



**A SECURE, TRACEABLE, AND EFFICIENTLY  
REVOCABLE CLOUD-BASED ACCESS CONTROL  
SCHEME USING CIPHERTEXT POLICY ATTRIBUTE-  
BASED ENCRYPTION AND BLOCKCHAIN**

**BY**

**KHANADECH WORAPALUK**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF MASTER OF  
SCIENCE (ENGINEERING AND TECHNOLOGY)  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY  
THAMMASAT UNIVERSITY  
ACADEMIC YEAR 2023**

THAMMASAT UNIVERSITY  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

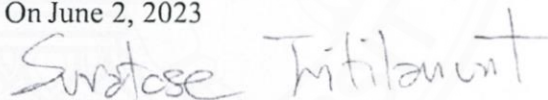
KHANADECH WORAPALUK

ENTITLED

A SECURE, TRACEABLE, AND EFFICIENTLY REVOCABLE CLOUD-BASED  
ACCESS CONTROL SCHEME USING CIPHERTEXT POLICY ATTRIBUTE-  
BASED ENCRYPTION AND BLOCKCHAIN

was approved as partial fulfillment of the requirements for  
the degree of Master of Science (Engineering and Technology)

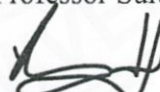
On June 2, 2023



Chairperson

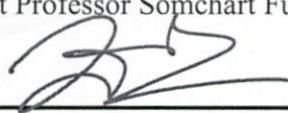
(Associate Professor Suratose Tritilanunt, Ph.D.)

Member and Advisor



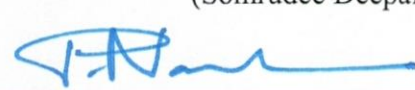
(Assistant Professor Somchart Fugkeaw, Ph.D.)

Member



(Somrudee Deepaisarn, Ph.D.)

Director



(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	A SECURE, TRACEABLE, AND EFFICIENTLY REVOCABLE CLOUD-BASED ACCESS CONTROL SCHEME USING CIPHERTEXT POLICY ATTRIBUTE-BASED ENCRYPTION AND BLOCKCHAIN
Author	Khanadech Worapaluk
Degree	Master of Science (Engineering and Technology)
Faculty/University	Sirindhorn International Institute of Technology/ Thammasat University
Thesis Advisor	Assistant Professor Somchart Fugkeaw, Ph.D.
Academic Years	2023

## ABSTRACT

Ciphertext Policy Attributed-based Encryption (CP-ABE) is considered as a suitable solution for supporting secure and fine-grained access control for outsourced data. Considering the revocation problem in cloud-based access control, existing revocable CP-ABE based schemes still have limitations. First, most solutions did not support both user and attribute revocation with auditability of revocation transactions. Second, they still relied on data owners or data users to generate the crypto components such as ciphertext update key to support the revocation process, resulting in the dependency of the availability of the data owner and computation overhead in both the data owner and the data user. Third, they did not provide the formal procedure for invoking the affected ciphertexts. Finally, most of them did not tackle the attributes hiding to support privacy-preserving policy outsourcing. To this end, we proposed a cloud-based access control scheme by leveraging CP-ABE, AES symmetric encryption, and blockchain technology to deliver an efficient user and attribute revocation with the ciphertext retrieval mechanism and transaction traceability. In addition, we introduced the attribute hiding method based on hidden vector encryption (HVE) to preserve the privacy of access policy content. To evaluate the efficiency of our proposed scheme, we conducted experiments to show that our proposed scheme is efficient and practical for real implementation.

**Keywords:** Access Control, CP-ABE, Blockchain, Attributes Hiding, Symmetric-Key  
Cryptography



## ACKNOWLEDGEMENTS

I want to express my deepest gratitude to everyone who provided me with invaluable support and assistance throughout the completion of this thesis. With their contributions, this graduate degree is possible.

First of all, words cannot express my gratitude to my thesis advisor, Dr.Somchart Fugkeaw, who provided invaluable support and technical guidance throughout my graduate studies. His encouragement and inspiration kept me on track in my research, even when I lost myself when facing many problems during the research. Furthermore, he has always been willing and enthusiastic to assist me in every way possible. He was the first person who taught me the formal methods of scientific research and the principles of academic paper writing. I am highly grateful for the opportunity he gave me to choose a research topic that can open a path for my future career, precisely blockchain-based access control with CP-ABE and cyber security awareness. I appreciate his insightful suggestions and technical guidance, which have significantly contributed to my research output, including this thesis and research papers. With his help, this thesis and my research paper were possible.

