Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 9506796, 15 pages
https://doi.org/10.1155/2021/9506796

Research Article

A Novel User Collusion-Resistant Decentralized Multi-Authority Attribute-Based Encryption Scheme Using the Deposit on a Blockchain

Siwan Noh,[1] **Donghyun Kim**,[2] **Zhipeng Cai**,[2] **and Kyung-Hyune Rhee**[3]

[1]*Department of Information Security, Graduate School, Pukyong National University, Busan 48513, Republic of Korea*
[2]*Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA*
[3]*Department of IT Convergence and Application, Pukyong National University, Busan 48513, Republic of Korea*

Correspondence should be addressed to Kyung-Hyune Rhee; khrhee@pknu.ac.kr

Received 23 April 2021; Revised 23 May 2021; Accepted 18 June 2021; Published 7 July 2021

Academic Editor: Weizhi Meng

Recently, the concept of a decentralized data marketplace is getting much attention to exchange user data. Multi-authority attribute-based encryption (ABE), which can provide flexibility and user-centric access control, is previously widely used in decentralized data sharing applications and also becoming a foundation to build decentralized data trading applications. It is known that users in a multi-authority ABE system can collude by sharing their secret information for malicious purposes. To address this issue, the collusion-resistant multi-authority ABE model was introduced in which a unique global identifier (GID) is issued by the central authority (CA) to each user. Unfortunately, such approach cannot be used directly to build a decentralized data marketplace as (a) such intervention of the CA is directly against the main motivation of the decentralized trading platform and, mostly importantly, (b) the CA can exploit its full knowledge on users' GID to launch various attacks against users. Motivated by these observations, this paper introduces a novel user collusion-resistant decentralized multi-authority ABE scheme for privacy preserving data trading systems. In the existing multi-authority ABE systems, users utilize his/her GID that is solely assigned by the CA to generate his/her secret keys throughout the collaboration with authorities and a user can compute multi-authority keys by combining the secret keys (stem from the same GID) in various ways. In the proposed system, the CA only has a partial knowledge of users' GIDs, and thus, users' privacy can be protected. On the other hand, we set the user's own partial GID as a secret which can be used to withdraw his/her deposit to discourage any possible collusion among users.

1. Introduction

The development of the Internet of Things (IoT) has led people to generate larger amounts of data in their daily lives. Experts predict that the amount of these data will explosively increase each year [1]. One major use of the massive amount of data is the training of various machine learning algorithms to build artificial intelligence-empowered applications for our daily lives [2–4]. As acquiring sufficient amounts of data from individuals to train the machine learning algorithms has many limitations, the concept of a data marketplace is introduced; a data marketplace is an online trading platform where people can trade data, and it can be considered suitable for legally acquiring the data required for learning. Google [5] and Amazon [6] provide an online data marketplace service based on their cloud infrastructure. On these platforms, peoples can buy qualified data and analyze it on the cloud computing service. However, there is a significant issue that the most centralized marketplace does not support the user-to-user data trade and all rights to sales data are controlled by the central administrator. A decentralized marketplace platform [7–9] where the concept has recently been proposed can provide decentralized data trading among users without a centralized administrator based on the blockchain smart contract. A decentralized data trading platform achieves decentralization by excluding the participation of

trusted intermediaries. However, the decentralized platform does not support data sharing through trusted administrator among the data seller and multiple buyers. Therefore, considering the problem that sellers must always be online for data sharing, the most efficient way is for the seller to outsource sales data to the cloud server and the server to provide data only to users authorized by the seller. However, traditional one-to-one cryptographic schemes (e.g., symmetric key encryption and public key encryption) are not suitable for decentralized data trading. If a one-to-one scheme is used, the seller will need to transform (or re-encrypt) data stored on the cloud server into new ciphertexts for the buyer every time.

An attribute-based encryption (ABE) is a cryptographic scheme that can provide one-to-many encryptions, which satisfies the utmost requirements of data trading. The seller can specify the buyer's job or position in the process of generating a ciphertext, and buyers can also efficiently find the desired data based on specified attributes in the ciphertext. In the ciphertext-policy attribute-based encryption (CP-ABE) [10–13], the message sender defines the attributes required for the decrypting of the message as an access tree. The ciphertext can be decrypted by any user who has appropriate attributes. To decrypt the message, firstly, the recipient must prove his/her attribute set and gets corresponding secret keys from the key authority. However, in the multi-authority environment [14–17], authorities manage attributes of users belonging to them privately unless there is a predefined communication channel among them and users can belong to more than one authority with multiple secret keys. The message sender can define a set of attributes issued by different authorities as conditions for decrypting data. Therefore, in the multi-authority ABE system, recipients must be able to generate a secret key by combining his/her multiple attributes generated by multiple authorities.

However, if the combination of secret keys is allowed, the system must consider the following two issues: first, authorities do not know a secret key generated by other key authorities. If the key authority receives a request to combine secret keys from the user, the authority must be able to verify that the requested secret keys are all held by the same user. In other words, all authorities must be able to distinguish between the honest user's request and the malicious attacker's request. The second issue is a revocation of the secret key. If the user leaves the system or an attribute is updated, the corresponding user's secret key should no longer be used. If the system does not provide the key revocation, attackers may be able to attempt a collusion attack using a revoked user's secret key.

Chase [14] proposed a global identifier (GID) to distinguish between the honest request and collusion attack. The user uses his/her unique GID issued by the central authority as an input parameter for the key generation algorithm, and only combinations of secret keys generated from the same GID are allowed in the system. Subsequent works [15, 17–20] have improved Chase's approach, where instead of identifiers, the CA chooses a random secret value for each user and generates the secret key based on this random value with each authority. This random value is revealed only when the user has enough secret keys. If the user attempts to combine the secret keys generated from different random values, it is not revealed and the algorithm does not return a valid computational result. However, these approaches assumed a trusted CA that issues a unique identifier for each user and there was a limit to the system being overly dependent on CA. A decentralized attribute-based encryption (DABE) [10, 18–20] was proposed to solve the concentration of secrets in the CA during the key generation process, but there remained a problem of the centralized GID. Even if the power of the CA has been weakened, the user's GID is still determined by CA, so the CA can collude with the user to violate the privacy of other users [21].

2. Related Work

Sahai and Waters [22] first proposed an ABE as an extension concept of identity-based encryption [23]. In ABE cryptosystems, a user is represented by a set of attributes instead of unique identities. Therefore, ABE can provide flexible access control based on the user's attributes. In [22], the ciphertext can be decrypted if the recipient has at least d attributes in the entire set of attributes. To generate the secret key, first, the user proves his/her identity with attributes to the trust authority. After validation of attributes is completed, the key authority uses its master key to generate the requester's secret key corresponding to the set of attributes that the requester has.

Many researchers have recently used ABE to achieve decentralized data sharing. Gao et al. [24] and Zhang et al. [25] proposed a decentralized data sharing system. They propose a method that combines the blockchain and CP-ABE to resolve problems caused by the untrusted cloud service provider in traditional data sharing systems. Instead of storing personal data on the untrusted cloud server, Gao et al. [24] suggested that InterPlanetary File System (IPFS) nodes store a chunk of encrypted data. In his data sharing scheme, only users with appropriate attributes can collect chunks from the decentralized file system and reconstruct the original decrypted data. Zhang et al. [25] used an attribute-based signature (ABS), CP-ABE, and the blockchain smart contract to share IoT sensor data on the untrusted cloud server. In [25], the data owner stores encrypted data on the cloud server and generates a smart contract that manages the access control table of the data. If the IoT device that wants access to data has appropriate attributes, it can submit ABS to the smart contract and receive the encrypted private key. Subsequently, the IoT device submits ABS to the cloud server and obtains encrypted data via a secure channel established through mutual authentication. However, [22, 24, 25] assumed a single attribute key authority in their system, so it failed to present available ABE in the multi-authority model in which several different authorities operate simultaneously. A single-authority ABE model violates our goal of mitigating the dependence of the centralized authority. Therefore, we require consideration of a multi-authority environment in which multiple authorities manage users' attributes individually.

Chase [14] proposed a multi-authority ABE model based on the global identifier. In Chase's ABE model, all users in the system have a unique GID issued by the central authority (CA) and they use it as a secret seed to issue secret keys from each attribute authority. In the process of the key generation, the requester's GID and his/her attributes and a pseudorandom function (PRF) that each key authority has uniquely are used. Upon receiving a request to combine several secret keys from the user, the authority reconstructs this identifier from the requested secret keys. If all secret keys are generated from the same GID, the authority combines the requested secret keys. Since the reconstruction of GID requires information about PRF that each authority uses to generate secret keys, Chase's model necessitates a fully trusted entity that maintains the state of all authorities in the system. However, this entity will have a significant impact on the entire system if it is compromised (known as a single point of failure).

