Towards Dynamic and Reliable Private Key Management for Hierarchical Access Structure in Decentralized Storage

Zhang, Yifang; Wang, Mingyue; Guo, Yu; Guo, Fangda

Published in:
CIKM '23

Published: 01/10/2023

Document Version:
Final Published version, also known as Publisher’s PDF, Publisher’s Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1145/3583780.3615090

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.
Towards Dynamic and Reliable Private Key Management for Hierarchical Access Structure in Decentralized Storage

Yifang Zhang  
Beijing Normal University  
Beijing, China  
zyfyydgq@mail.bnu.edu.cn  

Yu Guo  
Beijing Normal University  
Beijing, China  
yuguo@bnu.edu.cn  

Mingyue Wang  
City University of Hong Kong  
Hong Kong, China  
mingyue.wang@my.cityu.edu.hk  

Fangda Guo  
Institute of Computing Technology, Chinese Academy of Sciences  
Beijing, China  
guofangda@ict.ac.cn  

ABSTRACT

With the widespread development of decentralized storage, it is increasingly popular for users to store their data to the decentralized database systems for the well-understood benefits of outsourced storage. To ensure the data privacy, systems commonly require users to securely keep their private keys. Thus, the secure storage of private keys is an important issue in these systems. However, existing key-management schemes commonly rely on a Trusted Third Party (TTP), which raises critical security concerns such as the single point of failure and Distributed Denial of Service (DDoS) attacks. In this paper, we propose HasDPSS, a secure and efficient blockchain-based key-management scheme for decentralized storage systems. It uses secret sharing, a lightweight cryptographic technique, to build the decentralized key-management scheme. Considering that the reliability of managing participants has inherent heterogeneity, we introduce the hierarchical access structure to achieve fine-grained key management. Meanwhile, to adapt the node churn of decentralized key management, HasDPSS enables a dynamic management committee to provide reliable services with a proactive refresh mechanism while protecting the integrity and security of private keys. In our design, we use the dimension switch method of polynomials in the evolving process to achieve the committee change of the hierarchical access structure. The reliability of participants is guaranteed by the customized commitment protocol and the immutable property of the blockchain. We thoroughly analyze security strengths and conduct extensive experiments to demonstrate the practicality of our design.

CCS CONCEPTS

• Security and privacy → Key management; • Information systems → Distributed storage.

KEYWORDS

Decentralized storage, Secret sharing, Hierarchical access structure, Dynamic committee

ACM Reference Format:


1 INTRODUCTION

Decentralized storage with privacy protection has emerged as a promising direction in revolutionizing existing outsourcing storage systems with centralized service providers. Such systems are typically based on blockchain [1] that employs cryptographic techniques to provide the security guarantee. Key management is an important component for providing security in decentralized storage. To preserve their privacy information, users are required to securely store their private keys. The loss of keys has severe and often irreversible consequences. Thus, the secure management of private keys is a principal foundation for these systems.

Key management has attracted much attention in recent years. A typical solution is to store keys in centralized key-management systems with a trusted third party [2]. Such a practice seems attractive, but it raises critical privacy concerns. This method is contrary to the design intent of decentralized storage since the existence of the centralized party destroys the decentralized feature. It is thus practically valuable to explore how to securely store private keys for decentralized systems without centralized trust.

Some existing works have focused on applying cryptographic techniques as an attractive alternative to achieve decentralized key management schemes. Li et al. [3] presented a secure key-management scheme based on secret sharing technique, which enables users to delegate private keys to a decentralized management committee of multiple participants. In this design, each participant in the committee only holds a share of keys. However, the scheme cannot deal with security concerns arising from the issues of node churn and diversified credibility of committee members. For these reasons, some schemes [4, 5] have been proposed to achieve proactive share updates, where each share can be updated periodically.

This work is licensed under a Creative Commons Attribution International 4.0 License.

