Blockchain-Based Data Sharing With Key Update for Future Networks

Liang Xue, *Member, IEEE*, Dongxiao Liu, *Member, IEEE*, Cheng Huang, *Member, IEEE*, Xuemin Shen, *Fellow, IEEE*, Weihua Zhuang, *Fellow, IEEE*, Rob Sun, *Member, IEEE*, and Bidi Ying, *Member, IEEE*

***Abstract*— Future networks incorporate artificial intelligence to enable smart resource management and adaptive service provisioning. With a heterogeneous architecture and a large number of users in future networks, transparent and decentralized data sharing is required to promote data circulation and break data silos, for which blockchain is a potential solution to allow intelligent access permission control. However, it remains a challenging task to achieve flexible authorization management for blockchain-based data sharing and efficient key update for multi-users in case of key exposure. In this paper, we propose an intelligent blockchain-based data-sharing scheme with key update for future networks. First, we design a new encryption scheme, where keywords of data are extracted using machine learning algorithms that are published on the blockchain. Then, keywords of data and time validity are used to encrypt different types of data for flexible data authorization. Second, using hierarchical identity-based encryption, we construct an efficient key update mechanism, where update tokens are generated by invoking a smart contract deployed on the blockchain to facilitate key and ciphertext updates. We formally prove that the proposed scheme can guarantee three essential security properties: forward security, post-compromise security, and collusion attack resistance. On-chain and off-chain experiment results are provided to demonstrate that the proposed scheme can achieve computational and communication efficiency for key and ciphertext updates.**

***Index Terms*— Blockchain, data sharing, key update, access control.**

I. Introduction

THE continuous advancement of communication networks and the Internet of Things (IoT) has promoted the development of new application fields such as smart factories and autonomous driving, which put forward higher requirements for the real-time, reliable, and massive data processing in the future networks. Traffic in future networks will be generated not only by humans but also by connected intelligent machines and bots. With the increased scale, complexity, and dynamics of networks, innovations in telecommunications and access networking are needed to cope with the stringent Quality of Service (QoS) requirements and to respond to various abnormalities timely. Artificial Intelligence (AI) provides a data-driven approach that can analyze a mass of data, train machine learning models, and facilitate informed decision-making. AI has the potential to proactively predict and effectively manage the network traffic flows based on the large volume of historical traffic data. Innovative AI-based solutions can be explored for resource management [2], network protocol design [3], network security [4], and so on. AI will play an important role in future intelligent networks to provide services and support applications far beyond the current communication systems.

To achieve intelligent network management, different network managers can train AI models based on the data from their owned network infrastructures [5]. However, due to the data barriers from different network managers, AI models can learn knowledge only from a part of network data and cannot exploit the full potential of AI [6]. To break the data barriers from various data owners, trusted, secure and fine-grained data sharing is required. Blockchain, as a distributed database that is shared among nodes in a blockchain network, can be utilized as a reliable and transparent controller of data sharing [7], [8]. By integrating blockchain and AI, intelligent blockchain has a potential to streamline data management functions and enable flexible data access control. Data owners and data requesters can join a blockchain network and publish the description of their data and the data requirements. To ease data management, data owners can associate their data with different keywords or attributes, such as vehicle kinetics (velocity, acceleration, etc), location, and wireless network information. The attributes can be obtained by machine learning algorithms in Natural Language Processing (NLP), which extract keywords from data [9], [10]. The algorithms are published on blockchain for data owners to use. For different types of data, there are different ML algorithms. By using a specific NLP algorithm on the blockchain, data owners with the same type of data can obtain the same attributes. It is convenient for data users to search for all data providers who have certain types of data. After obtaining the attributes, the data owner can encrypt the data with AND/OR combinations over the corresponding

Manuscript received 14 March 2022; revised 15 June 2022; accepted 30 June 2022. Date of publication 10 October 2022; date of current version 22 November 2022. This work was supported by the Research Grants from Huawei Technologies Canada and from the Natural Sciences and Engineering Research Council (NSERC) of Canada. An earlier version of this paper was presented in part at the 2022 IEEE International Conference on Communications (ICC 2022) [DOI: 10.1109/ICC45855.2022.9838811]. *(Corresponding author: Dongxiao Liu.)*

Liang Xue, Dongxiao Liu, Cheng Huang, Xuemin Shen, and Weihua Zhuang are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: liang.xue@uwaterloo.ca; dongxiao.liu@uwaterloo.ca; c225huan@uwaterloo.ca; sshen@uwaterloo.ca; wzhuang@uwaterloo.ca).

Rob Sun and Bidi Ying are with Huawei Technologies Canada, Ottawa, ON K2P 0J9, Canada (e-mail: rob.sun@huawei.com; bidi.ying@huawei.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2022.3213312.

Digital Object Identifier 10.1109/JSAC.2022.3213312

0733-8716 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

attributes. Considering that the size of data may be large and the storage of a blockchain transaction is limited, a data owner can encrypt the data and store the resulting ciphertext at a storage platform, such as a cloud server. A data owner can authorize a data requester to access the data and provide him with the decryption key if the requester pays for it, which can be achieved by using the cryptocurrency on the blockchain. When a user is authorized to access the data, the data owner sends only the corresponding attribute keys to the user.

Considering that secret keys of a data owner may be exposed due to side-channel attacks [11], [12] or system vulnerabilities, which poses a threat to data security, it is desired that the data owner can update its secret key periodically to improve data security, i.e., achieving forward security and post-compromise security. Forward security guarantees that, if a current private key is obtained by an attacker, the attack cannot exploit the key to decrypt the ciphertexts that are encrypted with the previous keys. Post-compromise ensures that the leakage of historical keys will not affect the security of ciphertexts correlated to the current time period. When a data owner updates its private key, the outsourced ciphertext should also be updated such that the old secret key cannot decrypt it. To achieve ciphertext update, a simple way is to download the ciphertext and encrypt it using a new secret key. However, this can bring large computation and communication overhead for the data owner to decrypt the original ciphertext, re-encrypt the data, and send the new ciphertext to the storage platform. Another issue is that, when the key and ciphertexts of a data owner are updated, how to efficiently distribute new keys to the authorized users so that they can continue accessing the data.

Updatable encryption (UE) can be used to achieve key and ciphertext updates. A data owner can send an update token to the data storage platform, which can rotate the ciphertext from the old secret key to the new secret key by using the update token. Most UE schemes employ symmetric encryption primitives to achieve the functionality. For example, Dan et al. propose two UE schemes, one is based on authenticated encryption, and the other one is based on a key-homomorphic pseudorandom function [13]. In the existing UE schemes, fine-grained data access control is not supported, and both symmetric and asymmetric systems are involved. Fugkeaw presents a secure data sharing scheme with key update, where attribute certificates are utilized to support the authorization and a third party is introduced to carry out key update [14]. The main focus of the scheme is to achieve attribute revocation with low cost for key update, instead of achieving forward security and post-compromise security.

In this paper, we propose a blockchain-based data access control scheme with key update, which enables a data owner to specify access policies for its data and update its key periodically. In our scheme, to achieve flexible data access control, different types of data are encrypted with different access policies over attributes. A data owner can authorize users to access its data by sending corresponding attribute keys. To enable efficient key and ciphertext update, there are time validity components in keys and ciphertexts. Only the time period of a key matches the time period of a ciphertext, and the access policy of the ciphertext is satisfied by the attributes correlated with the key, the ciphertext can be decrypted by the key. Based on the hierarchical identity-based encryption (HIBE), our scheme not only can derive the keys in a hierarchical way, but also can facilitate a new ciphertext being derived from an original ciphertext. Furthermore, user revocation can be supported. For the key and ciphertext update, the cloud server can deploy a smart contract, and committee members of the consortium blockchain can cooperatively generate the update tokens. Data owners are not involved in the update token generation process, which reduces the burden on the data owners. The contributions of this paper are as follows:

- We propose a scheme that achieves flexible data access control and efficient key and ciphertext update. Data owners can predefine access policies for their data based on a set of attributes, such that only authorized users can decrypt the data. By associating keys and ciphertexts with time periods, the time validity of a key is required for decrypting data. Data owners and authorized users can update their keys periodically to defend against key exposure risks;
- Using blockchain-based data sharing, data owners can broadcast data supply information, and data demanders can publish data requests on the blockchain. Key and ciphertext update tokens are generated by invoking a smart contract deployed on the consortium blockchain, which reduces the communication cost and computation cost of data owners for key updating and distribution. By utilizing the hierarchical tree-based structure, our scheme can support a large number of key and ciphertext updates at low cost;
- Under the decisional bilinear Diffie-Hellman exponent (DBDHE) assumption, we formally prove that the proposed scheme can achieve forward security, post-compromise security, and that key components of different users cannot be combined to decrypt a ciphertext. We evaluate the performance of the proposed scheme. The on-chain and off-chain simulation results demonstrate that the computation and communication costs of the scheme are low.

We organize the remainder of this paper as follows. In Section II, we describe the system model, design goals, and the security model. We introduce the building blocks in Section III and present the detailed construction of our scheme in Section IV. In Section V, we formally prove the security of the proposed scheme. The simulation results are shown in Section VI. We recap the related works in Section VII and summarize this study in Section VIII.

II. System Model and Security Model

In this section, we first describe our system model and design goals. Then, we present a security model for fine-grained data sharing with key and ciphertext update.

A. System Model

There are four parties involved in our system, i.e., a data owner (DO), data users, a storage platform (SP), and a blockchain, as shown in Fig. 1, and detailed in the following:

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

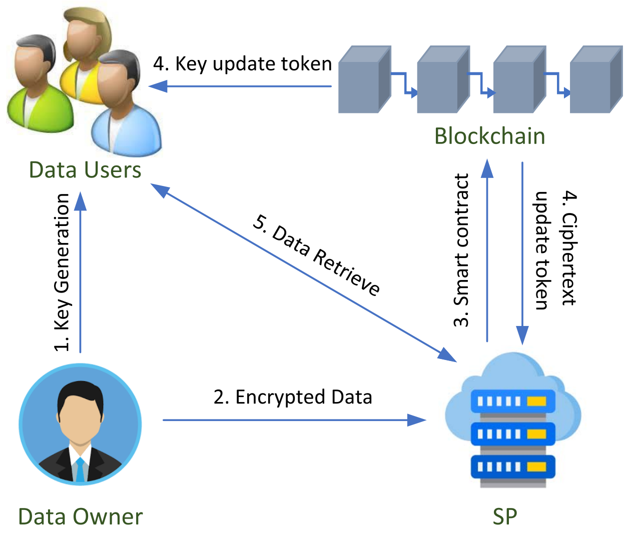


Fig. 1. System model.

- *DO* collects data from its smart devices. To reduce the data storage and management cost, DO outsources its data to a storage platform, such as a cloud server. To make better use of its data and gain profit, DO can share its data to data requesters if they pay for the data. To guarantee the confidentiality and flexible access of the data, different types of data are associated with different attributes, and DO encrypts data with an access policy built on various attributes. DO can generate keys for itself and authorized users. The users will obtain corresponding attribute keys for the data they need. By taking advantage of the blockchain and a smart contract, key and ciphertext update tokens are generated by committee members of the consortium blockchain. DO is not involved in the update token generation.
- *Data users* can obtain data supply information from the blockchain. Once a data user is authorized to access the data of DO, DO sends the corresponding attribute keys to the data user. A data user who own the attribute keys that satisfy with the policy of a ciphertext can decrypt the ciphertext. When the key and the ciphertext are updated, the authorized data users can obtain the key update token from the blockchain and update their keys based on the token;
- *SP*, which can be a cloud server, provides data storage services for DO. Data users and DO can retrieve data from SP. For ciphertext update, SP deploys a smart contract on the consortium blockchain. A set of committee members can invoke the smart contract to generate the ciphertext update token. Based on the token, SP can finish the ciphertext update by itself without learning any information of data. We assume that SP is honest but curious;
- *Blockchain* is a distributed ledger that can record data, such as Hyperledger Fabric, which is a consortium blockchain [15], [16]. All the nodes connected to the blockchain can read the data on the blockchain. Committee members of the consortium blockchain are responsible for maintaining the blockchain and validating transactions via consensus protocols.