Likewise, I would like to extend my sincere thanks to my thesis committee members, Dr.Suratose Tritilanunt and Dr.Somrudee Deepaisarn, for giving valuable suggestions on the research which were crucial for the improvement of this thesis.

Besides, I would like to extend my special thanks to my senior, Mr.Pattavee Sanchol, who helped me set up the experimentation environment, helped me debug the benchmark, and suggested the experimentation.

Additionally, I would like to thank Sirindhorn International Institute of Technology for providing me with the *Faculty Quota Scholarship* for my master's degree. The scholarship was an enormous contribution to my post-graduate study.

Lastly, I would be remiss not to mention my family and friends. Their support has been a valuable source of keeping my spirits and motivation high during my study.

Khanadech Worapaluk

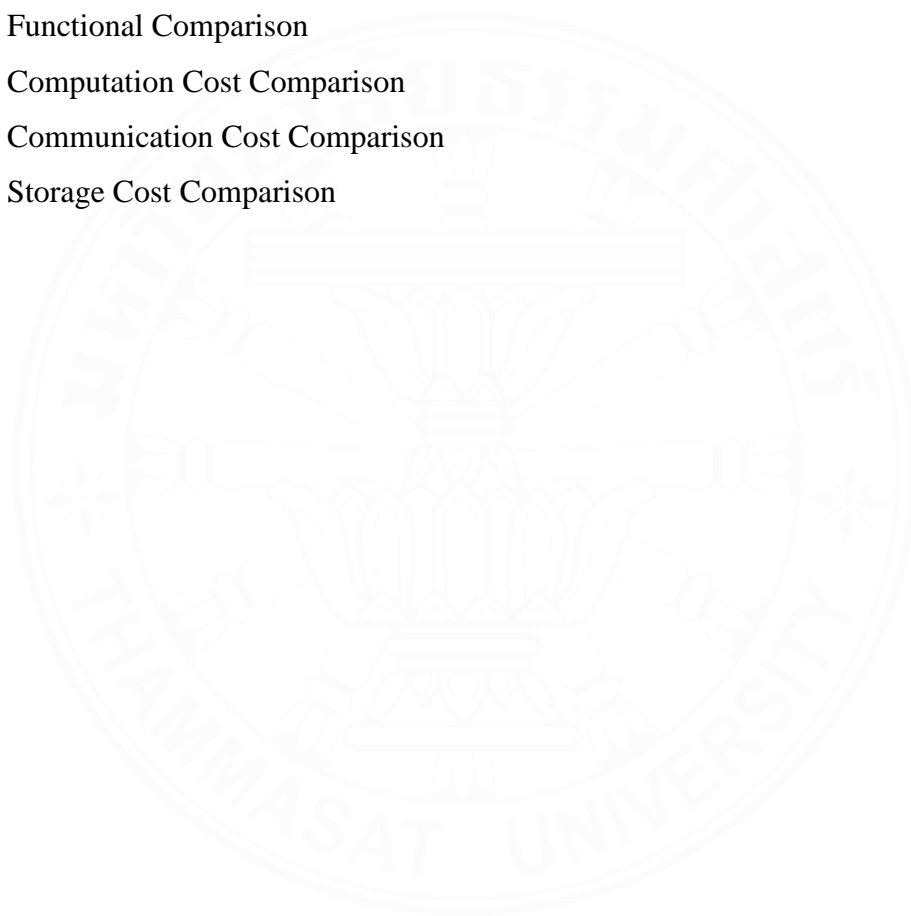
## TABLE OF CONTENTS

	Page
ABSTRACT	(1)
ACKNOWLEDGEMENTS	(3)
LIST OF TABLES	(6)
LIST OF FIGURES	(7)
LIST OF SYMBOLS/ABBREVIATIONS	(8)
CHAPTER 1 INTRODUCTION	1
1.1 Background Information	1
1.2 Problem statement	3
1.3 Our contributions	3
1.4 Organization of this thesis	4
CHAPTER 2 LITERATURE REVIEW	5
2.1 Attribute-based Access Control (ABAC)	5
2.2 Identity-based Encryption (IBE)	5
2.3 Attribute-based Encryption (ABE)	6
2.4 Non-Blockchain Cloud-based Access Control	7
2.5 Cloud-based Access Control with Blockchain Integration	8
CHAPTER 3 PRELIMINARIES	17
3.1 Blockchain	17
3.2 Smart Contract	19
3.3 Bilinear Mapping	19
3.4 Advance Encryption Standard (AES)	20