CIKM ’23, October 21–25, 2023, Birmingham, United Kingdom  
© 2023 Copyright held by the owner/author(s).  
https://doi.org/10.1145/3583780.3615090
Recently, to solve the node churn issue, Maram et al. proposed CHURP [6], a decentralized secret-sharing scheme based on a dynamic committee, which is more practical for blockchain systems. However, CHURP only focuses on the situation where committee members have the same priority. It cannot adapt to the dynamic key-management committee with the hierarchical access structure. Hierarchical Access Structure (HAS) has been widely used in practical applications and has attracted much attention [7, 8]. To achieve fine-grained key management, shares held by committee members at higher levels are more significant in this management structure. In practice, the reliability and priority of blockchain nodes has inherent heterogeneity. For example, in a blockchain system based on the consensus of Proof of Stake (PoS) [9], priorities of block packaging are according to the amount of cryptocurrencies held by each node. Thus, it is necessary to design a general key-management scheme that supports hierarchical access structure on decentralized storage systems.

There is a great challenge in implementing the dynamic key-management committee based on the hierarchical access structure. Existing schemes that support the dynamic committee [4, 6] implement the change of committee members via secure Multi-Party Computation (MPC). The basic assumption of these schemes is that the key-management shares have the same priority. When the committee changes, a random number is generated by the new committee via MPC and added to the reconstructed new shares to distinguish between the old and new committee shares. However, these methods cannot work in a key-management committee with the hierarchical access structure. Simply adding the same random number to shares cannot guarantee that the set of new shares still satisfies the hierarchical access structure. To realize the dynamic committee in hierarchical access structure, the random values added to shares of different priorities must meet certain conditions. Meanwhile, for security purposes, the information of node priorities cannot be leaked during the committee change.

In this paper, we propose a decentralized key-management scheme, HasDPSS, which can achieve Dynamic Proactive Secret Sharing in Hierarchical access structure on blockchain. Specifically, HasDPSS employs a bivariate polynomial to share the secret s, i.e., user’s private key. The distribution of shares is based on the Birkhoff interpolation theorem to achieve hierarchical access structure. During the evolving process, when participants of a committee change, it uses the dimension switch method of polynomials to perform the committee switch. Considering the threat of time dimension, HasDPSS also realizes the proactive randomization of shares without TTP. HasDPSS introduces a hybrid communication model to bring more savings on communication for key management on blockchain. It consists of a co-design of on-chain and off-chain communication. Both communication overheads are lower than other blockchain. It consists of a co-design of on-chain and off-chain communication models.

- We make hybrid on-chain and off-chain communications to boost communication efficiency in HasDPSS. We also present practical mechanisms to make our scheme more robust, achieving stronger security against a powerful and active adversary.
- We implement a prototype and conduct comprehensive evaluations. Results demonstrate that our design achieves a low latency and practically affordable on-chain cost.

2 RELATED WORK

Secret sharing was proposed by Shamir [10]. It is a polynomial-based decentralized secret-managing scheme, where the distribution and reconstruction of the secret are based on the Lagrange interpolation. In a (t, n) threshold scheme [10], secret shares are distributed among n participants of committee, and only the authorized subset consisting of over t participants can reconstruct the secret. In recent years, it has been widely used in numerous practical applications, such as the Data Aggregation in Mobile Crowdsensing and Federated Learning.

Compared with Shamir’s secret sharing scheme, the Proactive Secret Sharing (PSS) scheme proposed by Herzberg et al. [11] further enhances the security assumption. It introduces the time dimension and assumes a powerful adversary can constantly compromise the members of committee. Some studies have proposed improved PSS schemes for application in practice [12, 13]. PSS scheme and its variants assume that the powerful adversary can continuously compromise participants in committee. The members of committee proactively refresh the shares after a fixed epoch to withstand compromise, while maintaining the secret. However, most PSS schemes are accompanied by a significant amount of communication overhead that increases exponentially.

Hierarchical access structure for secret sharing has been proposed in [14]. Tassa proposed Hierarchical Secret Sharing (HSS), and later secret sharing in HAS has been studied in many works [7, 15, 16]. It is an important key management technique since it is specially designed for hierarchical organizations. Compared with PSS schemes, HSS schemes can realize multilevel and compartmented access structures. For example, Yuan et al. proposes a hierarchical scheme for multi-secret based on linear homogeneous recurrence relations in [8]. In general, these schemes employ the integer polynomials or the linear homogeneous recurrence. Most of these works only focus on hierarchical management and security guarantee of the committee inside, but ignore the security threat from external adversaries. Moreover, these schemes cannot handle the problem of node churn in blockchain systems. Most of those schemes that support dynamic committee have a significant amount of communication overhead that increases exponentially.

