IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 June 2023, accepted 14 August 2023, date of publication 24 August 2023, date of current version 30 August 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3308208*

RESEARCH ARTICLE

# KeyScrub: A Reliable Key Backup and Recovery Method for Blockchain

**WOOCHANG JEONG, HAESUNG PARK, AND CHANIK PARK**
Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang 37673, South Korea

Corresponding author: Chanik Park (cipark@postech.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korea Government through MSIT (Development of Big Blockchain Data Highly Scalable Distributed Storage Technology for Increased Application and Core Technologies for Hybrid P2P Network-Based Blockchain Services) under Grant 2021-0-00136 and Grant 2021-0-00484.

**ABSTRACT** On a blockchain platform, if a user has exposed or lost his/her private key, it may result in the loss of all the cryptocurrencies that he/she owns. It is critical to back up a user's private key and recover it at any time, without compromising the decentralized feature of the blockchain. The *t-out-of-n* secret sharing technique is typically used to support key backup and recovery in a blockchain, where a user's private key is divided into *n* partial keys, and the original private key can be restored only when at least *t* partial keys are available. This paper proposes a key scrubbing method, KeyScrub, to guarantee that at least *t* partial keys are available at any time in *t-out-of-n* secret sharing. The key scrubbing operation periodically (i.e., the scrubbing interval) checks how many partial keys are correctly maintained, and issues the recovery of partial keys without full recovery of the key. The reliability analysis of the proposed method shows a tradeoff relationship between the scrubbing interval and the values of *t* and *n* to meet the given reliability requirement for a key backup and recovery service.

**INDEX TERMS** Key scrubbing, t-out-of-n secret sharing, blockchain, key backup.

## I. INTRODUCTION

In blockchain networks, every user must have private and public-key pairs. Every transaction is signed by the user's private key and the users are identified by their public key. If a user loses their private key, it can result in the loss of all cryptocurrencies because there is no way to prove that the transaction was generated by the user. For example, a significant amount of Bitcoin has been lost due to key losses [1], [2], [3]. Unlike centralized services, users are responsible for managing private keys in decentralized blockchain services. Therefore, it is critical for each user to keep their private key secure and accessible in the blockchain [4].

For this purpose, a promising method called *t-out-of-n* secret sharing ((*t, n*)-secret sharing) [5]. It divides a user's private key into n pieces (partial keys) and stores each partial key separately in remote servers. The original private key can be recovered by correctly collect *t* partial keys. Each remote server stores a single partial key, rendering it useless. Note that the *t-out-n* secret-sharing technique assumes that *t* partial keys are accessible whenever needed. However, this assumption does not hold in real-world situations. Blockchain networks assume that certain percentage of nodes can be malicious. This accounts for the scenario where, despite distributing *n* partial keys to each node in the blockchain, the availability of more than *t* partial keys could be compromised due to the presence of such malicious nodes. Therefore, for *t-out-of-n* secret-sharing method to function properly, it is essential to have at least *t* correct partial keys available at all times. With this intention, it is necessary to verify whether each server correctly maintains its partial key. This verification operation is called 'scrubbing'. This term is from data scrubbing, which is the error correction technique to periodically analyze data for errors. Moreover, the *t-out-of-n* secret-sharing technique has a drawback during the scrubbing operation, each partial key is revealed in plain text to a server.

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

This study proposes an efficient key-backup method called KeyScrub, based on the *t-out-of-n* secret-sharing method. The proposed method involves (1) dividing a user's private key into n partial keys, (2) storing each partial key separately on a remote server, (3) enabling each server to conduct key scrubbing periodically (referred to as the scrubbing interval), and (4) enlisting a new server to reconstruct the corresponding partial key if a server fails to validate its partial key correctly during the key scrubbing process. Through the utilization of scrubbing and rebuilding of partial keys, KeyScrub ensures that the number of honest servers (storing their partial keys correctly) exceeds threshold number $t$. Furthermore, during the scrubbing and rebuilding process, Partial keys could potentially become exposed in plaintext during these scrubbing and rebuilding processes. To mitigate the security vulnerabilities caused by scrubbing and rebuilding, the proposed KeyScrub method also incorporates publicly verifiable secret sharing (PVSS) [6], [7] and bivariate secret sharing [8], [9].

The contributions of this paper are as follows:

- **Define the key scrubbing operation for monitoring the correctness of partial keys**: We introduce the concept of a key scrubbing operation to verify if a server stores its designated partial key correctly.
- **Maintain the number of honest servers above the threshold number*t***: To ensure the effectiveness of the *t-out-of-n* secret sharing approach, it is crucial to maintain a presence of at least t honest servers (storing their partial keys accurately). Through the implementation of the key scrubbing operation, the proposed KeyScrub method achieves the objective of upholding the number of honest servers beyond the threshold t.
- **Rebuild partial keys with no information leakage**: We employed PVSS [6], [7] and bivariate secret sharing [8], [9] to prevent the disclosure of any partial keys during the process of partial-key rebuilding.
- **Analyze the reliability requirement provided by KeyScrub**: Utilizing a Markov chain analysis model, we demonstrated that the proposed method can achieve the necessary level of reliability by configuring $t$, $n$, and the scrubbing interval.
- **Evaluate performance by prototype**: A prototype is built based on the Ed25519 [10] elliptic curve, which is typically used by blockchain. The proposed KeyScrub method is designed to support blockchain wallets [11], [12], [13] utilizing Ed25519. Additionally, the proposed method may be applicable to blockchain platform such as Cardano [14] and Solana [15] employing the Ed25519 curve for its cryptographic operations. The performance evaluation shows that the proposed method incurs an insignificant overhead.

The remainder of this paper is organized as follows. Section II explains the backgrounds, and Section III explains assumptions and threat model. Section IV describes the proposed KeyScrub method. Section V describes the implementation and evaluation of the proposed system. Finally, the conclusions and future work are presented in Section VI.

## II. BACKGROUNDS

### A. ELLIPTIC CURVE CRYPTOGRAPHY

An elliptic curve [16], [17], [18] is a cryptographic system adopted by most blockchains. Assume $G$ is the base point on an elliptic curve, and $A = G^q$ is the point obtained by multiplying $q$ times over $G$. In an elliptic curve, multiplication is equivalent to addition; therefore, $A$ is obtained by adding $q$ times to the base point $G$ on the elliptic curve. However, it is extremely difficult to determine $q$ given $G$ and $A$. Based on this feature, if we consider $x_i$ as a private key, then $X_i = G_i^x$ can be defined as a public key. Using this private-public key pair, scalar a can be encrypted as $X_i^a$ using public key $X_i$, and $X_i^a$ can be decrypted only if private key $x_i$ is available. General data messages other than scalar data can be encrypted via ECIES [19]. The encrypted message, $M$ is denoted as $E_{x_i}(M)$.