DABE can mitigate the impact of the aforementioned issues by dividing the role of the fully trusted CA in traditional ABE systems into multiple entities in the system [10, 18–20, 26, 27]. Hur and Kang [19] proposed a DABE model that improves the model of Bethencourt et al. [12]. In [19], the CA and a group of attribute authorities A_i cooperatively generate user's secret keys. Each authority generates only a part of the user's secret key in the key generation phase, so, nobody knows the entire user secret key except the user. The CA securely generates a portion of the user's secret key while maintaining privacy for their input data by running a two-party computation protocol with key authorities in the system. The user completes his/her secret keys by combining portions of the secret key issued by the CA and key authorities. Wang et al. [20] proposed the DABE model in which a key authority (KA) and a cloud service provider (CSP) cooperatively issue user's secret keys in the single-authority environment based on Water's model [13]. Wang mitigated the key escrow problem due to the key generation of the single authority by splitting the key generation operations separately between KA and CSP, similar to [19]. Similar to the scheme in [19], Wang's scheme runs a two-party computation protocol between KA and CSP, so only the user knows his/her secret key. Lin et al. [10] proposed a collaborative key management protocol based on Water's model [13]. In [10], the key authority and cloud server issue a user's secret key and a decryption server helps the user's decryption process. In Lin's scheme, the key authority (KA), cloud server (CS), and decryption server (DS) generate secret keys for each user during the key generation process. Unlike previous studies, in Lin's scheme, no one in the system has a user's secret key. Instead of issuing a secret key to the user, encryption and decryption operations based on the user's secret key issued by the CA are divided by KA, CS, and DS. In [27], Rahulamathavan et al. proposed a strong privacy-preserving DABE model with an anonymous key-issuing protocol to prevent key authorities from tracing users' GIDs and violating users' privacy. In Rahulamathavan's scheme, users can get a secret key from the key authority without revealing their GID to key authorities. However, most of the proposed schemes still rely on the GID generated by the CA to prevent collusion attacks among users. The security of the above systems has the disadvantage of relying entirely on a single fully trusted authority. Therefore, rather than simply distributing the running of traditional ABE algorithms across multiple entities, we need a method to efficiently prevent collusion attacks.

Blockchain is a variant of the distributed database. Users in the blockchain network (called a full node) manage a local copy of the blockchain ledger themselves without the database manager. In the blockchain network, a consensus protocol is used to synchronize ledgers even if malicious nodes are participating in the network. The blockchain can be divided into two types depending on the network model [28]. The first type is a public blockchain, where all nodes in the network are untrustworthy nodes. Therefore, the public blockchain uses consensus algorithms such as a Proof-of-Work (PoW), Proof-of-Stake (PoS), and Delegated Proof-of-Work (DPoS), which can provide strong network security. A strong consensus algorithm can protect a blockchain ledger from malicious nodes in the network; however, it leads to tradeoffs between security and performance [29, 30]. The second type, a private blockchain, improves the performance of the blockchain network by constructing nodes in the network only as authorized users. The private blockchain network is managed by a network administrator, and only users authorized by the administrator can participate in the network. Therefore, the private blockchain can improve the network performance based on efficient consensus algorithms such as PBFT [31].

A security deposit means money held as an initial payment of the purchase process. It is used in various fields, such as leasing, and is used as a means of ensuring fairness in the contract. For example, if a lessee damages an apartment or cancels a lease contract during an apartment lease, the lessor will deduct the amount from the lessor's deposit. Poon and Dryja [32] proposed a method to share cryptographic proofs that allow the counterparty to withdraw their deposit whenever the channel is updated. If a malicious user attempts to close the channel abnormally, the counterparty will be able to withdraw the malicious user's deposit without the original owner's consent. McCorry et al. [33] proposed a monitoring solution that improves the problem of channel participants being always online and monitoring the network. Participants in the channel always have to monitor the blockchain network to know if the counterparty closes the channel abnormally. McCorry et al. have delegated these monitoring roles to third parties and proposed a monitoring solution that allows users to validate the results. A Hashed Timelock Contract (HTLC) [34] is a method to support a cross-chain transaction between heterogeneous blockchain networks (e.g., exchange Bitcoin and Ethereum). Unlike traditional cryptocurrency transactions that use the digital signature as a proof of ownership, HTLC uses knowledge of preimage r of hash values $H(r)$ recorded in the blockchain ledger as proof of ownership. A preimage for the proof of ownership is automatically disclosed at the phase of consuming the counterparty's transaction after both sides send the transaction to the counterparty. As such, the security deposit is used as a motivation to induce honest behavior among untrusted users.

2.1. Our Contribution. In this paper, we propose a fair data trading system on the multi-authority ABE model. In the proposed system, we adopt a blockchain-based security deposit to prevent collusion attacks and a data trading protocol based on HTLC to guarantee fairness between the buyer and seller. To summarize, our contributions are listed as follows:

(i) In the proposed model, any user can attempt a collusion attack. However, if anyone tries to attack, he/she will lost his/her security deposit. People will act as honestly as possible to keep their deposits.

(ii) In the proposed system, the trading and sharing of sales data are done without the participation of a trusted administrator. We propose a decentralized data trading protocol that can guarantee reliability between untrusted sellers and buyers.

(iii) The proposed system is controlled only by smart contracts operating on the blockchain. Nobody can control the blockchain network that is controlled only by nodes in the network, and the proposed system is secure unless a fatal attack on the blockchain network is known.

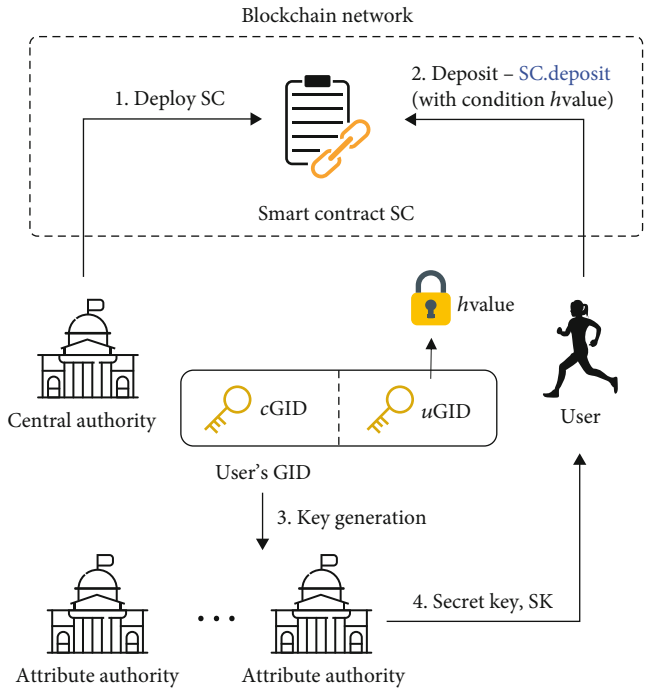


Figure 1: Proposed system overview.

3. System Architecture

In this section, we describe our proposed system architecture with our security considerations.

3.1. System Description. There are five system entities in our system, the central authority (CA), attribute authorities (A_i), the user (buyer and seller), the blockchain network, and the cloud service provider (CSP) as shown in Figure 1.

(i) *Central authority (CA)* generates a secret key to the user in cooperation with A_i that can be used to decrypt the ciphertext in the system. The CA periodically updates all secret keys to revoke a malicious user or attributes. Moreover, the CA deploys a security deposit contract (SC) for each user that motivates users to act honestly in the system

(ii) *Attribute authorities (A_i)* are responsible for verifying the user's possession of attributes and issuing attributes to the user. In our system, a secret key means a secret value corresponding to a set of attributes and the user can obtain multiple attributes and its secret keys from multiple authorities

(iii) *User* is classified as a seller or buyer according to their role in the trading protocol. All users must set up a security deposit as a constraint on their behavior and then participate in the system

(iv) *Blockchain* is a decentralized P2P network without a network administrator. A smart contract is a program that runs on the blockchain network, which is controlled only by network nodes, making control by third parties practically difficult. Therefore, it is practically difficult to control the operation of the program by malicious third parties

(v) *Cloud service provider (CSP)* is a cloud server that stores ciphertexts, which in our system CSP serves to provide ciphertexts to buyers and deploys a trading support contract (TSC) that supports users' data trading. Furthermore, we assume that in the proposed system, CSP is a semitrusted entity and honest but curious

3.2. Security Goals. In this paper, we assume a collusion attack as a major threat to the proposed system. In the CP-ABE model, a user's secret key represents a set of attributes authenticated by the key authority. Even if users share their secret keys with the collusion attacker, there is no damage to colluders unless the system administrator detects and punishes the colluder who shared the secret key. The attacker may be able to get secret keys through a trusted colluder who wants to access the same data. However, realistically, it would be more desirable to assume that a third-party user who has no contact with the attacker has this secret key. Therefore, we assume that a group of collusion attackers in the proposed system does not have enough attributes to decrypt the target data (also, the attacker's group consists of reasonable users and has no trust among them). We also assume that an attacker pays a reward and gets a secret key to a third-party user who has insufficient attributes.