B. Design Goals

Our design goals are as follows:

- Efficient key and ciphertext update – Considering the key exposure attacks, DO should be able to update its keys multiple times. Meanwhile, ciphertexts should also be updated synchronously. There should be minimal costs for DO to update the key and outsourced ciphertexts;
- Flexible access control – DO can share data with others and control who can access its data. Fine-grained access policies on data can be enforced. DO can also revoke the access right of users;
- Data security – Unauthorized users cannot obtain the plaintext data. Moreover, if an attacker obtains the current key of DO, it cannot decrypt the old ciphertexts, i.e., the ciphertexts correspond to a previous time period. Furthermore, the attacker cannot use the current key to decrypt the ciphertexts that correspond to a future time period.

C. Components of the Proposed Scheme

Our proposed flexible data access control scheme with key and ciphertext update consists of nine algorithms, which are listed as follows:

- Setup – In the Setup algorithm, DO defines the overall attribute set (SS) and generates the public parameters $(params)$ for various attributes and different time periods. Based on the public parameters, DO generates its public key (PK) and master secret key (MSK);
- Key Generation – In this algorithm, DO creates private keys for authorized data users. A hierarchical time tree with depth d is built by DO. A data user first submits a key generation request to DO, which includes an attribute set (S) that specifies the required data, and the time identity of the first time period $\vec{\tau}_d = \{\tau_1, \ldots, \tau_d\}$, where $\{\tau_i\}_{i\in[d]}$ are integers. Based on the attributes and the time identity, DO generates a private key (SK) for the user;
- Key Derivation – Given time identity $\vec{\tau}_{d+1} = \{\tau_1, \ldots, \tau_{d+1}\}$ and a private key of an entity for time identity $\vec{\tau}_d = \{\tau_1, \ldots, \tau_d\}$, the entity can generate a private key for time identity $\vec{\tau}_{d+1}$;
- Encrypt – DO defines an access policy for its data and encrypts the data with the access policy and the time identity of the first time period. Then, DO uploads the resulting ciphertext to SP;
- Decrypt – An entity (DO or a data user) can decrypt a ciphertext only if its attributes satisfy the defined access policy and the time validity of the private key matches the time period related to the ciphertext;
- Ciphertext Update Token Generation – Based on the $params$ generated by data owners, committee members of the blockchain can generate a ciphertext update token (Λ_j) by invoking a smart contract deployed on the blockchain. SP can update the ciphertext based on Λ_j;
- Ciphertext Update – Given a ciphertext with the decryptable time period j and Λ_j, SP can generate a new

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

ciphertext corresponding to time period $j+1$ without learning the plaintext;
- Key Update Token Generation – Key update tokens are also generated by committee members of the blockchain, but they are in a ciphertext form, and only unrevoked users can obtain the key update token Υ_j;
- Key Update – Based on Υ_j and a private key for time period j, an authorized user can generate a private key for time period $j+1$.

D. Security Model

We use a security game to describe our security model. In the security game, a challenger ($\mathcal{C}$) and an adversary ($\mathcal{A}$) are involved. At the beginning of the game, $\mathcal{A}$ needs to submit a challenge time period ($\mathbb{T}^*$) and a challenge access structure ($\mathbb{A}^*$), which are embedded in the challenge ciphertext. Then, $\mathcal{C}$ sets system parameters and sends public parameters to $\mathcal{A}$. To depict the ability of $\mathcal{A}$, $\mathcal{A}$ is allowed to make queries for private keys that cannot be utilized directly to decode the ciphertext given by the challenger. Adversary $\mathcal{A}$ can also make ciphertext update queries by submitting a ciphertext and the corresponding time period to $\mathcal{C}$. The formal security game is given as follows:

Init – $\mathcal{A}$ sends $\mathbb{A}^*$ and $\mathbb{T}^*$ to C;

Setup – $\mathcal{C}$ runs the Setup algorithm and sends PK of the scheme to $\mathcal{A}$;

Phase 1 Queries – $\mathcal{A}$ can submit the following queries to $\mathcal{C}$:
- $\mathcal{A}$ sends a private key query to $\mathcal{C}$, where the key query corresponds to an attribute set (S) and a time period ($\mathbb{T}$), which should not satisfy $S \in \mathbb{A}^* \wedge \mathbb{T} = \mathbb{T}^*$;
- $\mathcal{A}$ sends a ciphertext update query, which includes a ciphertext corresponding to time period $\mathbb{T}$. $\mathcal{C}$ computes and returns a ciphertext corresponding to time period $\mathbb{T}+1$;

Challenge – $\mathcal{A}$ sends two messages (m_0, m_1) that have the same length to $\mathcal{C}$. $\mathcal{C}$ randomly selects $b \in \{0,1\}$, and encrypts m_b under $\mathbb{A}^*$ and the first time period, then updates the ciphertext to time period $\mathbb{T}^*$. The resulting ciphertext (CT^*) is sent to $\mathcal{A}$;

Phase 2 Queries – $\mathcal{A}$ makes private key queries and ciphertext update queries to $\mathcal{C}$ as in Phase 1;

Guess – $\mathcal{A}$ returns a guess (b') of b.

For the above game, the advantage of $\mathcal{A}$ is calculated as $Adv_A = Pr[b' = b] - 1/2$.

Definition 1: A flexible data sharing scheme with key and ciphertext update is secure if, for any adversary, it can only have the advantage that is negligible for winning the above game.

III. Building Blocks

A. Linear Secret Sharing Scheme (LSSS)

An LSSS scheme (Π) over Z_p satisfies that: 1) the generated shares for a secret are vectors over Z_p; and 2) For Π, there is a share generating matrix (M) [17] that specifies and enforces the access policy of the secret, and there exists a function (ρ) that maps each row to an associated party, i.e., $\{\rho(i)\}_{i\in[l]}$ is the party corresponding to row i [18]. Assuming z is the secret to be shared, one first generates vector $\vec{\mu} = (z, t_2, \cdots, t_n)$, where $t_2, \cdots, t_n$ are selected randomly from Z_p. Then, it computes $M\vec{\mu}$ as the shares of z, and element $(M\vec{\mu})_i$ in the vector is the share of party ρ_i.

LSSS has the linear reconstruction property, which means if a set of parties satisfy the policy corresponding to M, they can reconstruct the secret (z).

B. HIBE

HIBE [19] is proposed to reduce the overhead of the key generation center (KGC) in an IBE system, where KGC needs to verify users' identity, and generate and transmit the private keys securely. In HIBE, identities are constructed according to a tree structure, and users with high-level identities can generate keys for lower-level identities. There are four algorithms in a HIBE system: (1) The Setup algorithm creates system parameters and a master key for users according to a security parameter and the depth of a tree; (2) Given a private key for ID_{k-1} of the depth $k-1$, the KeyGen algorithm can create a private key (d_{ID_k}) for a user (ID_k) of depth k; (3) The Encrypt algorithm can return a ciphertext (C) based on an identity (ID_k) and a message (M); (4) With d_{ID_k} and C, the Decrypt algorithm can output the plaintext of C.

C. DBDHE Assumption

DBDHE Assumption [17] is described below:

For a group (G) of prime order p, choose $a \in Z_p, h \in G$ randomly, and denote g^{β^i} as g_i, where g is a generator of G. Given $\vec{y} = (g, h = g^s, g_1, \cdots, g_q, g_{q+2}, \cdots, g_{2q}, B)$, an adversary needs to distinguish whether B is $e(g,g)^{\beta^{q+1}s}$ or a random group element (R) in G_T.

The advantage (ϵ) of an adversary ($\mathcal{B}$) in solving the above problem with an output ($out \in \{0,1\}$) is defined as

$$|Pr[\mathcal{B}(\vec{y}, B = e(g,g)^{\beta^{q+1}s}) = 0] - Pr[\mathcal{B}(\vec{y}, B = R) = 0]| \geq \epsilon.$$

Definition 2: The DBDHE assumption holds if no adversary can have an advantage that is non-negligible for solving the above DBDHE problem.

D. Zero-Knowledge Proof (ZKP)

ZKP allows a prover (P) to generate a proof π for a statement $R(Y, w) = 1$, where w is a witness for Y such that the statement holds [20]. By verifying π, a verifier (V) is convinced that P knows w without learning any information about w. For example, the statement can be $Y = g^w$, where g is a generator for a multiplicative cyclic group (G). Σ-protocols are 3-move ZKP protocols for proving statements related to discrete logarithms in prime-order groups.

A Σ-protocol consists of the following three algorithms [21]:
- Σ.Setup: Given a security parameter λ, the algorithm outputs a common reference string crs;
- Σ.ProofGen: This algorithm is run by a prover. Given a statement $R(Y, w) = 1$ and crs, the algorithm generates a proof π;

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

- Σ.Verify: This algorithm is run by a verifier. Given Y, crs, and π, the algorithm outputs 1 if π is a valid proof for the statement.

IV. Our Construction

A. Overview

To achieve flexible access control to DO's data, the data are encrypted with an access policy, which is built based on attributes related to the data. The attributes can be extracted by using a machine learning algorithm in NLP [9], [22]. The committee members of blockchain can choose an NLP algorithm and publish it on the blockchain, so that different data owners can use the same NLP algorithm to extract the attributes and encrypt the data with the policy over the attributes, and others can find data that matches their interests based on the attributes. Moreover, DO can also determine the attributes by itself. For data access, an authorized user's private key is associated with a set of attributes. Only the entities whose attributes satisfy the policy can decrypt the ciphertext. Meanwhile, to achieve time period-based key update, we add time validity into an entity's private key, and correlate a ciphertext with a decryptable time period. When an entity has attributes that satisfy the defined policy, and a private key that corresponds to the decryptable time period, the entity can decrypt the ciphertext. To defend against collusion attacks, i.e., two private keys that cannot decrypt a ciphertext separately could be combined to decrypt the ciphertext, we attach the time validity and the attributes tightly in an entity's private key.

For key updates, we construct a hierarchical time tree (T) [23], [24] and use its leaf nodes to represent different time periods. The leaf nodes are numbered from left to right to denote sequential time periods. Let the depth of T be d. For each node in T, there is a label that can be represented as $\{0,1\}^{\leq d}$. The label of the root node ($Root$) is 1. If the label of a node (n) is l, the left child of node n is under label $l0$, and the right child is labeled with $l1$. An example of T is shown in Fig. 2, where d is 4, and the labels of leaf nodes are $\{1000, \ldots, 1111\}$. The identity of a node in T is a vector, in which the elements are the labels of nodes from the root to that node. For example, the identity of a node under label 1000 is $(1, 10, 100, 1000)$. For a node in T, it can have a corresponding private key component. By using HIBE, with a private key component of a node and an identity of its child node, we can derive a private key component for the child node. In the tree, the sibling of a node is the other child of the node's parent. We denote the siblings of the nodes that are on the path from lf_i to $Root$ and are not ancestors of lf_j as set SN_i, where $i, j \in [1, 2^{d-1}]$, and $j < i$. For example, in Fig. 2, SN_1 includes nodes $(1001, 101, 11)$, and SN_3 includes nodes $(1011, 11)$.

To achieve key updates, in the first time period, an entity preserves a private key component corresponding to the first leaf node (lf_1) and private key components corresponding to the nodes in SN_1. The latter is used to derive the private key components for future time periods. When it comes to the next time period, an authorized user can update the private key,

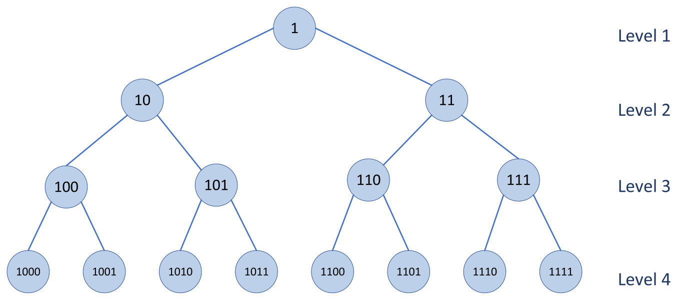


Fig. 2. Hierarchical tree.

i.e., deleting the private key components corresponding to the current time period, and generating private key components for the next time period.