3 PRELIMINARIES

3.1 Hierarchical Access Structure

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of n participants and consist of m levels. The set of participant in (i)th level is denoted by $U_i$, where $U = \bigcup_{i=0}^{m} U_i$ and $U_i \cap U_j = \phi$ for all $0 \leq i < j < m$. Let $k = (k_0, k_1, \ldots, k_{m-1})$, and $\mathcal{K} = (\sum_{j=0}^{k_0} k_j, \frac{1}{\sum_{j=0}^{k_0}} k_j, \ldots, \frac{1}{\sum_{j=0}^{k_{m-1}}} k_j)$ be a monotonically increasing sequence of integers to denote the
threshold of each level. The access structure of hierarchical secret-sharing scheme is described as follows:

$$\Gamma = \left\{ V \subset U : V \cap \bigcup_{j=0}^{i} U_j \geq \sum_{j=0}^{i} k_j, 0 \leq i < m \right\}. \quad (1)$$

A secret-sharing scheme that achieves this access structure is a \((\mathcal{K}, n)\)-hierarchical threshold scheme [14].

### 3.2 Birkhoff Interpolation

Let \(X = \{x_1, x_2, \cdots, x_n\}\) denote a series of points in \(R\), where \(x_1 < x_2 < \cdots < x_n\). Let \(E = (e_{i,j})_{1 \leq i \leq j \leq l}\) be a matrix with binary entries, \(I(E) = \{(i, j) : e_{i,j} = 1\}\) and \(k = |I(E)|\). The set \(C = \{c_{i,j} : (i, j) \in I(E)\}\) is consistent of \(k\) real values.

The Birkhoff interpolation corresponding to the triplet \((X, E, C)\) is to find a polynomial \(P(x) \in R_{k-1}[x]\) satisfying \(k\) equalities:

$$p^{(j)}(x_1) = c_{i,j}, (i, j) \in I(E), \quad (2)$$

where \(R_{k-1}[x]\) is a set of polynomials with degrees \(k-1\) and \(p^{(j)}(\cdot)\) is the \((j)\)-th derivative of \(P(x)\).

If the pair \((X, E)\) is regular, the set of equalities (2) has a unique solution for any choice of \(C\).

### 3.3 Lagrange Interpolation

Lagrange interpolation can be viewed as a special case of Birkhoff interpolation. The interpolation matrix in Lagrange interpolation has only one column, i.e., all data corresponds to the zeroth-order derivative. Specifically, given the points set \(X = \{x_1, x_2, \cdots, x_k\}\) and the real values set \(C = \{c_1, c_2, \cdots, c_k\}\), the Lagrange interpolation is the problem of finding a polynomial \(P(x) \in R_{k-1}[x]\) satisfying \(k\) equalities as follows:

$$P(x_i) = c_i. \quad (3)$$

For any choice of \(C\), the problem of Lagrange interpolation always has a unique solution.

### 4 PROBLEM STATEMENT

#### 4.1 System Architecture

The basic service model of our design is illustrated in Fig. 1. We consider two parties in our design, namely users and the multi-level key-management committee. For privacy protection, the user’s private key is submitted in encrypted form. Specifically, secret sharing, a lightweight cryptographic primitive is employed to encrypt the private key. Each share is expected to be received by a participant in the committee. We then have two basic assumptions: 1) participants of the committee differ in their levels of reliability; 2) they may periodically leave/join the committee or change their levels in hierarchical access structure.

HasDPSS adopts the hierarchical access structure [14], where the presence of higher-level participants is more important to the management of the private key. Let \(C\) denote the committee. We divide members of \(C\) into \(m\) different levels and \(\{C(i,j)\}_{j=1}^{n}\) denote the \(n\) nodes of level \(i\). Let \(k = (k_0, k_1, \cdots, k_{m-1})\), and the number of nodes is at least \(3k_i\) for level \(i\). We further define \(\mathcal{K} = (\sum_{j=0}^{0} k_j, \sum_{j=0}^{1} k_j, \cdots, \sum_{j=0}^{m-1} k_j)\), and the managed key can be reconstructed only if the number of shares obtained from each level satisfies the parameter \(\mathcal{K}\) of hierarchical access structure.

In a dynamic committee, time in HasDPSS is divided into a fixed interval of predefined length, called epoch. Fig. 2 shows the committee transition and the evolving process along time.