A collusion attack can be classified into several types of attacks according to the role of participants in the system [21], and we consider that the following two types of attacks are a threat to the proposed system.

(i) *Among users*: an attacker's secret key (i.e., a set of attributes) cannot decrypt the target data that he wants to access. The attacker can obtain insufficient attribute keys from colluders to achieve the condition to decrypt the target data

(ii) *Authority and user*: the CA does not issue secret keys to users. Instead, it can be the target of many attacks as an administrator of the system. A compromised CA can generate a secret key for any user without attribute authorities based on the information that can be obtained from the secret key of the user in the system

Under the threat model noted above, we consider the following security goals for a fair data trading system against collusion attacks on the multi-authority ABE model:

(i) *Collusion resistance*: although collusion attacks can occur among entities in the system, resistance to collusion attacks should be guaranteed

(ii) *User revocation*: the secret key of a user who has left the system or been revoked by the CA must no longer be valid on the system. Furthermore, the revoked secret key should not be used in the system even though it is still valid

(iii) *Decentralization*: all data trading transactions occurring on the proposed system are made without the participation of the trusted intermediary. It is a trade between untrusted users, but no user should be able to harm the other party by malicious behavior

3.3. Cryptocurrency Deposit. The proposed model uses cryptocurrency as a deposit. However, cryptocurrency may not be appropriate to use as a deposit due to its unique floating exchange rate. For example, on January 31, 2020, Ethereum was priced at $183.68 per dollar. However, a year later, the price increased sevenfold to $1,317.58 on January 31, 2021. If the value of the cryptocurrency decreases, the deposit will not prevent collusion attacks, and if the value increases, it will be a factor that makes it difficult for new users to participate in the system. Therefore, cryptocurrency-based deposits can have a significant impact on the reliability of the system. To prevent issues caused by the volatility of cryptocurrencies, we assume the use of Tether (USDT) instead of ordinary cryptocurrencies such as Bitcoin (BTC) and Ethereum (ETH) as deposits used in the system. As of February 22, 2021, Tether's volatility over the past 30 days was 0.01, while Bitcoin was 0.88 and Ethereum was 0.94 [35]. Tether tokens exist as digital tokens built on Bitcoin (Omni and Liquid Protocol), Ethereum, EOS, Tron, Algorand, SLP, and OMG blockchains. Tether is implemented in the Ethereum blockchain as an ERC-20 token and supports smart contracts. Furthermore, the low variability of the tether is suitable for use in the proposed system.

4. Decentralized CP-ABE

A common framework of the CP-ABE scheme consists of the following four algorithms:

(i) *Setup*: the authority runs a setup algorithm to generate public parameters for the system and then generates its master key (MK) and public key (PK)

(ii) *Key generation*: The authority validates attributes that the user has and then issues the user's attribute key using its master key

(iii) *Encryption*: the user (message sender) defines the access policy to decrypt the ciphertext. Then, the user uses the access policy and authority's public key to generate the ciphertext

(iv) *Decryption*: the user (recipient) uses his/her attribute key and the authority's public keys to decrypt the ciphertext. If the user has enough attributes defined in the access policy of the ciphertext, he can acquire the plaintext, but otherwise, he will fail to decrypt it.

Hur and Kang [19] proposed a decentralized attribute-based encryption model that improves the CP-ABE model proposed by Bethencourt et al. [12]. He improved the key generation algorithm in the Bethencourt model to collaborate with the central authority and attribute authorities. This model is as follows.

4.1. Setup

(i) *Global setup*: the trusted initializer chooses a bilinear group $\mathbb{G}_0$ of prime order p with generator g and a cryptographic hash function $H : \{0,1\}^* \longrightarrow \mathbb{G}_0$

(ii) *Authority setup*: a central key authority (CA) chooses a random exponent $\beta \in \mathbb{Z}_{\mathbb{p}}$ as its master key and computes its public key g^{β}. CA's master private and public key pair is given by $(MK_{\mathrm{CA}} = \beta, PK_{\mathrm{CA}} = g^{\beta})$. Each local key authority (A_i) chooses a random exponent $\alpha_i \in \mathbb{Z}_{\mathbb{p}}$ as its master private key and computes its public key $e(g,g)^{\alpha_i}$ (where e is a bilinear map and is denoted by $e : \mathbb{G}_0 \times \mathbb{G}_0 \longrightarrow \mathbb{G}_1$). Key authorities' master private and public key pairs are given by $(MK_{A_i} = e(g,g)^{\alpha_i}, PK_{A_i} = \alpha_i)$. Then, it publishes a public parameter $\mathrm{param} = \{\mathbb{G}_0, g, H\}$

4.2. Key Generation. In Bethencourt's model, the CA generates all parts of the user's secret key by itself. However, in Hur and Kang's model, the CA and key authorities generate users' secret keys cooperatively shown in Figure 2. Therefore, none of them can acquire the entire part of the user's secret key. The key generation protocol is as follows:

(1) The user u requests the CA, to generate a secret key

(2) The CA chooses random exponents $\gamma_i \in \mathbb{Z}_p^*$ for each key authority A_i and sets $\mathrm{GID}_{\mathrm{u}} = \sum_{i=1}^{m} \gamma_i$. Then, the CA and each key authority A_i run a secure two-party computation (2PC), where the private input of the CA is (γ_i, β), authority A_i's private input is α_i, and the protocol returns the private output $x = (\alpha_i + \gamma_i)\beta$ to A_i

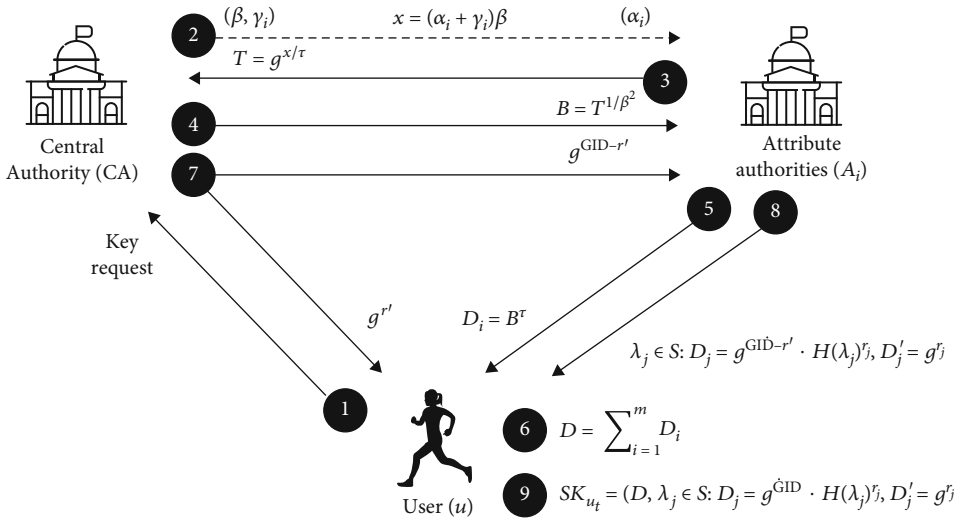


Figure 2: Key generation phase in the Hur and Kang model [19].

(3) A_i randomly chooses $\tau \in \mathbb{Z}_p^*$ and computes $T = g^{x\tau} = g^{(\alpha_i+\gamma_i)\beta\tau}$ and then sends it to the CA

(4) CA computes $B = T^{1/\beta^2} = g^{\alpha_i+\gamma_i\tau\beta}$ and sends it to A_i

(5) A_i computes $D_i = B^\tau = g^{(\alpha_i+\gamma_i)\beta}$ and sends it to the user u

(6) After receiving D_i from all key authorities, the user u computes the part of his/her secret key $D = \prod_{i=1}^{m} D_i = g^{(\alpha_1+\cdots+\alpha_m)+\mathrm{GID}_u\beta}$

(7) The CA randomly chooses $r' \in \mathbb{Z}_{\mathbb{p}}$ and then sends $g^{\mathrm{GID}_u - r'}$ to A_i and $g^{r'}$ to u.