### B. PUBLICLY VERIFIABLE SECRET SHARING (PVSS)

In the *(t,n)*-secret sharing method, the owner of secret $s$ first divides $s$ into $n$ partial secrets, and then shares each partial secret disjointly with each participant. To recover the original secret $s$, the user must collect at least $t$ correct partial keys from $t$ participants. PVSS [6], [7] is a *(t,n)*-secret sharing method based on Shamir secret sharing [5].

In PVSS, that is, *(t,n)*-Shamir secret sharing, the owner of secret s first creates a $t-1$ deg polynomial with a constant term of s as shown in Equation 1.

$$P(x) = s + a_1x^1 + \ldots + a_ix^i + \ldots + a_{t-1}x^{t-1} \quad (1)$$

Then, the point $(x_i, P(x_i))$ on the polynomial is sent to the $i^{th}$ participant. If $t$ points are collected, the $t-1$ deg polynomial can be solved to calculate the constant term $s$ using Lagrange interpolation. For this computation, we must ensure that all $t$ points collected from participants come from the same polynomial equation $P(x)$. This is the main reason that the PVSS has been applied in KeyScrub. Further details are provided in lemma 2. In many systems [9], [20], the PVSS applies the fat-shamir technique [21] for noninteractive zero-knowledge proofs. We modified this protocol to an interactive protocol to validate partial data. The details are presented in Section III. With the help of zero-knowledge proofs, we can support no information leakage on secret s or points from participants during partial key rebuilding.

### C. BIVARIATE SECRET SHARING

Bivariate secret sharing is based on a bivariate polynomial, where the highest degree is represented by $t-1$ and $u-1$, as expressed in Equation 2.

$$B(x, y) = s + a_1x^1 + \ldots + a_ix^i + \ldots + a_{t-1}x^{t-1} + b_1y^1 + \ldots + b_iy^i + \ldots + b_{u-1}x^{u-1} \quad (2)$$

There are two threshold values, $t$ and $u$ in bivariate secret sharing, whereas there is one threshold value, $t$ in *(t, n)*-secret

sharing. Bivariate secret sharing is denoted by *(t, u, n)*-secret sharing. In *(t, u, n)*-secret sharing, each participant can autonomously restore its partial share by collecting at least $u$ partial shares from $u$ neighboring participants. We must collect at least $t$ partial shares to recover the original secret value, $s$. The $i^{th}$ participant stores two partial shares, each of which is represented by two polynomials of degree $t-1$ and $u-1$, denoted as $B(i, y)$ and $B(x, i)$, respectively.

Partial share $B(i, y)$ of degree $t-1$: When $t$ or more participants send their own $B(i, 0)$, which is one point on their partial share, to a dealer, the dealer can solve $B(x, 0)$ to compute the constant secret $s\ (= B(0, 0))$ using Lagrange interpolation.

Partial share $B(x, i)$ of degree $u-1$: $B(x, i)$ is used when each participant wants to autonomously recover their partial share. For example, to restore $B(x, j)$, the partial share of Participant $j$, participant j collects $B(i, j)$ from participant $i$. If more than $u$ participants have sent their own $B(*, j)$, $B(x, j)$ can be restored using Lagrange interpolation, which means that the share value of participant $j$ from the original bivariate polynomial is restored.

## III. ASSUMPTIONS AND SYSTEM THREAT MODEL

We assume that there are $n$ servers and that each server fails independently at a constant rate $\lambda$. Network failure is regarded as a malicious attack. A user's private key is divided into $n$ partial keys, and each partial key is stored separately on a server.

When we say failure, it may result in the following attacks.

- Tampering (i.e. delete or modify) partial keys
- Making partial keys unavailable by hardware crash

We assume that the system manager handles failures by rebooting the server and rebuilding partial keys as soon as a failure is detected.

## IV. THE PROPOSED METHOD: KeyScrub

The set of $n$ servers is denoted as $S = \{1, 2, \ldots, n\}$. The proposed method is based on *(t, u, n)*-secret sharing to back up and restore a user's private key $K$. Consider bivariate polynomial $B(x, y) = K + \sum_{i=1}^{t-1} a_i x^i + \sum_{j=1}^{u-1} b_j y^j$, as shown in Equation 2.

For the $i^{th}$ server $S_i$, $t$-data is defined as the univariate polynomial $B(i, y)$, and $u$-data is defined as the univariate polynomial $B(x, i)$. The user private key $K$ is divided into $n$ partial keys using this bivariate polynomial, where each partial key consists of two parts denoted as $t$-data and $u$-data.

Then, the following conditions are satisfied:

1. The user private key $K$ can be restored if we collect *t-data* from $t$ or more servers.

2. A server can recover its own *t-data* if *u-data* from $u$ or more servers.

Note that the message generated by server $i$ (i.e., $S_i$) is denoted as $<M>_i$. Without loss of generality, it is assumed that a server signature is attached to $<M>_i$ for message authentication.

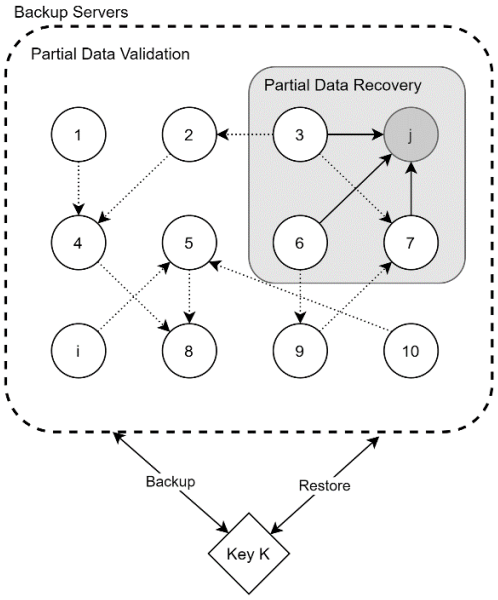


**FIGURE 1. Overview of KeyScrub: The dotted arrow means partial data validation and the bold arrow means partial data recovery. All servers are validating each other whether they keep their share of partial key correct. For example, $S_3$, $S_6$, and $S_7$ are cooperating to recover partial data (t-data) of $S_j$ that fails in the partial data validation step.**

Figure 1 shows the overall architecture of the proposed method KeyScrub method. Each circle represents a backup server that stores the partial key. The dotted line and solid line arrow between circles represent the partial data validation and partial data recovery operations, respectively. For example, in Figure 1, the dotted line arrow between $S_1$ and $S_4$ indicates that $S_1$ invokes a partial data validation over $S_4$.

