Attribute-based Private Data Sharing with Script-driven Programmable Ciphertext and Decentralized Key Management in Blockchain Internet of Things

Hongjian Yin, E Chen, Yan Zhu, *Member, IEEE*, Chengwei Zhao, Rongquan Feng, Stephen S. Yau, *Fellow, IEEE*

***Abstract*—In this paper, we address the problem of secure sensitive data sharing for the specified recipients in Blockchain Internet of Things (BIoT). To do it, we present a cryptographic solution to meet the requirements of decentralization and convenience through key management and programmable ciphertext. Firstly, we design a new Ciphertext-Policy Decentralized-Key Attribute-based Encryption (CP-DK-ABE) scheme. After the master secret key is shared into all full nodes in the form of threshold secret sharing, a decentralized multi-party computation protocol is used to generate the user's private key in an interactive way. Meanwhile, the attribute sub-keys associated with the private key can be reconstructed by obtaining fragment from each of full nodes, so as to achieve the cooperative management of attribute key through all of full nodes. Furthermore, following the blockchain's script system, we introduce five new opcodes to represent ciphertext in the programmable format. Such a mechanism provides flexible capability to represent the logical relationship of access control policy among attribute sub-ciphers in the CP-DK-ABE ciphertext by the scripting language. As a result, the processes of encryption and decryption are implemented entirely by the script interpreter on the blockchain node, thereby greatly improving the convenience of programming in BIoT devices. In addition, we prove that the proposed CP-DK-ABE scheme is key private and semantically secure for a limited number of corrupted full nodes under the decisional linear and bilinear Diffie-Hellman assumption, respectively.**

***Index Terms*—Blockchain IoT, ABE, decentralized key, script system, programmable ciphertext.**

I. Introduction

As an exciting technology, Blockchain Internet of Things, known as BIoT, has attracted much attention since it was proposed and has been widely implemented across industries, such as medical, vehicle, agriculture, etc.. Benefiting from the decentralized blockchain network, BIoT has a lower operating cost and decentralized resource management in comparison with the traditional IoT. As a combination of two popular technologies, BIoT has not only broad application prospects, but also great commercial value [1]. According to Aftrex market report 2018[1], the global BIoT market is estimated to reach USD 254.31 billion by 2026.

The large-scale application of BIoT technology has caused a series of security threats. In particular, IoT network is replaced by a distributed broadcast network through blockchain, which will increase the risk of eavesdropping, denial of service and botnet attacks. Among these security threats, the most serious one is data privacy leakage. Considering that 98% of all IoT device traffic is unencrypted according to the Palo Alto IoT Threat Report in 2020[2], these data exposed on the network is easy to be monitored and collected by attackers, and then exploited for profit on the Dark Web. Some works have studied on user authentication [2] and privacy protection [3] in IoT, but it is necessary to pay special attention to secure sensitive data sharing. Therefore, our main focus is the protection of BIoT data privacy in this work.

A. Motivations

Although digital signature and Hash function in blockchain can protect the BIoT data from being tampered, most of the existing blockchain systems have not adopted encryption technique. The reason is that the existing encryption algorithms (such as RSA, ECC) are built on "one-to-one" cryptosystem over PKI system, that is, one user's public key corresponds to one unique private key. It means that a ciphertext generated by the recipient's public key can only be decrypted by the private key of the designated recipient. Under this encryption mechanism, in order to share BIoT data among multiple users, the data needs to be encrypted and transmitted repeatedly. The number of encryptions and transmissions will be as large as that of target users. Obviously, this "one-to-one" unicast mode is not only inefficient, also contrary to the blockchain broadcast network.

In view of the broadcast network of BIoT, group-oriented encryption is a more effective solution to share the sensitive data securely. As a "one-to-many" cryptosystem, the advantage of group-oriented encryption is that the authorized recipients are no longer a single individual but a specified set. It means that a ciphertext, generated by group public-key and broadcasted throughout the whole network, can only be decrypted by anyone in the authorization set. In addition, new users belonging to the authorization set can still decrypt the previous ciphertext without any other operations after obtaining their

H. Yin, E Chen and Y. Zhu are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, 100083 China (e-mail: honjanyin@163.com; chene5546@163.com; zhuyan@ustb.edu.cn).

C. Zhao is with the Chinese Academy of Science and Technology for Development (CASTED), Beijing, 100038 China (email: zhaocw@bupt.edu.cn).

R. Feng is with the School of Mathematical Science, Peking University, Beijing, 100871 China (e-mail: fengrq@math.pku.edu.cn).

Stephen S. Yau is with the School of Computing and Augmented Intelligence, Arizona State University, Tempe, 85287-8809 USA (email: yau@asu.edu).

Manuscript received month day, 2021; revised month day, 2021; accepted month day, 2021. (Corresponding author: Yan Zhu)

[1]https://www.aftrexmarketresearch.com/report-details.php?id=548

[2]https://start.paloaltonetworks.com/unit-42-iot-threat-report

private key. Therefore, group-oriented encryption is more suitable for sharing the sensitive data in BIoT.

At present, group-oriented encryption can be roughly divided into Identity-based Broadcast Encryption (IBBE), Role-based Encryption (RBE) and Attribute-based Encryption (ABE). Intuitively, the main difference among these types is the authorization way in which identity, role, and attribute are used as the basic unit to describe the authorized users for IBBE, RBE and ABE, respectively. For example, chief physicians of cardiology are Alice and Bob in a hospital, IBBE describes the authorization set as {Alice, Bob}. RBE uses the role *"Chief physicians of Cardiology"* to describe the authorized users. ABE represents the same set by access policy *"department = cardiology" and "level = Chief physician"*. In comparison with identity and role, attribute is a more fine-grained and flexible unit to describe authorized users [4]. Therefore, we choose ABE to protect sensitive data in BIoT in this paper.

Taking Ciphertext-Policy ABE (CP-ABE) as an example, such as schemes in [5]–[7], it allows one to encrypt data under the policies defined over some attributes of data recipients. For instance, a health record policy can be defined as

(Hospital = "Rehabilitation Hospital" and Role = "Clinician") or Institution = "Insurance company".

It means that in addition to clinicians in rehabilitation hospital, the staffs in insurance company allow access to health records. Compared with IBBE and RBE, the description way of authorization users in CP-ABE is more fine-grained and flexible. It is suitable for protecting sensitive data in BIoT, which has more flexible authorization methods.

In general, the CP-ABE system is deployed in a centralized environment, so that key management and distribution can be performed by a trusted administrator. In order to adapt for the decentralized feature of blockchain, the trusted administrator should be removed and the key management and private key-generation should be decentralized. Therefore, CP-ABE scheme needs to meet the following requirements.

1) **Decentralized key distribution**. The public-key in CP-ABE is disclosed, but the corresponding master secret key needs to be decentralized management rather than the trusted administrator;
2) **Distributed key generation**. The user's private key needs to be generated collaboratively by multiple nodes, and a central node is not involved in the process of private key generation;
3) **Programmable ciphertext**. The form of ciphertext is not only compact but also programmable. Moreover, the processes of encryption and decryption are executed in blockchain node by program codes.

In the previous CP-ABE schemes, there is almost no research on the storage forms of ciphertext. Most of the ciphertexts are stored in a numeric form that will bring inconvenience to the decryption operation with complicated access policy. In the process of ciphertext decryption, for example, a user first searches all attribute sub-keys corresponding to attribute sub-ciphers, and then performs logical operations among attributes until the message is opened. As a consequence, we consider using programmed ciphertext to replace numeric ciphertext, so as to simplify the above decryption process. In addition, the encryption and decryption processes in the existing CP-ABE schemes are executed by clients. These client-based operations increase the difficulty of program design in BIoT devices. Considering that the blockchain nodes has become a module of BIoT, our work hopes to transfer encryption and decryption processes into the blockchain nodes, which is called node-based encryption and decryption.

B. Related Works

In 2005, Sahai and Waters [8] proposed the first ABE scheme on the basis of identity-based encryption. Since then, ABE has attracted widespread attention. ABE schemes can be roughly divided into two categories: Ciphertext-Policy ABE (CP-ABE) [9] and Key-Policy ABE (KP-ABE) [10]. In the CP-ABE scheme, user's private key is bound to attribute and ciphertext is related to access policy; whereas it is opposite in the KP-ABE scheme. Subsequently, Li et al. [11] proposed the first adaptive chosen-ciphertext security ABE with equality test, which only adds one dummy attribute to access structure. Similarly, Jiang and Susilo et al. [12] addressed the "key-delegation abuse" problem in CP-ABE systems. However, almost all of the above schemes are built on single authority, that is, the users' private keys are issued by a trusted center. This is inconsistent with our requirements of the distributed management about master key.

In order to relieve the users' trust on a single authority, the first Multi-Authority ABE (MA-ABE) was proposed by Chase in [13], which allows arbitrary polynomial number of independent authority to manage attributes and issue private keys for users. After that, Li et al. [14] constructed a MA-CP-ABE scheme with decryption outsourcing, which largely eliminates the user's decryption overloads. And they designed an attribute-level user revocation approach with less computation costs. Belguith et al. [15] presented a policy-hidden MA-ABE scheme, which not only ensures the security of hidden policies, but also protects users' privacy. However, in order to prevent collusion attacks, almost all of the above MA-ABE schemes need a trusted central authority to participate in the generation of user's private key, which goes against the decentralization of Blockchain.

In order to remove the trusted central authority, Lewko and Waters proposed a new MA-ABE named decentralized ABE (DABE) [16]. In their scheme, each authority manages a set of attributes and issues attribute secret keys to users independently. However, each authority needs to collaborate with other authorities to initialize the system. To improve it, Li et al. [17] provided a Decentralized MA-CP-ABE scheme. In this scheme, each of the attribute authorities works independently without any interaction with other authorities, but it only applies to domain management and needs an obfuscated program with any mediated interaction. Subsequently, Liang et al. [18] proposed a privacy-preserving DABE scheme with policy hiding and global identity hiding. This scheme removes random oracle, but it relies on composite order group, which leads to low efficiency and security. Additionally, the similar

work [19] has been put forward in the recent years. In these schemes, the central authority is removed, and each authority independently issues attribute subkeys to users. Although these schemes can realize the decentralized generation of user's private key, it has low fault tolerance. This means that the whole system may not be able to run if an authority node is destroyed, because each authority node manages part of attributes separately.

Recently, some other kinds of ABE schemes have been proposed to achieve various of functional purposes. For example, Li et al. [20] presented a hierarchical ABE against continuous master key leakage and users' secret key leakage. In the same year, based on lattice hard problem, Tian et al. [21] proposed a MA-ABE scheme with hidden policies, which can resist quantum computer attacks. Han et al. [22] proposed a traceable and revocable CP-ABE scheme, which realizes the partially hidden policy as well as the tracking and revoking of malicious users. Islam et al. [23] first proposed revocable ABE with verifiable outsourced decryption, which allows any number of users to revoke and join without affecting the secret membership keys of non-revoked users.