(8) A_i randomly chooses $r_j \in \mathbb{Z}_{\mathbb{p}}$ and sends the following secret value to the user u (value r_j related to the set of attributes λ_j issued to the user u by authority A_i):

$$\forall \lambda_j \in S : D_j = g^{\mathrm{GID}_u - r'} \cdot H(\lambda_j)^{r_j}, \quad D'_j = g^{r_j} \tag{1}$$

(9) The user u computes $g^{r'} \cdot D_j$ for all attribute keys that he has. The secret key that the user u obtains is as follows: (where $D = g^{(\alpha_1+\cdots+\alpha_m)+\mathrm{GID}_u\beta}$)

$$\mathrm{SK}_u = \left(D, \forall \lambda_j \in S : D_j = g^{\mathrm{GID}_u} \cdot H(\lambda_j)^{r_j}, D'_j = g^{r_j}\right) \tag{2}$$

In the Hur and Kang model, a decentralized key generation is possible because each authority generates only a part of the user's secret key. Moreover, each user's secret key uses a different secret value GID_u randomly chosen by the CA, which prevents the collusion attack among users. Without knowledge of this secret value GID_u, collusion attack among users is impossible.

However, if the CA is compromised, it is possible to generate a secret key for another user by colluding with the user without the participation of key authorities [21]. The user u gives the CA the value D that is part of his/her secret key $\mathrm{S K}_u$. The CA can compute the value g^α required to generate another user's secret key from the value D received from the user u as follows:

$$\frac{D^\beta}{g^{\mathrm{GID}_u}} = \left(g^{(\alpha_1+\cdots+\alpha_m)+\mathrm{GID}_u\beta}\right)^\beta \times g^{-\mathrm{GID}_u} = g^{(\alpha_1+\cdots+\alpha_m)+\mathrm{GID}_u+(-\mathrm{GID}_u)} = g^{(\alpha_1+\cdots+\alpha_m)}. \tag{3}$$

A simple solution for this vulnerability is to set the value GID_u, which the CA randomly determines for each user, to secret information that the CA does not know. However, if users choose the value GID_u themselves, it leads to a problem which is a collusion attack among users. And if the user gives a value GID_u as well as a value D to the CA, the solution no longer guarantees resistance to collusion attacks.

5. Proposed Scheme

In this section, we present a decentralized ABE model that improves the Hur and Kang model [19]. To solve the aforementioned problem, we propose a security deposit protocol to avoid collusion attacks. We use a security deposit on the blockchain as a precaution against malicious behavior by users in the attribute-based encryption system. In the proposed model, the user sets up a security deposit to the blockchain smart contract as a guarantee of his/her honest behavior to participate in the system. If the user colludes with another user, during the exchange of information for the attack, he exposes the secret value that is required to withdraw his/her security deposit from the smart contract. Table 1 shows notations used in our scheme.

Table 1: Notations and descriptions.

Notation	Description
$c\mathrm{GID}_{u,t}$	CA-generated global identifier for user u at period t
$u\mathrm{GID}_{u,t}$	User-generated global identifier at period t
$\mathscr{D}_u$	Deposit from user u
$k_{u,t}$	Random value for proof of ownership of the deposit $\mathscr{D}_u$
H	Cryptographic hash function

5.1. Deposit Setup. All users must set up their security deposit to participate in the system. The security deposit is managed by a blockchain smart contract, and if a user shares his/her secret key to another user or authority for the collusion attack, the colluder who receives the secret key can withdraw the security deposit set by the original owner of the shared secret key. We allow users to choose a part of the global identifier $u\mathrm{GID}_u$, which was previously determined uniquely by the CA for each user. And knowledge of this $u\mathrm{GID}_u$ is used as a condition for withdrawing the user's deposit. The user requests the CA to generate his/her secret key and then runs a deposit function shown in Figure 3.

CA deploys a deposit smart contract SC to the blockchain network for user u who requests key generation. At this time, the state of the smart contract is initialized (init). Then, the user u chooses random exponents $\delta_i, k_{u,t_1} \in \mathbb{Z}_p^*$ for each key authority A_i and for proof of ownership (where $u\mathrm{GID}_{u,t_1} = \sum_{i=1}^{m} \delta_i$). The user u computes and sends hash results $h\mathrm{value} = H(u\mathrm{GID}_{u,t_1})$ and $h\mathrm{index} = H(k_{u,t_1})$ to the smart contract SC at the period t_1 (where H is a cryptographic hash function). After a simple verification process, the state of SC transits from the init to the active.

Then, the user u, CA, and each key authority A_i run a key generation algorithm as shown in Figure 4.

(1) The user u requests the CA, to generate a secret key

(2) The CA chooses random exponents $\gamma_i \in \mathbb{Z}_p^*$ for each key authority A_i and sets $c\mathrm{GID}_{u,t_1} = \sum_{i=1}^{m} \gamma_i$

(3) The user u securely sends δ_i for each key authority A_i

(4) The CA and each key authority A_i run a secure two-party computation (2PC), where the private input of the CA is (γ_i, β), authority A_i's private input is $(\alpha_i + \delta_i)$, and the protocol returns the private output $x = (\alpha_i + \gamma_i + \delta_i)\beta$ to each A_i

The subsequent key generation process is the same as Hur and Kang's model [19], and the encryption and decryption algorithms are the same. After completing the key generation algorithm, the user u gets a his/her secret key (where $D = \prod_{i=1}^{m} D_i = g^{(\alpha_1 + \cdots + \alpha_m) + c\mathrm{GID}_{u,t_1} + u\mathrm{GID}_{u,t_1}\beta}$):

$$\mathrm{SK}_u = \left(D, \forall \lambda_j \in S : D_j = g^{\left(c\mathrm{GID}_{u,t_1} + u\mathrm{GID}_{u,t_1}\right)} \cdot H\left(\lambda_j\right)^{r_j}, D_j' = g^{r_j}\right). \tag{4}$$

5.2. Get Ciphertext. After completing the deposit setup, users can participate in the trade with other users. In the proposed system, the seller stores his/her encrypted data on the cloud server, and after the transaction is completed, the cloud service provider provides the data to the buyer. However, since transactions do not go through trusted intermediaries, the trading between users on the external channel cannot guarantee the fairness of the trading. Therefore, we applied HTLC to the proposed system to ensure the fairness of the trading as shown in Figure 5. In the system, data trading and data sharing between the buyer and the seller are as follows:

(1) The seller stores encrypted data CT on the cloud server

(2) The buyer requests the seller to sell the data CT

(3) The seller randomly chooses $R \in \mathbb{Z}_p^*$ and computes $h\mathrm{proof} = H(R)$ and then sends $h$proof to the CSP and the buyer. In the proposed system, the CSP manages lists for managing each user's data. The list consists of $\{\mathrm{ID}_{\mathrm{seller}}, CT, \mathrm{txID}, h\mathrm{proof}\}$ and the CSP provides corresponding data only if the buyer has provided valid proof (i.e., preimage of $h$proof)

(4) The buyer deposits the transaction amount to the trading support contract (TSC) with the $h$proof

(5) The seller submits his/her digital signature σ_{seller} and the preimage of $h$proof (i.e., R) to TSC and receives the transaction amount

(6) To request access to the data CT, the buyer submits the value R exposed by the seller in the process of receiving the transaction amount to the CSP. The buyer generates the following message and sends it to the CSP to show that the state of his/her smart contract is active and that the trade with the seller has been completed

$$\mathrm{msg}_{\mathrm{req}} = \left\{\mathrm{SC}, \mathrm{txID}, R', k_{u,t_1}, CT\right\} \tag{5}$$

(7) If SC.state = active, SC.hindex $= H(k_{u,t_1})$, and $h$proof $= H(R')$, the CSP returns the requested ciphertext CT to the buyer

5.3. State Transition. The activated SC can be transited to three states: active, close, and revoke. We have five state transition scenarios as shown in Figure 6. A detailed description of each state transition is as follows:

(i) [active ⟶ active]: for the user revocation, all users in the system must periodically update their secret keys. The update period t is determined by the CA, which is inserted into the user's contract SC during the deposit setup process. Therefore, all contracts in the system have the same timer of update period t.

Contract SC

Init: CA deploy the contract SC and set the initial state and the next key update period t_{expired} and the user address addr_u.
1) set state = inti
2) set $t_{\text{expired}} = t + 1$
3) set user = addr_u

Deposit: Upon receiving $\{\sigma_{sk_u}, h\text{value}_{u,t}, h\text{index}_{u,t}, \text{deposit}_u\}$ from the user u (where $h\text{value}_{u,t} = H(u\text{GID}_{u,t})$, $h\text{index}_{u,t} = H(k_{u,t})$).