Given a user's private key $K$ to be saved, the following operations are defined in Figure 1:

*1. Backup of a User Private Key K:* This divides $K$ into $n$ partial keys and distributes each partial key to $n$ backup servers. Partial keys were calculated using bivariate and publicly verifiable secret sharing. After this step, each server maintains two pieces of information, *t-data* and *u-data* as partial keys.

*2. Partial Key Validation:* It checks whether the target servers maintain their partial data (i.e., *t-data*) correctly. This process is called 'scrubbing' and is carried out periodically, with a scrubbing interval defining the frequency. Initiating scrubbing on a server is referred to as invocation or validation of the server. Through the use of a publicly verifiable scheme, this can be accomplished without disclosing any partial data in plain text to the invoking server. In the event that a server fails to uphold its partial data (*t-data*) accurately, the partial data recovery operation is triggered.

*3. Partial Key Recovery:* The partial data (*t-data*) of a target server are recovered using an invoking server. Notably, this recovery can be autonomously executed by a target server due to each server maintaining its own *u-data* through bivariate secret sharing. We assume that a target server undertakes this step on behalf of an invoking server if the

partial data (e.g., *t-data*) of the target server are found to be invalid.

*4. Restore of a User Private Key K:* This restores private key $k$ at the user's request.

### A. KEY BACKUP AND RESTORE

KeyScrub is based on *(t, u, n)*-secret sharing.

For key backup, first, a user must decide two threshold parameters, $t$ and $u$, each of which defines the minimum number of servers to recover $K$ and the partial key (*t*-data), respectively. Second, the user generates a bivariate polynomial (specified in Equation (2)): $B(x, y) = K + \sum_{i=1}^{t-1} x^i a_i + \sum_{j=1}^{u-1} x^j b_j$. Third, the user generates partial data, called *t-data* and *u-data* which are then distributed to each server. Note that *t-data* and *u-data* are sets of points on the univariate polynomials of *B(*, y)* and *B(x, *)*, respectively. For example, in case of server $S_i$, $t\text{-}data_i$ and $u\text{-}data_i$ are specified as $\cup_{m=1}^{u} B(i, m)$ and $\cup_{n=1}^{t} B(n, i)$. Both *t-data* and *u-data* are denoted as $D_i$. Therefore, $D_i = (t\text{-}data_i, u\text{-}data_i)$. Upon receiving $u$ and $t$ points, Si can set up two univariate polynomials, *B(i, y)* and *B(x, i)*, via Lagrange interpolation. Partial data validation checks whether each server stores these two sets of points (i.e., $t\text{-}data_i$ and $u\text{-}data_i$) correctly.

For key scrubbing, each server requires all coefficients of the bivariate polynomial information. This includes constant K and the coefficients $a_i$ $(i = 1, \ldots, t-1)$ and $b_j$ $(j = 1, \ldots, u-1)$. This information is sent to each server as a commitment to not reveal sensitive information in plain text. Commitment is defined as C in Equation (3).

$$C = (H, H^K, \cup_{i=1}^{t-1} H^{a_i}, \cup_{j=1}^{u-1} H^{b_j}) \tag{3}$$

where $H$ denotes one random point and $H^m$ denotes the multiplication of $H$ by $m$ times on the elliptic curve. We can see that constant $K$ and coefficients $a_i$ and $b_j$ are securely sent to a server as commitments $H^K$, $H^{a_i}$, $H^{b_j}$ together with $H$.

KeyScrub successfully generates all the necessary information to back up a user private key $K$. Then, a user has to distribute this information to each server to complete the key backup.

To describe message communication, we introduce additional notations to describe message communication. Each server is denoted as *ServerInfo* consists of *address* and *index* where *address* represents the server network connection address, and *index* represents the server identification. For example, the $i^{th}$ server $S_i$ is denoted as $ServerInfo_i$ which consists of its network connection address and index $i$. *ServerInfo* is the union of $ServerInfo_i$ for all backup servers.

A user is specified as *UserInfo* consisting of the user identity and his/her private key.

For secure communication, the message $M$ is encrypted using the recipient's public key. The encrypted message is denoted by $(M)_{X_i}$ when the recipient's public key is $X_i$. To ensure message integrity, a message is signed using the user's private key. If a message is $M$ and the private key is user, the signed message is denoted as $<M>_{user}$.

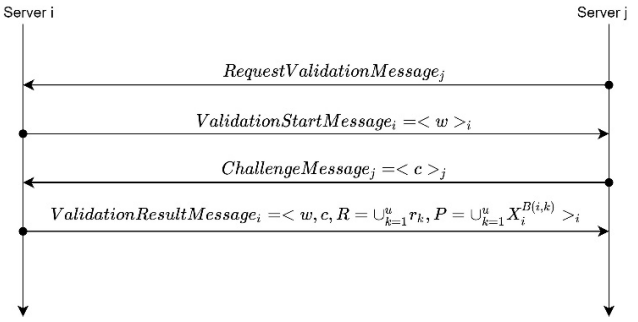


**FIGURE 2.** The protocol of key scrubbing by Server $S_j$ for Server $S_i$.

With this notation, the message for $S_i$ (whose public key is $X_i$) generated by a user (specified as *UserInfo* storing the private key user) is specified in Equation (4).

$$DM_i =< (D_i)_{X_i}, C, UserInfo, ServerInfo >_{user} \tag{4}$$

Note that $DM_i$ consists of encrypted partial *t-data* and *u-data* $(D_i)$, commit values $(C)$, user information (*UserInfo*), and server information (*ServerInfo*). Based on *ServerInfo*, each server can identify which server is responsible for the given partial key information (i.e., *t-data* and *u-data*).

Upon receiving $DM_i$ from a user, recipient server $S_i$ decrypts $DM_i$ and stores $D_i$, $C$, *UserInfo*, and *ServerInfo*. Next, $S_i$ calculates *B(i, y)* and *B(x, i)* from the bivariate polynomial points in $D_i$ using the Lagrange interpolation. When all servers receive *DM* and complete the calculations *B(i, y)* and *B(x, i)*, it is considered that the key backup is completed.