C. Our Approaches

However, none of the existing ABE schemes can meet our requirements: decentralized key management and distributed key generation. Therefore, it is necessary to design a new ABE to prevent the sensitive data from leaking in BIoT. In response to the aforementioned requirements, we will intent to utilize the following technologies for designing a new ABE scheme:

1) Using the public ledger, the public-key is known by every node in the BIoT system. In addition, by utilizing the secret sharing, the master key will be distributed to each full node in blockchain for decentralized management (see Section IV-A);
2) In order to realize the decentralized generation of the user's private key, it needs to design a new key generation algorithm through multi-party interaction and cooperation (see Section IV-B);
3) Ciphertext is expressed in the scripting language. The type identifiers and operation codes are introduced into the ciphertext to make the expression more compact, facilitate execution of script interpreter, and implement blockchain's node decryption (see Section V-C and V-D).

Considering script mechanism in Bitcoin system can realize advance operations, such as flow control, bitwise logic, crypto, etc. [24]–[26], we expect to combine ABE with the Bitcoin script mechanism for describing ciphertext in the scripting language. Through programming, this kind of ciphertext is more compact and efficient.

D. Our Contributions

In this paper, our goal is to design a new attribute-based encryption scheme to share the sensitive data with the specified recipients in BIoT. Considering the requirements of decentralization and convenience, we present a private data sharing solution through decentralized key generation and programmable ciphertext. Exactly, our contributions are summarized as follows:

1) We design a new Ciphertext-Policy Decentralized-Key ABE (CP-DK-ABE) scheme to support decentralized key distribution and generation. In this scheme, the master secret key is distributed into all full nodes in blockchain by using (t, n)-threshold secret sharing. Based on it, a decentralized multi-party computation technique is used to generate the user's private key in an interactive way. Meanwhile, the attribute sub-keys associated with the private key can be reconstructed by obtaining fragment from each of full nodes, so as to achieve the cooperative management of attribute key through all of full nodes.
2) Aim at the requirement of programmable ciphertext, we introduce five new opcodes and the corresponding algorithms into Bitcoin's script system. Such a mechanism provides flexible capability to represent the logical relationship of access control policy among attribute sub-ciphers in the CP-DK-ABE ciphertext by the scripting language. Furthermore, the processes of encryption and decryption are implemented entirely by the script interpreter on the blockchain node. Therefore, this kind of script-driven programmable ciphertext can greatly improve the convenience of programming in BIoT devices.

In addition, we prove that the proposed CP-DK-ABE scheme is key private and semantically secure for a limited number of corrupted full nodes under the decisional linear and bilinear Diffie-Hellman assumption, respectively.

II. Preliminaries

In this section, we will introduce some preliminaries including bilinear mapping and hardness assumptions as below.

A. Bilinear Mapping

Let $\Phi = (p, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot))$ be the bilinear group with prime order p, where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order p. g is the random generator of $\mathbb{G}$. The mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is called a bilinear mapping if the following properties are satisfied:

1) Bilinearity. For all $g, h \in \mathbb{G}$, and $a, b \in \mathbb{Z}_p^*$, $e(g^a, h^b) = e(g, h)^{ab}$ holds;
2) Non-degeneracy. $e(g, g) \neq 1$;
3) Computability. $\forall g, h \in \mathbb{G}$, there are efficient algorithms to compute $e(g, h)$.

B. Hardness Assumptions

Decision Linear Diffie-Hellman assumption. Let $\{a_i\}$, $\{c_i\}$, b and Z be random elements in $\mathbb{Z}_p^*$, where $i \in [0, t-1]$. And G is a generator in group $\mathbb{G}$. The Decision Linear Diffie-Hellman (DLDH) assumption holds in group $\mathbb{G}$ if no probabilistic polynomial-time (PPT) algorithm can distinguish the tuple $[G, G^b, \{G^{b(a_i+c_i)}, G^{-c_i}\}_{i=0}^{t-1}, G^{ab}]$ from $[G, G^b, \{G^{b(a_i+c_i)}, G^{-c_i}\}_{i=0}^{t-1}, Z]$ with a non-negligible advantage.

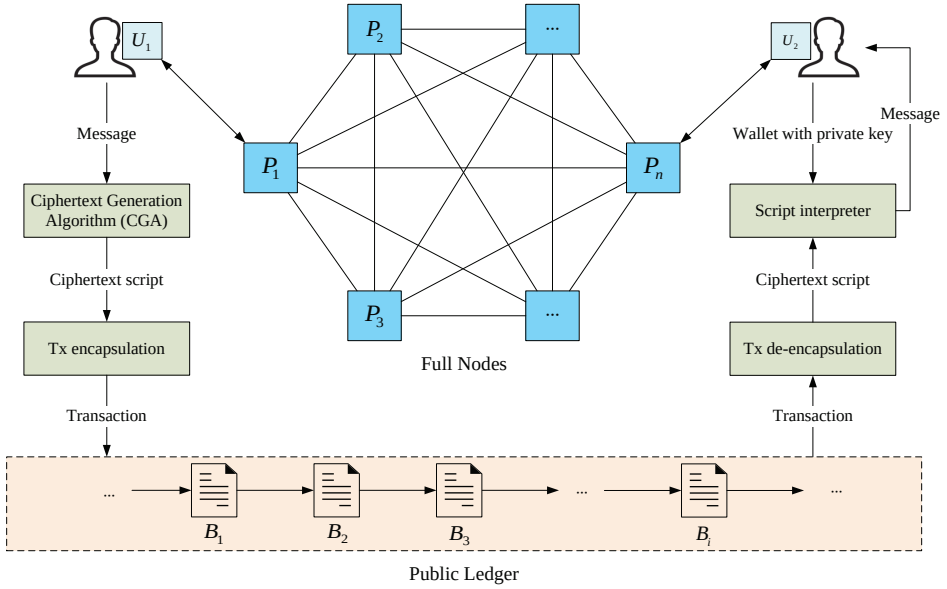

Fig. 1: The blockchain-based IoT data sharing system

Decisional Bilinear Diffie-Hellman assumption. Let a, b, c and z be random element in $\mathbb{Z}_p^*$, and G be a generator in $\mathbb{G}$. The Decisional Bilinear Diffie-Hellman (DBDH) assumption holds in group $\mathbb{G}$ if no PPT algorithm can distinguish the tuple $[G, G^a, G^b, G^c, e(G,G)^{abc}]$ from $[G, G^a, G^b, G^c, e(G,G)^z]$ with a non-negligible advantage.

III. System Model

Before describing our Blockchain-based IoT data sharing system, we first give some abbreviations and notations in TABLE I, which will be used in this paper.

TABLE I: Some main notations in this paper

Symbol	Description
P_i	blockchain full node
$Att_i^{(k)}$	the i-th attribute of user ID_k
$\mathbb{A}$	the set of attribute
$\mathcal{T}$	access tree
$\tilde{x}$	node in access tree
PK	the system public key
SK_i	the secret key of the full node P_i
sk_k	the private key of user ID_k
$N_{\tilde{x}}$	the decryption result of node $\tilde{x}$
$a \leftarrow_R A$	a is the random element picked in set A

As shown in Fig. 1, our BIoT data sharing system is built on a blockchain, which contains two entities: blockchain network and public ledger. Next, we will describe the above two entities in detail.

Blockchain network. In this work, the goal is to propose a new encryption mechanism to protect sensitive data in BIoT, and achieve the flexible selection of content recipients through the control of authorization. To end it, we will propose a new scheme based on public blockchain or consortium blockchain. In our scheme, the blockchain contains two types of nodes:

- **Full node** P_i has strong computing power to manage keys, but not all full nodes are required to be fully trusted. Moreover, full nodes store a copy of the blockchain's history, and thus guarantee the security and correctness of transaction data by participating in consensus-based data verification. Denote $P = \{P_i\}_{i\in[1,n]}$ as the set of full nodes, and n is the number of full nodes.
- **Light node** U_k represents the user whose BIoT device is connected to full nodes to synchronize with the current state of the blockchain network. Moreover, the user U_k owns a wallet to store his private keys and assets. In particular, a user can be either data owner or data consumer, that is,
  - Data owner, who wants to share data with specified users secretly;
  - Data consumer, who wants to obtain shared data from the public ledger.

Public ledger. Public ledger, which is composed of a series of blocks B_i in chronological order, is used to record transaction (or just "Tx" for short) in blockchain system. Specifically, in our system, public ledger is denoted as $B = (B_1, \cdots, B_i, \cdots)$. Moreover, each block B_i contains a lot of transactions, i.e. $B_i = \{T_{i,j}\}_{j\in[1,m]}$, and each transaction $T_{i,j}$ stores transaction content and other parameters (e.g. previous Tx ID, timestamp, etc.) in form of "key-value" pairs. As maintained above, the structure of public ledger can be seen in Fig. 2. Specially, as ciphertext, private data of transaction can be expressed by using script codes in our public ledger.

A. System Workflow

In our system, taking the blockchain as a medium, user shares his BIoT sensitive data (e.g. health records) to specified recipients by the public-key encryption mechanism. The main difficulty in this work is not the designing of encryption or

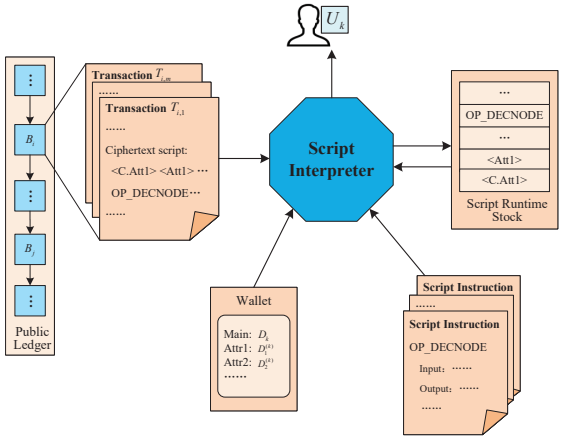

Fig. 2: The script system

decryption, but the key generation. Because, in the traditional sense, the user's private key is generated by a trusted central, but in the decentralized environment of blockchain, the master secret key, used to generate the user's private key, should be shared among all full nodes. Therefore, the user's private key is generated by interacting between user and full nodes. Besides, for reducing the user's computing burden, the process of encryption and decryption is completed at the blockchain nodes, which is called node encryption and decryption.

According to the discussion above, the workflow of the BIoT sensitive data sharing system is described as follows.

1) **Full node's key generation**. At the beginning of setting up the blockchain system, administrator generates system public key PK and master secret key MSK. Then, it distributes the fragments information of MSK for realizing the co-management of the master key among full nodes. Subsequently, the administrator will store PK into the public ledger and exit this system permanently (see Section IV-A);
2) **User's private key generation**. For the user U_k with attribute set $\mathbb{A}$, his private key sk_k is generated by interacting between U_k and all of full nodes P_i. Next, U_k stores sk_k into his wallet (see Section IV-B);
3) **Sensitive data encryption**. The data owner utilizes the CP-DK-ABE to encrypt the message M under the policy $\mathcal{T}$ and public key PK to generate the ciphertext $CT_{\mathcal{T}}$. With the help of the script system, $CT_{\mathcal{T}}$ can be encoded as the script codes, and encapsulated into the transaction. Finally, the data owner submits the transaction into the public ledger (see Section IV-C);
4) **Ciphertext decryption**. The k-*th* consumer U_k first downloads the corresponding transaction Tx from public ledger, then extracts the ciphertext script from Tx by executing the de-encapsulation. As shown in Fig. 2, the script interpreter supporting decryption opcodes, extracts the private key of the consumer from wallet to recover the message through the script running stack. If the attribute set $\mathbb{A}$ satisfies the access policy $\mathcal{T}$, that is, $\mathbb{A} \vDash \mathcal{T}$, then the user U_k can recover the message M from the ciphertext $CT_{\mathcal{T}}$ by running the script interpreter (see Section IV-D).