To release the burden of DO for generating the key and ciphertext update tokens, in our design, a set of committee members of the blockchain cooperatively generate the update tokens by invoking the token generation smart contract. SP and authorized users can obtain the ciphertext and key update tokens from the blockchain.

B. Detailed Construction

1) Setup: DO defines an overall attribute set (SS) for its data, and the number of attributes in SS is denoted by U. DO builds a hierarchical time tree (T) with depth d. The identity of a node (n) at level k in T is denoted as $\vec{\tau}_k = (\tau_1, \tau_2, \ldots, \tau_k)$, where the elements are labels of the nodes from the Root to n. DO chooses two multiplicative cyclic groups G and G_T of prime order p, and $e : G \times G \rightarrow G_T$ is a bilinear map. Denote g to be a generator of G. $\hat{g}$ is a generator of the subgroup $\hat{G}$ of prime order Q of Z_p^*, where $Q = (p-1)/\gamma$, and γ is a small integer [25]. To generate system parameters for attributes in system and different time periods, DO randomly chooses $\alpha, \beta \in Z_p$, and computes g^β and $e(g,g)^\alpha$. Then, DO randomly selects $h_1, \ldots, h_U \in G$, $F_0, F_1, \ldots, F_d, \eta_1, \ldots, \eta_d, F'_2, \ldots, F'_d \in G$, and defines two hash functions $H : G_T \rightarrow Z_Q^*$ and $H_1 : \{0,1\}^* \rightarrow Z_p^*$.

A consortium blockchain is utilized to finish the ciphertext and key update token generation, where a number of committee members are involved. DO chooses an integer N as the number of committee members required. The public parameters are $param = (g, \hat{g}, g^\beta, e(g,g)^\alpha, F_0, \ldots, F_d, h_1, \ldots, h_d, \eta_1, \ldots, \eta_d, F'_2, \ldots, F'_d, H, H_1)$. The master secret key (MSK) is g^α. Some parameters and their definitions are listed in Table I.

2) Key Generation: In this algorithm, DO creates private keys for the authorized data users. For key generation, an entity first generates a key generation request, $Resq$, which includes its attribute set S, and the time identity of the first time period $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_d)$. DO first checks the validity of S. If it is invalid, DO refuses the request. Otherwise, DO generates a private key for the entity, which is the user's first time period key. The private key includes two components, one for attributes of the entity, the other for time validity. The first time period corresponds to the first leaf node (lf_1). DO generates a private key component for lf_1. Moreover, DO generates the

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SUMMARY OF SYMBOLS

Symbol	Definition
U	The number of attributes in system
g	A generator of G
g^{α}	The master secret key of DO
lf_i	The i-th leaf node
SN_i	The siblings of the nodes that are on the path from lf_i to $Root$, excluding the ancestors of lf_j, $j \le i$
$\vec{\tau}_k = \{\tau_1, \ldots, \tau_k\}$	The identity of a node n at level k
$D_{i,lf}$	The private key component for leaf node lf_i
$D_{i,\vec{\tau}_k}$	The private key component for a node in SN_i
CT_i	A ciphertext for time period i
C''_{lf_i}	The ciphertext component for leaf node lf_i
C''_{i,l_k}	The ciphertext component for a node at level k in SN_i

private key components for the nodes that are in set SN_1, i.e., the siblings of the nodes on the path from lf_1 to $Root$. For key generation, DO first randomly selects $t, v_\tau \in Z_p$, and computes a private key for the user as follows:

1) DO calculates

$$L = g^t, \qquad \forall x \in S \; K_x = h_x^t$$
$$D_0 = g^{v_\tau} \quad \{Q_j = (F'_j F_j)^{v_\tau}\}_{j=2,\ldots,d};$$

2) DO computes the private key component for the first leaf node (lf_1). Let the identity of lf_1 be $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_d)$. DO calculates

$$D_{1,lf} = g^{\alpha} g^{\beta t}(F_0 \prod_{i=1}^{d} F_i^{\tau_i})^{v_\tau};$$

3) DO computes private key components for the nodes in SN_1. For a node in SN_1 with identity $\vec{\tau}_k = (\tau'_1, \tau'_2, \ldots, \tau'_k)$, where $k \le d$, DO computes

$$D_{1,\vec{\tau}_k} = g^{\alpha} g^{\beta t}(F_0 \prod_{i=1}^{k} F_i^{\tau'_i})^{v_\tau}.$$

The private key of the user for the first time period is $SK_1 = \{L, \{K_x\}_{x\in S}, D_0, \{Q_j\}_{j=2,\ldots,d}, D_{1,lf}, \{D_{1,\vec{\tau}_k}\}_{\vec{\tau}_k \in SN_1}\}$.

3) Key Derivation: Given an entity's private key component of a node (N) with identity $\vec{\tau}_{k-1} = (\tau_1, \tau_2, \ldots, \tau_{k-1})$, and a child node of N with identity $\vec{\tau}_k = (\tau_1, \tau_2, \ldots, \tau_k)$, the entity can derive a private key component for the child node. To be specific, the entity chooses a random $\psi \in Z_p$, and calculates

$$D_{1,\vec{\tau}_k} = D_{1,\vec{\tau}_{k-1}} Q_k^{\tau_k} g^{\psi}.$$

Other components of the entity's private key remain the same.

4) Encrypt: When DO outsources a file to SP, it first generates a user identifier $uid \in Z_p$ and a file identifier $fid = H_1(fname)$, where $fname$ is the file name. Then, it encrypts the file using a symmetric encryption algorithm, such as AES, with a secret key ($K \in G_T$). After that, DO encrypts K with an access policy and the identity of lf_1. We represent the access policy as an LSSS access structure (M, ρ), in which M is an access matrix with size $l * n$, and ρ is a mapping function that maps a row of M to an attribute. We assume there are no two rows that map to the same attribute. To generate an encryption of K, DO randomly chooses $\vec{v} = (s, y_2, \ldots, y_n) \in Z_p^n$, and for $i = 1$ to l, it computes $\lambda_i = \vec{v} \cdot M_i$, where M_i is the i-th row of M. Then, DO computes a ciphertext as follows:

1) DO calculates

$$C = K \cdot e(g,g)^{\alpha s}, \quad C' = g^s$$
$$for\ i = 1, \ldots, l, \quad C_i = g^{\beta \lambda_i} h_{\rho_i}^{-s},$$
$$\{Q'_j = (F'_j F_j)^s\}_{j=2,\ldots,d},$$
$$E' = \eta_1^s, \quad \{E_j = \eta_j^s\}_{j=2,\ldots,d};$$

2) For the identity $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_d)$ of lf_1, DO computes

$$C''_{lf_1} = (F_0 \prod_{j=1}^{d} F_j^{\tau_j})^s;$$

3) DO computes a ciphertext component for each node in the sibling node set (SN_1) of lf_1. To be specific, for a node in SN_1 with identity $\vec{\tau}_k = (\tau'_1, \tau'_2, \ldots, \tau'_k)$, where $k \le d$, DO computes

$$C''_{1,l_k} = (F_0 \prod_{j=1}^{k} F_j^{\tau'_j})^s.$$

The ciphertext for the first time period is $CT_1 = \{C, C', C_1, \ldots, C_l, \{Q'_j\}_{j=2,\ldots,d}, E', \{E_j\}_{j=2,\ldots,d}, C''_{lf_1}, \{C''_{l_k}\}_{k\in SN_1}\}$ along with the description of $\mathbb{AS}$. Here, CT_1 denotes that the decryptable time period for the ciphertext is the first time period. After data encryption, DO randomly chooses k_f from G_T, and encrypts k_f by using the *Encrypt* algorithm such that only the authorized users can obtain k_f from the ciphertext ($Enc(k_f)$). Then, DO generates $pk_f = \hat{g}^{H(k_f)}$. Note that ($sk_f = H(k_f), pk_f = \hat{g}^{H(k_f)}$) is a private-public key pair. DO uploads uid, fid, $Enc(k_f)$, pk_f, the ciphertext of K, and the ciphertext of its data to SP, and sends the file storage information to the authorized users.

In the ciphertext, $\{Q'_j\}_{j=2,\ldots,d}$ and $\{E_j\}_{j=2,\ldots,d}$ are introduced to allow ciphertext update, so that an updated ciphertext can be matched with an updated key.

5) Decrypt: An entity can decrypt a ciphertext (CT) only if its attribute set (S) satisfies the defined access policy, which is denoted as $\mathbb{AS} = (M, \rho)$, and it has a private key corresponding to the decryptable time period of CT. Let c denote the decryptable time period of CT. To decrypt CT, the entity first defines set $I = \{i : \rho(i) \in S\}$, and $I \subset \{1, 2, \ldots, l\}$. Then, the entity computes $\{\omega_i \in Z_p\}_{i\in I}$, which are constants that satisfy $\sum_{i\in I} \omega_i \lambda_i = s$, where λ_i are shares of s. After that, the entity calculates

$$\mu = \prod_{i\in I}(e(L, C_i)e(C', K_{\rho(i)}))^{\omega_i}$$
$$= \prod_{i\in I} e(g,g)^{t\beta\lambda_i\omega_i}$$
$$= e(g,g)^{\beta s t}.$$

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

Then, the entity uses the private key component of leaf node lf_c, whose identity is $\vec{\tau}_d = \{\tau_1, \ldots, \tau_d\}$, to decrypt the ciphertext corresponding to lf_c. That is, the entity computes

$$K = C \cdot \mu \cdot e(D_0, C''_{lf_c})/e(C', D_{c,lf})$$

where $D_{c,lf}$ denotes the private key component corresponding to lf_c, and C''_{lf_c} is the ciphertext component associated with lf_c.

When $c = 1$, the correctness can be validated by:

$$\begin{aligned} &C \cdot \mu \cdot e(D_0, C''_{lf_1})/e(C', D_{1,lf}) \\ &= \frac{Ke(g,g)^{\alpha s} \cdot e(g,g)^{\beta st} \cdot e(g^{v_\tau}, F_0 \prod_{j=1}^{d} F_j^{\tau_j})^s}{e(g^s, g^\alpha g^{\beta t}(F_0 \prod_{j=1}^{d} F_j^{\tau_j})^{v_\tau})} \\ &= K. \end{aligned}$$

After obtaining K, the entity can decrypt the ciphertext using K.