	(5)
3.5 Rivest-Shamir-Adleman Cryptosystem (RSA)	23
3.6 Hidden Vector Encryption (HVE)	24
<b>CHAPTER 4 OUR PROPOSED SCHEME</b>	<b>27</b>
4.1 System Overview	27
4.2 Ciphertext-Attribute-User Ethereum Account Mapping	29
4.3 Attributes Hiding	29
4.4 Smart Contract Design	31
4.5 Cryptographic Constructs	36
<b>CHAPTER 5 SECURITY ANALYSIS</b>	<b>47</b>
5.1 Security Model of our proposed scheme	47
5.2 Security Proof of our proposed scheme	49
5.3 Forward Security	53
5.4 Backward Security	53
5.5 Confidentiality of Ciphertexts on Cloud and Blockchain Storage	53
5.6 Proxy's Key Security	53
<b>CHAPTER 6 COMPARATIVE ANALYSIS AND EVALUATION</b>	<b>54</b>
6.1 Functionality Analysis	54
6.2 Computation Cost Analysis	55
6.3 Communication Cost Analysis	56
6.4 Storage Cost Analysis	59
6.5 Experimental Analysis	60
<b>CHAPTER 7 CONCLUSION AND FUTURE WORK</b>	<b>70</b>
<b>REFERENCES</b>	<b>71</b>
<b>BIOGRAPHY</b>	<b>76</b>

## LIST OF TABLES

Tables	Page
3.1 Attributes Vector Table Example	26
4.1 Attributes Vector Table Example	30
4.2 Notation used in our model	36
6.1 Notation for comparative analysis section	54
6.2 Functional Comparison	55
6.3 Computation Cost Comparison	55
6.4 Communication Cost Comparison	57
6.5 Storage Cost Comparison	59





## LIST OF FIGURES

Figures	Page
2.1 Taxonomy of A Cloud-based Access Control with Blockchain Integration	9
2.2 A permission-less blockchain in cloud	10
2.3 A permission-based blockchain in cloud	14
3.1 Blockchain Block Structure	18
4.1 System Model	28
4.2 Ciphertext-Attributes-User Ethereum Account Mapping Model	29
4.3 Attribute-Tree with Attribute Hiding	31
4.4 User Revocation Process Diagram	41
4.5 Attribute Revocation Process Diagram	44
6.1 Encryption Performance	62
6.2 Decryption Performance	62
6.3 User Revocation Performance based on number of attributes in policy	64
6.4 Attribute Revocation Performance based on number of attributes in policy	64
6.5 User Revocation Performance based on number of ciphertexts	66
6.6 Attribute Revocation Performance based on number of ciphertexts	66
6.7 Query Performance Per Attribute	68
6.8 Revocation Performance with Query Time	68

## LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
AA	Attribute-Authority
ABE	Attribute-based Encryption
ABAC	Attribute-based Access Control
AES	Advanced Encryption Standard
CP-ABE	Ciphertext-Policy Attribute-base Encryption
$CT_M$	Data Ciphertext
$CT_K$	Key Ciphertext
$CT_{SK,DU}$	DU's Key Ciphertext
DO	Data Owner
DU	Data User
HVE	Hidden Vector Encryption
IBE	Identity-based Encryption
IPFS	Interplanetary File System
KP-ABE	Key-Policy Attribute-base Encryption
MSK	Master Secret Key
PRE	Proxy-Re-Encryption
PRX	Proxy
PK	Public-Key
RSA	Rivest-Shamir-Adleman Public-Key Cryptosystem
$RSA_{DU, PubK}$	Data User RSA public key
$RSA_{DU, PrivK}$	Data User RSA private key
SHA	Secure Hash Algorithm
$SK_{DU}$	User's Secret Key
$SK_{Proxy}$	Proxy's Secret Key

# CHAPTER 1

## INTRODUCTION

This chapter provides background information of cloud computing, cloud-based access control, problem statements, and our contributions to the thesis.