B. Script System

Scripting is a simple and lightweight programming language that can interpret tasks automatically by using stack and Reverse Polish Notation. Based on the scripting language, a novel encryption/decryption framework is designed in this work. Thanks to the above features of scripting language, in our CP-ABE scheme, user's encryption and decryption operations are automatically executed by the blockchain node, so as to reduce the difficulty of programming in BIoT devices. Specifically, as shown in Fig. 2, our script system contains the following entities.

- **Transaction**. It is an entity which is used to share sensitive data in BIoT. The private content of transaction is expressed by the script ciphertext.
- **Wallet**. It is an entity which stores the user's private key and assets in the form of "key-value" pairs.
- **Script Running Stack**. In an entity, script codes can be operated according to stack rules. Finally, it returns the results at the top of the stack.
- **Script Instruction**. It is an entity which is used to describe the flow of algorithm corresponding to operation codes.
- **Script Interpreter**. It is similar to a "virtual processor", which extracts relevant components from *Transaction*, *Wallet* and *Script Instruction*, and then returns the results through *Script Runtime Stock* processing.

C. The Definition of CP-DK-ABE

The ciphertext-policy decentralized-key attribute-based encryption (CP-DK-ABE) scheme is defined under the following four algorithms. Without loss of generality, in this definition, there are n full nodes $P = \{P_i\}_{i\in[1,n]}$. Next, we will show the formal definition of CP-DK-ABE scheme.

- **Setup algorithm**. This algorithm is executed by the system administrator. After inputting security parameters, the administrator generates system public key PK, and assigns the private key SK_i for each full node $P_i \in P$. After that, the administrator exits this system and no longer participates in any other operations. I.e., $Setup(1^\lambda) \rightarrow \{PK, \{SK_i\}_{i\in[1,n]}\}$.
- **Key-generation algorithm**. Different with two-party key exchange protocol (such as [27]), this algorithm is used to generate user's private key by interacting between all of nodes in P and the user U_k. It takes the user's attribute set $\mathbb{A}$, identity ID_k, PK and all SK_i as inputs. Then the results of interaction between all of nodes in P and the user U_k, i.e. $\langle U_k \leftrightarrow \{P_i\,(SK_i)\}_{i\in[1,n]}\rangle\,(ID_k, \mathbb{A}, PK) \rightarrow sk_k$, is treated as the outputs of the key-generation algorithm. I.e., $KeyGen(\mathbb{A}, ID_k, PK, \{SK_i\}_{i\in[1,n]}) \rightarrow sk_k$.
- **Encryption algorithm**. The algorithm is run by data owner. Taking PK, message M and access policy $\mathcal{T}$ as inputs, this algorithm outputs the relevant ciphertext $CT_{\mathcal{T}}$. I.e., $Enc(M, \mathcal{T}, PK) \rightarrow CT_{\mathcal{T}}$.
- **Decryption algorithm**. The data consumer executes this algorithm and inputs user's private keys and the ciphertext $CT_{\mathcal{T}}$. Then, it outputs the message M if and only if the

user's attribute set $\mathbb{A}$ satisfies the access policy under the ciphertext, that is, $\mathbb{A} \vDash \mathcal{T}$. I.e., $Dec(CT_{\mathcal{T}}, sk_k) \rightarrow M$.

Correctness. For any user ID_k with attribute set $\mathbb{A}$ and an access policy $\mathcal{T}$, we can get the private key $sk_k \leftarrow KeyGen(\mathbb{A}, ID_k, PK, SK_i)$ and ciphertext $CT_{\mathcal{T}} \leftarrow Encryption(M, \mathcal{T}, PK)$. If user attribute set satisfies the access policy, i.e. $\mathbb{A} \vDash \mathcal{T}$, then the algorithm $Dec(CT_{\mathcal{T}}, sk_k)$ will returns M, that is,

$$\Pr\left[Dec(CT_{\mathcal{T}}, sk_k) \rightarrow M : \mathbb{A} \vDash \mathcal{T}\right] = 1. \tag{1}$$

IV. The proposed scheme

In this section, we improve BSW's CP-ABE scheme [9] to construct a new CP-DK-ABE in order to make it more suitable for BIoT. The CP-DK-ABE also consists of four algorithms, i.e., *setup, key-generation, encryption* and *decryption*, in consistent with common CP-ABE. Specially, compared with the BSW's scheme, we redesign setup and key-generation algorithms for the decentralization setting based on the Secret Sharing Scheme (SSS) and secure computing, but the encryption and decryption algorithms remain almost unchanged.

A. Setup Algorithm

This algorithm is executed by the system administrator to generate public key and private key of full node. After executing this algorithm, the administrator will exit this program permanently. Specially, the setup algorithm inputs security parameter 1^{λ} and generates the public key PK and the private key SK_i for each full node P_i in $P = \{P_i\}_{i\in[1,n]}$.

Step 1. The system administrator runs the setup algorithm to get the master secret key and some other public parameters. Firstly, administrator executes this algorithm to achieve a bilinear group Φ with prime order p. Then it randomly picks the generator $g \in \mathbb{G}$ and the collision-resistant hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Finally, the administrator randomly chooses $\alpha, \beta \in \mathbb{Z}_p^*$ as the master secret key of this system.

Step 2. This step is the process of public key generation. The administrator randomly chooses $p_i \in \mathbb{Z}_p^*$ for each full node P_i and computes $Y = e(g,g)^{\alpha}$, $h = g^{\beta}$, where $i \in [1,n]$. Then it sets the public key PK as

$$PK = \left\{\Phi, g, H_1, H_2, \{p_i\}_{i\in[1,n]}, Y, h\right\}. \tag{2}$$

Step 3. In order to decentralize the authority of the blockchain system, we should avoid all of the master secret keys being managed by only one node. In this step, the administrator shares the master key α among all of P_i by using Shamir's (t,n)-threshold secret sharing scheme [28]. In this case, each of nodes can obtain a shared segment of α, and the master key α can only be recovered if and only if at least t full nodes cooperate.

For sharing the system master key α, the administrator first randomly selects a polynomial $f(x) = b_0 + b_1 x + \cdots + b_{t-1}x^{t-1} \pmod p$ of degree $t-1$, such that $\alpha = b_0$ and $b_1, \cdots, b_{t-1}$ are random elements in a certain finite field. Then the administrator computes $\alpha_1 = f(p_1)$, $\alpha_2 = f(p_2)$, $\cdots$, $\alpha_n = f(p_n)$ and distributes α_i to the shareholder P_i secretly. Finally, the private key of full node P_i is defined as

$$SK_i = \left\{g^{\frac{1}{\beta}}, f(p_i)\right\} \in \mathbb{G} \times \mathbb{Z}_p^*. \tag{3}$$

B. Key-generation Algorithm

With the help of all of SK_i from P, the key-generation algorithm inputs the user's identity ID_k, attribute set $\mathbb{A}$ and PK, outputs the private key sk_k for user ID_k. This algorithm is described by the interactive process between private key requester and each blockchain node P_i. As shown in Fig. 3, this algorithm includes the four following steps.

Step 1. For the user ID_k with attribute set $\mathbb{A} = \left\{Att_1^{(k)}, Att_2^{(k)}, \cdots, Att_m^{(k)}\right\}$, he first sends $\mathbb{A}$ to P_i and computes $H_2(ID_k) = \varphi_k$. Then the user computes identity's parameters g^{φ_k}, $g^{\frac{\varphi_k}{\beta}}$ and sends them to P_i.

Step 2. This step is used to describe the generation of full node's parameters. Firstly, for each attribute $Att_l^{(k)} \in \mathbb{A}$, the node P_i chooses a random number $r_{i,l} \in \mathbb{Z}_p^*$. Then P_i picks a random number θ_i and shares it among all of P_i by using Shamir's (t,n)-threshold secret sharing scheme as follows. P_i randomly chooses a degree $t-1$ polynomial,

$$h_i(x) = \theta_i + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1} \pmod p, \tag{4}$$

such that $a_{i,1}, \cdots, a_{i,t-1}$ are random elements in a certain finite field. Finally, P_i computes n shared segments $h_i(p_1)$, $h_i(p_2)$, $\cdots$, $h_i(p_n)$ and distributes $h_i(p_j)$ to shareholder P_j secretly, where $j \in [1,n]$. When the node P_i receives all the shared segments $h_j(p_i)$ from P_j, it computes $\sum_{j=1}^{n} h_j(p_i)$ individually.

Step 3. Each node P_i uses its private key SK_i and the parameters created in Steps 1-2 to compute the fragment information of user's private key.

For fragments of the main private key, P_i computes

$$D_{k,i} = \left(g^{\frac{1}{\beta}}\right)^{f(p_i)} \cdot \left(g^{\frac{\varphi_k}{\beta}}\right)^{\sum_{j=1}^{n} h_j(p_i)} \in \mathbb{G}. \tag{5}$$

For attribute $Att_l^{(k)} \in \mathbb{A}$, P_i computes fragments of attribute subkey as

$$\begin{cases} D'^{(k)}_{i,l} = \left(g^{\varphi_k}\right)^{\sum_{j=1}^{n} h_j(p_i)} H_1\left(Att_l^{(k)}\right)^{r_{i,l}} \in \mathbb{G}, \\ D''^{(k)}_{i,l} = g^{r_{i,l}} \in \mathbb{G}. \end{cases} \tag{6}$$

Finally, P_i sends $D_{k,i}$, $D'^{(k)}_{i,l}$ and $D''^{(k)}_{i,l}$ to the user ID_k.

Step 4. In this step, by using the Lagrange interpolation formula, users will reconstruct their main private key and attribute subkey from the received key fragments. We define the Lagrange coefficient as $L_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-p_j}{p_i-p_j}$ for $i \in \mathbb{Z}_p$ and a set $S \subseteq \{1, 2, \cdots, n\}$.