6) Ciphertext Update Token Generation: In this algorithm, SP deploys an update token generation (UTG) smart contract on the consortium blockchain, by which a set of committee members can cooperate to generate a ciphertext update token [26]. We assume that the majority of the committee members are honest and do not collude with the users and SP. The SP can use the resulting token to update the stored ciphertext. In this way, DO is not involved in ciphertext update and token generation. If for a time period t, where $t \in [2^{d-1}]$, there are users that are revoked, i.e., the access policy is changed, DO randomly chooses a $k'_f \in G_T$, and encrypts it with the new policy such that only the unrevoked users can obtain k'_f. The resulting ciphertext is denoted as $Enc(k'_f)$. DO sends $Enc(k'_f)$ and $pk'_f = \hat{g}^{H(k'_f)}$ to the SP, which will be used to generate the update token for the next time period. As shown in Figure 3, there are five functions in smart contract UTG, which are described as follows:

- *Create*: This function is invoked when smart contract UTG is deployed on the blockchain. The SP publishes public parameters $pp = \{uid, g, \eta_1, \ldots, \eta_d, N, d_r, d_m\}$, where uid is the identifier of DO, and $g, \eta_1, \ldots, \eta_d, N$ are included in $param$ of DO. d_r denotes the reward for committee members, and d_m represents the deposit for each committee member.
- *Setup*: The SP uploads uid, fid, the current time period t, and $\{\overline{C_{k'_f}} = (Enc(k'_f), \overline{pk_f} = \hat{g}^{H(k'_f)}\}$. It also deposits reward d_r for token generation.
- *Register*: In this function, N committee members register themselves on the smart contract by submitting their public keys ($\{PK_i\}_{i\in[N]}$) and account IDs ($\{acc_i\}_{i\in[N]}$). Moreover, each committee member deposits funds d_m on the smart contract.
- *Upload*: In this function, each committee member uploads its token share $\{TK_{i,1}, TK_{i,2}, C_{i,1}, C_{i,2}, \pi_i\}$ to the smart contract, where $TK_{i,1}$ and $TK_{i,2}$ are used for token generation, $\{C_{i,1}, C_{i,2}\}$ is an ElGamal ciphertext and used to recover the key update token by authorized users, and π_i is a zero-knowledge proof that demonstrates the form of $\{TK_{i,1}, TK_{i,2}, C_{i,1}, C_{i,2}\}$ is correct. The details of generating token share by committee member Mem_i, where $i \in [N]$, are given in Algorithm 1. After receiving a share from Mem_i, this function first checks whether π_i is valid. If so, $\{TK_{i,1}, TK_{i,2}, C_{i,1}, C_{i,2}\}$ are recorded. After N committee members upload their parameters, the function computes the final ciphertext update token by calculating $TK_1^t = \prod_{j=1}^{N} TK_{i,1}$, and $TK_2^t = \prod_{j=1}^{N} TK_{i,2}$.
- Timeout: If a committee member (M_i) registers its public key on smart contract UTG but does not upload $\{TK_{i,1}, TK_{i,2}, C_{i,1}, C_{i,2}, \pi_i\}$ within a specified amount of time, its deposit is transferred to the SP.

```
Create:   Set $pp := \{uid, g, \eta_1, \ldots, \eta_d, N, d_r, d_m\}$
          Set $Deposit := \{\}$,  $Regs := \{\}$
          Set $T_{period} := \{\}$,  $Timer := \{\}$
          Set $TK1s := \{\}$,  $TK2s := \{\}$, $Enc := \{\}$
          Set $C_{k_f} := \{\}$,  $pk_f := \{\}$
Setup:    Upon receiving from $SP(uid, fid, t, \$d_r,$
          $\overline{C_{k_f}}, \overline{pk_f}, T_1, T_2, T_3)$
            $T_{period} := t$, $Timer := \{T_1, T_2, T_3\}$
            $n_1 = 0$, $n_2 = 0$
            $C_{k_f} := \overline{C_{k_f}}$, $pk_f := \overline{pk_f}$
            Assert $Ledger[CS] \le \$d_r$
            $Deposit[CS] := \$d_r$
            $Ledger[CS] := Ledger[CS] - \$d_r$
            $State := INIT$
Register: Upon receiving from $M_i(fid, \$d_m, PK_i,$
          $acc_i)$
            Assert $T_{now} \le T_1$ and $State = INIT$
            Assert $Ledger[M_i] \le \$d_m$ and $Regs[M_i] = 0$
            $Deposit[M_i] := \$d_m$
            $Ledger[M_i] := Ledger[M_i] - \$d_m$
            $Regs[M_i] = 1$
            $n_1 := n_1 + 1$
            IF $n_1 = N$
            $State := RegFinish$
Upload:   Upon receiving from $M_i(fid, TK_{i,1}, TK_{i,2}, C_{i,1},$
          $C_{i,2}, \pi_i)$
            Assert $T_1 < T_{now} < T_2 \wedge Regs[M_i] = 1 \wedge$
            $State := RegFinish$
            IF $1 = ZK.Verify(\pi_i)$
             $Deposit[M_i] := 0$
             $Ledger[M_i] := Ledger[M_i] + \$d_r/N$
             $TK1[M_i] := TK_{i,1}$, $TK2[M_i] := TK_{i,2}$
             $Enc[M_i] := C_{i,1}, C_{i,2}$
             $n_2 := n_2 + 1$
          IF $n_2 = N$
             $TK_1 := \prod_{i=1}^{N} TK1[M_i]$, $TK_2 := \prod_{i=1}^{N} TK2[M_i]$
             $TK1s[fid][t].push(TK_1)$
             $TK2s[fid][t].push(TK_2)$
             $State := UplFinish$
Timeout:  Upon receiving from $CS(M_i)$
            Assert $T_3 \le T_{now}$
            Assert $State := RegFinish \wedge Deposit[M_i] := \$d_m$
            $Deposit[M_i] := 0$
            $Ledger[CS] := Ledger[CS] + \$d_m$
```

Fig. 3. Pseudocode of smart contract UTG.

When a ciphertext is updated from time period t to time period $t+1$, the ciphertext update token generated by committee members is $\Lambda_t = (TK_1^t, TK_2^t)$, which is recorded on the blockchain.

7) Ciphertext Update: Given the update token Λ_t, which is $(TK_1^t = g^{s'}, TK_2^t = g^{\psi_t} \cdot \prod_{i=1}^{t} \eta_i^{s'})$, and the ciphertext stored

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Token Share Generation($Mem_i, \eta_1, \ldots, \eta_t$)

```
1: Randomly choose $s'_i, \psi_{t_i} \in Z_p^*$, and compute $TK_{i,1} = g^{s'_i}$, $TK_{i,2} = g^{\psi_{t_i}} \cdot \prod_{i=1}^{t} \eta_i^{s'_i}$.
2: Encrypt $g^{\psi_{t_i}}$ with $pk_f$ in contract UTG, which is actually $\overline{pk_f}$, using ElGamal encryption, i.e., randomly select $k_i \in Z_p$, and compute $C_{i,1} = g^{k_i}$, $C_{i,2} = g^{\psi_{t_i}} \cdot pk_f^{k_i}$.
3: Generate a zero-knowledge proof $\pi_i$ with the witness $(s'_i, \psi_{t_i})$ for statement $st = \{TK_{i,1} = g^{s'_i} \wedge TK_{i,2} = g^{\psi_{t_i}} \cdot \prod_{i=1}^{t} \eta_i^{s'_i} \wedge C_{i,1} = g^{k_i}, C_{i,2} = g^{\psi_{t_i}} \cdot pk_f^{k_i}\}$. The proof can be generated by using the $\Sigma.ProofGen$ algorithm.
```

on SP corresponding to time period t, SP first computes

$$\frac{e(C', g^{\psi_t} \cdot \prod_{i=1}^{t} \eta_i^{s'})}{e(g^{s'}, E')} = \frac{e(g^s, g^{\psi_t} \cdot \prod_{i=1}^{t} \eta_i^{s'})}{e(g^{s'}, \prod_{i=1}^{t} \eta_i^{s})}$$
$$= e(g^s, g^{\psi_t}).$$

Then, SP updates the ciphertext (CT_t) as follows:

1) Calculate $\overline{C} = C \cdot e(g^s, g^{\psi_t})$;

2) Calculate $\overline{E}' = E' \cdot E_{t+1}$ and delete E_{t+1};

3) If lf_t is a left child node of its parent, SP deletes C''_{lf_t}, and set C''_{t,l_d} in CT_t as $C''_{lf_{t+1}}$; Otherwise, the process is as follows: SP first deletes C''_{lf_t}. Letting the identity of lf_{t+1} be $(\tau_1, \tau_2, \ldots, \tau_{d_{t+1}})$ and the identity of lf_{t+2} be $(\tau_1, \tau_2, \ldots, \tau_{d_{t+2}})$, SP generates the new ciphertext for nodes lf_{t+1} and lf_{t+2} as $\overline{C}''_{lf_{t+1}} = C''_{t,l_{d-1}} \cdot (Q'_d)^{\tau_{d_{t+1}}}$ and $\overline{C}''_{lf_{t+2}} = C''_{t,l_{d-1}} \cdot (Q'_d)^{\tau_{d_{t+2}}}$, which is also $\overline{C}''_{t+1,l_d}$. After that, SP deletes $C''_{t,l_{d-1}}$. If there is no $C''_{t,l_{d-1}}$ stored, SP generates $\overline{C}''_{lf_{t+1}}$ and $\overline{C}''_{lf_{t+2}}$ based on its ancestor node in SN. Given the ancestor node N_k at level k of the tree, SP derives the ciphertext components for lf_{t+1} and the nodes in SN_{t+1}. For nodes in SN_{t+1} with identity $\vec{\tau}_j = (\tau_1, \tau_2, \ldots, \tau_j)$, $j \leq d$, SP calculates C''_{t+1,l_j} as $C''_{t,l_k} \prod_{i=k+1}^{j} (Q'_i)^{\tau_i}$. For node lf_{t+1} with identity $(\tau_1, \tau_2, \ldots, \tau_{d_{t+1}})$, SP computes

$$C''_{lf_{t+1}} = C''_{t,l_k} \prod_{i=k+1}^{d} (Q'_i)^{\tau_i}.$$

Then, it deletes the ciphertext components related to the ancestors of lf_{t+1}.

8) Key Update Token Generation: DO and authorized users can obtain the key update token from the blockchain. The process is as follows: 1) For time period t, it downloads the corresponding $\{C_{i,1}, C_{i,2}\}_{i \in [N]}$ and $Enc(k'_f), pk'_f\}$ from the blockchain; 2) It decrypts $Enc(k'_f)$ and obtains k'_f. Then, it generates $sk'_f = H(k'_f)$ and decrypts $\{C_{i,1}, C_{i,2}\}_{i \in [N]}$ with sk'_f; 3) After obtaining $g^{\psi_{t_i}}$, it calculates $g^{\psi_t} = \prod_{i=1}^{N} g^{\psi_{t_i}}$, which is the key update token and can be used to update the current private key.

9) Key Update: Given private key SK_t for time period t and a key update token (g^{ψ_t}), DO or an authorized user can generate private key SK_{t+1} for time period $t+1$. The process is as follows:

If leaf node lf_t is a left child node of its parent, it deletes $D_{t,lf}$. Then, it updates the private key components of the nodes ($D_{t,\vec{\tau}_d}$) in SN_t. That is, for $D_{t,\vec{\tau}_k}$, where $k \leq d$, it computes $D_{t+1,\vec{\tau}_k} = D_{t,\vec{\tau}_k} \cdot g^{\psi_t}$. After that, it sets $D_{t+1,\vec{\tau}_d}$ as $D_{t+1,lf}$, which is the private key component corresponding to leaf node lf_{t+1};

Otherwise, i.e., leaf node lf_t is a right child node of its parent, it deletes $D_{t,lf}$. Then, for $D_{t,\vec{\tau}_k}$, where $k \leq d$, it calculates $D_{t+1,\vec{\tau}_k} = D_{t,\vec{\tau}_k} \cdot g^{\psi_t}$. After that, based on the identities of lf_{t+1} and lf_{t+2}, it derives $D_{t+1,lf}$ and $D_{t+2,lf}$ from $D_{t+1,\vec{\tau}_{d-1}}$ using Key Derivation algorithm. Note that $D_{t+2,lf}$ is also $D_{t+1,\vec{\tau}_d}$. After that, it deletes $D_{t+1,\vec{\tau}_{d-1}}$. If the parent node (PN) of lf_{t+1} is not in SK_t, it first derives the private key component for PN using the private key component of a PN's ancestor that is in SK_t. Then, it derives the private key components for lf_{t+1} and lf_{t+2} using PN's private key component. After that, it deletes the private key components of lf_{t+1}'s ancestors.

V. Security Proof

In this section, we prove that the proposed scheme achieves the desired security properties.

Theorem 1: The proposed scheme is selective-identity secure under the chosen plaintext attacks (IND-sID-CPA) if the decisional BDHE assumption holds.

Suppose an adversary, $\mathcal{A}$, has the advantage, ϵ, in attacking the proposed scheme, then we can construct a Challenger, $\mathcal{C}$, which has a non-negligible probability in solving the decisional BDHE problem.

