \sum_{j=0}^{n-1} \overline{[0(\iota)]_j}$. An HBC adversary can forge and transmit unverifiable values $\overline{[D]_J}' = \left(\overline{[P]_J} - \overline{[P']_J}\right) \sum_{j=0}^{n-1} \overline{[r(\iota)]_j} + \overline{[s]_j} + \sum_{j=0}^{n-1} \overline{[0(\iota)]_j}$ to users. In our scheme, an HBC adversary cannot interfere with key recovery without knowing the PPSS value and the hash parameter for calculating $s = (e_s)^{-p}$

6.2 Efficiency

We here compare the efficiencies of the Bagherzandi, Ogata, and our schemes (Tab. 4) using the following symbols.

- W: Size of the shared SSS data.
- S: The SSS computational complexity over 0.218 s [Sugianto, Sugianto and Nico (2018)]
- E: The computational complexity of the ElGamal cryptosystem. Message size 1 kB, encryption and decryption operation speed 0.120 s [Shaina and Pooja (2014)]
- k: The computational complexity of random numbers.
- n: The number of secret key pieces.



Number of n	2	5	10
Bagherzandi et al.	1.156	1.81	2.9
Ogata et al.	0.436	1.09	2.18
Proposed Scheme	0.536	1.19	2.28

Figure 3: An efficiency comparison of two existing schemes and our new scheme

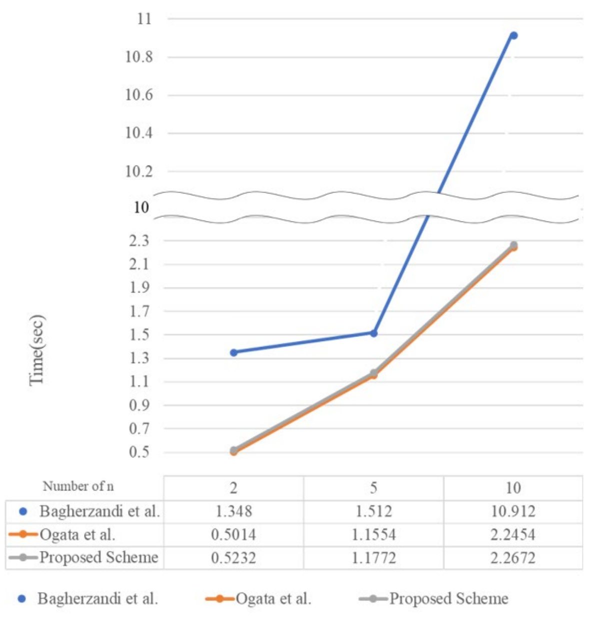
In terms of efficiency comparison, multiple passwords and secrets are in play, and the owner and user are engaged in key distribution and reconstruction. The distribution protocols include user registration and secret sharing. Bagherzandi et al. [Bagherzandi, Jarecki, Saxena et al. (2011)] were the first to introduce the PPSS concept, but if an attacker attacks with a strange value more than once, the password may leak. To solve this problem, Ogata did not divide passwords by counting 0-n numbers but, rather, by dividing them by 1-n numbers. However, there was no way to prevent malicious server behavior. Our scheme preserves Ogata's password leakage prevention, but identifies malicious nodes using random values. A user must perform key recovery calculations several times if a node is malicious. By recognizing such nodes, it is possible to reduce the number of recovery calculations.

In terms of the number of communications, the Bagherzandi scheme allows the user to receive a key recovery value via four communications; the Ogata scheme requires only two. The comparisons are made as follows:

The Bagherzandi scheme uses the Shamir secret (t, n) threshold technique and ElGamal cryptographic operations. In contrast, Ogata's scheme uses the x, and the Threshold scheme when distributing, requiring the following calculations.

In our scheme, verification of normal server behavior increases the amount of computation. Fig. 3 shows the execution times of the various schemes, which are influenced by the number and complexity of communications. If the communication

complexity <S, E, k, n> of each scheme is set to 1, the execution time will be proportional to the following constants. The Bagherzandi scheme requires the user to perform recovery correctly; the server then returns the key. This is inefficient. Both the Ogata scheme and our scheme feature verification using a symmetrical PPSS secret value; key fragment integrity can be verified without key recovery. However, the Ogata scheme does not defend against attacks if a server incorrectly stores or provides a symmetrical secret value of a fragment. Our method uses a hash function to calculate the secret. This adds a step to Recovery. However, our scheme is much more efficient than the Bagherzandi scheme and somewhat more efficient than the Ogata scheme (see Fig. 4).