A user can recover his/her private key $K$ by collecting at least $t$ *t-data* from the backup servers. Without loss of generality, let us consider that server $S_i$ (whose private key is $x_i$) receives a request for key recovery from the user. From *B(i, y)*, $S_i$ builds a message $< B(i, 0) >_{x_i}$ $(KeyRestoreMessage_i)$ and sends it to the user. If a user collects at least $t$ *KeyRestoreMessage* from backup servers, the user can recover the user's private key $K$. (See Lemma 1)

*Lemma 1:* If there are $t$ or more *t-data*, the user's private key $K$ can be restored.

*Proof:* Refer to Appendix for detailed description.

### B. KEYSCRUB–PARTIAL DATA VALIDATION

Because each server maintains partial *t-data* and *u-data* $(D)$, commit values $(C)$, user information (*UserInfo*), and all servers' information (*ServerInfo*), any server can autonomously check whether its neighboring servers store their partial data correctly. This operation is called 'key scrubbing'. For example, to verify whether server $S_i$ maintains its $t\text{-}data_i$ (i.e., *B(i, y)*) correctly, any neighboring server (e.g., server $S_j$) can invoke key scrubbing for $S_i$. We call $S_i$ target server and $S_j$ the invoking server and validation server, respectively.

Figure 2 shows the key scrubbing operation of $S_j$ for $S_i$.

The detailed steps are as follows:

1. $S_j$ sends a request for partial validation of the data to $S_i$. The request is denoted as $RequestValidationMessage_j$.

2. $S_i$ sends back a randomly chosen scalar value $w$ to $S_j$ as $ValidationStartMessage_i$

3. $S_j$ sends back a randomly chosen scalar value $c$ to $S_i$ as $ChallengeMessage_j$.

4. $S_i$ proves that $S_i$ maintains a valid $B(i, y)$. For this purpose, it is sufficient to show that $S_i$ maintains valid $u$ points $B(i, k)$ $(k = 1, 2, \ldots, u)$ on $B(i, y)$. Therefore, for $B(i, k)$ $(k = 1, 2, \ldots, u)$, $S_i$ computes the proof $P$ by computing $X_i^{B(i,k)}$ where $X_i$ is the public key of $S_i$. In addition, $S_i$ builds a proof $R$ for freshness by computing $r_k = w^*B(i,k) - c$. Then, $S_i$ sends the proofs $P$ and $R$ together with $w$ and $c$ to $Sj$ as $ValidationResultMessage_i$.

5. $S_j$ can validate that $S_i$ maintains valid $u$ points for $B(i, y)$ by checking proofs $P$ and $R$. (See Lemma 2).

*Lemma 2:* We assume that $S_j$ invokes a key scrubbing operation for $S_i$. After receiving the message $ValidationResultMessage_i$ from $S_i$, $S_j$ can validate that $S_i$ maintains its partial data (that is, $t\text{-}data_i$ or $B(i, y)$) correctly.

*Proof:* Refer to Appendix for detailed description.

### C. KEY RECOVER–PARTIAL DATA RECOVERY

Note that any server can invoke a key scrubbing protocol, typically at regular intervals, for safety. This interval is denoted as *ScrubbingInterval*. If $S_j$ detects that $S_i$ fails to validate its own partial data during key scrubbing, $S_j$ recruits a new backup server $S_i^*$ replacing $S_i$. Then, $S_i^*$ recovers $t\text{-}data_i$ autonomously. Without a loss of generality, we can assume that a standby server replacing a faulty server can be recruited at any time.

The steps of partial data recovery are as follows:

1. $S_i^*$ sends a request message to neighbor servers to collect a sufficient amount of $u\text{-}data$ to recover partial data ($t\text{-}data_i$). This request message is called $RecoverRequestMessage_i$.

2. Upon receiving $RecoverRequestMessage_i$ from $S_i$, each neighbor server (e.g., $S_j$) sends back to $S_i$ a reply of $< B(i, j) >_j$ as $RecoverDataMessage_j$. $<B(i, j)>_j$ can be constructed by using $B(x, j)$ in $S_j$.

3. When $S_i^*$ collects at least $u$ *RecoverDataMessage* from neighboring servers, then $S_i^*$ can recover $B(i, y)$ via Lagrange interpolation. (See Lemma 3).

*Lemma 3:* If a server $i$ receives *RecoverDataMessage*(containing one point on the univariate polynomial $B(i, y)$) from $u$ or more number of servers, then it is guaranteed that sever $i$ can recover its own $t\text{-}data$ correctly.

*Proof:* Refer to Appendix for detailed description.

The time required to recover partial data through key scrubbing is denoted as *ScrubbingProcessTime*. This includes the latency of fault detection by key scrubbing and the time required for partial data recovery.

## V. RELIABILITY ANALYSIS

We assume that each server fails independently at a constant failure rate, λ. For a reliability analysis, two parameters should be considered: *ScrubbingInterval and ScrubbingProcessTime*.

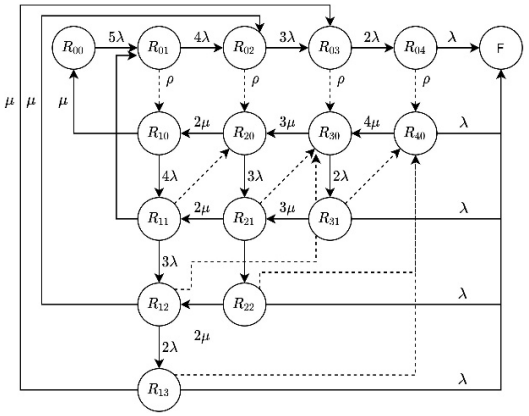


**FIGURE 3. The Markov chain model of KeyScrub when there are five servers and t = 4. The state transition is made either by a failure or a server replacement correct.**

Because *ScrubbingProcessTime* is used to detect a faulty server and recover partial data, it is important to estimate how many servers fail during the time interval of *ScrubbingProcessTime*. If the number of failed servers exceeds the threshold number, $t$, the proposed *KeyScrub* method may be unable to recover the original private key.

We consider the well-known reliability model of repairable systems based on the Markov birth–death chain [22], [23], [24]. With this model, we can estimate the number of servers that may fail during the time interval of *ScrubbingProcessTime*, that is, the probability that the proposed KeyScrub method fails to recover a user private key $K$.