Based on our security model, two parties, the challenger ($\mathcal{C}$) and the adversary ($\mathcal{A}$) are involved in the security game. $\mathcal{C}$ has a decisional BDHE challenge

$$\vec{y} = (g, g^s, g^{\beta}, \ldots, g^{\beta^q}, , g^{\beta^{q+2}}, \ldots, g^{\beta^{2q}}, B \in G_T).$$

$\mathcal{C}$ needs to identify whether B is $e(g,g)^{\beta^{q+1}s}$ or a randomly chosen number from G_T. If $\{\vec{y}, B\}$ is a BDHE tuple, i.e., $B = e(g,g)^{\beta^{q+1}s}$, $\mathcal{C}$ returns 1; Otherwise, $\mathcal{C}$ returns 0.

Following the conventions of security proof for public-key encryption schemes [19], [27], as shown in Fig. 4, there are six phases in our security proof: *Init*, *Setup*, *Phase 1 Queries*, *Challenge*, *Phase 2 Queries*, *Guess*, which are briefly described as follows:

- **Init** – $\mathcal{A}$ sends a challenge access structure and a challenge time period to $\mathcal{C}$;
- **Setup** – $\mathcal{C}$ generates system parameters and sends public parameters to $\mathcal{A}$;
- **Phase 1 Queries** – $\mathcal{A}$ can send private key queries and ciphertext update queries to $\mathcal{C}$, which returns corresponding responses to $\mathcal{A}$;
- **Challenge** – $\mathcal{A}$ submits two messages (K_0, K_1) to $\mathcal{C}$, which randomly selects one message ($\{K_b\}_{b \in \{0,1\}}$) and generates an encryption (C_b) of the chosen message. Then, $\mathcal{C}$ sends C_b to $\mathcal{A}$;
- **Phase 2 Queries** – $\mathcal{A}$ sends the same types of queries as Phase 1. $\mathcal{C}$ returns the responses;
- **Guess** – Given C_b, $\mathcal{A}$ sends a guess of b to $\mathcal{C}$.

The detailed security proof is as follows:

Init – Let the depth of the hierarchical tree be d. $\mathcal{A}$ generates a challenge access structure ($\mathbb{AS} = (M^*, \rho^*)$), in which M^*

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

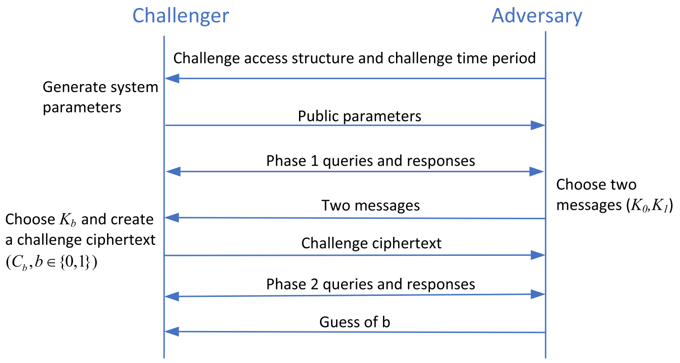

Challenger
Adversary
Challenge access structure and challenge time period
Generate system parameters
Public parameters
Phase 1 queries and responses
Choose two messages (K_0,K_1)
Two messages
Choose K_b and create a challenge ciphertext (C_b, $b \in \{0,1\}$)
Challenge ciphertext
Phase 2 queries and responses
Guess of b

Fig. 4. Phases of the security proof.

is of size $l^* \times n^*$, and $l^*, n^* \leq q$. $\mathcal{A}$ also generates a challenge time period $\mathbb{T}^*$ under identity ($\vec{\tau}^* = \{\tau_1^*, \ldots, \tau_d^*\}$). $\mathcal{A}$ sends (M^*, ρ^*) and $\vec{\tau}^*$ to $\mathcal{C}$.

Setup – In this algorithm, $\mathcal{C}$ generates the public parameters. $\mathcal{C}$ first sets g^β to the same element in $\vec{y}$. Then, $\mathcal{C}$ randomly selects $\alpha' \in Z_p$, and generates $e(g,g)^\alpha = e(g^\beta, g^{\beta^q})e(g,g)^{\alpha'}$, which implicitly sets

$$\alpha = \alpha' + \beta^{q+1}.$$

For F_0, $\mathcal{C}$ generates a random number, ξ, and sets $F_0 = g^\xi \prod_{i=1}^q g^{a^{q-i+1}\tau_i^*}$. For $j \in [1,d]$, $\mathcal{C}$ chooses a random number, γ_j, and generates $F_j = g^{\gamma_j}/g^{\beta^{q-j+1}}$. Then, $\mathcal{C}$ generates F_j' as

$$F_j' = g^{\theta_j} \cdot g^{\beta^{q-j+1}}.$$

For h_x, where $x \in [1,U]$, $\mathcal{C}$ first randomly selects $z_x \in Z_p$. If x does not appear in the challenge access set, $\mathcal{C}$ sets $h_x = g^{z_x}$; Otherwise, there exists an i, $i \in \{1,\ldots,l\}$, which satisfies $\rho^*(i) = x$. $\mathcal{C}$ generates h_x as

$$h_x = g^{z_x} \cdot g^{\beta M_{i,1}^*} \cdot g^{\beta^2 M_{i,2}^*} \ldots g^{\beta^{n^*}} M_{i,n^*}^*.$$

Finally, for $i \in [1,d]$, $\mathcal{C}$ randomly picks $\zeta_i \in Z_p$, and generates $\eta_i = g^{\zeta_i}$. Then, $\mathcal{C}$ sends $param = (g, g^\beta, e(g,g)^\alpha, F_0, \ldots, F_d, h_1, \ldots, h_U, F_2', \ldots, F_d', \eta_1, \ldots, \eta_d)$ to $\mathcal{A}$.

Phase 1 – In this phase, $\mathcal{C}$ responds to private key queries and ciphertext queries of $\mathcal{A}$.

For the private key queries, consider that $\mathcal{A}$ sends a private key query which corresponds to attribute set S and time period $\mathbb{T}$, and at least one of the following conditions must hold for the query:

- S does not satisfy $\mathbb{AS}$;
- $\mathbb{T}$ is not equal to $\mathbb{T}^*$.

Case 1: S does not match $\mathbb{AS}$. $\mathcal{C}$ creates a private key as follows.

For an unauthorized set (S), according to the properties of LSSS, there exists a vector ($\vec{\omega} = (\omega_1, \ldots, \omega_{n^*})$), where $\omega_i \in Z_p$ and $\omega_1 = -1$ [17]. Moreover, for each i that satisfies $\rho^*(i) \in S$, $\omega \cdot M_i^*$ holds. $\mathcal{C}$ finds the vector ($\vec{\omega}$) and randomly chooses $\mu \in Z_p$. Then, $\mathcal{C}$ implicitly defines t as

$$t = \mu + \omega_1\beta^q + \omega_2\beta^{q-1} + \cdots + \omega_{n^*}\beta^{q-n^*+1}.$$

In the private key, L is set to be

$$L = g^t = g^\mu \prod_{i=1,\ldots,n^*} (g^{\beta^{q-i+1}})^{\omega_i}.$$

For K_x, where $x \in S$, if x does not appear in the challenge access set, $\mathcal{C}$ sets $K_x = L^{z_x}$; Otherwise, attribute x is used in $\mathbb{AS}$. To generate K_x, there must not exist the term, $g^{\beta^{q+1}}$, since $\mathcal{C}$ cannot simulate it. For $h_x = g_x^z \cdot g^{\beta M_{i,1}^*} \cdot g^{\beta^2 M_{i,2}^*} \ldots g^{\beta^{n^*}} M_{i,n^*}^*$, and

$$t = \mu + \omega_1\beta^q + \omega_2\beta^{q-1} + \cdots + \omega_{n^*}\beta^{q-n^*+1}$$

notice that, in h_x^t, all terms in the form $(g^{\beta^{q+1}})$ derive from $g^{\beta^j M_{i,j}^* \cdot \omega_j a^{q-j+1}}$ for some j. Since we have $M_i^* \cdot \vec{\omega} = 0$, the terms that include $g^{\beta^{q+1}}$ cancels. Thus, the resulting K_x for x such that $\rho^*(i) \in S$ has the following form

$$K_x = L^{z_x} \prod_{j=1,\ldots,n^*} \left(g^{r\cdot\beta^j\cdot M_{i,j}^*} \prod_{k=1,\ldots,n^*,k\neq j} g^{\beta^{j+q+1-k}\cdot\omega_k\cdot M_{i,j}^*} \right).$$

For private key components for time period $\mathbb{T}$, $\mathcal{C}$ randomly chooses $v_\tau \in Z_p$, and sets $D_0 = g^{v_\tau}$. Then, $\mathcal{C}$ computes private key components as follows:

(1) $\mathcal{C}$ computes the private key component for the first leaf node (lf_1)

$$D_{1,lf} = g^\alpha g^{\beta t}(F_0 \prod_{j=1}^d F_j^{\tau_j})^{v_{\tau_k}}$$
$$= g^{\alpha'} g^{\beta\mu} \prod_{i=2,\ldots,n^*} (g^{q-i+2})^{\omega_i} (F_0 \prod_{j=1}^d F_j^{\tau_j})^{v_{\tau_k}};$$

(2) DO computes private key components for the nodes in SN_1. For a node in SN_1 with identity $\vec{\tau}_k = (\tau_1', \tau_2', \ldots, \tau_k')$, where $k \leq d$, DO computes

$$D_{1,\vec{\tau}_k} = g^\alpha g^{\beta t}(F_0 \prod_{j=1}^k F_i^{\tau_i'})^{v_\tau}$$
$$= g^{\alpha'} g^{\beta\mu} \prod_{i=2,\ldots,n^*} (g^{q-i+2})^{\omega_i} (F_0 \prod_{j=1}^k F_j^{\tau_j'})^{v_\tau};$$

(3) $\mathcal{C}$ randomly chooses ψ_i, where $i \in \{2,\ldots,\mathbb{T}-1\}$, and updates the private key based on Key Update algorithm until $\mathcal{C}$ obtains the private key for time period $\mathbb{T}$.

Note that, in g^α, there exits a term, $g^{a^{q+1}}$, which cannot be simulated by $\mathcal{C}$. However, since $\omega_1 = -1$, we have $g^{-a^{q+1}}$ in $g^{\beta t}$, making the terms of $g^{a^{q+1}}$ cancel out.

Then, $\mathcal{C}$ computes $Q_j = (F_j F_j')^{v_\tau}$, for $j = 2, \ldots, d$.

Case 2: $\mathbb{T}$ is not equal to $\mathbb{T}^*$. $\mathcal{C}$ computes the private key as follows:

(1) $\mathcal{C}$ randomly picks $t \in Z_p$, and calculates $L = g^t$, and $K_x = h_x^t$ for $\forall x \in S$.