### 1.1 Background Information

Cloud computing has shifted the traditional information technology operation from requiring essential resources available on-premises to the service-based platform where the computing resources are provided by the cloud service providers. Cloud service offers resilient and unlimited computational resources with zero operation and maintenance costs. One of the key services cloud providers offer is cloud storage or storage as a service where enterprises or data owners can outsource and share their data with multiple users. Although the cloud storage service renders flexibility and accessibility for data sharing, security and privacy are of paramount concern. To this end, CSPs usually provide a basic access control system and auditing function for their subscribers (Z. Ying, L. Wei, Q. Li, X. Liu, and J. Cui, 2018). Nevertheless, the providers are still regarded as “*honest but curious*,” and the degree of user control of their data is limited. Consequently, additional security mechanisms such as encryption, data integrity checking, and authorization policy enforcement are essential, especially when the outsourced data is sensitive.

In addition to the strong authentication generally provided by the cloud, ABAC access control (V. Hu et al., 2014) and cryptographic-based access control that integrates fine-grained authorization enforcement and encryption are desirable to support security and privacy for outsourced data. Implementing both an access control environment and encryption requires double operation and administrative costs, such as the expense of access policy management and key management.

To date, CP-ABE has been regarded as an effective cryptographic-based access control approach for data outsourcing environments since it encompasses both authorization and encryption features. CP-ABE offers one-to-many encryption,

allowing the data owner to encrypt the data as a single ciphertext for multiple users. In CP-ABE, the data owner is able to define the access structure or policy constructed from a set of attributes through the logical gates *AND*, *OR*, *M of N* to encrypt the data. The data users with the secret key with attributes that satisfy the policy can decrypt the ciphertext. Here, it offers secure and fine-grained data access control. Nevertheless, user revocation and attribute revocation are non-trivial in CP-ABE-based access control. There are subsequent costs, including ciphertext re-encryption, key re-generation, and key re-distribution, that occur from both revocation levels.

Existing CP-ABE-based access control schemes that support revocation generally focus on the design and development of the revocation mechanism either by the ciphertext update (R. Guo, G. Yang, H. Shi, Y. Zhang, and D. Zheng, 2021; Y. Jiang, X. Xu, and F. Xiao, 2022) or the ciphertext re-encryption (X. Liu, Y. Zheng, and X. Li, 2021; X. Wang et al., 2021; S. Fugkeaw & S. Sato, 2017; S. Maiti & S. Misra, 2020; D. Sethia, A. Shakya, R. Aggarwal, and S. Bhayana, 2019; X. Wang, Y. Chi, and Y. Zhang, 2020) with the proposed user key update methods. For the ciphertext update, the data owner and/or the data user needs to generate a ciphertext update key or a ciphertext transformation key and send it to the cloud to update the affected ciphertexts. This deals with the processing cost of bilinear pairing and the communication cost for transferring crypto objects. For ciphertext re-encryption, the proxy is employed to support ciphertext re-encryption if there is a revocation case. In this method, the proxy is typically given a decryption key and encryption components to use in the re-encryption process. This deals with the handling of the secure delegation of decryption key and encryption components to the proxy.

Recently, blockchain (M. Xu, X. Chen, and G. Kou, 2019) has been integrated into the existing access control systems (R. Kumar, B. Palanisamy, and S. Sural, 2021; X. Liang, N. An, D. Li, Q. Zhang, and R. Wang, 2022; S. Fugkeaw, 2022) to achieve the additional requirements mentioned above. It structurally stores data in a series of blocks where multiple blocks are chained together based on the hashing method. Therefore, auditability, integrity preservation, transparency, and accessibility are key benefits offered by the blockchain. Blockchain-as-a-Service (W. Zheng et al., 2019)

has become the alternative service that many cloud service providers provide to their customers. Empowering cloud-based access control with robust decentralized authentication, immutable records of access transactions, and the assisted revocation function is also promising.

### **1.2 Problem statement**

Existing revocable cloud-based access control solutions employ CP-ABE as their core cryptographic construct. However, there are four major problems that have not been resolved in an integrated manner.