**Figure 2: Each epoch ends with an evolving process where the old committee transfers the key to a new committee.**

In an epoch \(e - 1\), the committee, denoted \(C^{(e-1)}\), manages the private key \(s\) and transfers it to a new committee \(C^{(e)}\) at the end of the epoch. The evolving process depicted in Fig. 3 contains several protocols to achieve the committee transition. Our scheme is designed to support the dynamic committee in hierarchical access structure while keeping the integrity of the private key.

In practice, the same set of nodes actively participate in committees of different epochs, i.e., the nodes with deep color in Fig. 3. As the committees may intersect, here we logically separate their roles for clear and flexible architectural illustration.

#### 4.2 Adversarial Model

Following prior work on the decentralized key management based on secret sharing [6], we consider a powerful adversary \(\mathcal{A}\) in our design. The node may be caught by \(\mathcal{A}\) at any time and be assumed to be corrupted until the end of each epoch. When the current epoch ends, \(\mathcal{A}\) needs to regain control of the nodes.

We assume the adversary is computationally bounded and limit \(\mathcal{A}\) to corrupt less than the thresholds \(k = (k_0, k_1, \cdots, k_{m-1})\) of the nodes in committee. Let \(k_i\) denote the threshold of level \(i\). It also reflects that nodes at higher level are harder to be corrupted. \(\mathcal{A}\) gets the shares from corrupted nodes and may control nodes to deviate from the protocol arbitrarily. The number of shares held by \(\mathcal{A}\) may exceed the threshold, but only less than \(k_i\) of each level are controlled in each epoch (i.e., \(\mathcal{A}\) may corrupt less than \(k_i\) nodes of level \(i\)). As noted above, shares need to be refreshed proactively to prevent leakage of the managed key.

Furthermore, observe that members of committees \(C^{(e-1)}\) and \(C^{(e)}\) are active during the evolving process between the epoch \(e - 1\) and \(e\). We further assume that \(\mathcal{A}\) may corrupt less than \(k\) nodes in the evolving process and control up to \((2 \cdot k - 1)\) nodes of each level at this time.
5 THE PROPOSED DESIGN

Our design goal is to establish a blockchain-based secret-managing system with hierarchical access structure where users can securely store their private keys.

It is known that Shamir’s (k, n)-threshold scheme [10] is perfect and ideal. Shares of the secret s are points at a univariate polynomial \( P(x) \) such that \( P(0) = s \). The scheme employs Lagrange interpolation to reconstruct s. To achieve the dynamic committee, CHURP [6] employs a bivariate polynomial \( B(x, y) \) with the degree \((k, 2k)\) such that \( B(0, 0) = s \) to manage the secret and proposes the handoff protocol, a dimension switch process to transfer the secret s to a new committee. In that scheme, a share is a univariate polynomial \( B(x, i) \) or \( B(i, y) \) where i is the index of a node.

Inspired by the above designs, our idea to resolve the security threat is to similarly use a bivariate polynomial to construct the basic secret-sharing polynomial. Instead, to enable its hierarchical access structure, HasDPSS employs Birkhoff interpolation to reconstruct the managed key and redesigns the process of dimension switch. Taking a bivariate polynomial \( B(x, y) \) with the degree \((k − 1, 2k − 1)\) such that \( B(0, 0) = s \). Let \( \delta = \sum_{i=0}^{k-1} k_i \), shares of the key s are the univariate polynomial \( B^{(\delta)}(x, v_j) \) or \( B^{(\delta)}(v_j, y) \), where \( i \) and \( v_j \) respectively represent the level and the identification of node j. \( B^{(\delta)}() \) is the \((\delta)\)-th partial derivative of \( B(x, y) \) respect to the parameter x (we define \( k_{−1} = 0 \)). The dimension switch process consists of three phases: Dimension Attenuation, Randomization and Dimension Recovery. Firstly, HasDPSS uses an attenuation protocol to switch the share \( B^{(\delta)}(x, v_j) \) to \( B^{(\delta)}(v_j, y) \). The attenuated share \( B^{(\delta)}(v_j, y) \) needs to be twice as many as the complete share \( B^{(\delta)}(x, v_j) \) when reconstructing the key s. To realize the system proactive refresh, HasDPSS then generates a random polynomial \( Q(x, y) \) with the degree \((k − 1, 2k − 1)\) such that \( Q(0, 0) = 0 \). The polynomial \( Q(x, y) \) is added to \( B(x, y) \) during the Randomization phase. Note that the additional polynomial \( Q(x, y) \) does not affect the reconstruction of s, but it randomizes the polynomial \( B^{(\delta)}(x, \cdot) \) or \( B^{(\delta)}(\cdot, y) \) of current shares. The share will be recovered to the complete share \( B^{(\delta)}(x, v_j) \) with the degree \((k − 1, 0)\) for the new committee in the last phase.