(2) For the private key component for $lf_{\mathbb{T}}$, $\mathcal{C}$ first generates private key components for lf_1. Then, $\mathcal{C}$ derives the key components for $lf_{\mathbb{T}}$ using Key Update algorithm. If the challenge time period is not the first time period, the identities of lf_1 and the nodes in SN_1 are different from identity

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

$\vec{\tau}^* = \{\tau_1^*, \ldots, \tau_c^*\}$ of the challenge time period. For the identity of lf_1 or the identity of a node in SN_1, there exists a value of u such that $\tau_u \neq \tau_u^*$. Thus, $\mathcal{C}$ can create a private key for identity $\vec{\tau}_u = (\tau_1, \ldots, \tau_u)$, and then utilize it to obtain the key for $\vec{\tau}$, which is $\vec{\tau} = (\tau_1, \ldots, \tau_u, \ldots, \tau_k)$. To create the private key for $(\tau_1, \ldots, \tau_u)$, $\mathcal{C}$ randomly picks $\bar{u} \in Z_p$, and sets v_τ as $\beta^u/(\tau_u - \tau_u^*) + \bar{u}$. Then,

$$D_0 = g^{v_\tau} = g^{\frac{\beta^u}{\tau_u - \tau_u^*}} g^{\bar{u}}.$$

For $D_{1,\vec{\tau}_u}$, let y_i be g^{β^i}. We have

$$\begin{aligned}
&D_{1,\vec{\tau}_u}\\
&= g^{\alpha} g^{\beta t}(F_0 F_1^{\tau_1} \ldots F_u^{\tau_u})^{v_\tau}\\
&= g^{\alpha'} \cdot y_{q+1} \cdot (g^{\xi + \sum_{i=1}^{u} \tau_i \gamma_i} \prod_{i=1}^{u} y_{q-i+1}^{\tau_i^* - \tau_i} \prod_{i=u+1}^{q} y_{q-i+1}^{\tau_i^*})^{v_\tau}\\
&= g^{\alpha'} \cdot y_{q+1} \cdot (y_{q-u+1}^{\tau_u^* - \tau_u})^{v_\tau} \cdot (g^{\xi + \sum_{i=1}^{u} \tau_i \gamma_i} \prod_{i=1}^{u-1} y_{q-i+1}^{\tau_i^* - \tau_i}\\
&\qquad \times \prod_{i=u+1}^{q} y_{q-i+1}^{\tau_i^*})^{v_\tau}\\
&= g^{\alpha'} y_{q-u+1}^{\bar{u}(\tau_u^* - \tau_u)} \cdot (g^{\xi + \sum_{i=1}^{u} \tau_i \gamma_i} \prod_{i=1}^{u-1} y_{q-i+1}^{\tau_i^* - \tau_i} \prod_{i=u+1}^{q} y_{q-i+1}^{\tau_i^*})^{v_\tau}.
\end{aligned}$$

After obtaining $D_{1,\vec{\tau_u}}$, $\mathcal{C}$ can obtain the private key for node lf_1 with identity $\vec{\tau}_d = (\tau_1, \ldots, \tau_d)$ by calculating $D_{1,lf} = D_{1,\vec{\tau}_u} \cdot \prod_{i=u+1}^{d} (F_i^{v_\tau})^{\tau_i}$, where $F_i^{v_\tau}$ can be calculated by $\mathcal{C}$. Similarly, for a node in SN_1 with identity $\vec{\tau}_k = (\tau_1', \ldots, \tau_k')$, where $k \leq d$, $\mathcal{C}$ calculates $D_{1,lf} = D_{1,\vec{\tau}_u} \prod_{i=u+1}^{k} (F_i^{v_\tau})^{\tau_i'}$. Then, $\mathcal{C}$ can obtain the private key components for $lf_{\mathbb{T}}$ and the nodes in $SN_{\mathbb{T}}$ by using Key Update algorithm with the chosen g^{ψ_i}, where $i \in \{2, \ldots, \mathbb{T}-1\}$.

If the challenge time period is the first time period, $\mathcal{C}$ generates the private key components only for the nodes in SN_1, and generates the private key components for $lf_{\mathbb{T}}$ and the nodes in $SN_{\mathbb{T}}$ by using the preceding method.

For Q_j, where $j = 2, \ldots, d$, $\mathcal{C}$ can compute $Q_j = (F_j' F_j)^{v_\tau}$.

For a ciphertext update query, given a ciphertext corresponding to time period $\mathbb{T}-1$, $\mathcal{C}$ generates a ciphertext corresponding to time period $\mathbb{T}$ as follows.

Based on the $g_{\mathbb{T}}^{\psi}$ value used in the Key Update algorithm, $\mathcal{C}$ updates the ciphertext based on the Ciphertext Update algorithm. Then, $\mathcal{C}$ returns the resulting ciphertext.

Challenge – When $\mathcal{A}$ finishes Phase 1, it submits two messages, $K_0, K_1 \in G$, to $\mathcal{C}$. $\mathcal{C}$ picks a random number, $b \in \{0, 1\}$, and creates a ciphertext for K_b as follows:

(1) Note that there are g^s and B in the decisional BDHE challenge $\vec{y}$. $\mathcal{C}$ sets $C = K_b \cdot B \cdot e(g^s, g^{\alpha'})$, and $C' = g^s$.

For $i = 1, \ldots, l$,

$$\begin{aligned}
C_i &= g^{\beta \lambda_i} h_{\rho_i}^{-s}\\
&= g^{\beta \lambda_i} (g^{z_{\rho_i}} \cdot g^{\beta M_{i,1}^*} \cdot g^{\beta^2 M_{i,2}^*} \ldots g^{\beta^{n^*}} M_{i,n^*}^*)^{-s}.
\end{aligned}$$

Since $\mathcal{C}$ does not know the terms associated with $g^{\beta^i s}$, it cannot directly compute C_i values. To simulate them, $\mathcal{C}$ can carefully craft λ_i such that $g^{\beta^i s}$ can be canceled out. Since $\lambda_i = \vec{v} \cdot M_i^*$, $\mathcal{C}$ can randomly choose $\epsilon_2, \ldots, \epsilon_{n^*} \in Z_p$, and set $\vec{v}$ as

$$\vec{v} = (s, s\beta + \epsilon_2, s\beta^2 + \epsilon_3, \ldots, s\beta^{n^*-1} + \epsilon_{n^*}).$$

Thus,

$$\begin{aligned}
g^{\beta \lambda_i} &= g^{\beta \cdot \vec{v} \cdot M_i^*}\\
&= g^{(s\beta) M_{i,1}^* + (s\beta^2 + \epsilon_2 \beta) M_{i,2}^* + \cdots + (s\beta^{n^*} + \epsilon_{n^*} \beta) M_{i,n^*}^*}.
\end{aligned}$$

The resulting C_i is given by $C_i = (\prod_{j=2,\ldots,n^*} (g^{\beta})^{\epsilon_j M_{i,j}^*})(g^s)^{-z_{\rho*(i)}}$.

(2) For $C''_{lf_{\mathbb{T}^*}}$ and $C''_{\mathbb{T}^*,l_k}$ in the ciphertext, $\mathcal{C}$ first generates ciphertext components C''_{lf_1} and C''_{1,l_k}. Then, $\mathcal{C}$ uses Ciphertext Update algorithm to obtain $C''_{lf_{\mathbb{T}^*}}$ and $C''_{\mathbb{T}^*,l_k}$.

For lf_1 with identity $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_d)$, $\mathcal{C}$ first pads $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_d)$ as $\vec{\tau}_d = (\tau_1, \tau_2, \ldots, \tau_q)$, and computes

$$\begin{aligned}
C''_{lf_1} &= (F_0 \prod_{j=1}^{q} F_j^{\tau_j})^s\\
&= \left(\prod_{j=1}^{q} (g^{\gamma_i / y_{q-i+1}})^{\tau_j} \cdot (g^{\xi} \prod_{j=1}^{q} y_{q-i+1}^{\tau_j}) \right)^s\\
&= (g^s)^{\xi + \sum_{i=1}^{q} \gamma_i \tau_j}.
\end{aligned}$$

For a node in SN_1 with identity $\vec{\tau}_k = (\tau_1', \tau_2', \ldots, \tau_k')$, $\mathcal{C}$ first pads $\vec{\tau}_k = (\tau_1', \tau_2', \ldots, \tau_k')$ as $\vec{\tau}_k = (\tau_1, \tau_2, \ldots, \tau_q')$, and computes

$$C''_{1,l_k} = (F_0 \prod_{j=1}^{q} F_j^{\tau_j'})^s = (g^s)^{\xi + \sum_{i=1}^{q} \gamma_i \tau_j'}.$$

After that, based on the chosen g^{ψ_i} value, $i \in \{2, \mathbb{T}^* - 1\}$, $\mathcal{C}$ generates the ciphertext components $C''_{lf_{\mathbb{T}^*}}$ and $C''_{\mathbb{T}^*,l_k}$, where $k \leq d$.

(3) For Q_j', where $j = 2, \ldots, d$, $\mathcal{C}$ calculates Q_j' as

$$Q_j' = (F_j' F_j)^s = g^{(\gamma_j + \theta_j)} s = (g^s)^{(\gamma_j + \theta_j)}.$$

Further, $\mathcal{C}$ computes $E' = \prod_{i=1}^{c} (g^s)^{\zeta_i}$. Then, $\mathcal{C}$ calculates $E_j = (g^s)^{\zeta_i}$ for $j = c+1, \ldots, d$. Finally, $\mathcal{C}$ returns the ciphertext components to $\mathcal{A}$ as the challenge ciphertext.

We can see that, if $B = e(g,g)^{\beta^{q+1} s}$, C generated by $\mathcal{C}$ is

$$\begin{aligned}
C &= K_b \cdot B \cdot e(g^s, g^{\alpha'})\\
&= K_b \cdot e(g,g)^{\beta^{q+1} s} \cdot e(g^s, g^{\alpha'})\\
&= K_b \cdot e(g,g)^{\alpha s}.
\end{aligned}$$

Thus, the returned ciphertext is valid for K_b. Or else, when B is randomly chosen from G_T, C is independent of K_b.

Phase 2 – $\mathcal{A}$ submits the same types of queries to $\mathcal{C}$. $\mathcal{C}$ responds based on the queries.

Guess – For the challenge ciphertext, $\mathcal{A}$ outputs a guess, b'. If $b' = b$, $\mathcal{C}$ sends 1 to the challenger $\mathcal{C}$, which means B is equal to $e(g,g)^{\beta^{q+1} s}$. Otherwise, $\mathcal{C}$ sends 0 to $\mathcal{C}$, which represents that B is a random number in G_T.

We can observe that, when $B = e(g,g)^{\beta^{q+1} s}$, what $\mathcal{A}$ receives is indistinguishable from a real attack. Hence, we have

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

$|Pr[b = b'] - 1/2| \leq \epsilon$. However, when B is a randomly chosen number in G_T, $|Pr[b = b'] = 1/2|$. Therefore,

$$|Pr[\mathcal{C}(\vec{y}, B = e(g,g)^{\beta^{q+1}s}) = 0] - Pr[\mathcal{C}(\vec{y}, B = R) = 0]| \geq |(1/2 + \epsilon) - 1/2| = \epsilon$$

VI. Performance Evaluation

In this section, we first analyze the complexity of the proposed scheme. Then, we present simulation results using the algorithms in our scheme.

A. Scheme Complexity Analysis

The proposed scheme includes 9 algorithms. For the computational complexity, Setup algorithm and Key Generation algorithm are run by DO. In Setup algorithm, DO needs only one exponentiation operations in group G and one bilinear pairing operation. For Key Generation algorithm, DO generates a first time period private key for an entity with attribute set S. With b attributes in S and depth d of time tree, DO needs to computes $d^2/2 + 9d/2 + b$ exponentiations in G and $d^2/2 + 9d/2 - 2$ multiplications G. For the encryption algorithm, to encrypt symmetric key K, DO needs to perform one exponentiation and one multiplication in G_T, and $d^2/2 + 5d/2 + l - 2$ multiplications and $d^2/2 + 9d/2 + 2l - 1$ exponentiations in G, for l rows in matrix M. For decryption, given the number (v) of attributes owned by an entity that satisfy the access structure, the entity needs to perform $2v + 2$ pairing operations, and $v + 3$ multiplications and v exponentiations in G_T. For key update from time period t to $t + 1$, if lf_t is a left child node, DO or an authorized user needs to perform $d - 1$ multiplications and exponentiations in G; otherwise, it needs $d + 2$ multiplications and d exponentiations in G.

For key and ciphertext update token generation, each committee member needs to perform $2t + 2$ multiplications and 12 exponentiations in G. For ciphertext update algorithm, when the ciphertext for time period t needs to update to the next period, and lf_t is a left child node, SP needs to accomplish $t + 2$ multiplications in G, t exponentiations in G, and 2 bilinear pairing operations. If lf_t is a right child node, $t + d + 1$ multiplications in G, $t + d - 1$ exponentiations in G, and 2 bilinear pairing operations are needed. The computational complexity of algorithms is shown in TABLE II, where exp_G and exp_{G_T} denote exponentiation operation in group G and group G_T respectively. mul_G and mul_{G_T} denote multiplication operation in group G and group G_T respectively.