We model our proposed KeyScrub method as a system with a set $S$ of independent elements (servers), tolerating up to $M$ element failures. The size of $S$ should be equal to or larger than $t + M$, where $t$ and $M$ represent the threshold value of key recovery and the maximum number of failures to tolerate, respectively. The initial state of the system assumes that all the servers in $S$ are operable.

Fig. 3 shows the Markov chain model of the proposed KeyScrub method, where

1. $\lambda$ = Failure rate of the server.
2. $\mu$ = Recovery rate of the faulty server
3. $\rho$ = Detection rate of faulty servers
4. $R_{ij}$ = System state, where $i$ is the number of faulty servers being replaced and $j$ is the number of faulty servers not yet detected by key scrubbing.

The recovery rate ($\mu$) is negligible compared to the server fault detection rate ($\rho$) and the failure rate ($\lambda$). This occurs due to *ScrubbingProcessTime* is in the range of several seconds. Therefore, the Markov chain in Figure 3 can be simplified to that shown in Figure 4.

Accordingly, the Kolmogorov-Chapman differential equation system for the simplified Markov chain, shown in

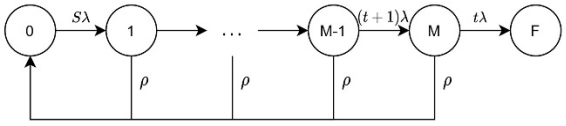


**FIGURE 4.** The simplified Markov model of KeyScrub considering short latency for scrubbing and partial data recovery when the number of servers is S and the threshold number is t. The integer number of a system state denotes the number of faulty servers detected.

Figure 4, is as follows:

$$\begin{aligned} P_0(0) &= 1; P_1(0) = P_2(0) = \ldots = P_M(0) = P_F(0) = 0; \\ P_0(t) &+ P_1(t) + \ldots + P_M(t) + P_F(t) = 1; \\ dP_k(t)/dt &= -S\lambda P_0(t) + \rho P_1(t) \\ &+ \rho P_2(t) + \ldots + \rho P_M(t) \, (k = 0) \\ dP_k(t)/dt &= (S - k + 1)\lambda P_0(t) - (S - k)\lambda P_k(t) \\ &- \rho P_k(t) \, (k = 1 \ldots M) \\ dP_k(t)/dt &= -S\lambda P_0(t) + \rho P_1(t) + \rho P_2(t) \\ &+ \ldots + \rho P_M(t) \, (k = F) \end{aligned} \quad (5)$$

Based on these equations, the state change probability can be calculated in a simple manner.

For reliability analysis in the simplified Markov model, we must set parameters $\lambda$, $\rho$, and $\mu$, which are the failure rate, server fault detection rate, and server recovery rate, respectively.

Failures may occur owing to various causes such as disk, memory, and server hacking. For simplicity, we consider the failure rate of a server in a cloud-computing environment in our reliability analysis. It was described in [25] that failures occur at a rate of approximately 8% per year in a cloud computing environment. In other words, a server fails at a rate of $9.53E^{-6}$ per hour, that is, $\lambda = 9.53E^{-6}$.

For the partial key recovery time, we must consider both the server fault detection latency and server recovery latency. The server recovery latency was insignificant compared with the server fault-detection latency. Therefore, it is sufficient to consider only the server fault-detection latency for partial key recovery time. In our model, key scrubbing was executed during the scrubbing interval period of *ScrubbingInterval*.

Equation (6) defines $C_t$, the probability of a server failing at time $t$. Equation (7) shows an equation for calculating the average failing time, *MeanFailTime*. Because the time required for key recovery is *ScrubbingInterval –MeanFailTme*, the server fault detection rate ($\rho$) is the inverse.

$$\begin{aligned} C_t &= \left(1 - (1 - \lambda)^t\right) / (1 - (1 - \lambda)^{ScrubbingInterval}) \\ &- (1 - (1 - \lambda)^{t-1}) / (1 - (1 - \lambda)^{ScrubbingInterval}) \end{aligned} \quad (6)$$

$$MeanFailTime = \sum_{t=1}^{ScrubbingInterval} tC_t \quad (7)$$

Figure 5 shows the reliability of the proposed method, that is, the probability that the original user key cannot be recovered when $n = 10$, $\lambda = 9.53E^{-6}$, and $\rho$ is set, as shown in Table 1. Under constant scrubbing interval, the probability of failure rate is increasing as the threshold is increasing.

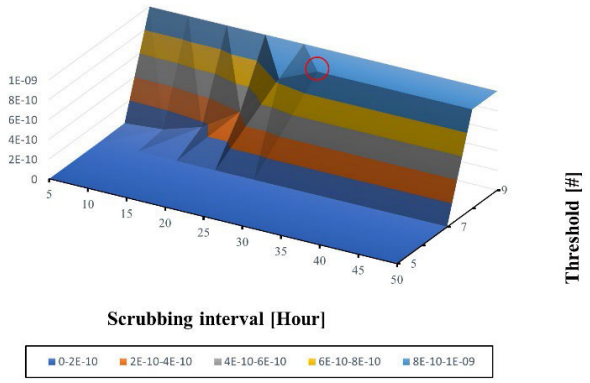


**FIGURE 5.** The reliability of the proposed method when n = 10: z-axis represents the probability of key recovery failure, x-axis represents the scrubbing interval (ScrubbingInterval), and y-axis shows the threshold value t. The circle indicates that the reliability is calculated to $1.0E^{-9}$ when scrubbing interval is 27 hour and the threshold t is 8.

**TABLE 1.** Detecting rate ($\rho$) according to scrubbing interval (in hour).

| | scrubbing interval | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| $\rho$ | 0.499 | 0.222 | 0.142 | 0.105 | 0.083 | 0.068 | 0.058 | 0.051 | 0.045 | 0.040 |

n=10 and $\lambda$=9.53E$^{-6}$

Table 2 shows required scrubbing interval to guarantee reliability requirements. For an example, in Figure 5, the red circle indicates that the system failure probability is $1.0E^{-9}$ when *ScrubbingInterval* is set to 27 and the threshold $t$ is set to 8.

Table 2 shows that the scrubbing interval should be set to 26 hours (or less) to guarantee the reliability requirement of 99.9999999% (nine nines).

## VI. PERFORMANCE EVALUATION

To evaluate the performance overhead of the proposed method, a prototype was implemented in approximately 2,400 lines of Go code, and all source codes were available at GitHub [26]. The Kyber [27] cryptography library was used for cryptographic primitives, and the Ed25519 [10] curve was chosen in the prototype. It is used for polynomial, commitment, and message encryption [19]. In addition, Blake [28] and SHA256 were used for hashing. The data size of the points and scalars used in the validation method is 32 bytes.