In addition, to detect and rule out the invalid shares during the evolving process, our idea is to craft a joint off-chain and on-chain design to post the commitment of polynomial B or Q on the blockchain, via customizing the Protocol 1 based on KZG Commitment [24], and the receiver verifies messages via this commitment.
5.2 System Initialization

The initialization stage of HasDPSS is mainly conducted by the user $U$ and the initial/current committee noted $C^{(0)}$. Protocol 2 illustrates the procedure of system initialization. Specifically, $U$ uses its own server to compute and distribute the shares of the private key $s$. $U$ firstly distributes the unique identification $v$ for each node in $N$. The determinant of the interpolation matrix consisting of all nodes’ $v$ based on $k$ is not equal to 0. The set $P$ is the points at the $(\delta)$-th partial derivative polynomial $B^{(\delta)}(x, y)$ of $B(x, y)$ with respect to the parameter $x$, where $B(0, 0) = s$ and $\delta = \sum k_i$. The node in $C^{(0)}$ receives the set $P$ and then employs the Lagrange interpolation to construct the polynomial $B^{(\delta)}(x, y)$ that is the share of the node. Using their shares, members of $C^{(0)}$ generate the commitments by the function $CreateCom$ in Protocol 1 and post them on chain.

5.3 The Evolving Process

The evolving process consists of three protocols: Attenuation Protocol, Randomization Protocol and Recovery Protocol. Without loss of generality, we describe here the evolving process for epoch $e = 1$ to $e$ in one round.

5.3.1 Phase 1. Dimension Attenuation. HasDPSS employs the Attenuation Protocol to construct the attenuated share $B^{(\delta)}(x, v_j)$ with the degree $(k - 1, 0)$ as shown in Protocol 3. Firstly, the honest nodes in $C^{(e-1)}$ compute the points of current share and generate the corresponding witnesses using the function $CreateWitness$ in Protocol 1. The points and witnesses are sent to corresponding nodes in the new committee via the off-chain channel. Hereafter, all operations related with secret shares take place in $C^{(e)}$. The node needs to verify the validity of received points via the on-chain KZG commitment and chooses any $(k - \delta + 1)$ verified points to construct the polynomial of attenuated share $B^{(\delta)}(x, v_j)$ via the Lagrange interpolation. Note that the attenuated share held by the node at this time is temporary and will not be used to restore the complete share. The node does not need re-compute the commitment of attenuated share and post it on chain.

5.3.2 Phase 2. Randomization. HasDPSS aims to achieve the decentralized randomization of shares without exposing its random polynomial. Protocol 4 shows the whole procedure of randomization. Members of the committee $C^{(e)}$ first generate their polynomials $Q_j(x, y)$ with the degree $(k - 1, 2k - 1)$ such that $Q_j(0, 0) = 0$. Let $Q(x, y) = \sum_{j=1}^{n} Q_j(x, y)$ be the randomized polynomial and be added to the basic polynomial $B(x, y)$. Each node $C_{(i,j)}$ respectively generates the sets $\{P^{(1)}_{i,j}\}_{i=1}^{n}$ including at least $(k - \delta)$ points at $Q_j(x, v_j)$, where $\delta = \sum k_i$ and $i$ is the level of node $C_{(i,j)}$. Then the set $P^{(1)}_{i,j}$ is sent to $C_{(i,j)}$ with the witnesses. $C_{(i,j)}$ needs to post the correspond commitment on chain. The receiving node $C_{(i,j)}$ first verifies the validity of all points and witnesses in each received message from $C_{(i,j)}$. We denote $\Lambda$ be the set consisting index of the verified message’s sender. The verified messages are used to construct $Q^{(2)}_{i,j} (x, v_j). C_{(i,j)}$ only needs to compute $Q^{(2)}_{i,j} (x, v_j) = \sum_{i\in \Lambda} \hat{Q}^{(2)}_{i,j} (x, v_j)$ and adds it to the current attenuated share $B^{(1)}(x, v_j)$. We use $B^{(1)}(x, v_j)$ to denote the randomized
attenuated share for $C_{(i,j)}$. Then $C_{(i,j)}$ computes the commitment of $B^{(\delta)}(x, v_j)$ and posts it on chain.