For the main private key, it reconstructs:

$$D_k = \prod_{i=1}^{t} D_{k,i}^{L_{i,S}(0)} \in \mathbb{G}. \tag{7}$$

For the attribute subkey, the user reconstructs:

$$\begin{cases} D'^{(k)}_{l} = \prod_{i=1}^{t} D'^{(k)}_{i,l}{}^{L_{i,S}(0)} \in \mathbb{G}, \\ D''^{(k)}_{l} = \prod_{i=1}^{t} D''^{(k)}_{i,l}{}^{L_{i,S}(0)} \in \mathbb{G}. \end{cases} \tag{8}$$

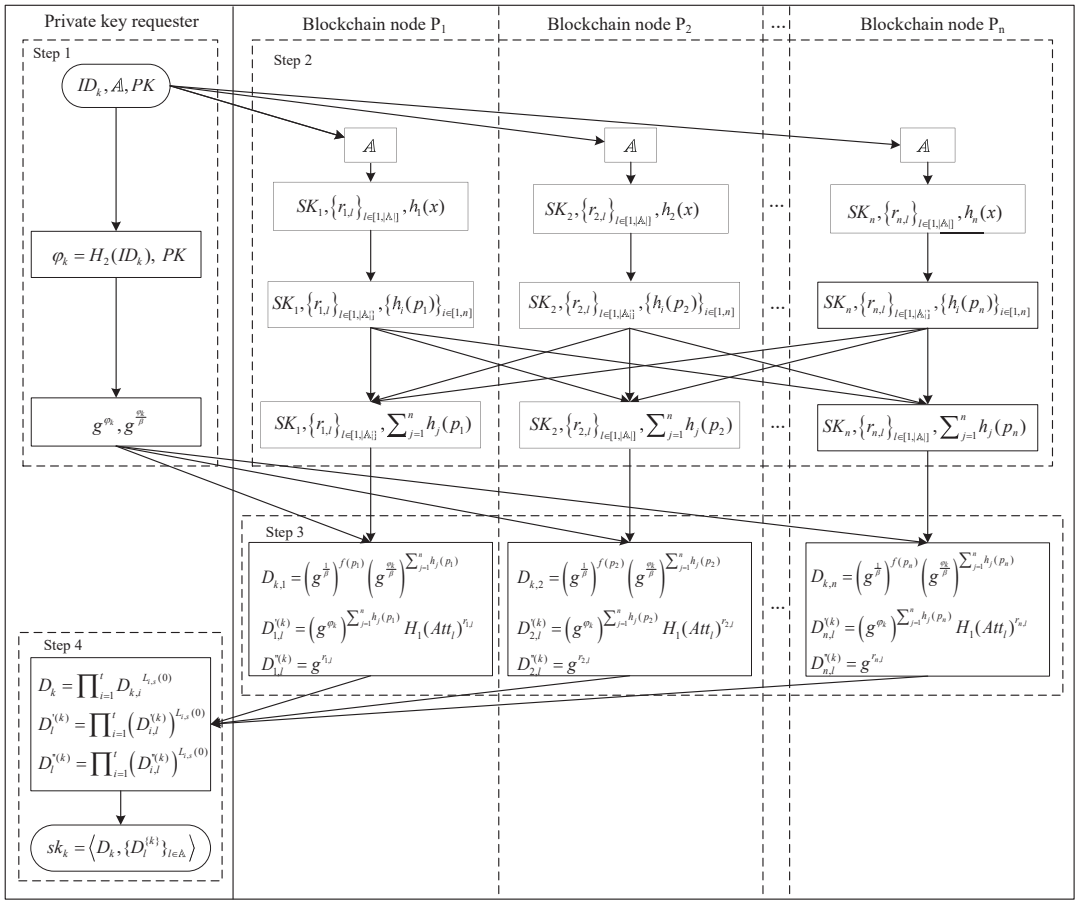

Fig. 3: The process of user's private key by our key-generation algorithm

Finally, let $D_l^{(k)} = \langle D'^{(k)}_l, D''^{(k)}_l \rangle$ and the user gets his private key sk_k as follows,

$$sk_k = \{D_k, \{D_l^{(k)}\}_{l \in \mathbb{A}}\} \in \mathbb{G} \times \mathbb{G}^{m \times 2}. \tag{9}$$

C. Encryption Algorithm

This algorithm is used by light nodes to encrypt sensitive data individually. Specially, our encryption is the same as $Encrypt$ algorithm in the BSW's CP-ABE scheme [9]. For clarity, we only consider the (k, n)-threshold secret sharing in the case of $n = 2$, that is, our encryption algorithm will only focus on the simple logic AND and OR gates for only two outputs[1]. Before describing this algorithm, we will introduce some preliminaries about access tree.

Given an access policy built on Boolean function, its access structure [9] can be described by a binary tree $\mathcal{T}$. In this tree, each non-leaf node is a logic AND or OR gate which has two child nodes, and each leaf node relates to a Boolean predicate for attribute matching. More details are described in Section V-C. Moreover, each node, including non-leaf or leaf node, is associated with a secret value. Let $v_{\tilde{x}}$ denote the secret value of the node $\tilde{x}$, and $v_{\tilde{x},l}, v_{\tilde{x},r}$ be the secret values of its left and right child, respectively.

[1]The AND and OR gates can be implemented by $(2, 2)$- and $(1, 2)$-threshold secret sharing, respectively.

To encrypt a massage M with the access tree $\mathcal{T}$, the encryption algorithm executes the following steps:

Step 1. At first, the root node in $\mathcal{T}$ is assigned to a random number $s \in \mathbb{Z}_p^*$ as the initial secret value, that is, $v_{root} = s$. Then, the algorithm uses the "from top to down" approach to share this secret value into all nodes in the tree. This process adopts the following "AND/OR" secret sharing schemes to share the value $v_{\tilde{x}}$ of node $\tilde{x}$ into its children, $v_{\tilde{x},l}$ and $v_{\tilde{x}.r}$, i.e.,

1) **Logic AND gate:** picks a new random number $r \in \mathbb{Z}_p^*$, and sets the left child's value $v_{\tilde{x},l} = r$ and the right child's value $v_{\tilde{x},r} = v_{\tilde{x}} - v_{\tilde{x},l} = s - r$;
2) **Logic OR gate:** sets the equivalence of the left and right children, i.e., $v_{\tilde{x},l} = v_{\tilde{x},r} = v_{\tilde{x}} = s$.

Finally, each of leaf nodes is assigned a secret value after the above process stops.

Step 2. This step is used to encrypt the leaf node, called $EncNode()$. Let $\mathcal{Y}$ be the set of leaf nodes in the access tree $\mathcal{T}$. For any node $\tilde{x} \in \mathcal{Y}$, the algorithm computes the attribute subcipher as $C_{\tilde{x}} = \langle C'_{\tilde{x}}, C''_{\tilde{x}} \rangle \leftarrow EncNode(PK, \tilde{x}, v_{\tilde{x}})$, where

$$C'_{\tilde{x}} = g^{v_{\tilde{x}}} \quad and \quad C''_{\tilde{x}} = H_1(att(\tilde{x}))^{v_{\tilde{x}}}. \tag{10}$$

So that, it produces the attribute subcipher $\{C_{\tilde{x}}\}_{\tilde{x} \in \mathcal{Y}}$.

Step 3. For the root node in $\mathcal{T}$ and a message $M \in \mathbb{G}_T$, the encryption algorithm computes data subciphers $\tilde{C} = M \cdot Y^s$ and $C_0 = h^s$ by using the initial secret value s. Finally, the

TABLE II: Several examples for opcodes in Bitcoin script

Word	Opcode	Input	Output	Description
OP_DUP	0x76	x	$x\ x$	Duplicates the top stack item.
OP_HASH160	0xa9	x	hash(x)	The input is hashed twice, first with SHA-256 and then with RIPEMD-160.
OP_CHECKSIG	0xac	sig,pubkey	True/False	Checks that the input signature is a valid signature using the input public key for the hash of the current transaction.
OP_EQUALVERIFY	0x88	a, b		Checks if two top-most items are equal, if not the entire script execution fails.

ciphertext $CT_{\mathcal{T}}$ under $\mathcal{T}$ is generated as

$$CT_{\mathcal{T}} = \left\{\tilde{C}, C_0, \{C_{\tilde{x}}\}_{\tilde{x}\in\mathcal{Y}}\right\} \in \mathbb{G}_T \times \mathbb{G} \times \mathbb{G}^{|\mathcal{Y}|\times 2}. \quad (11)$$

D. Decryption Algorithm

Similar to the $Decrypt$ in the BSW's CP-ABE scheme [9], our decryption algorithm can be regarded as the reverse of the above encryption. Specifically, this algorithm can be divided into three steps: *leaf-node matching*, *logical-node decryption* and *data subcipher decryption*.

Step 1 (Leaf-node matching). This step is used to decrypt the leaf node in $\mathcal{T}$, called $DecNode()$. For the leaf node $\tilde{x}$, let $att(\tilde{x})$ denote the corresponding attribute and $C_{\tilde{x}}$ be the attribute subcipher about $\tilde{x}$. If the attribute $att(\tilde{x})$ is in $\mathbb{A}$, the attribute subkey $D^{(k)}_{att(\tilde{x})}$ could be used to decrypt the attribute subcipher $C_{\tilde{x}}$ as

$$\begin{aligned} N_{\tilde{x}} &= DecNode(C_{\tilde{x}}, D^{(\tilde{x})}_{att(\tilde{x})}) \\ &= \frac{e(D'^{(k)}_{att(\tilde{x})}, C'_{\tilde{x}})}{e(D''^{(k)}_{att(\tilde{x})}, C''_{\tilde{x}})} = e(g,g)^{\varphi_k \cdot v_{\tilde{x}} \cdot \Theta} \in \mathbb{G}_T, \end{aligned} \quad (12)$$

where $\Theta = \sum_{i=1}^{n} \theta_i \mod p$. Otherwise, the decryption result is invalid, i.e., $DecNode(C_{\tilde{x}}, D^{(k)}_{att(\tilde{x})}) = \perp$.

Step 2 (Logical-node decryption). For the logical node (non-leaf node) $\tilde{x}$ in $\mathcal{T}$, the set of its child node is denoted as $S_{\tilde{x}}$. Let $DceLogic()$ be the decryption algorithm of logic node, which inputs $\tilde{x}$, $\{N_{\tilde{z}}\}_{\tilde{z}\in S_{\tilde{x}}}$ and the Lagrange coefficient $L_{i,S'_{\tilde{x}}(0)}$, outputs the decryption result $N_{\tilde{x}}$ of $\tilde{x}$,

$$\begin{aligned} N_{\tilde{x}} &= DceLogic(\tilde{x}, \{N_{\tilde{z}}\}_{\tilde{z}\in S_{\tilde{x}}}, L_{i,S'_{\tilde{x}}(0)}) \\ &= \prod_{\tilde{z}\in S_{\tilde{x}}} N_{\tilde{z}}^{L_{i,S'_{\tilde{x}}}(0)} = e(g,g)^{\varphi_k \cdot v_{\tilde{x}} \cdot \Theta} \in \mathbb{G}_T, \end{aligned} \quad (13)$$

where i is the index of $\tilde{z}$ in $S_{\tilde{x}}$, $S'_{\tilde{x}}$ is the set of the index.

Finally, if the attribute set $\mathbb{A}$ satisfies the access tree $\mathcal{T}$, we can simply call the $DecLogic$ algorithm to calculate the decryption result of the root node in $\mathcal{T}$. $N_{root} = DecLogic(root, N_{\tilde{z}\in root}, L_{i,S'_{root}}(0)) = e(g,g)^{\varphi_k \cdot s \cdot \Theta}$.