1) If verifySign(σ_{sk_u}, user) = true
   - set $h$value = $h\text{value}_{u,t}$
   - set $h$index = $h\text{index}_{u,t}$
   - set balance = deposit_u
2) set state = active

Withdraw: Upon receiving { proof_1, proof_2, deposit$u$, addr_x} from the user x
1) If state = active && balance ≥ deposit_u
   - If $h$value = $H(\text{proof}_1)$
     - set state = close
     - send deposit_u to addr_x
2) If state = revoke && balance ≥ deposit_u
   - If $h$value = $H(\text{proof}_1)$ || $h$index = $H(\text{proof}_2)$
     - set state = close
     - send deposit_u to addr_x

Key Update: Upon receiving $\{\sigma_{sk_u}, h\text{value}_{u,t+1}, h\text{index}_{u+t+1}, g^r\}$ from the user x, $\{\sigma_{CA}, g^{r'}\}$ from CA
1) If t_{expired} > current block height
2) If verifysign(σ_{sk_u}, addr_u) ∧ verifysign(σ_{CA}, addr_{CA}) ∧ $(g^r = g^{r'})$ = true
   - set $h$value = $h\text{value}_{u,t+1}$
   - set $h$index = $h\text{index}_{u,t+1}$
   - set $t_{\text{expired}} = t + 2$

Contract TSC

Init: CSP deploy the trade support contract TSC. The seller stores the ciphertext CT in CSP. CSP manages access control lists for the seller. The list consisted of the seller's ID $\text{ID}_{\text{seller}}$, the identifier of the stored data ID_{CT}, an index of the transaction txID, and the proof needed to access the data $h$proof.
Deposit: Upon receiving {$h$proof, $\text{ID}_{\text{seller}}$, ID_{CT}, $\text{deposit}_{\text{buyer}}$} from the buyer
1) create new trade
   - trade[txID] = Trade ($h$proof, ID_{CT}, $\text{deposit}_{\text{buyer}}$)
2) send txID to the buyer and seller

Withdraw: Upon receiving {proof, $\text{deposit}_{\text{buyer}}$, add$r_{\text{seller}}$, txID} from the seller.
1) If trade[txID].$h$proof = H(proof) && trade[txID].balaance ≥ $\text{deposit}_{\text{buyer}}$
   - trade[txID].balance = trade[txID].balance – $\text{deposit}_{\text{buyer}}$
   - send $\text{deposit}_{\text{buyer}}$ to addr_{seller}
2) If trade[txID].balance ≤ 0
   - delete trade[txID]

Figure 3: Contract design for SC and TSC.

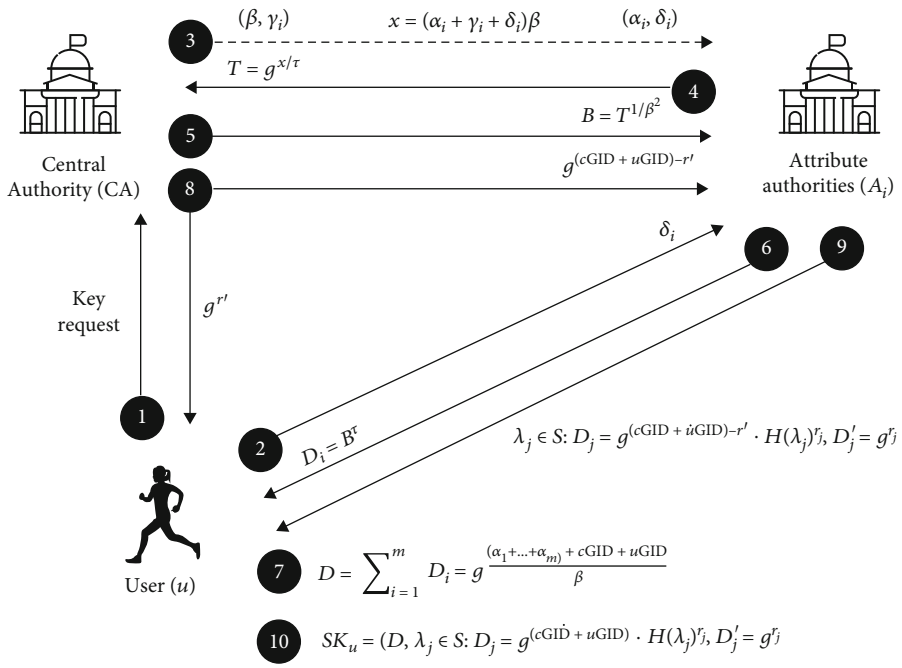
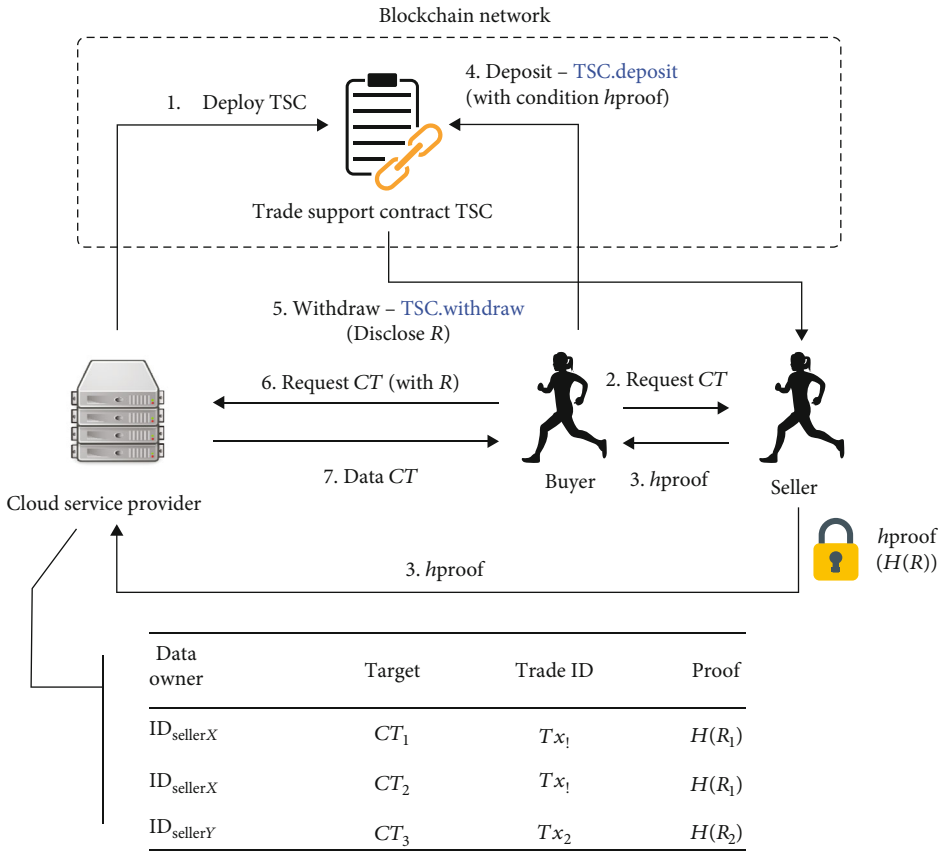


Figure 4: Key generation phase in the proposed system.



Data owner	Target	Trade ID	Proof
$ID_{sellerX}$	CT_1	Tx_1	$H(R_1)$
$ID_{sellerX}$	CT_2	Tx_1	$H(R_1)$
$ID_{sellerY}$	CT_3	Tx_2	$H(R_2)$

Figure 5: Data transaction overview.

Since implementing the synchronized timer in an asynchronous network is hard, we measure this period t as the nth block is appended to the blockchain (e.g., update every 100 blocks appended to the blockchain). If the user takes no action until the nth block is appended, his/her contract will automatically be transited to the revoke state. In the revoke state contract, the user loses ownership of the deposit locked in the contract, so to avoid this situation, all users will participate in the key update function honestly before the contract timer is expired as shown in Figure 6 (case 1). To update the secret key, the user chooses a new random value $\delta_i, k_{u,t_2} \in \mathbb{Z}_p^*$ for each attribute authority and the new proof of ownership. Then, set $u\text{GID}_{u,t_2} = \sum_{i=1}^{m} \delta_i$ and compute $h\text{value} = H(u\text{GID}_{u,t_2}), h\text{index} = H(k_{u,t_2})$. The user uses $u\text{GID}_{u,t_2}$ to generate a new secret key as in the deposit setup phase. At the end of the key generation process in Hur and Kang's model, the CA generates $g^{r'}$ and sends it to the user. The user finally gets the secret key using $g^{r'}$ received from the CA and the part of the secret key D_j received from the attribute authorities. The user and CA send this value $g^{r'}$ to the contract after key generation is completed, and if the values submitted by the CA and the user are the same, the contract is transited to the active state of the new period t_2