Experiments with the prototype were conducted in a system of six physical machines, each equipped with an Intel Xeon E5-2630 (6 cores at 2.3 GHz) × 2, 40 GBytes of RAM, and a 100Mbps network link. To evaluate the prototype in a large-scale distributed system, 20–120 virtual machines were created. Each virtual machine is assigned to one logical core.

**TABLE 2. Scrubbing interval in the given threshold.**

| | number of nines | | | | | | |
|---|---|---|---|---|---|---|---|
| | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| $t = 8$ | 26 | 55 | 117 | 252 | 544 | 1186 | 2625 |
| 7 | 181 | 322 | 574 | 1027 | 1846 | 3356 | 6225 |
| 6 | 634 | 1008 | 1609 | 2579 | 4164 | 1186 | 11385 |
| 5 | 1557 | 2301 | 3414 | 5092 | 7661 | 11687 | 18256 |

Scrubbing interval in the given threshold t to meet the reliability requirement specified by the number of 9s. For example, if the number of 9s is 5, the reliability requirement is 99.999%.

**TABLE 3. Elapsed time (time in MSEC) and message size (data in KB) required for partial data validation on a single key in the given number of servers.**

| | PVSS | | DKBS | | *prob*-DKBS | |
|---|---|---|---|---|---|---|
| | *time* | *data* | *time* | *data* | *time* | *data* |
| $t = 20$ | 138.51 | 15.66 | 7139.67 | 4.97+$p$(86.25) | 294.84 | 4.97+$p$(11.25) |
| 40 | 138.44 | 16.28 | 14062.33 | 6.22+$p$(161.25) | 281.01 | 6.22+$p$(11.25) |
| 60 | 138.39 | 16.91 | 21943.47 | 7.47+$p$(236.25) | 296.13 | 7.47+$p$(11.25) |
| 80 | 138.48 | 17.53 | 28196.12 | 8.72+$p$(311.25) | 284.93 | 8.72+$p$(11.25) |
| 100 | 138.62 | 18.16 | 35659.94 | 9.97+$p$(386.25) | 277.63 | 9.97+$p$(11.25) |

There are 120 servers, that is, $n = 120$, and $t$ is the threshold value. $p(x)$ represents the data size $x$ observed in a single operation of partial data validation.

### A. COMPLEXITY ANALYSIS

Partial data validation checks whether all the partial data sent from a target server are valid points on the polynomial. Because any partial data sent from $S_i$ is a point on polynomial $B(i, y)$ of $u-1$ °, it is necessary to check the validity of $u$ points. Therefore, the time complexity of partial data validation on a validation server is $O(u)$.

Instead of validating all points from a target server, we devised a probabilistic partial data validation that validates a randomly chosen single point (named the 'challenge' point). A 'challenge' point is randomly selected by a validation server and sent to a target server. If the target server is able to compute partial data for the challenge point, we can probabilistically conclude that the target server keeps $B(i, y)$ correct. The false-negative probability of probabilistic partial data validation was approximately $2^{-252}$ based on Ed25519 using a group with a prime range of $2^{252}$. Because the prime range is approximately $2^{252}$, the probability of obtaining the same random number by chance is $2^{-252}$.

Moreover, because each server continuously validates its neighboring servers, the false negative probability may be further reduced in real applications.

### B. PERFORMANCE OVERHEAD

The performance of the prototype for $(t, u, n)$-secret sharing was evaluated by measuring the time required for partial data validation. For security reasons, we set $u \geq t$. In this evaluation, $u = t$. PVSS [7] does not support key scrubbing, for example, partial data validation, and is considered the basis for comparison.

**TABLE 4. Elapsed time (time in MSEC) and message size (data in KB) required for partial data validation on a single key in the given threshold.**

| | PVSS | | DKBS | | *prob*-DKBS | |
|---|---|---|---|---|---|---|
| | *time* | *data* | *time* | *data* | *time* | *data* |
| $servers = 20$ | 23.85 | 2.84 | 870.47 | 1.22+$p$(8.13) | 47.93 | 1.22+$p$(1.88) |
| 40 | 47.86 | 5.34 | 1140.96 | 1.84+$p$(16.25) | 92.46 | 1.84+$p$(3.75) |
| 60 | 72.34 | 7.84 | 2005.10 | 2.47+$p$(24.38) | 138.97 | 2.47+$p$(5.63) |
| 80 | 92.25 | 10.34 | 2585.16 | 3.09+$p$(32.50) | 191.04 | 3.09+$p$(7.50) |
| 100 | 115.77 | 12.84 | 3211.41 | 3.72+$p$(40.63) | 231.36 | 3.72+$p$(9.38) |
| 120 | 138.30 | 15.34 | 3688.54 | 4.34+$p$(48.75) | 277.99 | 4.34+$p$(11.25) |

t=10.

In Table 3, we measured the elapsed time (ms) and message size (KB) required for partial data validation on a single key, where $n = 120$. As expected, in the case of the PVSS, the elapsed time does not change as $t$ increases. In the case of partial data validation using KeyScrub, it increased linearly as the threshold increased. In the case of probabilistic partial data validation in KeyScrub (*prob*-KeyScrub), the elapsed time does not change significantly because one point on the polynomial must be validated in the case of probabilistic partial data validation.

In Table 4, we measured the elapsed time (ms) and message size (KB) required for partial data validation using a single key, where $t = 10$. Compared with PVSS, *prob*-KeyScrub takes approximately twice as long, and KeyScrub takes approximately 30-40 times longer. This is because KeyScrub must collect at least t points for partial data validation, whereas *prob*-KeyScrub requires only one point for partial data validation.

## VII. CONCLUSION

In decentralized blockchain systems, it is critical to back up and recover a user's private key to address the key losses. Shamir secret sharing, which is denoted as $(t, n)$-secret sharing, is typically used to back a user's private key in a blockchain system. In $(t, n)$-secret sharing, a user's private key is partitioned into $n$ partial keys, which can be recovered by collecting at least $t$ partial keys. Therefore, $t$ was the threshold value.

$(t, n)$-Secret sharing may be extended to $(t, u, n)$-secret sharing, where there are two threshold values: $t$ and $u$. In $(t, u, n)$-secret sharing, a user private key is partitioned into $n$ partial keys, and each partial key consists of two partial data, *t-data* and *u-data*. We need to collect at least $t$ amounts of *t-data* to recover the user's private key, and we need to collect at least $u$ *u-data* to recover the associated *t-data*. In both cases, partial keys are distributed over networked servers. For security reasons, $u \geq t$ and we only consider the case of $u = t$ for simplicity.