Step 3 (Data subcipher decryption). Finally, the data subcipher is decrypted as follows,

$$\begin{aligned} \frac{\tilde{C} \cdot N_{root}}{e(C_0, D_k)} &= \frac{M \cdot e(g,g)^{\alpha \cdot s} \cdot e(g,g)^{\varphi_k \cdot s \cdot \Theta}}{e\left(g^{\beta \cdot s}, g^{(\varphi_k \cdot \Theta + \alpha)/\beta}\right)} \\ &= M \in \mathbb{G}_T. \end{aligned} \quad (14)$$

V. Script Implementation

Almost blockchains provide a built-in programming language to support multiple capabilities (e.g., payments, exchanges) and allow special operations on transaction, such as various cryptographic algorithms. Especially, Bitcoin's scripting language is one of the most representative blockchain's languages. In this section, we will apply this kind of scripting language to implement our CP-DK-ABE scheme for more convenient and concise encryption/decryption. Before implementing script-based cryptographic algorithms, we first give some descriptions of the scripting language.

A. Scripting Language

Bitcoin's script is a Forth-like, stack-based, reverse-polish programming language, which is used to describe the execution of a certain algorithm in a blockchain transaction. The scripting language is simple and intentionally not Turing-complete, but decentralized validation. Its complexity is limited since the number of executable operations can be predicted. We call it **minimalist program**. The requirements and limitations of the minimalist program is a necessary part of the functional design of the cryptocurrency.

More importantly, the script will be executed by the *interpreter* in blockchain nodes, rather than user's device. No organization can override its executed results, nor can any organization save the scripting intermediate states during the execution. Therefore, the script is suitable to implement encryption and signature mechanism for avoiding bugs and malicious codes.

Bitcoin script defines a rich set of operational codes (opcodes) for various advanced operations classified into flow control, stack, splice, bitwise logic, arithmetic, crypto, locktime, etc. These opcodes are represented as a combination of prefix "OP_" and suffix "operation name", e.g., OP_DUP, OP_HASH160, OP_CHECKSIG, OP_EQUALVERIFY, as showed in TABLE II. In this table, we list some Bitcoin script opcodes with basic description. For a given opcode, the corresponding operands, enclosed in angle brackets (e.g., $\langle$sig$\rangle$), will be pushed onto the stack before the opcode in terms of Reverse Polish Notation (RPN), which ensures that the script interpreter performs operations defined by opcode on the items of the stack.

B. New Opcodes

Although Bitcoin scripting system has introduced some kinds of opcodes (such as opcodes in TABLE II), it still

TABLE III: New added opcodes

Word	Opcode	Input	Output	Description
OP_WQUERY	0xb1	x	out	Returns the private key corresponding to x from wallet.
OP_DEC_NODE	0xb2	a, b	out	Returns the node decryption value.
OP_DEC_AND	0xb3	a, b	out	Returns the AND gate node decryption value.
OP_DEC_OR	0xb4	a, b	out	Returns the OR gate node decryption value.
OP_DECRYPT	0xb5	a, b, c, d	out	Returns the message M.

needs new instructions to support the implementation of our scheme. In TABLE III, we define five new opcodes for our CP-DK-ABE scheme by replacing the reserved opcodes from OP_NOP1 to OP_NOP5. The detailed descriptions of these instructions are presented as follows.

1) OP_WQUERY: It is a wallet query operation. A blockchain wallet is a collection of private keys, but it may also refer to a special program used to manage these keys. In our CP-DK-ABE scheme, the wallet stores the private key $sk_k = \{D_k, \{D_l^{(k)}\}_{l\in\mathbb{A}}\}$ with attribute set $\mathbb{A} = \{Att_1^{(k)}, Att_2^{(k)}, \cdots, Att_m^{(k)}\}$ for the user ID_k. For clarity, the private key will be represented by Key-Value Pairs (KVPs) in lookup table, e.g., $< Mainkey, D_k >$, $< Att_i^{(k)}, D^{(k)}_{Att_i^{(k)}} >$. Here, we define the identity of D_k as "Mainkey".

Algorithm 1 Wallet query (OP_WQUERY)

Input: the attribute x
Output: the key component out associated with x
1: **if** $x \in \{Att_1^{(k)}, Att_2^{(k)}, \cdots, Att_m^{(k)}\}$ **then**
2: $out \leftarrow$ extract the attribute key $D_x^{(k)}$ from wallet
3: **else if** x is "Mainkey" **then**
4: $out \leftarrow$ extract the main private key D_k from wallet
5: **else**
6: $out \leftarrow \perp$
7: **end if**

We show the program of OP_WQUERY in Algorithm 1. By running it, script interpreter executes this opcode to query a certain private key from wallet. It takes the attribute x as input and extracts the corresponding attribute subkey $D_x^{(k)}$ from the wallet's KVPs. For example, taking the attribute $Att_i^{(k)}$ as input, the interpreter outputs the subkey $D^{(k)}_{Att_i^{(k)}}$ and pushes it onto the stack. In addition, it outputs an empty symbol $\perp$ if the attribute x cannot be found in the KVPs.

2) OP_DEC_NODE: This is leaf-node matching operation in accordance with the Step 1 of Section IV-D. The interpreter runs this opcode as Algorithm 2. It pops the two most-top items, $C_{\tilde{x}}$ and $D_x^{(k)}$, from the stack, where $C_{\tilde{x}}$ is the attribute subcipher and $D_x^{(k)}$ is the attribute subkey. If $D_x^{(k)}$ is not $\perp$, then interpreter outputs the decryption result $N_{\tilde{x}}$ according to Equation (12), otherwise it pushes $\perp$ onto the stack.

3) OP_DEC_AND: This is a simplified Lagrange interpolation opcode with two inputs, which is used to decrypt non-leaf nodes with AND gates. Let $\tilde{x}$ represent a node, $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$ be the decryption results of $\tilde{x}$'s left and right child nodes, respectively. As shown in Algorithm 3, the interpreter

Algorithm 2 Node decryption (OP_DEC_NODE)

Input: attribute subcipher $C_{\tilde{x}} = \langle C'_{\tilde{x}}, C''_{\tilde{x}} \rangle$ and the attribute subkey $D_x^{(k)} = \langle D'^{(k)}_x, D''^{(k)}_x \rangle$
Output: the decryption result $N_{\tilde{x}}$ associated with $C_{\tilde{x}}$
1: **if** $D_x^{(k)}$ is not $\perp$ **then**
2: $E \leftarrow e\big(D'^{(k)}_{att(\tilde{x})}, C'_{\tilde{x}}\big)$
3: $F \leftarrow e\big(D''^{(k)}_{att(\tilde{x})}, C''_{\tilde{x}}\big)$
4: $F' \leftarrow \text{inverse}(F)$
5: $N_{\tilde{x}} \leftarrow E \cdot F'$
6: **else**
7: $N_{\tilde{x}} \leftarrow \perp$
8: **end if**

pops two top-most items, $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$, from the stack as inputs. If neither of them is $\perp$, the interpreter returns the decryption result of $N_{\tilde{x}}$, otherwise it pushes $\perp$ onto the stack.

Algorithm 3 AND gate decryption (OP_DEC_AND)

Input: the decryption results $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$ of AND-gate node $\tilde{x}$'s child nodes
Output: decryption result $N_{\tilde{x}}$ of node $\tilde{x}$
1: **if** neither $N_{\tilde{x}.left}$ nor $N_{\tilde{x}.right}$ is $\perp$ **then**
2: $N_{\tilde{x}} \leftarrow N_{\tilde{x}.left} \cdot N_{\tilde{x}.right}$
3: **else**
4: $N_{\tilde{x}} \leftarrow \perp$
5: **end if**

4) OP_DEC_OR: This is also a simplified Lagrange interpolation opcode with two inputs, and it is used to decrypt non-leaf nodes with OR gates. As shown in Algorithm 4, the interpreter pops two top-most items, $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$, from the stack as inputs. If neither $N_{\tilde{x}.left}$ nor $N_{\tilde{x}.right}$ is $\perp$, the interpreter randomly selects one of them as the decryption result of $\tilde{x}$; if only one of them is $\perp$, the interpreter outputs the other one as the decryption result of $\tilde{x}$; if both $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$ are $\perp$, the interpreter pushes $\perp$ onto the stack.

5) OP_DECRYPT: This opcode is used to decrypt the message from data subcipher complying with Equation (14). The script interpreter runs Algorithm 5 by popping the four top-most items, i.e., data subcipher C_0 and $\tilde{C}$, the main private key D_k and the decryption result N_{root} of root node. If neither D_k nor N_{root} is $\perp$, the interpreter returns the decryption result of data subcipher, otherwise it pushes $\perp$ onto stack.

Algorithm 4 OR gate decryption (OP_DEC_OR)

Input: the decryption results $N_{\tilde{x}.left}$ and $N_{\tilde{x}.right}$ of OR-gate node $\tilde{x}$'s child nodes
Output: decryption result $N_{\tilde{x}}$ of node $\tilde{x}$

if neither $N_{\tilde{x}.left}$ nor $N_{\tilde{x}.right}$ is $\perp$ **then**
  $N_{\tilde{x}} \leftarrow_R \{N_{\tilde{x}.left}, N_{\tilde{x}.right}\}$
else if one of them is $\perp$ **then**
  if $N_{\tilde{x}.left}$ is $\perp$ **then**
    $N_{\tilde{x}} \leftarrow N_{\tilde{x}.right}$
  else
    $N_{\tilde{x}} \leftarrow N_{\tilde{x}.left}$
  end if
else
  $N_{\tilde{x}} \leftarrow \perp$
end if

Algorithm 5 Data subcipher decryption (OP_DECRYPT)

Input: the data subcipher C_0, $\tilde{C}$, the main private key D_k and the decryption result of root node N_{root}
Output: the description result out of data subcipher

if neither D_k nor N_{root} is $\perp$ **then**
  $E \leftarrow \tilde{C} \cdot N_{root}$
  $F \leftarrow e(C_0, D_k)$
  $F' \leftarrow \text{inverse}(F)$
  $out \leftarrow E \cdot F'$
else
  $out \leftarrow \perp$
end if

C. Script-driven Encryption

In this section, we will describe the process of generating ciphertext script under an access policy tree in terms of our new opcodes. Since any multi-branches tree can be converted into a binary tree, without loss of generality, we set the access tree in our CP-DK-ABE scheme as a binary tree for the convenience of description.

Taking the following access policy as an example, we first convert it into the policy tree $\mathcal{T}$ in Fig. 4(a).

$$\begin{aligned} Policy ::&= (Att_1 \wedge Att_2) \vee Att_3 \\ &= (\text{“CS”} \wedge \text{“student”}) \vee \text{“professor”}. \end{aligned}$$

In Fig. 4(a), each leaf node denotes an attribute. Let I_{21} denote the attribute “CS”, I_{22} is “student” and I_{12} is “professor”. I_0 is a logic OR gate and I_{11} is a logic AND gate. It means that a *CS student* or a *professor* can satisfy access policy tree $\mathcal{T}$.