5.3.3 Phase 3. Dimension Recovery. Members of $C^{(e)}$ perform Protocol 5 to recover the new complete share $B^{(\delta)}(x, v_j)$ with the degree $(0, 2k - 1)$. Specifically, nodes compute the points of attenuated share and send them to other corresponding nodes along with the witnesses. The receiving node verifies the validity of received points and interpolates any $(2k - \delta)$ verified points to construct the polynomial of new complete share $B^{(\delta)}(x, v_j)$.

After above operations, $B^{(\delta)}(x, v_j)$ is the proactively refreshed share for dynamic-committee in the new epoch. Nodes post the new commitments of $B^{(\delta)}(x, v_j)$ on chain, which will be used in the next evolving process.

6 SECURITY ANALYSIS

For each evolving process $e$ as shown in Fig 2, we consider the three phases of HasDPSS: Dimension Attenuation, Randomization and...
Towards Dynamic and Reliable Private Key Management for Hierarchical Access Structure in Decentralized Storage

Protocol 5: Recovery: Recover complete shares and restore the system to the stable state

Input: Threshold $k = (k_0, k_1, \ldots, k_{m-1})$, committee $C^{(e)}$, nodes number $n^{(e)} = (n_{i_0}^{(e)}, n_{i_1}^{(e)}, \ldots, n_{i_{m-1}}^{(e)})$.

Output: Complete share $B^{(i)}(x, y)$ of each node in $C^{(e)}$.

$k \leftarrow \sum_{t=0}^{m-1} k_t, n = \sum_{t=0}^{m-1} n_{i_t}^{(e)}$, then $C^{(e)} = \left\{ C^{(e)}_{(i_t)}, i_t \right\}_{i_t=1}^{n}$;

Each node $C^{(e)}_{(i_t)}$ in $C^{(e)}$;

$\delta \leftarrow \sum_{t=0}^{i-1} k_t, pk_j \leftarrow$ public key of $C^{(e)}_{(i_t)}$;

for each node $C^{(e)}_{(i_t)}$ in $C^{(e)}$ with the same level $i$ do

$(\sigma, \tau) \leftarrow \text{CreateWith}(B^{(i)}(x, y), i, pk_j)$;

Send $(\sigma, \tau)$ to $C^{(e)}_{(i_t)}$;

end

(After honest nodes have completed above procedure)

Each node $C^{(e)}_{(i_t)}$ in $C^{(e)}$;

$\delta \leftarrow \sum_{t=0}^{i-1} k_t$;

Receive $n_{i_t}^{(e)}$ messages including points $\sigma$ and witnesses $\tau$;

for each message $(\sigma, \tau)$ from $C^{(e)}_{(i_t)}$ do

$pk_j \leftarrow$ public key of $C^{(e)}_{(i_t)}$;

$\mu \leftarrow \text{Com}_B^{(e)}$ of $C^{(e)}_{(i_t)}$ retrieved on-chain;

VerifyCom($\mu, i, \sigma, \tau, pk_j$);

end

Interpolate any $(2k - \delta)$ verified points to construct the complete polynomial $B^{(i)}(x, y)$;

Compute $\text{Com}^{(e)}_{B_y} \leftarrow \text{CreateCom}(B^{(i)}(x, y), pk_j)$ and post it on chain;

Dimension Recovery. Our communication model consists of on-chain communication and off-chain communication. The on-chain communication is only used to post the commitments of shares in different phases. The adversary $\mathcal{A}$ learns no extra information from these commitments by the hiding property of the commitment scheme based on the discrete log assumption [24]. Therefore, we only need to consider the secrecy of the off-chain communication.

Except for public inputs of the system, the information that can be caught by $\mathcal{A}$ includes: 1) Complete share $B^{(i)}(v, j)$ from corrupted nodes in committee $C^{(e-1)}$. 2) Attenuated share $B^{(i)}(x, v_j)$ during Dimension Attenuation. 3) Randomized polynomial $Q_j(x, y)$ and randomized attenuated share $B^{(i)}(x, v_j)$ during Randomization. 4) Complete share $B^{(i)}(x, v_j)$ during Dimension Recovery.