For the communication cost of DO, when an entity requests a private key, DO needs to send $2d + b + l$ elements in G. For the communication cost of the ciphertext of K, DO needs to transmit $3d + l$ elements in G and 1 elements in G_T.

For the storage cost, DO and users need to store their private keys. For a private key, an entity with b attributes in S needs to store $2d + b + 1$ elements in G. For the storage overhead of SP, it needs to store the outsourced data and the ciphertext of K, which includes $l + 3d$ elements in G and one element in G_T.

TABLE II
Computational Complexity

Algorithms	Complexity
Setup	$1\ exp_G + 1\ pairing$
Key Generation	$(d^2/2 + 9d/2 - 2)\ mul_G + (d^2/2 + 9d/2 + b)\ exp_G$
Key Derivation	$2\ mul_G + 2\ exp_G$
Encrypt	$(d^2/2 + 5d/2 + l - 2)\ mul_G + (d^2/2 + 9d/2 + 2l - 1)\ exp_G + 1\ mul_{G_T} + 1\ exp_{G_T}$
Decrypt	$(2v + 2)\ pairing + v\ exp_{G_T} + (v + 3)\ mul_{G_T}$
Key Update	$d\ mul_G + d\ exp_G$
Ciphertext Update Token Generation	$t\ mul_G + (t + 2)\ exp_G$
Ciphertext Update	$(t + d/2 + 2)\ mul_G + (t + d/2)\ exp_G + 2\ pairing$

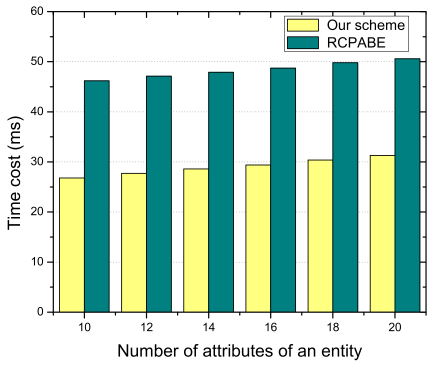
B. Experiment Results

1) Off-Chain Overheads: To evaluate the off-chain overheads of proposed scheme, we conduct simulations on a computer equipped with Intel Core 2.9 GHz i7-7500U CPU and 8 GB RAM. Miracl library [28] is employed to achieve bilinear mapping. In the experiments, we choose BN elliptic curves and set the security parameter as 128; D_0, L, C' in the proposed scheme are elements in G_1, $\{C_i\}_{i \in [l]}, \{K_x\}_{x \in S}, D_{c,lf}, \{Q_j\}_{j \in 2,\ldots,d}$ are elements in G_2, and C in a ciphertext is an element in G_T.

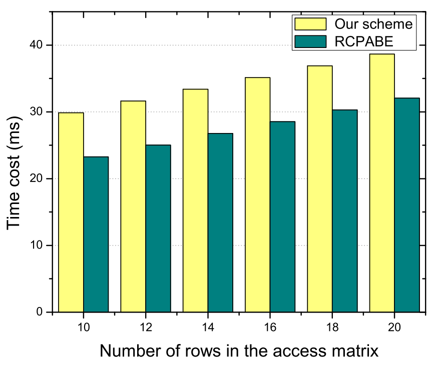
We compare the performance of our scheme with [29], which is called RCPABE. RCPABE enables revocable ciphertext-policy attribute-based encryption with short revocation list by combining key update and revocation list. For the proposed scheme and RCPABE, in the key generation algorithms, a user first needs to apply for a private key from DO, which generates a key for the user based on the attributes of the required data. We test the computational cost for key generation when the number of the attributes varies from 10 to 20. We set the depth of the tree (d) to 6, as this allows 2^5 key updates for a file of DO, which is sufficient for real-world applications. If d is too large, there would be more storage cost and computation cost for SP and authorized data users, since SP needs to store more ciphertext components for internal nodes, and data users need more computations to derive a new key. In the evaluation, private keys for the same number of nodes in the tree are generated. Fig. 5(a) shows that, for a given depth of the tree, computational cost for key generation increases linearly with the number of attributes of an entity. Compared with RCPABE, our scheme has less computational cost, since RCPABE embeds a revocation list in ciphertexts and needs to generate private key components for future revocation.

In the encryption algorithm, DO needs to encrypt symmetric key K with access matrix M. We test the computational overhead of encryption algorithms for the proposed scheme and RCPABE. Fig. 5(b) shows that, when the depth of tree is fixed, the computational cost for encryption increases with the number of rows in M. Moreover, in our scheme, DO needs to

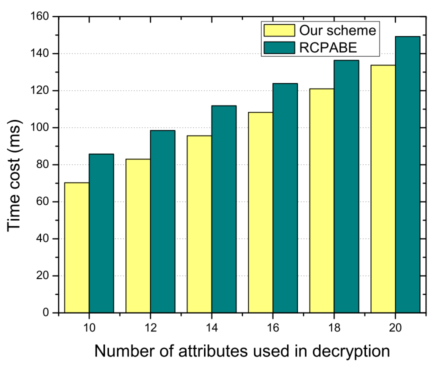
Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.



(a) Computational cost for key generation when the number of attributes varies



(b) Computational cost for encryption when the number of rows in M varies



(c) Computational cost for decryption when the number of attributes increases

Fig. 5. Computational cost for key generation, encryption, and decryption.

generate ciphertext components for the nodes in SN_1, which is used for subsequent ciphertext update. Thus, the time cost of encryption for the proposed scheme is slightly larger than that of RCPABE.

We measure the computational cost of the decryption algorithms of the proposed scheme and RCPABE. Fig. 5(c) shows that the decryption overhead increases linearly as the number of attributes used in decryption grows from 10 to 20. The computational cost for decryption is independent of the depth of the time tree because, to verify the time validity, only the ciphertext component corresponding to the decryptable time period is included. From Fig. 5(c), the proposed scheme has less computational overhead for decryption, as RCPABE needs to further check where a user is in the revocation set for a ciphertext.

2) On-Chain Overheads: We deploy a consortium blockchain based on Hyperledger Fabric that consists of two peer nodes and one order node [30]. The blockchain relies on the Raft-ordering service and is configured with 10 transactions per block and a fixed block confirmation delay of 2 seconds. The blockchain is run in a testbed equipped with an Apple M1-chip (8 cores) and 16 GB RAM. We develop the smart contract UTG and the client application using Java [31], and the simulation results are shown in Table III. To measure the three core functions of the contract, we use the end-to-end transaction confirmation delay as the on-chain performance index. It is observed that SP and committee members can receive the confirmation of transactions (that successfully trigger these functions) in around 2 seconds if the blockchain network is not blocked. In addition, as the number of committee members is another parameter that may affect *Upload* function, we evaluate the total running time of two sub-functions, verification and aggregation, of *Upload* with the different number of committee members, as shown in Table IV. It is observed that, with the increase in the committee members, the total running time increases as well. With the participation of 90 committee members, the total running time can be less than 290 ms, which is efficient. Moreover, committee members' off-chain computation costs are low. It takes less than 45 ms for each committee member to construct a transaction that can be utilized to trigger *Upload* function.

TABLE III
TRANSACTION CONFIRMATION DELAY OF THREE CORE FUNCTIONS DEFINED IN THE CONTRACT UTG

Functions	Setup	Register	Upload
Transaction Confirmation (s)	2.076	2.103	2.081

TABLE IV
TOTAL RUNNING TIME OF VERIFICATION AND AGGREGATION IN *Upload* FUNCTION

Sub-functions	$N = 10$	$N = 30$	$N = 50$	$N = 70$	$N = 90$
Verification (ms)	38.035	110.156	183.969	251.932	289.314
Aggregation (ms)	0.03	0.08	0.16	0.19	0.26

VII. RELATED WORK

A. Attribute-Based Encryption (ABE)

The concept of ABE is first presented by Goyal et al. [32]. ABE is a type of public-key encryption, where the private key of a user and ciphertexts are related to attributes. There are two categories for ABE. One is key-policy ABE, in which a user's private key is generated according to a policy, which is built on attributes and defines the access right of a user, and a ciphertext is encrypted with several attributes. When the attributes of a ciphertext satisfy the policy corresponding to a user's private key, the user can decrypt the ciphertext. The other is ciphertext-policy ABE, in which a user's private key is created based on its attributes, and a ciphertext is encrypted with an access policy [33], [34]. Only if a user owns the necessary attributes that satisfy the access policy embedded in a ciphertext, the user can decrypt the ciphertext.

Since in an ABE system, each attribute is shared by multiple users, revoking a user will influence others who own the same attributes with the user [29], [35]. Yu et al. address the attribute revocation problem by integrating CP-ABE and proxy re-encryption [36]. Users' private keys are updated except for the users who are revoked. Xiong et al. propose a partially policy-hidden attribute-based broadcast encryption scheme, which achieves direct revocation of users [37]. The revocation list is embedded in the ciphertext such that only users who are

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

TABLE V
COMPARISON WITH RELATED WORKS

Scheme	Data Sharing Policy	User Revocation	Ciphertext Update	Key Update without Proxy	Time Validity
[38]	N/A	N/A	✓	✓	×
[13]	N/A	N/A	✓	✓	×
[36]	AND-gates	✓	✓	×	×
[29]	LSSS	✓	N/A	×	✓
[39]	AND-gates	✓	✓	✓	×
This work	LSSS	✓	✓	✓	✓

not in the list and satisfy the predefined access policy can decrypt the ciphertext.

The existing ABE schemes with user revocation can achieve data sharing and key update. However, there is no time validity for the data sharing. Our proposed scheme supports fine-grained data access control and embeds the time validity in the shared data. Moreover, by utilizing the blockchain, the communication costs of key updates for authorized users are reduced, since DO does not need to distribute key update tokens to each user. The comparison with the related works is summarized in Table V.

B. Updatable Encryption

Updatable encryption enables the updates of a secret key and a ciphertext encrypted with the key [40]. To be specific, a user creates a secret key and uses the key to encrypt a ciphertext. At the beginning of the next time period, the user creates a new secret key and generates a conversion key, which is used to convert the original ciphertext to a new ciphertext that can be decrypted by the new secret key. Updatable encryption can be ciphertext-dependent, i.e., the ciphertext update token is generated with the involvement of partial ciphertext, or ciphertext-independent, i.e., the update token can be generated without a ciphertext and can be used to update any ciphertext.

For the direction of key updates, Lehman and Tackmann define bi-directional key updates and uni-directional key updates [40]. The former allows an adversary to upgrate and downgrade keys, i.e., an adversary can derive a secret key for the next time period (k_{u+1}) from an update token (δ_{u+1}) and the current time period key (k_u), and recover k_u from δ_{u+1} and k_{u+1}. The uni-directional key updates mean that we can generate k_{u+1} from k_u and δ_{u+1}. While uni-directional updatable encryption is preferable, Jiang et al. [41] prove that bi-directional key update schemes are equivalent to the schemes that support uni-directional key updates under the most advanced updatable encryption model presented by Boyed et al. [42]. Shen et al. propose a data privacy preservation scheme with key rotation and ciphertext update, in which a symmetric encryption-based updatable encryption is adopted to achieve key rotation, and an identity-based encryption is used to assign access right for users [43]. Slamanig et al. propose an updatable encryption scheme with no-directional key updates, where the update tokens cannot be used to upgrade or downgrade secret keys [38].

Most UE schemes employ symmetric encryption primitives to achieve key update and ciphertext update. Thus, they cannot achieve fine-grained data sharing. By using a tree-based structure, our scheme not only can support flexible data access authorization, but also can enable multiple key updates.

Based on the conference version [1], first, we improve the blockchain-based data sharing scheme by utilizing a tree-based structure to generate private keys for authorized users and ciphertexts. As a result, for the same public parameters ($params$), our scheme enables 2^{d-1} key updates instead of d updates, where d denotes the depth of the time tree. Second, we utilize the committee members of the consortium blockchain to cooperatively generate the key and ciphertext update token by invoking smart contract UTG, and DO is not involved in the update token generation. Third, we formulate a security model and formally prove the security of the proposed scheme. Finally, we evaluate the on-chain and off-chain performance of the proposed scheme and compare its performance with an existing scheme. The experiment results show that our scheme can achieve high computational efficiency.