To encrypt a sensitive data under $\mathcal{T}$, encryptor picks a random number s and runs the Ciphertext Generation Algorithm (CGA) (Algorithm 6) to generate ciphertext $CT_{\mathcal{T}}$. In detail, the CGA inputs a node $\tilde{x}$, a secret value s, public key PK and out. If $\tilde{x}$ is leaf node, the algorithm computes attribute subcipher $C_{\tilde{x}} \leftarrow \langle C'_{\tilde{x}}, C''_{\tilde{x}} \rangle = \langle g^s, H_1(att(\tilde{x}))^s \rangle$ and appends $(C_{\tilde{x}}\ \langle att(\tilde{x}) \rangle$ OP_WQUERY OP_DEC_NODE$)$ to the out. If $\tilde{x}$ is a logic node with AND gate, the algorithm chooses a random number r and runs CGA($\tilde{x}$.left, r, out) and CGA($\tilde{x}$.right, $s - r$, out), then appends OP_DEC_AND to the out; if $\tilde{x}$ is a logic node with OR gate, it runs CGA($\tilde{x}$.left, s, out) and CGA($\tilde{x}$.right, s, out), and then appends OP_DEC_OR to the out. Moreover, if $\tilde{x}$ is the root node, the CGA computes the data subcipher, $\tilde{C} = e(g,g)^{\alpha s}$ and $C_0 = g^{\beta s}$, and then appends $(\tilde{C}\ C_0\ \langle Mainkey \rangle$ OP_WQUERY OP_DECRYPT$)$ to the out. Note that, $\langle Mainkey \rangle$ is the identity of data subcipher which is only used to indicate the ciphertext type.

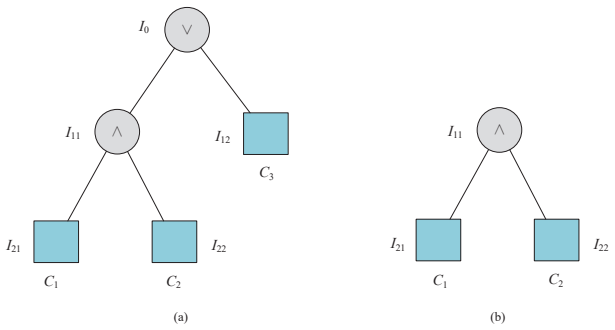

Fig. 4: Two examples on access tree

For instance, the ciphertext under the access policy tree $\mathcal{T}$ in Fig. 4(a) can be generated by CGA as follows.

$$\begin{aligned} CT_{\mathcal{T}} = \{ &\langle C_1 \rangle\ \langle Att_1 \rangle \text{ OP\_WQUERY OP\_DEC\_NODE } \langle C_2 \rangle \\ &\langle Att_2 \rangle \text{ OP\_WQUERY OP\_DEC\_NODE} \\ &\text{OP\_DEC\_AND } \langle C_3 \rangle\ \langle Att_3 \rangle \text{ OP\_WQUERY} \\ &\text{OP\_DEC\_NODE OP\_DEC\_OR } \langle \tilde{C} \rangle\ \langle C_0 \rangle \\ &\langle Mainkey \rangle \text{ OP\_WQUERY OP\_DECRYPT} \}. \end{aligned}$$

Algorithm 6 Ciphertext Generation Algorithm (CGA)

Input: node $\tilde{x}$, a secret value s, PK, message M, out
Output: ciphertext script

if node $\tilde{x}$ is leaf **then**
  Get the attribute $Att(x)$ of the node $\tilde{x}$
  $C_{\tilde{x}} \leftarrow \langle C'_{\tilde{x}}, C''_{\tilde{x}} \rangle = \langle g^s, H_1(att(\tilde{x}))^s \rangle$
  out.append $(C_{\tilde{x}}\ \langle Att(x) \rangle$ OP_WQUERY OP_DEC_NODE$)$
else
  if node.logic is AND gate **then**
    Pick up a random number r
    CGA($\tilde{x}$.left, r, out)
    CGA($\tilde{x}$.right, $s - r$, out)
    out.append(OP_DEC_AND)
  else
    CGA($\tilde{x}$.left, s, out)
    CGA($\tilde{x}$.right, s, out)
    out.append(OP_DEC_OR)
  end if
  if node $\tilde{x}$ is root **then**
    $\tilde{C} = M \cdot e(g,g)^{\alpha s}$, $C_0 = g^{\beta s}$
    out.append$(\tilde{C}\ C_0\ \langle Mainkey \rangle$ OP_WQUERY OP_DECRYPT$)$
  end if
end if

As shown in Fig. 5, the relationship among the five new opcodes in Fig. 4(b) can be roughly described as follows. Firstly,

TABLE IV: The execution process about script decryption of attribute subcipher under $\mathcal{T}_{I_{11}}$

	Stack	Script	Description
1	Empty	$\langle C_1\rangle$ $\langle Att_1\rangle$ OP_WQUERY OP_DEC_NODE $\langle C_2\rangle$ $\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	
2	$\langle C_1\rangle$	$\langle Att_1\rangle$ OP_WQUERY OP_DEC_NODE $\langle C_2\rangle$ $\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	$\langle C_1\rangle$ added onto the stack.
3	$\langle C_1\rangle$ $\langle Att_1\rangle$	OP_WQUERY OP_DEC_NODE $\langle C_2\rangle$ $\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	Attribute $\langle Att_1\rangle$ added onto the stack.
4	$\langle C_1\rangle$ $\langle sk_1\rangle$	OP_DEC_NODE $\langle C_2\rangle$ $\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	Query $\langle sk_1\rangle$ matched with $\langle C_1\rangle$.
5	$\langle N_{Att_1}\rangle$	$\langle C_2\rangle$ $\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	Decrypt the ciphertext $\langle C_1\rangle$.
6	$\langle N_{Att_1}\rangle$ $\langle C_2\rangle$	$\langle Att_2\rangle$ OP_WQUERY OP_DEC_NODE OP_DEC_AND	$\langle C_2\rangle$ added onto the stack.
7	$\langle N_{Att_1}\rangle$ $\langle C_2\rangle$ $\langle Att_2\rangle$	OP_WQUERY OP_DEC_NODE OP_DEC_AND	Attribute $\langle Att_2\rangle$ added onto the stack.
8	$\langle N_{Att_1}\rangle$ $\langle C_2\rangle$ $\langle sk_2\rangle$	OP_DEC_NODE OP_DEC_AND	Query $\langle sk_2\rangle$ matched with $\langle C_2\rangle$.
9	$\langle N_{Att_1}\rangle$ $\langle N_{Att_2}\rangle$	OP_DEC_AND	Decrypt the ciphertext $\langle C_2\rangle$.
10	$\langle N_{I_{11}}\rangle$	Empty	Decrypt the root node I_{11}.

by utilizing the private key extracted by the OP_WQUARY, the OP_DEC_NODE algorithm can be executed to decrypt the leaf node. Then the OP_DEC_AND and OP_DEC_OR can be executed to decrypt logic nodes with AND and OR gate, respectively. Finally, the decryption results of logic node are used by the OP_DECRYPT algorithm to recover the message from the ciphertext.

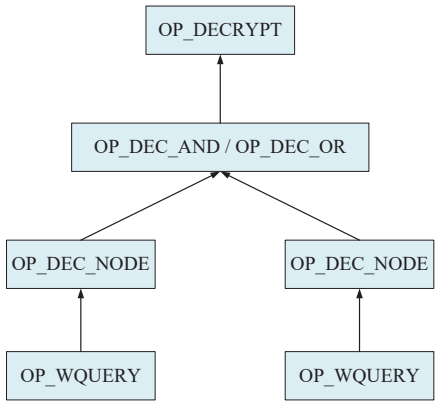

Fig. 5: The relationship among opcodes for access tree in Fig. 4(b)

D. Script-driven Decryption

In our construction, sensitive data is encrypted and submitted into the public ledge in the form of scripting code. A user who wants to open the protected data should run script interpreter to decrypt the ciphertext script with the help of his wallet and script instructions. In the following sections, we will show the process of script decryption in detail.

As shown in Fig. 4(b), we take the subtree $\mathcal{T}_{I_{11}}$ as an example to describe the process of decryption. The access policy under $\mathcal{T}_{I_{11}}$ is defined as $Policy ::= Att_1 \wedge Att_2 = (\text{“CS”}) \wedge (\text{“student”})$. According to CGA, we can easily get the ciphertext under the access tree $\mathcal{T}_{I_{11}}$. For convenience, we divide the ciphertext into two parts: attribute subcipher and data subcipher. The attribute subcipher under $\mathcal{T}_{I_{11}}$ is

$$\{\langle C_1\rangle\,\langle Att_1\rangle\,\text{OP\_WQUERY OP\_DEC\_NODE}\,\langle C_2\rangle\,\langle Att_2\rangle\,\text{OP\_WQUERY OP\_DEC\_NODE OP\_DEC\_AND}\};$$

and the data subcipher is

$$\{\langle \tilde{C}\rangle\,\langle C_0\rangle\,\langle Mainkey\rangle\,\text{OP\_WQUERY OP\_DECRYPT}\}.$$

Corresponding to the above categories, we divide the process of decryption into two steps: *script decryption of attribute subcipher* and *script decryption of data subcipher*. Next, taking the ciphertext under $\mathcal{T}_{I_{11}}$ as an example, we will show the process of script decryption in detail.

1) Script decryption of attribute subcipher: After reading the attribute subcipher from ciphertext script, the interpreter will execute this subcipher from left to right. As shown in TABLE IV, the execution process is described as follows.

- **Init**. The stack is initialized and empty it in Step 1.
- **The decryption of** C_1. In Step 2 and 3, script interpreter pushes $\langle C_1\rangle$ and $\langle Att_1\rangle$ onto the stack in turn. Then it runs the OP_WQUERY algorithm to obtain the private key $\langle sk_1\rangle$ corresponding to attribute $\langle Att_1\rangle$ in Step 4. In Step 5, the interpreter runs the OP_DEC_NODE algorithm and inputs $\langle C_1\rangle$, $\langle sk_1\rangle$, then returns the decryption result $\langle N_{Att_1}\rangle$ of $\langle C_1\rangle$.
- **The decryption of** C_2. Same as the decryption of C_1, script interpreter decrypts $\langle C_2\rangle$ and pushes the decryption result $\langle N_{Att_2}\rangle$ of $\langle C_2\rangle$ onto the stack in Step 6-9.
- **The decryption of the root node**. The interpreter takes $\langle N_{Att_1}\rangle$, $\langle N_{Att_2}\rangle$ as inputs and runs OP_DEC_AND algorithm to get the decryption result $N_{I_{11}}$ of the root node I_{11}.

2) Script decryption of data subcipher: After decrypting the attribute subcipher, the interpreter can use the decryption result $N_{I_{11}}$ of the root I_{11} to recover the message from data subcipher. As shown in TABLE V, the execution process is described as follows.

TABLE V: The execution process about script decryption of data subcipher under $\mathcal{T}_{I_{11}}$

	Stack	Script	Description
1	Empty	$\langle N_{I_{11}} \rangle$ $\langle \widetilde{C} \rangle$ $\langle C_0 \rangle$ $\langle Mainkey \rangle$ OP_WQUERY OP_DECRYPT	
2	$\langle N_{I_{11}} \rangle$ $\langle \widetilde{C} \rangle$ $\langle C_0 \rangle$	$\langle Mainkey \rangle$ OP_WQUERY OP_DECRYPT	$\langle \widetilde{C} \rangle$ $\langle C_0 \rangle$ $\langle A \rangle$ added onto the stack.
3	$\langle N_{I_{11}} \rangle$ $\langle \widetilde{C} \rangle$ $\langle C_0 \rangle$ $\langle Mainkey \rangle$	OP_WQUERY OP_DECRYPT	$\langle Mainkey \rangle$ added onto the stack.
4	$\langle N_{I_{11}} \rangle$ $\langle \widetilde{C} \rangle$ $\langle C_0 \rangle$ $\langle D_k \rangle$	OP_DECRYPT	The data private key $\langle D_k \rangle$ is exacted and pushed onto the stack.
5	M	Empty	Decrypt ciphertext and get the message M.