(ii) [active ⟶ close]: a deposit in the contract can be withdrawn by showing a preimage of $h$value (i.e., $u\text{GID}_{u,t}$) using the withdraw function shown in Figure 3. The user can withdraw their deposit before the contract timer t is expired himself (case 2). To withdraw the deposit, the user u submits the secret value $u\text{GID}_{u,t}$ used to generate his/her secret key to the contract. When the deposit is withdrawn, the contract transits to a close state automatically. The withdrawal algorithm of the proposed model does not verify the original owner of the deposit but simply verifies that it has valid proof as shown in Figure 3. Therefore, anyone with valid proof can withdraw the deposit without the original owner's consent. As mentioned earlier, in the proposed model, the user who wants to join the collusion attack must disclose his/her secret value $u\text{GID}_{u,t}$ to the colluder. That is, if the secret value $u\text{GID}_{u,t}$ is disclosed to another user and the deposit is withdrawn, the contract is equally transited to the close state (case 3)

(iii) [active ⟶ revoke]: a transit to the revoke state occurs due to the user's revocation. As mentioned earlier, we use a revocation scheme that periodically regenerates a new secret key for all valid users in the system. However, if a user does not complete a key update (case 5) or is revoked from the system before the key update (case 4), the contract will be transited

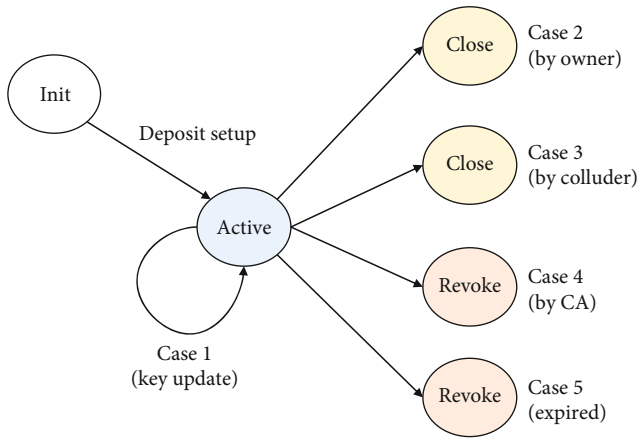


Figure 6: A state transition diagram of the proposed system.

to the revoke state. After the state of the contract is transited to revoke, *h*index is added as an option to the conditions for deposit withdrawal. In other words, the deposit of the revoked user can be withdrawn with knowledge of the preimage of either h value or *h*index as shown in Figure 3

6. Analysis and Evaluation

6.1. Security Analysis

(1) *Collusion resistance*: our proposed system ensures resistance to collusion (a) *among users* and (b) *among a user and central authority*. All users in the system can share their secret keys with collusion attackers. An attacker will pay a certain reward $\text{price}_{\text{reward}} \times n$ to induce users to share their secret keys (where n is the number of colluders). They will be able to obtain unauthorized data through attacks and benefit pric e_{data} from it. In the end, the benefit of the collusion attacker from the attack would be

$$\text{price}_{\text{attack}} = \text{price}_{\text{data}} - (\text{price}_{\text{reward}} \times n) \tag{6}$$

However, if the reward to colluders $\text{price}_{\text{reward}} \times n$ is greater than the benefits from the attack (i.e., $\text{price}_{\text{data}} < (\text{price}_{\text{reward}} \times n)$), the attackers will lose the motivation of the attack, because they will suffer financial damage even if the attack succeeds.

(a) *Collusion among users*: a collusion attack among users can be prevented by making it impossible to combine if the global identifier GID_u used for key generation is not the same. In the existing scheme, the CA chooses random exponents for each attribute authority A_i and sets $\text{GID}_u = \sum_{i=1}^{m} \gamma_i$ (where GID_u is uniquely determined by each user). Then, each attribute authority A_i and CA cooperatively generate the user's secret key based on their master key and given γ_i. Even though the attribute authority generates a secret key corresponding to the same attribute set, each user has a different secret key because the exponent γ_i is different. Therefore, GID_u used to generate a user's secret key can never be reconstructed from the secret keys of different users. Previous models used a CA-determined global identifier GID_u for secret key generation. Therefore, the CA knew the parameters that each attribute authority used to generate a user's secret key and could access the ciphertexts in the system without having enough attributes. In contrast, in our proposed system, the CA and user choose and combine the global identifier to be used for key generation separately. In more details, the CA generates a global identifier $c\text{GID}_{u,t} = \sum_{i=1}^{m} \gamma_i$ the same for previously proposed models and the user u also generates a global identifier $u\text{GI D}_{u,t} = \sum_{i=1}^{m} \delta_i$ at period t. The user u keeps $u\text{GID}_{u,t}$ secret from the CA and sends δ_i to each attribute authority A_i. Each A_i performs a two-party computation with the CA using its master key and δ_i received from the user as a private input. Finally, the user u receives $D_i = g^{(\alpha_i+\gamma_i+\delta_i)\beta}$ from each attribute authority and computes a part of his/her secret key $D = \prod_{i=1}^{m} D_i = g^{(\alpha_1+\cdots+\alpha_m)+c\text{GID}_{u,t}+u\text{GID}_{u,t}\beta}$ from it. In the proposed model, $\text{GID}_{u,t} = c\text{GID}_{u,t} + u\text{GID}_{u,t} = \sum_{i=1}^{m}(\gamma_i + \delta_i)$ is computed from the random exponents chosen by the CA (γ_i) and the user (δ_i). In other words, the CA only knows a part of the user's global identifier and the random parameters used by each attribute authority to generate secret keys. To achieve the collusion among users requires that attackers and colluders should know their global identifier used to generate their secret keys. If attackers know the global identifier used to generate secret keys, they can combine secret keys generated from the same global identifier. An attacker can combine colluders' secret keys generated from the same GID or create a new secret key using the same GID of the colluder to attempt the collusion attack. In other words, the attacker should be able to recover the colluder's global identifier. However, $c\text{GID}_u$ chosen by the CA is not disclosed to the user on the key generation process, and in contrast, the colluder may disclose his/her $u\text{GID}_u$ to the attacker but this will be shared with a trustworthy user, the ownership of his/her deposit until the next key update period $t + 1$. That is, the colluder has the risk of losing his/her deposit by the collusion attacker until the end of the period t, whether or not the collusion attack is successful. The colluder will only join the attack if the attacker pays at least more than the deposit he may lose (i.e., $\text{price}_{\text{reward}} \geq \mathscr{D}_u$). Theoretically, the attacker will not be able to obtain the GID for the attack, neither from the CA nor from colluders, so the proposed system can provide resistance to the collusion attack among users

(b) *Compromised authority*: Hur's model had a vulnerability that a compromised CA could generate a secret key for the unauthenticated user from the secret key

of the colluder. In this attack scenario, the CA receives the part of the secret key $D = g^{(\alpha_1+\cdots+\alpha_m)+\mathrm{GID}_u\beta}$ from colluder u and computes secret information of the attribute authorities $g^{\alpha} = g^{(\alpha_1,\cdots,\alpha_m)}$ that required to generate a new secret key (where α_i is a master secret key of the attribute authority A_i). The CA can generate a new secret key alone without the cooperation of attribute authorities based on g^{α}. However, in our proposed system, the CA needs the global identifier of the colluder $\mathrm{GID}_u = c\mathrm{GID}_u + u\mathrm{GID}_u$ to extract g^{α} from his/her secret key. In other words, the CA must receive uGID$_u$ along with the secret key from the colluder. However, $u\mathrm{GID}_u$ is secret information needed to withdraw the deposit from the colluder's smart contract. If the colluder u shares his/her global identifier $u\mathrm{GID}_u$ with the CA, the CA may withdraw the colluder's deposit by submitting the preimage of $h$value, $u\mathrm{GID}_u$, to the smart contract

(2) *User revocation*: for situations where a user leaves the system (by themselves or by force), the system must provide a revocation of the user's secret key. If the system does not provide the key revocation, our deposit protocol will not guarantee resistance to collusion attacks. Our proposed deposit protocol relies on activated security deposits to prevent users from malicious behavior. All users of the system will always act rationally because they have the risk of losing their deposit due to their misbehavior. This causes an attacker to lose motivation for the attack by making it more costly for the attack. However, assuming the key revocation is not provided, the user's deposit will be withdrawn if the user leaves the system, while the secret key will remain valid in the system. In other words, the system cannot prevent a user who has already left the system from using his/her secret key in the collusion attack. A user revocation prevents the secret key of the user who has left the system from being used in the system. In our proposed system, we apply a user revocation to the system, which periodically regenerates and reencrypts secret keys and ciphertexts of all users in the system so that revoked user's secret keys can no longer be used. However, even if the user is revoked, his/her secret key is still valid until the next key update period. This can be solved by shortening the key update period, but it results in a high computational cost on the system. We adopt a method to prevent revoked users from using their secret keys until the next key update period based on the security deposit. In the proposed system, a user revocation can be divided into two cases:

(i) *Revoked by the CA*: the CA can revoke a user from the system and prevent that user from participating in the system. the CA can transit the state of the user's contract SC from active to revoke. However, even if the contract is transited to the revoke state, the revoked user still retains ownership of the deposit locked in the contract and his/her secret key can also be used until the next key update period. However, revoked users may ignore the revoked state and participate in data trading. The revoked user submits his/her knowledge of the preimage of SC.$h$index to show ownership of the deposit while depositing the transaction amount in TSC for the data trading. In the active state, withdrawals of a security deposit require submission of SC.$h$value's preimage (i.e., uGID at period t), but in the revoke state, the deposit withdrawal can be made by submission of SC.$h$index's preimage. TSC approves deposits of the transaction amount only when the buyer's SC.state is active. In other words, the revoked user is no longer able to proceed with the trading process. Furthermore, his/her secret information k (preimage of SC.$h$index) is exposed to the blockchain network, so his/her security deposit could be withdrawn by a third-party user in the system

(ii) *Expired*: all users must periodically update their secret keys. If the user did not update the secret key at period t, his/her SC is automatically transited to the revoke state as the next period $t + 1$ begins. The revoked user can withdraw his/her deposit and participate in the system again with a new deposit

(3) *Decentralization*: in the proposed system, data trading among users is made through a smart contract TSC. In each trade, the seller provides data to the buyer and the buyer pays the seller its price. However, on the decentralized trading platform where trusted intermediaries do not participate, fairness issues arise in the order of the trading protocol. Therefore, we adopt a method to ensure fairness of the decentralized trading based on HTLC. When the buyer requests the seller to sell the data CT, the seller chooses a random value R, then computes $h\mathrm{proof} = H(R)$, and sends it to the buyer and CSP. In our proposed system, the CSP manages a special list for each seller. This list is an access control list of the seller's data. The CSP provides requested data only when the buyer submits a valid proof (i.e., preimage of h proof $= H(R)$). The buyer sets the seller's digital signature σ_{seller} and the preimage of $h$proof as a withdrawal condition while depositing the transaction amount to TSC. The seller submits randomly selected R to TSC to withdraw the price, in which R is disclosed to all participants in the blockchain network. The buyer can get the purchased data by submitting the disclosed R to the CSP. In the proposed system, the random value R was used as a method to ensure the fairness of the trade transaction. In the trading protocol, two situations can be considered:

Table 2: Computational cost.

Phase	Entity	Operation	Cost
Phase 1 (deposit setup)	CA	*Deploy SC*	$1tx$
	User	*Randomly choose* $\delta_i, k \in \mathbb{Z}_p^*$	$(i+1)$ *RNG*
		Compute $uGID = \sum \delta_i$,	$1sum$
		$H(uGID), H(k)$	$2hash$
		Send Tx to SC (deposit)	$1tx$
Phase 2 (get ciphertext)	Seller	*Randomly choose* $R \in \mathbb{Z}_p^*$	1 *RNG*
		Compute $H(R)$	$1hash$
		Generate signature σ_{seller}	$1sig$
	Buyer	*Send Tx to TSC (deposit)*	$1tx$
	CSP	*Compute* $H(k), H\left(R'\right)$	$2hash$
Phase 3 (state transition: Update)	CA	—	—
	User	*Randomly choose* $\delta_{i,new}, k_{new} \in \mathbb{Z}_p^*$	$(i+1)$ *RNG*
		Compute $uGID_{new} = \sum \delta_{i,new}$,	$1sum$
		$H(uGID_{new}), H(k_{new})$	$2hash$
		Send Tx to SC (update)	$1tx$

Table 3: Computational cost notation.

Operation	Meaning
tx	Create blockchain transaction and broadcast it to the network
sig	Generate a digital signature (ECDSA)
hash	Cryptographic hash function (Keccak-256)
sum	Summation operation
RNG	Random number generate function

(i) *Malicious seller*: may not share the purchased data after receiving the price from the buyer

(ii) *Malicious buyer*: may not pay the price after receiving purchased data from the seller

In the proposed system, the seller and buyer set specific conditions *h*proof for data sharing and trade transactions, respectively. For more details, the seller sets specific conditions for data sharing, and then, the buyer also sets a copy of conditions (i.e., same conditions *h*proof = $H(R)$ set by seller) for withdrawing price in TSC. A buyer can get purchase data by submitting valid proof to the CSP, and a seller can also submit valid proof to TSC to withdraw the price from the contract. When the seller submits the random value R as proof to withdraw the price, it is recorded in the blockchain and automatically disclosed to all network nodes. The seller cannot withdraw the price from TSC without disclosing R. The buyer also cannot find out the R randomly chosen by the seller and submit it to the CSP without the transaction being completed. The buyer also cannot find out the random value R chosen by the seller and submit it to the CSP without the seller completing the withdrawal of the price.

6.2. Evaluation and Implementation. Our approach is to mitigate the dependence of fully trusted CA in the system by issuing only a portion of GID by the CA and the user, instead of entirely issuing GID by the fully trusted CA. In the key generation phase, the user randomly selects a secret value u GID to be used as a portion of his GID and submits the deposit to the smart contract. Deposits locked in the smart contract can be withdrawn by submitting this secret value u GID to the contract. In summary, compared to previous researches, our approach can mitigate the dependence of the CA but the cost of user participation in the protocol is inevitable. Table 2 shows the computational cost of the system entities at each phase of the proposed protocol. Table 3 lists the notations to express the computational cost.

Table 4: Costs required to execute the proposed contract SC and TSC.

Phase	Command	Cost (gas)	Cost ($)
Phase 1	Deploy contract SC	399816	63.84
	Deposit (SC.deposit)	69419	11.08
Phase 2	Deploy contract TSC	335323	53.54
	Deposit (TSC.deposit)	103479	16.52
	Withdraw (TSC.withdraw)	45322	7.23
Phase 3	Key update (SC.keyupdate)	65940	10.52
	Withdraw (SC.withdraw)	43661	6.97

To evaluate the computational cost, we consider only the computational cost of each entity except for the operations performed by the smart contract. We also consider only the operations that each entity performs to communicate with smart contracts as evaluation targets.

In phase 1, the CA deploys a smart contract SC for each user. The deployment of smart contracts requires 1tx operation, and the CA keeps the addresses of all smart contracts deployed (each address is 20 bytes in size). The user submits a security deposit to SC for CP-ABE key generation. The user

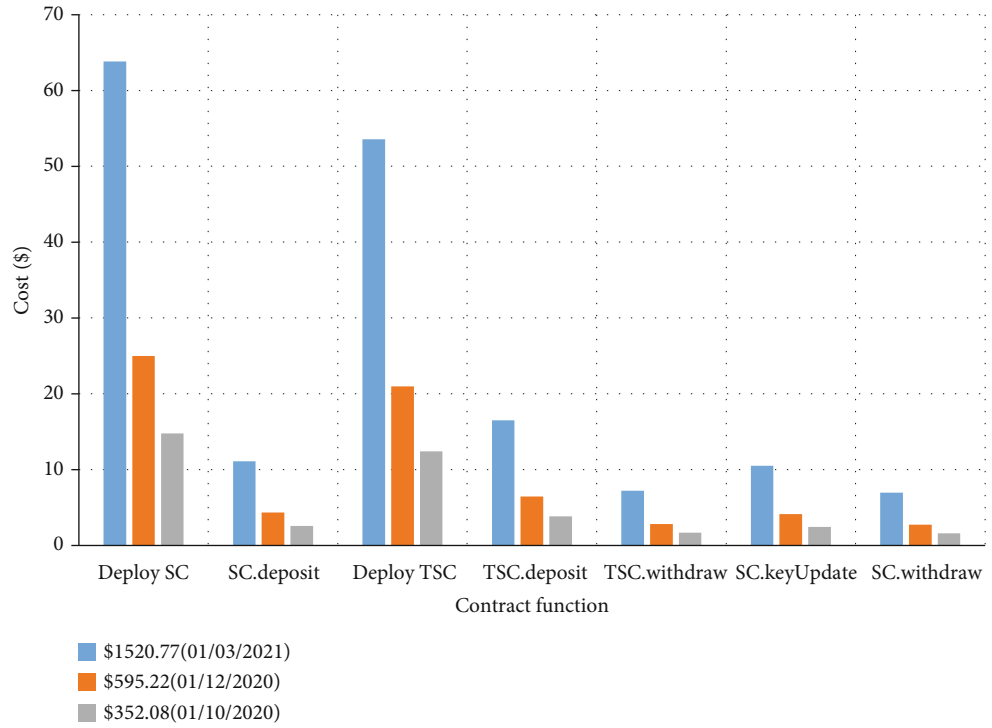


Figure 7: Cost of contract functions according to the exchange rate in the proposed system.

performs an i RNG operation that chooses the random number for i key authorities in the system to generate uGID, 1 RNG operation that generates a random number for the proof of ownership of the contract SC and 1tx operation that submits a deposit after 1sum operation for uGID generation.