In particular, we have following theorems.

Theorem 1. If $\mathcal{A}$ corrupts less than $k = (k_0, k_1, \ldots, k_{m-1})$ nodes of each level in epoch $e - 1$ and epoch $e$ respectively, it learns no information about the managed key in Dimension Attenuation phase from their obtained shares.

Proof. Recall that in our security design, the bivariate polynomial $B(x, y)$ is used to share the managed key. For each level, it uses a different partial derivative of $B(x, y)$ respected to the parameter $x$ to represent shares. Therefore, the polynomials maintained by members of each level are different. Given $k$, the key can be reconstructed only if the number of shares obtained from each level satisfies the parameter $K = (\sum_{t=0}^{i} k_t, \sum_{t=0}^{i} k_t, \ldots, \sum_{t=0}^{i} k_t)$ of hierarchical access structure.

In the worst case, $\mathcal{A}$ corrupts $(k_0 - 1, k_1 - 1, \ldots, k_{m-1} - 1)$ nodes of each level in $C^{(e-1)}$, and these corrupted nodes are also in the new committee $C^{(e)}$, i.e., it obtains $(k_0 - 1, k_1 - 1, \ldots, k_{m-1} - 1)$ complete shares and 2 times attenuated shares.

Specifically, the complete shares of level $i$ are different univariate polynomials $B^{(i)}(v, j)$ with the degree $(k - \delta - 1)$, where $v_j$ is the identification of $j$ and other parameter can be calculated as follow:

$$k = \sum_{t=0}^{i-1} k_t, i = \sum_{t=0}^{i-1} k_t. \tag{4}$$

For the lowest level $m - 1$, the share’s polynomial with the degree $(k_{m-1} - 1)$, i.e., at least $k_{m-1}$ complete shares of level $m - 1$ can reconstruct the bivariate polynomial $B^{(i)}(x, y)$ and satisfy the access structure of our scheme. $\mathcal{A}$ now keeps $k_{m-1} - 1$ complete shares in this level. Given the feature of hierarchical access structure, $\mathcal{A}$ can use the share from a higher level than $m$ as an alternative. The proof for the other levels is similar. However, for the level 0, there are no other shares that can be used as an alternative for this level, $\mathcal{A}$ cannot use other shares to satisfy the access structure.

The attenuated shares of level $i$ are polynomials $B^{(i)}(x, v_j)$ with the degree $(2k - \delta - 1)$. Using (4), we can calculate the degree of level $m$’s polynomial is $(2k_{m-1} - 1)$. As the above description of the worst case, $\mathcal{A}$ may obtain $(2k_{m-1} - 2)$ attenuated shares, and only over $(2k_{m-1})$ attenuated shares can reconstruct the polynomial with the degree $(2k_{m-1} - 1)$. \hfill $\Box$

Theorem 2. If $\mathcal{A}$ corrupts less than $k = (k_0, k_1, \ldots, k_{m-1})$ nodes of each level in epoch $e$, $\mathcal{A}$ learns no information about the managed key in this phase from their obtained shares.

Proof. As the design of the Randomization phase, shares in new committee $C^{(e)}$ are attenuated shares, and nodes realize the share randomization by exchanging their respective randomized polynomials (by points). In the former phase, $\mathcal{A}$ has obtained $(k_0 - 1, k_1 - 1, \ldots, k_{m-1} - 1)$ attenuated shares from the corrupted nodes. $\mathcal{A}$ can use the received points of each corrupted node $C^{(e)}_{(i_t)}$ to generate the randomized polynomial $Q^{(i)}(x, v_j)$ and compute $B^{(i)}(x, v_j)$.