For $(t, n)$-secret sharing or $(t, u, n)$-secret sharing to operate, it is important for the following condition to be satisfied: the number of partial keys available should exceed the threshold value, $t$.

In this paper, we propose an efficient and reliable user private key backup and recovery method, KeyScrub, based on (*t, u, n*)-secret sharing. The main idea of KeyScrub is to introduce a new operation of key scrubbing, which keeps monitoring servers to check whether each server stores its partially correct key. If it detects that a server does not store its partial key correctly, the proposed method rebuilds the partial key on a new server by replacing the failed server. With the help of key scrubbing, KeyScrub can satisfy the condition that the number of servers storing their partial keys correctly should exceed the threshold value $t$.

We analyzed the reliability of KeyScrub by using a Markov model. Through reliability analysis, we can determine the parameters $t$, $u$, and $n$ in (*t, u, n*)-secret sharing to satisfy the reliability requirement. The performance evaluation also showed that the proposed KeyScrub provides sufficient performance for practical use.

Our future work will include the application of KeyScrub to real blockchain platforms such as Bitcoin and Ethereum.

## APPENDIX

We prove the lemmas for KeyScrub.

*Lemma 1:* If there are $t$ or more *t-data*, the user's private key $K$ can be restored.

*Proof:* If there are $t$ or more *t-data*, $t$ or more *B(i, 0)* can be calculated from each *t-data*(= *B(i, y)*). Each *B(i, 0)* becomes $P(i)$, a point on $P(x) = K + a_1x^1 + a_2x^2 + \ldots + a_{t-1}x^{t-1}$, which is a *t-1* degree polynomial for $x$ with all $y$ removed from *B(x,y)*. Therefore, $P(x)$ can be created through $t$ or more points on $P(i)$ via Lagrange interpolation, and *P(0)* becomes $K$, the original key of the user.

*Lemma 2:* We assume that $S_j$ invokes a key scrubbing operation for $S_i$. After receiving the message $ValidationResultMessage_i$ from $S_i$, $S_j$ can validate that $S_i$ maintains its partial data (that is, $t\text{-}data_i$ or *B(i, y)*) correctly.

*Proof:* In order to prove possession of *B(i, k)*, which is the partial data of server $i$, Chaum-Pedersen protocol [29] is used as a sub protocol. This is discrete logarithm equality using zero knowledge proof. Therefore, server $i$'s any secret is not exposed during these steps. Proof of possessing *B(i, k)* without revealing the secret value processes through Equation (8) to Equation 8.7.

$$log_{g1}h1 = log_{g2}h2(h_1 = g_1^{B(i,k)}, h_2 = g_2^{B(i,k)}, g_1 = H, G_2 = X_i) \tag{8}$$

In Equation (8), $g_1$ is the random point on the elliptic curve in Commit, so $h_1$ is $H^{\mathrm{B(i,k)}}$, which is the *B(i, k)* value committed using $H$. $g_2$ and $h_2$ are the public key points of $i(X_i)$ and the encrypted value using this public key. Therefore, if Equation 8.1 is satisfied, it can be said that the prover knows *B(i, k)*, which means that it has valid data. Equation (8) can be proved if it satisfies the following Equation (9) and (10).

$$H^w = H^{rk}h_1^c \tag{9}$$

$$X_i^w = X_i^{rk}h_2^c \tag{10}$$

Equation (9) and (10) can be transformed $h_1$ and $h_2$ into Equation (11) and (12) using committed data and encrypted data in *CommitData*.

$$H^wH^{rk}h_1^c = H^r(H^{B(i,k)})^c \tag{11}$$

$$X_i^w = X_i^{rk}h_2^c = X_i^r(X_i^{B(i,k)})^c \tag{12}$$

At above equations, $H^{B(i,k)}$ can be calculated through Equation (13) even if *B(i, k)* is not known.

$$\begin{aligned} \mathrm{H}^{b(i,k)} &= H^{K+\sum_{m=1}^{t-1} a_m i^m + \sum_{n=1}^{u-1} b_n k^n} \\ &= H^K * \left(H^{a_1}\right)^1 * \ldots * \left(H^{a_{t-1}}\right)^{t-1} * \left(H^{b_1}\right)^1 \\ &\quad * \ldots * \left(\mathrm{H}^{b_{u-1}}\right)^{u-1} \end{aligned} \tag{13}$$

The value of $H^K$, $H^{am}$, $H^{bn}$ used in Equation (13) is included in *CommitData*, so Equation (13) can be calculated. Therefore, if $r_k = wB(i,k) - c$ is generated from valid *B(i, k)*, Equation (11) can be transformed to Equation (14).

$$H^w = H^{(w-B(i,k)^*c)}H^{B(i,k)^*c} = H^wH^{-B(i,k)^*c+B(i,k)^*c} = H^W \tag{14}$$

If the $r_k$ value in Equation (11) is not generated from valid *B(i, k)*, then the Equation (14) does not hold. Conversely, if equation (14) is held and equation (12) is held using *CommitData*, it can be said that it has valid *B(i, k)*. If server $j$ check $u$ point values for $B(i,k)(k = 1, \ldots, u)$, it means that *B(i, y)* can be recovered through the corresponding points via Lagrange interpolation, which can prove that server $i$ have valid partial data. Through the above process, server $j$ uses only the commit value, server $i$'s public key, $w$, $c$, and $R$ to validate that server $i$ possess a valid *B(i, y)*. By discrete logarithm problem, no one can get the secret value *B(i, y)* with only these values.

*Lemma 3:* If a server $i$ receives *RecoverDataMessage*(containing one point on the univariate polynomial *B(i, y)*) from $u$ or more number of servers, then it is guaranteed that sever $i$ can recover its own *t-data* correctly.

*Proof:* Having $u$ or more *RecoverDataMessage* means that there are points of *B(i, 1)*, *B(i, 2)*, …, *B(i, u)* on the bivariate polynomial *B(x, y)*. *B(x, y)* is a *t-1* degree of $x$ and *u-1* degree of y polynomial. If $x = i$ is fixed, it becomes a *u-1* degree of y polynomial *B(i, y)*. Since it has u points on *B(i, y)*, *B(i, y)*, that is t-data, can be calculated via Lagrange interpolation.

## REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Aug. 27, 2023. [Online]. Available: https://nakamotoinstitute.org/bitcoin/

[2] J. J. Roberts and N. Rapp. (Nov. 2017). *Exclusive: Nearly 4 Million Bitcoins Lost Forever, New Study Says*. Accessed: Aug. 21, 2023. [Online]. Available: https://fortune.com/crypto/2017/11/25/lost-bitcoins/

[3] CCN. (Feb. 2019). *$190 Million in Crypto Gone Forever, How Canada's Biggest Bitcoin Exchange Lost it All*. Accessed: Aug. 21, 2023. [Online]. Available: https://www.ccn.com/190m-gone-how-canada-biggest-bitcoin-exchange-lost-it/

[4] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better—How to make Bitcoin a better currency," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Bonaire, The Netherlands: Springer, 2012, pp. 399–414.
[5] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[6] M. Stadler, "Publicly verifiable secret sharing," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.* Saragossa, Spain: Springer, May 1996, pp. 190–199.
[7] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*. Santa Barbara, CA, USA: Springer, Aug. 1999, pp. 148–164.
[8] T. Tassa and N. Dyn, "Multipartite secret sharing by bivariate interpolation," *J. Cryptol.*, vol. 22, no. 2, pp. 227–258, Apr. 2009.
[9] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "CHURP: Dynamic-committee proactive secret sharing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., Nov. 2019, pp. 2369–2386.
[10] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Y. Yang, "High-speed high-security signatures," *J. Cryptogr. Eng.*, vol. 2, no. 2, pp. 77–89, 2012.
[11] SatoshiLabs. (2018). *Trezor Bitcoin Wallet—The Original and Most Secure Hardware Wallet*. Accessed: Aug. 21, 2023. [Online]. Available: https://trezor.io/
[12] Ledger. (2018). *Ledger Wallet—Hardware Wallets—Smartcard Security for Your Bitcoins*. Accessed: Aug. 21, 2023. [Online]. Available: https://www.ledgerwallet.com/
[13] Coinbase. (2018). *Coinbase—Buy/Sell Digital Currency*. Accessed: Aug. 21, 2023. [Online]. Available: https://www.coinbase.com
[14] Cardano. (2023). *Cardano Improvement Proposals*. Accessed: Aug. 21, 2023. [Online]. Available: https://cips.cardano.org/cips/cip16/
[15] Solana. (2023). *Solana Native Programs*. Accessed: Aug. 21, 2023. [Online]. Available: https://docs.solana.com/developing/runtime-facilities/programs
[16] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Conf. Theory Appl. Cryptogr. Techn.* Linz, Austria: Springer, 1985, pp. 417–426.
[17] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.
[18] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Proc. Int. Conf. Financial Cryptogr. Data Secur. (FC)*. Christ Church, Barbados: Springer, Mar. 2014, pp. 157–175.
[19] V. G. Martínez, L. H. Encinas, and C. S. Avila, "A survey of the elliptic curve integrated encryption scheme," *J. Comput. Sci. Eng.*, vol. 2, pp. 7–13, Jan. 2010.
[20] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "HydRand: Efficient continuous distributed randomness," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 73–89.
[21] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Conf. Theory Appl. Cryptogr. Techn. (CRYPTO)*. Santa Barbara, CA, USA: Springer, 1986, pp. 186–194.
[22] W. J. Anderson, *Continuous-Time Markov Chains: An Applications-Oriented Approach*. New York, NY, USA: Springer, 2012.
[23] Z. Wang, T. Wang, and X. Yang, *Birth and Death Processes and Markov Chain*. New York, NY, USA: Springer, 1992.
[24] S. Edge Mann, M. Anderson, and M. Rychlik, "On the reliability of RAID systems: An argument for more check drives," 2012, *arXiv:1202.4423*.
[25] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. ACM Symp. Cloud Comput. (SOCC)*. Indianapolis, IN, USA: ACM, Jun. 2010, pp. 193–204.
[26] SSLAB. (Dec. 2020). *DKBS Module*. Accessed Aug. 21, 2023. [Online]. Available: https://github.com/sslab-archive/dkms
[27] Decentralized and Distributed Systems Research Lab. (Dec. 2020). *Kyber, Advanced Crypto Library for Go*. Accessed Aug. 21, 2023. [Online]. Available: https://github.com/dedis/kyber
[28] J. P. Aumasson, S. Neves, Z. W. O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Banff, AB, Canada: Springer, 2013, pp. 119–135.
[29] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Proc. Annu. Int. Cryptol. Conf.* Santa Babara, CA, USA: Springer, Aug. 1992, pp. 89–105.
[30] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. Annu. Int. Cryptol. Conf.* Santa Barbara, CA, USA: Springer, Aug. 1995, pp. 339–352.
[31] S. Dolev, J. Garay, N. Gilboa, and V. Kolesnikov, "Swarming secrets," in *Proc. 47th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*. Monticello, IL, USA: IEEE, Sep./Oct. 2009, pp. 1438–1445.
[32] K. P. Birman, "The process group approach to reliable distributed computing," *Commun. ACM*, vol. 36, no. 12, pp. 37–53, Dec. 1993.
[33] A. Das, I. Gupta, and A. Motivala, "SWIM: Scalable weakly-consistent infection-style process group membership protocol," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, Washington, DC, USA, Jun. 2002, pp. 303–312.

**WOOCHANG JEONG** received the bachelor's degree in computer science and engineering from the Pohang University of Science and Technology, in 2018, where he is currently pursuing the Ph.D. degree with the Department of Computer Engineering. His research interests include blockchain, storage, and distributed computing.

**HAESUNG PARK** received the B.S. degree in computer science and engineering from Chungang University, in 2019, and the M.S. degree in computer science and engineering from the Pohang University of Science and Technology, in 2021. His research interests include blockchain and distributed computing.

**CHANIK PARK** received the B.S. degree in electronics engineering from Seoul National University, Seoul, South Korea, in 1983, and the M.S. and Ph.D. degrees in electronics and electrical engineering (computer engineering) from KAIST, Daejeon, South Korea, in 1985 and 1988, respectively.

He was a Visiting Scholar with the Parallel Systems Group, IBM T. J. Watson Research Center; and a Visiting Professor with the Storage Systems Group, IBM Almaden Research Center. He also visited Northwestern University and Yale University, in 2009 and 2015, respectively. Since 1989, he has been a Professor with the Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH). His research interests include storage systems, operating systems, system security, and blockchain. He has served as a program committee member for a number of international conferences and workshops.