Number of n	2	5	10
● Bagherzandi et al.	1.348	1.512	10.912
Ogata et al.	0.5014	1.1554	2.2454
Proposed Scheme	0.5232	1.1772	2.2672

Figure 4: Comparison of the recovery efficiencies of two existing schemes and our scheme

7 Conclusion

In blockchains such as cryptocurrency, key management is performed by wallets; key recovery methods for users of networks lacking trusted persons/servers have been well-studied. However, if the blockchain is a PBC, a trusted authority can be used for key recovery. Research on the safe storage and retrieval of private PBC keys continues. Most studies attempt to solve the problem by securely storing keys in trusted institutions; often, users alone cannot perform key recovery. Various schemes allowing users to participate in key recovery employing a PPSS password have been proposed, but passwords can be leaked and nodes can be malicious. We propose a new PPSS scheme that prevents these problems. We confirm that normal data are exchanged by adding a user-chosen random value during key recovery and erase that value before full recovery. Also, we detect malicious nodes, eliminate interference during key recovery, and reduce the number of user operations. In future, we will explore safe and efficient key recovery from the wallets of current blockchains with consideration given to calculation efficiency and communication time.

Funding Statement: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2020-2015-0-00403) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation) and This research was supported by the Soonchunhyang University Research Fund.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Abidin, A.; Rúa, E. A.; Preneel, B. (2016): An efficient entity authentication protocol with enhanced security and privacy properties. *International Conference on Cryptology and Network Security*, pp. 335-349, Springer, Cham.

Bagherzandi, A.; Jarecki, S.; Saxena, N.; Lu, Y. (2011): Password-protected secret sharing. *Proceedings of the 18th ACM conference on Computer and Communications Security*, pp. 433-444.

Dong, N.; Muller, T. (2018): The foul adversary: formal models. *International Conference on Formal Engineering Methods*, pp. 37-53, Springer, Cham.

Harn, L.; Xia, Z.; Hsu, C.; Liu, Y. (2020): Secret sharing with secure secret reconstruction. *Information Sciences*, vol. 519, pp. 1-8.

Hasegawa, S.; Isobe, S.; Iwazaki, J. Y.; Koizumi, E.; Shizuya, H. (2015): A strengthened security notion for password-protected secret sharing schemes. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 98, no. 1, pp. 203-212.

He, M.; Zeng, G.; Zhang, J.; Zhang, L.; Chen, Y. et al. (2018): A new privacy-preserving searching model on blockchain. *International Conference on Information Security and Cryptology*, pp. 248-266, Springer, Cham.

He, X.; Chen, X.; Li, K. (2019): A decentralized and non-reversible traceability system for storing commodity data. *Transactions on Internet and Information Systems*, vol. 13, no. 2, pp. 619-634.

Ismail, L.; Materwala, H. (2019): A review of blockchain architecture and consensus protocols: use cases, challenges, and solutions. *Symmetry*, vol. 11, no. 10, pp. 1198.

Jarecki, S.; Kiayias, A.; Krawczyk, H.; Xu, J. (2016): Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). *IEEE European Symposium on Security and Privacy*, pp. 276-291.

Li, H.; Zhang, F.; Luo, P.; Tian, H.; He, J. (2019): How to retrieve the encrypted data on the blockchain. *KSII Transactions on Internet and Information Systems*, vol. 13, no. 11, pp. 5560-5579.

Liu, B.; Yu, X. L.; Chen, S.; Xu, X.; Zhu, L. (2017): Blockchain based data integrity service framework for IoT data. *2017 IEEE International Conference on Web Services*, pp. 468-475.

Liu, Y.; Li, R.; Liu, X.; Wang, J.; Zhang, L. et al. (2017): An efficient method to enhance Bitcoin wallet security. *11th IEEE International Conference on Anti-counterfeiting, Security, and Identification*, pp. 26-29.

Nakamoto, S. (2019): Bitcoin: A peer-to-peer electronic cash system. Manubot.

Nawari, N. O.; Ravindran, S. (2019): Blockchain and the built environment: Potentials and limitations. *Journal of Building Engineering*.

Ogata, W. (2013): Improvement of IT-secure password-protected secret sharing. *30th Symposium on Cryptography and Information Security*.

Paverd, A. J.; Martin, A.; Brown, I. (2014): Modeling and automatically analyzing privacy properties for honest-but-curious adversaries. *Tech. Rep.*

Shaina, A.; Pooja (2014): Comparing results of various asymmetric cryptography algorithms using MATLAB. *International Journal of Computer Science and Technology*, vol. 5, no. 3, pp. 35-37.

Sugianto, S.; Suharjito, S.; Nico, S. (2018): Comparison of secret splitting, secret sharing and recursive threshold visual cryptography for security of handwritten images. *Telecommunication, Computing, Electronics and Control*, vol. 16, no. 1, pp. 323-333.

Sun, W.; Harayama, M. (2011): A proposal of key recovery mechanism for personal decryptographic keys. *International Conference on Internet Technology and Applications*, pp. 1-6.

Wazid, M.; Das, A. K.; Odelu, V.; Kumar, N.; Conti, M. et al. (2017): Design of secure user authenticated key management protocol for generic IoT networks. *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 269-282.

Xiong, F.; Xiao, R.; Ren, W.; Zheng, R.; Jiang, J. (2019): A key protection scheme based on secret sharing for blockchain-based construction supply chain system. *IEEE Access*, vol. 7, pp. 126773-126786.

Zhang, J. Y.; Zhong, S. Q.; Wang, T.; Chao, H. C.; Wang, J. (2020): Blockchain-based systems and applications: A survey. *Journal of Internet Technology*, vol. 21, no. 1, pp. 1-14.