In phase 2, the data seller performs 1 RNG operation that chooses a random R for data trading, 1 hash operation to generate an HTLC proof, and a digital signature to generate operation 1 sig for deposit acquisition. The buyer performs 1tx operation that submits a deposit, and the CSP performs 1hash operation for the seller's data table update and 1hash operation for the buyer's data request verification.

In phase 3, the user updates his smart contract SC by performing the same operations, as in phase 1, for new key generation.

In our protocol, users' deposits are controlled by HTLC based on simple hash operations instead of complex operations. Except for the $(i+1)$RNG operation required in the key generation (and key update) process, other operations can be seen efficiently in practical terms with low-cost operations. However, since this can be resolved by properly adjusting the key update cycle, the computational cost of the user is reasonable even if the proposed protocol is applied.

We implemented the smart contract SC and TSC on the *Kovan* Ethereum test network. The price of the gas that we set in the test is 1.6×10^{-7} ether (160 Gwei) and the cost required to execute the contract is calculated as the gas price × gas used. Unfortunately, however, it is not appropriate to show the execution cost of the contract at this point due to the soaring price of the cryptocurrency in 2021. Therefore, we present the costs with the current and previous exchange rates together for a more appropriate evaluation.

Table 4 shows the results of converting the costs required to execute our contract into gas and US dollars (exchange rate as of 01/03/2021: 1 ether = $1520.77) at each phase in the proposed system. Figure 7 shows a recalculation of the execution cost of each function of the contract shown in Table 3 according to the previous exchange rate (exchange rate as of 01/10/2020: 1 ether = $595.22 and 01/12/2020: 1 ether = $352.08). In the proposed system, the honest user pays for the data trading (SC.deposit, SC.withdraw) and a periodic key update (SC.keyUpdate), and the cost of each function, excluding contract deployment costs, is reasonable because it is less than $5 with the previous exchange rate.

7. Conclusions

In this paper, we propose a data trading protocol for decentralized user-to-user data trading and a security deposit protocol for preventing collusion attacks in the multi-authority ABE system. Our system is controlled by two smart contracts SC and TSC, and instead of centralized methods by a trusted third party, our system leads to honest behavior of users based on their security deposit. However, the management history of the user's deposit is transparent to everyone on the network. If a malicious third party can relate a user's identity in the ABE system to his/her deposit in the blockchain network, there is a possibility of a new attack resulting from this vulnerability. Applying many privacy-preserving techniques used in the blockchain, we think that it would be possible to further improve the proposed protocol in terms of security. Moreover, we expect the proposed system to be utilized as a way to ensure that the system operates

honestly without a central system administrator in the modern complex society.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program (2020-0-01596) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and partially supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2020-0-01797) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world from edge to core[white paper]," 2018, http://cloudcode.me/media/1014/idc.pdf.

[2] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 577–590, 2016.

[3] Z. Xiong, H. Xu, W. Li, and Z. Cai, "Multi-source adversarial sample attack on autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2822–2835, 2021.

[4] K. Li, G. Luo, Y. Yang, W. Li, S. Ji, and Z. Cai, "Adversarial privacy-preserving graph embedding against inference attack," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6904–6915, 2020.

[5] "What is Google cloud marketplace?," https://cloud.google.com/marketplace/docs.

[6] "AWS marketplace," https://aws.amazon.com/marketplace.

[7] B. Goertzel, S. Giacomelli, D. Hanson, C. Pennachin, and M. Argentieri, "SingularityNET: a decentralized, open market and inter-network for AIs[white paper]," 2017, http://airesearch.com/ai-research-papers/singularitynet-a-decentralized-open-market-and-inter-network-for-ais/.

[8] O. Protocol, "A decentralized data exchange protocol to unlock data for AI[white paper]," 2020, https://oceanprotocol.com/.

[9] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: managing data purchasing and data placement in a geo-distributed data market," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 893–905, 2018.

[10] G. Lin, H. Hong, and Z. Sun, "A collaborative key management protocol in ciphertext policy attribute-based encryption for cloud data sharing," *IEEE Access*, vol. 5, pp. 9464–9475, 2017.

[11] C. Li, J. He, L. Cheng, C. Guo, and K. Zhou, "Achieving privacy-preserving CP-ABE access control with multi-cloud," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 801–808, Melbourne, VIC, Australia, 2018.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy*, pp. 321–334, Berkeley, CA, USA, 2007.

[13] B. Waters, "Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization," in *Public Key Cryptography – PKC 2011. PKC 2011*, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds., vol. 6571 of Lecture Notes in Computer Science, pp. 53–70, Springer, Berlin, Heidelberg, 2011.

[14] M. Chase, "Multi-authority attribute based encryption," *Proceedings of the Theory of cryptography conference*, pp. 515–534, 2007.

[15] K. Yang and X. Jia, "Attributed-based access control for multi-authority systems in cloud storage," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pp. 536–545, Macau, China, 2012.

[16] A. Gao and Z. Li, "Free global ID against collusion attack on multi-authority attribute-based encryption," *Security and Communication Networks*, vol. 6, no. 9, p. 1152, 2013.

[17] K. Riad, "Multi-authority trust access control for cloud storage," in *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 429–433, Beijing, China, 2016.

[18] J. Hur, D. Koo, S. O. Hwang, and K. Kang, "Removing escrow from ciphertext policy attribute-based encryption," *Computers & Mathematics with Applications*, vol. 65, no. 9, pp. 1310–1317, 2013.

[19] J. Hur and K. Kang, "Secure data retrieval for decentralized disruption-tolerant military networks," *Proceedings of the IEEE/ACM transactions on networking*, vol. 22, no. 1, pp. 16–26, 2012.

[20] S. Wang, K. Liang, J. K. Liu, J. Chen, J. Yu, and W. Xie, "Attribute-based data sharing scheme revisited in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1661–1673, 2016.

[21] E. Meamari, H. Guo, C.-C. Shen, and J. Hur, "Collusion attacks on decentralized attributed-based encryption: analyses and a solution," 2020, https://arxiv.org/abs/2002.07811.

[22] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005*, R. Cramer, Ed., vol. 3494 of Lecture Notes in Computer Science, pp. 457–473, Springer, Berlin, Heidelberg, 2005.

[23] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology. CRYPTO 1984*, G. R. Blakley and D. Chaum, Eds., vol. 196 of Lecture Notes in Computer Science, pp. 47–53, Springer, Berlin, Heidelberg, 1984.

[24] H. Gao, Z. Ma, S. Luo, Y. Xu, and Z. Wu, "BSSPD: a blockchain-based security sharing scheme for personal data with fine- grained access control," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6658920, 20 pages, 2021.

[25] Y. Zhang, D. He, and K.-K. R. Choo, "BaDS: blockchain-based architecture for data sharing with ABS and CP-ABE in IoT," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 2783658, 9 pages, 2018.

[26] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Advances in Cryptology – EUROCRYPT 2011. EUROCRYPT 2011*, K. G. Paterson, Ed., vol. 6632 of Lecture Notes in Computer Science, pp. 568–588, Springer, Berlin, Heidelberg, 2011.

[27] Y. Rahulamathavan, S. Veluru, J. Han, F. Li, M. Rajarajan, and L. Rongxing, "User collusion avoidance scheme for privacy-preserving decentralized key-policy attribute-based encryption," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2939–2946, 2015.

[28] K. Wüst and A. Gervais, "Do you need a blockchain?," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 45–54, Zug, Switzerland, 2018.

[29] K. Croman, C. Decker, I. Eyal et al., "On scaling decentralized blockchains," in *Financial Cryptography and Data Security. FC 2016*, J. Clark, S. Meiklejohn, P. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds., vol. 9604 of Lecture Notes in Computer Science, pp. 106–125, Springer, Berlin, Heidelberg, 2016.

[30] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, pp. 1–10, Trento, Italy, 2013.

[31] M. Castro and B. Liskov, "Practical byzantine fault tolerance," *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, vol. 99, no. 1999, pp. 173–186, 1999.

[32] J. Poon and T. Dryja, "The bitcoin lightning network: scalable off-chain instant payments[White paper]," 2016, https://lightning.network/lightning-network-paper.pdf.

[33] P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn, "Pisa: arbitration outsourcing for state channels," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pp. 16–30, Zurich, Switzerland, 2019.

[34] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pp. 245–254, Egham, United Kingdom, 2018.

[35] "Tether price index - CoinDesk 20," https://www.coindesk.com/price/tether.