TABLE VI: Comparison between existing CP-ABE schemes and ours

Scheme	Decentralization	Access Structure	Standard Model	Hardness Assumption	Group Order	Programmable Ciphertext
Jiang et al. [12]	×	AND-gate	√	DBDH	Prime	×
Li et al. [14]	×	LSSS	√	Subgroup Decision	Composite	×
Belguith et al. [15]	×	LSSS	×	CDH, DBDH	Prime	×
Li et al. [17]	√	LSSS	√	q-PBDHE	Prime	×
Liang et al. [18]	√	LSSS	√	Subgroup Decision	Composite	×
Han et al. [22]	×	LSSS	√	q-BDHE, l-SDH	Prime	×
Islam et al. [23]	×	Tree	×	CDH	Prime	×
Our scheme	√	Tree	√	DLDH, DBDH	Prime	√

- **Init**. The stack is initialized and empty it in Step 1.
- **Wallet query**. The script interpreter pushes $\langle N_{I_{11}} \rangle$ and data subcipher $\langle \widetilde{C} \rangle$ and $\langle C_0 \rangle$ onto the stack in Step 2-4. Next, the interpreter executes the OP_WQUERY algorithm to query the private key about data subcipher in the wallet. Finally, the interpreter pushes the main private key D_k onto the stack.
- **The decryption of data subcipher**. The interpreter runs the OP_DECRYPT algorithm to recover the message M.

VI. System analysis

In this section, the performance and security of our system are analyzed. Specifically, in terms of performance, we compared the features, computation and storage overheads between some ABE schemes and our construction. In terms of security, we prove the privacy of our key-generation algorithm and the security of our CP-DK-ABE scheme against selective plaintext attack, respectively.

A. Performance Analysis

In TABLE VI, we make a comparison between seven existing CP-ABE schemes and ours in terms of decentralization, access structure, security model, hardness assumption, group order and ciphertext programmability. The schemes in [17], [18] and ours are distributed ABE, but a trusted authority center is necessary for managing the master secret key in other schemes. Besides, there are only two schemes, [23] and ours, to support tree-based access structure, which can express much more complex policy compared with either AND-gate [12] or LSSS [14], [15], [17], [18], [22]. In addition, the schemes, [14] and [18], are constructed in composite order groups, and the security proof of [15] and [23] depend on random oracle. However, our scheme is constructed in prime order groups and its security is reduced to the DLDH and DBDH assumptions under the standard model. It is worth noticing that only our scheme supports programmable ciphertext compared with other ABE schemes.

Let $E(\mathbb{G})$ and $E(\mathbb{G}_T)$ denote exponentiation operations in $\mathbb{G}$ and $\mathbb{G}_T$, respectively. $M(\mathbb{G}_T)$ is multiplication operation in $\mathbb{G}_T$ and B is bilinear pairing operation. Define $l_{\mathbb{G}}$, $l_{\mathbb{G}_T}$ and $l_{\mathbb{Z}_p^*}$ as the lengths of elements in $\mathbb{G}$, $\mathbb{G}_T$ and $\mathbb{Z}_p^*$, respectively. Besides, k is the number of attributes in the whole system, n is the number of full nodes and m is the number of attributes in access policy. We neglect operations in $\mathbb{Z}_p^*$, multiplication in $\mathbb{G}$ and hash operation, because they are much more efficient than exponentiation operations.

In TABLE VII, we provide the comparison of computation and storage costs between few existing DABE schemes ([17]–[19]) and ours. It shows that, the computation costs of our scheme, both encryption and decryption, are $\mathcal{O}(m)$, that means the costs are linear with the number of attributes in access policy. The encryption costs of [19] are similar as our scheme, but its decryption costs are $\mathcal{O}(m+n)$, which is larger than ours. The computation coats of [17] are also $\mathcal{O}(m+n)$ for either encryption or decryption. The costs of each of these two operations in scheme [18] are $\mathcal{O}(m)$, but they are larger than those in our scheme.

We turn our attention to storage costs. In our scheme, the size of the public key only relates to the number of full nodes, while that of [18] is linear with the number k of attributes. Considering practical DABE schemes, the number of attributes is generally more than the number of full nodes, i.e., $k >$

TABLE VII: Comparison of the computation and storage overheads

Scheme	Computation Costs		Storage Costs	
	Encryption	Decryption	Public Key	Ciphertext
Li et al. [17]	$(n+3m)\cdot E(\mathbb{G})+n\cdot E(\mathbb{G}_T)$	$(n+2m)\cdot B+(3+m)\cdot E(\mathbb{G}_T)$	$(2n+k)\cdot l_{\mathbb{G}}$	$(2m+n)\cdot l_{\mathbb{G}}+2\cdot l_{\mathbb{G}_T}$
Liang et al. [18]	$(4m+1)\cdot E(\mathbb{G})+2m\cdot E(\mathbb{G}_T)+1\cdot B+1\cdot M(\mathbb{G}_T)$	$4m\cdot B+m\cdot E(\mathbb{G}_T)+(3m-1)\cdot M(\mathbb{G}_T)$	$3k\cdot l_{\mathbb{G}}+k\cdot l_{\mathbb{G}_T}$	$(3n+1)\cdot l_{\mathbb{G}}+(n+1)\cdot l_{\mathbb{G}_T}$
Nasiraee et al. [19]	$(2m+1)\cdot E(\mathbb{G})+1\cdot E(\mathbb{G}_T)+1\cdot B+1\cdot M(\mathbb{G}_T)$	$4m\cdot B+2m\cdot E(\mathbb{G}_T)+(2n+2)\cdot E(\mathbb{G})$	$(3n+3k)\cdot l_{\mathbb{G}}$	$2m\cdot l_{\mathbb{G}}+1\cdot l_{\mathbb{G}_T}+m\cdot l_{\mathbb{Z}_p^*}$
Our scheme	$(2m+2)\cdot E(\mathbb{G})+1\cdot M(\mathbb{G}_T)$	$(2m+1)\cdot B+(m-1)\cdot E(\mathbb{G}_T)$	$2\cdot l_{\mathbb{G}}+1\cdot l_{\mathbb{G}_T}+n\cdot l_{\mathbb{Z}_p^*}$	$(2m+1)\cdot l_{\mathbb{G}}+1\cdot l_{\mathbb{G}_T}$

n. Therefore, our scheme is better than [18]. In addition, the public key storage costs of schemes, either [17] or [19], are $\mathcal{O}(m+n)$. On the other hand, the ciphertext storage costs of scheme [18] are linear with the number of full nodes, those of either [19] or ours are linear with the number of attributes in access policy, and those of [17] are linear with both the number of attributes in access policy and the number of full nodes. Therefore, our scheme has relative low storage costs.

To test the practical performance of our scheme, we implement it based on the well-known Java Pairing Based Cryptography Library (JPBC) and Intellij IDEA 2020.3.3. The experiments are executed on 64-bit Windows 10 under Intel(R) Core(TM) i5-4590S CPU @3.00GHz, 8G ROM. In our experiments, we select the Type-A pairings on JPBC with 160-bit group order, where the pairings are constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime $q = 3 \mod 4$ and the order r is some prime factor of $q+1$.

TABLE VIII: Encryption and decryption time of our scheme

The number of attributes	5	10	15	20	25
Encryption time (ms)	128.43	259.38	352.27	497.18	616.06
Decryption time (ms)	84.12	181.35	241.59	355.75	451.94

In TABLE VIII, we list the encryption and decryption time of our scheme in the experiments. The running time of encryption and decryption algorithms increases with higher number of attributes in access policy, which is consistent with the results of theoretical analysis in TABLE VII.

In Fig. 6 and 7, we further compare the encryption and decryption time overheads of our scheme with those of [17]–[19], respectively. For the sake of comparison, we set the number of full nodes as $n = 10$. As shown in Fig. 6, the encryption time of these four schemes increases with the increases of attribute numbers in access policy. Specifically, the encryption time of [17] is approximate to that of our scheme, and they are lower than other schemes. Next, in Fig. 7 we make a comparison on the time costs of decryption algorithms among the above-mentioned schemes. It is easy to see the same results in which the decryption costs of our scheme are better than other schemes. In summary, the experiments' results show that our scheme are efficient in encryption and decryption processes.

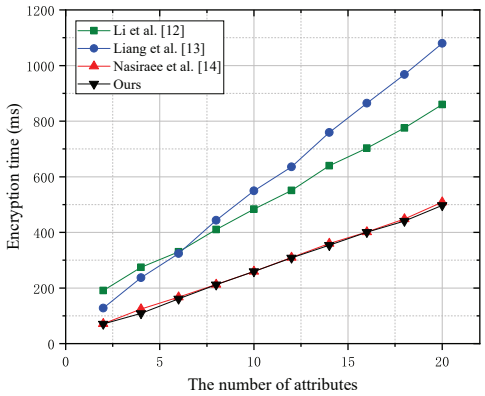

Fig. 6: Encryption time comparison with other schemes

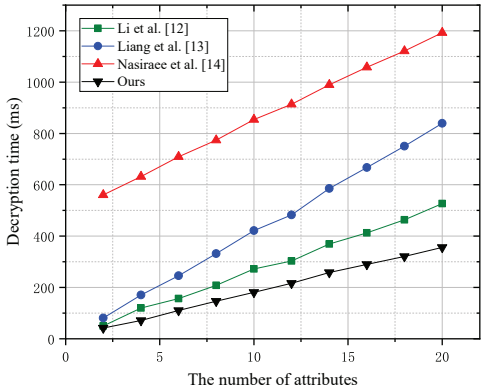

Fig. 7: Decryption time comparison with other schemes

B. Privacy Analysis of Key Generation Algorithm

In this section, the privacy of our key-generation algorithm will be analyzed from two aspects: *passive* and *active* adversary attack. *Passive* means that the adversary merely eavesdrops on the communication line; *active* means that the adversary who is monitoring the channel and may tamper with the messages sent on it. The key privacy of our key generation algorithm means that no information of the user's private-key, as the result of interactive process, as well as the master secret key held by all of full nodes will be leaked no matter whether the inputs of nodes are attacked by active or passive adversaries.

Theorem 1. *If both Shamir's (t, n)-threshold secret sharing scheme [28] and Decision Linear Diffie-Hellman (DLDH) assumption hold, our key-generation algorithm is key private*

for a limited number of corrupted full nodes. That is, in the case that at most of $t-1$ full nodes are corrupted, for an adversary $\mathcal{A}$, the advantage of correctly guessing the input value $\Theta = \sum_{i=1}^{n} \theta_i$ about the blockchain nodes is negligible.

The proof of this theorem is presented in the supplementary materials.