VIII. CONCLUSION

In this paper, we have proposed a blockchain-based flexible data sharing scheme that supports key and ciphertext updates. Data can be shared with authorized users based on data attributes and time validity. By using a tree-based structure, the proposed scheme can support an exponential number of key updates. The storage platform is utilized to facilitate ciphertext update, and no re-encryption of data is needed for the data owner. Security proof and simulation results have shown that our scheme is both secure and practical. The design of authorization and key update mechanisms can motivate further research on efficient key management. The proposed scheme has one limitation that the access policies are stored on the blockchain to achieve transparent data sharing management, which may leak the information of the outsourced data. In the future, we aim to design a blockchain-based policy hidden and auditable data sharing scheme.

REFERENCES

[1] L. Xue et al., "Secure and flexible data sharing for distributed storage with efficient key management," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2022, pp. 4408–4413.

[2] W. Wu et al., "Dynamic ran slicing for service-oriented vehicular networks via constrained learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2076–2089, Jul. 2020.

[3] K. B. Letaief, Y. Shi, J. Lu, and J. Lu, "Edge artificial intelligence for 6G: Vision, enabling technologies, and applications," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 5–36, Jan. 2022.

[4] P. Podder, S. Bharati, M. R. H. Mondal, P. K. Paul, and U. Kose, "Artificial neural network for cybersecurity: A comprehensive review," 2021, *arXiv:2107.01185*.

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

[5] G. Zhang, T. Li, Y. Li, P. Hui, and D. Jin, "Blockchain-based data sharing system for AI-powered network operations," *J. Commun. Inf. Netw.*, vol. 3, no. 3, pp. 1–8, 2018.

[6] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning: A big data-AI integration perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1328–1347, Apr. 2021.

[7] D. Liu, C. Huang, J. Ni, X. Lin, and X. Shen, "Blockchain-cloud transparent data marketing: Consortium management and fairness," *IEEE Trans. Comput.*, early access, Feb. 14, 2022, doi: 10.1109/TC.2022.3150724.

[8] Y. Liu, X. Guan, Y. Peng, H. Chen, T. Ohtsuki, and Z. Han, "Blockchain-based task offloading for edge computing on low-quality data via distributed learning in the Internet of energy," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 2, pp. 657–676, Feb. 2022.

[9] N. Firoozeh, A. Nazarenko, F. Alizon, and B. Daille, "Keyword extraction: Issues and methods," *Natural Lang. Eng.*, vol. 26, no. 3, pp. 259–291, 2020.

[10] S. Beliga, "Keyword extraction: A review of methods and approaches," *Univ. Rijeka, Dept. Inform. Radmile*, vol. 1, no. 9, pp. 1–9, 2014.

[11] M. Lipp et al., "PLATYPUS: Software-based power side-channel attacks on X86," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 355–371.

[12] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun, "SCARF: Detecting side-channel attacks at real-time using low-level hardware features," in *Proc. IEEE 26th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2020, pp. 1–6.

[13] D. Boneh, S. Eskandarian, S. Kim, and M. Shih, "Improving speed and security in updatable encryption schemes," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2020, pp. 559–589.

[14] S. Fugkeaw, "Secure data sharing with efficient key update for industrial cloud-based access control," *IEEE Trans. Services Comput.*, early access, Sep. 8, 2021, doi: 10.1109/TSC.2021.3110828.

[15] X. Shen et al., "Blockchain for transparent data management toward 6G," *Engineering*, vol. 8, pp. 74–85, Jan. 2021.

[16] G. S. Aujla and A. Jindal, "A decoupled blockchain approach for edge-envisioned IoT-based healthcare monitoring," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 491–499, Feb. 2021.

[17] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. Int. Workshop Public Key Cryptography*, 2011, pp. 53–70.

[18] Z. Liu, Z. Cao, and D. S. Wong, "Efficient generation of linear secret sharing scheme matrices from threshold access trees," *IACR Cryptol. ePrint Arch.*, vol. 2010, p. 374, 2010.

[19] D. Boneh, X. Boyen, and E. J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2005, pp. 440–456.

[20] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *J. Cryptol.*, vol. 7, no. 1, pp. 1–32, Dec. 1994.

[21] J. B. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider, "A certifying compiler for zero-knowledge proofs of knowledge based on σ-protocols," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2010, pp. 151–167.

[22] A. Onan, S. Korukoğlu, and H. Bulut, "Ensemble of keyword extraction methods and classifiers in text classification," *Exp. Syst. Appl.*, vol. 57, pp. 232–247, Sep. 2016.

[23] R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2003, pp. 255–271.

[24] F. Günther, B. Hale, T. Jager, and S. Lauer, "0-RTT key exchange with full forward secrecy," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2017, pp. 519–548.

[25] Y. Tsiounis and M. Yung, "On the security of ElGamal based encryption," in *Proc. Int. Workshop Public Key Cryptogr.*, 1998, pp. 117–134.

[26] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007.

[27] Y. Zhang, D. He, M. S. Obaidat, P. Vijayakumar, and K.-F. Hsiao, "Efficient identity-based distributed decryption scheme for electronic personal health record sharing system," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 384–395, Feb. 2021.

[28] *Miracl Library*. Accessed: Jan. 2022. [Online]. Available: https://github.com/miracl/MIRACL

[29] J. K. Liu, T. H. Yuen, P. Zhang, and K. Liang, "Time-based direct revocable ciphertext-policy attribute-based encryption with short revocation list," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2018, pp. 516–534.

[30] E. Androulaki et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[31] *Miracl Java*. Accessed: Jan. 2022. [Online]. Available: https://github.com/miracl/core

[32] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 89–98.

[33] S. Wang et al., "A fast CP-ABE system for cyber-physical security and privacy in mobile healthcare network," *IEEE Trans. Ind. Appl.*, vol. 56, no. 4, pp. 4467–4477, Jul./Aug. 2020.

[34] Q. Li, B. Xia, H. Huang, Y. Zhang, and T. Zhang, "TRAC: Traceable and revocable access control scheme for mHealth in 5G-enabled IIoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3437–3448, May 2022.

[35] Z. Liu, Q. Huang, and D. S. Wong, "On enabling attribute-based encryption to be traceable against traitors," *Comput. J.*, vol. 64, no. 4, pp. 575–598, Apr. 2021.

[36] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proc. 5th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2010, pp. 261–270.

[37] H. Xiong, Y. Zhao, L. Peng, H. Zhang, and K.-H. Yeh, "Partially policy-hidden attribute-based broadcast encryption with secure delegation in edge computing," *Future Gener. Comput. Syst.*, vol. 97, pp. 453–461, Aug. 2019.

[38] D. Slamanig and C. Striecks, "Puncture 'EM all: Updatable encryption with no-directional key updates and expiring ciphertexts," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 268, 2021.

[39] V.-H. Hoang, E. Lehtihet, and Y. Ghamri-Doudane, "Privacy-preserving blockchain-based data sharing platform for decentralized storage systems," in *Proc. IFIP Netw. Conf. (Networking)*, 2020, pp. 280–288.

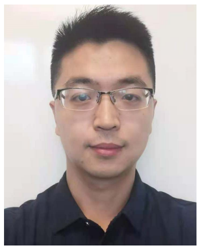
[40] A. Lehmann and B. Tackmann, "Updatable encryption with post-compromise security," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2018, pp. 685–716.

[41] Y. Jiang, "The direction of updatable encryption does not matter much," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2020, pp. 529–558.

[42] C. Boyd, G. T. Davies, K. Gjøsteen, and Y. Jiang, "Fast and secure updatable encryption," in *Proc. Int. Cryptol. Conf.*, 2020, pp. 464–493.

[43] S. Shen, Y. Yang, and X. Liu, "Toward data privacy preservation with ciphertext update and key rotation for IoT," *Concurrency Comput., Pract. Exper.*, p. e6729, 2021.

Liang Xue (Member, IEEE) received the B.S. and M.S. degrees from the School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC), China, in 2015 and 2018, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. Her research interests include applied cryptography, cloud computing, and blockchain.



Dongxiao Liu (Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2020. He is currently a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests include security and privacy in intelligent transportation systems, blockchain, and mobile networks.

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.

Cheng Huang (Member, IEEE) received the B.Eng. and M.Eng. degrees in information security from Xidian University, China, in 2013 and 2016, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, ON, Canada, in 2020. He is currently a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests are in the areas of applied cryptography, cyber security, and privacy in the mobile networks.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, ON, Canada. His research focuses on network resource management, wireless network security, the Internet of Things, 5G and beyond, and vehicular *ad-hoc* and sensor networks.

Dr. Shen is a registered Professional Engineer of Ontario, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society. He received the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R. A. Fessenden Award in 2019 from IEEE, Canada, and the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015 and the Education Award in 2017 from the IEEE Communications Society, and the Technical Recognition Award from Wireless Communications Technical Committee in 2019 and AHSN Technical Committee in 2013. He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario. He served as the Technical Program Committee Chair/Co-Chair for IEEE GLOBECOM'16, IEEE Infocom'14, IEEE VTC'10 Fall, IEEE GLOBECOM'07, and the Chair for the IEEE Communications Society Technical Committee on Wireless Communications. He is the President of the IEEE Communications Society. He was the Vice President for Technical and Educational Activities, the Vice President for Publications, the Member-at-Large on the Board of Governors, the Chair of the Distinguished Lecturer Selection Committee, a member of IEEE Fellow Selection Committee of the ComSoc. He served as the Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL, *IEEE Network*, and *IET Communications*.

Weihua Zhuang (Fellow, IEEE) received the B.Sc. and M.Sc. degrees from Dalian Marine University, China, and the Ph.D. degree from the University of New Brunswick, Canada, all in electrical engineering. She is a University Professor and a Tier I Canada Research Chair in wireless communication networks at the University of Waterloo, Canada. Her research focuses on network architecture, algorithms and protocols, and service provisioning in future communication systems. She is an Elected Member of the Board of Governors and the Executive Vice President of the IEEE Vehicular Technology Society. She is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, and the Engineering Institute of Canada. She was a recipient of 2021 Women's Distinguished Career Award from the IEEE Vehicular Technology Society, the 2021 Technical Contribution Award in Cognitive Networks from IEEE Communications Society, the 2021 R. A. Fessenden Award from IEEE Canada, and the 2021 Award of Merit from the Federation of Chinese Canadian Professionals in Ontario. She was the General Co-Chair of 2021 IEEE/CIC International Conference on Communications in China (ICCC), the Technical Program Chair/Co-Chair of 2017/2016 IEEE VTC Fall, the Technical Program Symposia Chair of 2011 IEEE GLOBECOM, an IEEE Communications Society Distinguished Lecturer from 2008 to 2011, and the Editor-in-Chief of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY from 2007 to 2013.

Rob Sun (Member, IEEE) is currently a Principal Engineer with Huawei Technologies Canada Company Ltd. He also coauthored a few books on wireless security technologies. His work primarily focuses on the advancement of the NG wireless, including 5G/6G and WiFi/IoT security architecture and standardization. He was the Vice Chair of IEEE Privacy Management Protection (PMP) Task Group which was to set out the best practices for protecting personal privacy information, and support efficient, adaptable, and innovative approaches for privacy governance. He was regarded as one of the core contributors to the standardizations of a series of NG Wi-Fi security protocols and certifications, including the most recent Wi-Fi WPA3 protocol suites.

Bidi Ying (Member, IEEE) received the Ph.D. degree. She is currently working at Huawei Technologies Canada Company Ltd., as a Senior Network Architecture Engineer. Before that, she was working at the University of Ottawa. For the past 15 years, she has published more than 200 papers in top conferences and reputable journals. Her main research interests include security and privacy in wireless networks.

Authorized licensed use limited to: University of Waterloo. Downloaded on March 14,2023 at 20:47:57 UTC from IEEE Xplore. Restrictions apply.