In the worst case, we assume that the shares caught by $\mathcal{A}$ in the Randomization phase are non-repetitive with the shares held from the corrupted node, i.e., $\mathcal{A}$ obtains 2 times of randomized attenuated shares. Recall the attenuated share with the degree $(2k - \delta - 1)$, it needs at least $2k_{m-1}$ shares of level $m$ to satisfy the hierarchical access structure. $\mathcal{A}$ only keeps $2k_{m-1} - 2$ shares of level $m$. However, it may use the shares from a higher level to be an alternative, the highest level of access structure cannot be satisfied. \hfill $\Box$

Theorem 3. If $\mathcal{A}$ corrupts less than $k = (k_0, k_1, \ldots, k_{m-1})$ nodes of each level in epoch $e$, and at least one node is honest in the Randomization phase, $\mathcal{A}$ learns no information about the managed key in Dimension Recovery phase from obtained shares.
7 EXPERIMENTS

7.1 Implementation

We implement the prototypes of HasDPSS and the comparative schemes with the same setting in Python. We compare the obtained results to show the superiority of HasDPSS. All the experiments are carried out on the machine with 2.60GHz Intel i7-6700HQ processors, 8 Cores and 16GB of RAM. The PBC library\(^1\) is used to execute polynomial operations of bilinear mapping. Besides, we fixed the irrelevant parameter in the experiment according to Table 1, and each experiment is repeated 1000 times to obtain the average value for statistical confidence.

7.2 Evaluation

7.2.1 Distribution Cost. In HasDPSS, users perform Protocol 2 to distribute shares for nodes in the initial committee.

The performance is shown in Fig. 4. We first fix the network size at 1000, and each increment of network hierarchy size is 1 until \(m\) reaches up to 10. A total of 10 sample points are selected for experiments. As shown in Fig. 4a, the fewer hierarchies lead to greater efficiency of share distribution. However, fewer hierarchies will undermine the security benefits of hierarchical management, and too many hierarchies cause the same problem. Following the same settings in [7], it is reasonable to divide the 1000-size network into 10 levels. Fig. 4b illustrates that HasDPSS is almost unaffected by the changing conditions of \(m\). When the network hierarchy size \(m\) increases from 1 to 10, the communication overhead of share distribution decreases slightly from 30.52MB to 30.33MB.

In the same network with 10 levels, Fig. 4c and Fig. 4d show the overhead comparison and variation trend in different network sizes \(N\). HasDPSS commonly requires more computing resources for share distribution, but less for communication than CHURP. Different from CHURP based on Lagrange interpolation, the Birkhoff interpolation-based scheme performs non-zero verification of the determinant for the interpolation matrix after generating identifications for nodes in \(\mathbb{N}\). Despite this, it only takes about 4.9 seconds for users to complete the calculation of share distribution for the 1000-size network with 10 levels, which can be acceptable.

7.2.2 Latency. The latency is measured by the total execution time per epoch, including the process of computing and communication. Fig. 5 describes the comparison of the latency of HasDPSS and CHURP under different conditions. When the number of nodes increases, the latency of HasDPSS does not increase greatly as shown in Fig. 5a. HasDPSS does not require full-node interaction in the process of switching attenuated and complete shares. The process is completed by the interaction of nodes at the same level. The performance of latency in HasDPSS is significantly better than that of CHURP for large committee sizes. HasDPSS outperforms CHURP in dwindling the latency by 34.51% with the committee of 100 nodes. An increase in the number of levels causes the latency to decrease significantly as shown in Fig. 5b. When the scale of committee is fixed, the more hierarchies divided, the fewer members of the same level, and the more obvious improvement in latency.

7.2.3 Communication Overhead. We simulate on-chain communication through shared memory and use total bits of the data to measure on-chain efficiency. Following the most existing blockchain

---

\(^1\)https://crypto.stanford.edu/pbc/
We compare HasDPSS with the following advanced and representative secret-sharing schemes to illustrate the feature and performance of HasDPSS as shown in Table 2.

We presented HasDPSS towards building a dynamic and reliable blockchain-based key-management scheme for hierarchical access structure in decentralized storage. HasDPSS can provide secure and efficient management of private keys without relying on a centralized authority, where users can securely store their keys in a decentralized form. HasDPSS introduces an evolving process to securely transfer and refresh shares, which achieves both the dynamic hierarchical key-management committee and the robustness against a powerful and active adversary. We conducted extensive experiments, and results showed the practical performance of our design. For future work, it would be interesting to explore the dynamic changing of thresholds. The support for adding or removing hierarchies without reconstructing the share polynomial will be extended in follow-up research.

9 ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grants 62102035, and China Postdoctoral Science Foundation (2022M713206). Besides, we thank for Junxin Liu (junxin@mail.bnu.edu.cn) to realize the implementation code of the system prototype.
REFERENCES