C. Security Analysis of CP-DK-ABE Scheme

In our CP-DK-ABE scheme, the security model of indistinguishability under chosen-plaintext attacks (IND-CPA) can be described by the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ as shown in Fig 8.

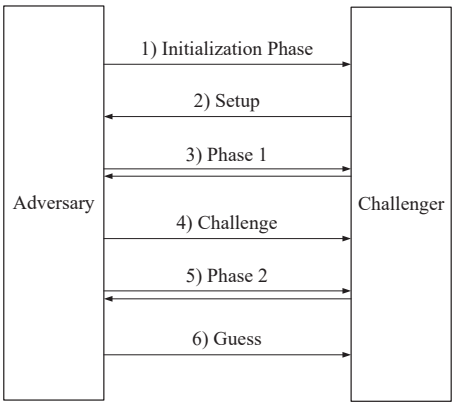

Fig. 8: Security game between adversary and challenger.

- **Init**. Adversary $\mathcal{A}$ announces the access policy W^* to be challenged.
- **Setup**. Challenger $\mathcal{B}$ runs the Setup algorithm according to the proposed scheme, then $\mathcal{B}$ sends the public key PK to $\mathcal{A}$ and keeps the master key SK_i as secrets.
- **Phase 1**. Adversary $\mathcal{A}$ makes private-key queries on its attribute set $\mathbb{A}$, where $\mathbb{A}$ does not satisfy the challenge policy W^*, i.e., $\mathbb{A} \nvDash W^*$.
- **Challenge**. $\mathcal{A}$ outputs two message M_0 and $M_1 \in \mathbb{G}_T$ with the same length, then the challenger randomly picks a number $\rho \in \{0,1\}$ and encrypts M_ρ under the access policy W^* following proposed Encryption algorithm. Finally, $\mathcal{B}$ outputs the ciphertext CT_{W^*}.
- **Phase 2**. Same as the Phase 1, adversary $\mathcal{A}$ makes private-key queries on some attribute sets. It requires that no queried attribute set satisfies the policy W^*.
- **Guess**. Adversary $\mathcal{A}$ outputs a guess ρ' of ρ and wins the game if $\rho' = \rho$.

The advantage of $\mathcal{A}$ in the above game is defined as

$$Adv_{\mathcal{A}} = \left| \Pr\left[\rho = \rho'\right] - \frac{1}{2} \right|, \tag{15}$$

where the probability is taken over the random bits used by the challenger and the adversary.

Definition 1. *A CP-DK-ABE scheme is secure against the IND-CPA, if for any PPT adversary $\mathcal{A}$, its advantage $Adv_{\mathcal{A}}$ is negligible.*

Theorem 2. *Under the Decisional Bilinear Diffie-Hellman assumption, our CP-DK-ABE scheme is semantically secure under chosen-plaintext attacks (IND-CPA).*

The proof of this theorem is presented in the supplementary materials.

VII. Conclusion

In this paper, we propose a CP-DK-ABE scheme with decentralized key generation, which is suitable to share the sensitive data in BIoT. Specially, the master secret key is managed by all of full nodes collectively. Moreover, we ensure that, by designing the programmable ciphertext, each of blockchain nodes can execute encryption and decryption automatically through running the script system. This kind of script-driven ciphertext can reduce the difficulty of programming in BIoT devices. In addition, our key-generation algorithm is proven to protect the secret key held by the full nodes, and our CP-DK-ABE scheme is semantically secure under the DBDH assumption.

Acknowledgment

This work was supported by the National Key Technologies R&D Programs of China (2018YFB1402702) and the National Natural Science Foundation of China (61972032).

References

[1] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. A. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2188–2204, 2019.

[2] M. F. Aziz, A. N. Khan, J. Shuja, I. A. Khan, F. G. Khan, and A. u. R. Khan, "A lightweight and compromise-resilient authentication scheme for IoTs," *Trans. Emerg. Telecommun. Technol.*, early access, Nov. 25, 2019.

[3] M. N. Alraja, H. Barhamgi, A. Rattrout, and M. Barhamgi, "An integrated framework for privacy protection in iot - applied to smart healthcare," *Comput. Electr. Eng.*, vol. 91, p. 107060, 2021.

[4] E. Chen, Y. Zhu, G. Zhu, K. Liang, and R. Feng, "How to implement secure cloud file sharing using optimized attribute-based access control with small policy matrix and minimized cumulative errors," *Comput. Secur.*, vol. 107, p. 102318, 2021.

[5] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Proceedings of Advances in Cryptology - EUROCRYPT 2010*. Springer, 2010, pp. 62–91.

[6] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proceedings of Public Key Cryptography - PKC 2011*. Springer, 2011, pp. 53–70.

[7] 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, 2021.

[8] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proceedings of Advances in Cryptology - EUROCRYPT 2005*. Springer, 2005, pp. 457–473.

[9] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of IEEE Symposium on Security and Privacy (S&P 2007)*. IEEE, 2007, pp. 321–334.

[10] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*. ACM, 2006, pp. 89–98.

[11] C. Li, Q. Shen, Z. Xie, X. Feng, Y. Fang, and Z. Wu, "Large universe CCA2 CP-ABE with equality and validity test in the standard model," *Comput. J.*, vol. 64, no. 4, pp. 509–533, 2021.

[12] Y. Jiang, W. Susilo, Y. Mu, and F. Guo, "Ciphertext-policy attribute-based encryption against key-delegation abuse in fog computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 720–729, 2018.

[13] M. Chase, "Multi-authority attribute based encryption," in *Proceedings of Theory of Cryptography, TCC 2007*. Springer, 2007, pp. 515–534.
[14] Q. Li, J. Ma, R. Li, X. Liu, J. Xiong, and D. Chen, "Secure, efficient and revocable multi-authority access control system in cloud storage," *Comput. Secur.*, vol. 59, pp. 45–59, 2016.
[15] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, and R. Attia, "PHOABE: securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted iot," *Comput. Networks*, vol. 133, pp. 141–156, 2018.
[16] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of Advances in Cryptology - EUROCRYPT 2011*. Springer, 2011, pp. 568–588.
[17] J. Li, S. Hu, Y. Zhang, and J. Han, "A decentralized multi-authority ciphertext-policy attribute-based encryption with mediated obfuscation," *Soft Comput.*, vol. 24, no. 3, pp. 1869–1882, 2020.
[18] P. Liang, L. Zhang, L. Kang, and J. Ren, "Privacy-preserving decentralized ABE for secure sharing of personal health records in cloud storage," *J. Inf. Secur. Appl.*, vol. 47, pp. 258–266, 2019.
[19] H. Nasiraee and M. Ashouri-Talouki, "Anonymous decentralized attribute-based access control for cloud-assisted iot," *Future Gener. Comput. Syst.*, vol. 110, pp. 45–56, 2020.
[20] J. Li, Q. Yu, and Y. Zhang, "Hierarchical attribute based encryption with continuous leakage-resilience," *Inf. Sci.*, vol. 484, pp. 113–134, 2019.
[21] Q. Tian, D. Han, X. Liu, and X. Yu, "Lwe-based multi-authority attribute-based encryption scheme with hidden policies," *Int. J. Comput. Sci. Eng.*, vol. 19, no. 2, pp. 233–241, 2019.
[22] D. Han, N. Pan, and K.-C. Li, "A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection," *IEEE Trans. Dependable Secure Comput.*, early access, Mar. 2, 2020.
[23] M. A. Islam and S. Madria, "Attribute-based encryption scheme for secure multi-group data sharing in cloud," *IEEE Trans. Serv. Comput.*, early access, Nov. 18, 2020.
[24] M. Möser, I. Eyal, and E. G. Sirer, "Bitcoin covenants," in *Proceedings of Financial Cryptography and Data Security - FC 2016*. Springer, 2016, pp. 126–141.
[25] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *Proceedings of Intelligent Vehicles Symposium, IV 2018*. IEEE, 2018, pp. 108–113.
[26] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *Proceedings of Symposium on Security and Privacy, SP 2014*. IEEE, 2014, pp. 443–458.
[27] S. Wang, Y. Zhu, D. Ma, and R. Feng, "Lattice-based key exchange on small integer solution problem," *Sci. China Inf. Sci.*, vol. 57, no. 11, pp. 1–12, 2014.
[28] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

Hongjian Yin received the M.S. degree from the School of Mathematics and Statistics, Xidian University, Xi'an, China. He is currently a Ph.D. student with the department of School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research interests include information security and cryptography. (Email: honjanyin@163.com)

E Chen received the B.S. degree from the department of School of Mathematics and Physics, University of Science and Technology Beijing, China. She is currently a Ph.D. candidate with the department of School of Computer and Communication Engineering, University of Science and Technology Beijing, China. Her research interests include attribute based system and lattice cryptography. (Email: chene5546@163.com)

Yan Zhu is currently a professor with the School of Computer and Communication Engineering, University of Science and Technology Beijing (USTB), China. He was an Associate Professor in Peking University, China, from 2007 to 2013. He was a visiting scholar with the Arizona State University from 2008 to 2009 and University of Michigan-Dearborn in 2012. His research interests include cryptography, secure computation, and network security. (Email: zhuyan@ustb.edu.cn)

Chengwei Zhao received the Ph.D. in Engineering from Beijing University of Posts and Telecommunications (BUPT) in 2019, and worked there from 2011 to 2020. Now, he is a postdoctoral fellow in the Chinese Academy of Science and Technology for Development (CASTED). His current research interests include scientific and technological innovation and regional collaborative innovation. (Email: zhaocw@bupt.edu.cn)

Rongquan Feng received the Ph.D. in Mathematics from the Institute of Systems Science, Chinese Academy of Sciences in 1994. He is currently a professor in Peking University. He was a post-doctorate fellow in Pohang University of Science and Technology (POSTECH), Korea from October 1995 to August 1997, and a visiting professor there from July 2002 to August 2003. His research interests are in the areas of algebraic combinatorics, cryptology and information security. He has published more than 100 papers on these fields. He is now an administrative committee member of Chinese Association for Cryptologic Research. (Email: fengrq@math.pku.edu.cn)

Stephen S. Yau is currently a professor of computer science and engineering in the School of Computing and Augmented Intelligence at Arizona State University (ASU), USA. He served as the chair of the Department of Computer Science and Engineering at ASU in 1994-2001. Previously, he was on the faculties of Northwestern University, Evanston, Illinois, and the University of Florida, Gainesville. He served as the president of IEEE Computer Society, and was on the board of directors of IEEE and of Computing Research Association. He served as the editor-in-chief of IEEE Computer magazine, organizing committee chair of 1989 World Computer Congress sponsored by IFIP, and the chair of COMPSAC 1977 and its steering committee chair in subsequent years sponsored by IEEE Computer Society. He was the general chair of 2018 IEEE World Congress on Services, and an honorary co-chair of 2017 IEEE Smart World Congress. His current research includes services and cloud computing systems, cybersecurity, software engineering, and internet of things. He received Tsutomu Kanai Award and Richard E. Merwin Award of IEEE Computer Society, and Outstanding Contributions Award of Chinese Computer Federation. He is a Fellow of the AAAS and IEEE. (Email: yau@asu.edu)