1. Inability to support both user and attribute revocation with full traceability of revocation transaction.
2. The dependency on the data owner and/or the data user to generate cryptographic components to support the revocation process.
3. Lack of attributes hiding mechanism while the access policy needs to be used in the cloud environment.
4. Lack of formal search or invocation of affected ciphertexts that need to be updated or re-encrypted when the revocation occurs.

### **1.3 Our contributions**

The contributions of our proposed scheme are summarized as follows.

1. Our proposed scheme provides the first attempt providing efficient attribute and user revocation with efficient key update mechanism in the blockchain-cloud based access control setting.
2. We devised the policy hiding method based on hidden vector machine that enables privacy-preserving policy enforcement with no additional computation overhead compared to traditional CP-ABE scheme.
3. Our proposed novel ciphertext-attribute-user Ethereum account mapping technique is practical for optimizing the ciphertext re-encryption cost when there is a revocation case in a large-scale data sharing.

4. We performed security analysis to substantiate that our proposed scheme is secure under the general security model as well as the proposed revocation technique supports both backward and forward security.
5. We conducted the experiments in real cloud and Ethereum blockchain environment where there are a high number of ciphertexts and access requests.

#### **1.4 Organization of this thesis**

The organization of this thesis is structured as follows. Chapter 2 discusses the related literature. Chapter 3 presents the preliminaries. Chapter 4 shows the components and construction of our proposed scheme. Chapter 5 explains the security analysis of our proposed scheme. Chapter 6 presents the comparative analysis of our proposed scheme and related works. Finally, the concluding remarks regarding our proposed scheme and possible future work are given in Chapter 7.

## CHAPTER 2

### LITERATURE REVIEW

This chapter discusses the related literature in the area of cloud-based access control. This chapter includes the basic definition of Attribute-based Access Control (ABAC), Identity-based Encryption (IBE), Attribute-based Encryption (ABE), and the related literature related to our proposed system.

#### 2.1 Attribute-based Access Control (ABAC)

Attribute-based access control (S. Rouhani, R. Belchior, R. Cruz, and R. Deter, 2021; V. Hu et al., 2014), or ABAC, is based on the characteristics of the users, resources, and policies defined for each data. The policies contain conditions, typically “*And*” and “*Or*” operators, alongside the attributes that must be satisfied. Users who have a set of attributes satisfying the access rule can access the resource. Otherwise, access is denied. ABAC is considered a fine-grained access control because policy enforcement is based on the user’s attributes and can be enforced at the individual user level.

#### 2.2 Identity-based Encryption (IBE)

Shamir (1985) originally proposed identity-based encryption. This scheme is based on asymmetric key encryption. This scheme uses the public identity of the user to generate the key pair. For example, an email address, social security number, home address, and network address can be used to generate the private-public key pair. A Private Key Generator (PKG) is required to generate the Identity-based key pair. PKG is assumed to be a fully trusted third-party entity. PKG first generates a *Master Private Key (MPK)*, which contains the public parameters and the corresponding users’ identities. When users request to generate the key pair, they usually need to submit their identity to the PKG, and they are registered to the system. PKG then takes the identity from the user and *MPK* to generate the key pair for the user. IBE is primarily adopted in IoT access control applications because the computation cost of IBE’s encryption and decryption algorithm is considered lightweight and efficient.

However, IBE has limitations. If the PKG gets corrupted, all the user identities and the messages encrypted by the key generated by the PKG will be exposed (D. Anand, V. Khemchandani, and R. Sharma, 2013). In addition, the encrypted data can only be decrypted by the users with a key containing the qualified set of identities. The expressiveness of IBE is limited as only the set of identity attributes can be used.

### **2.3 Attribute-based Encryption (ABE)**

Attribute-based encryption, or ABE, is a public key cryptographic primitive where the encryption and decryption process deals with a set of attributes and an access policy. The ABE offers fine-grained and expressive access control through the cryptographic protocol binding with user attributes and access policy enforcement. It is considered a one-to-many encryption since the encryptor can encrypt the message and share the single ciphertext with multiple users. There are two major types of ABE: Key-Policy Attribute-based Encryption (KP-ABE) (V. Goyal, O. Pandey, A. Sahai, and B. Waters, 2006) and Ciphertext-Policy Attribute-based Encryption (CP-ABE) (J. Bethencourt, A. Sahai, and B. Waters, 2007).

#### **2.3.1 Key Policy Attribute-based Encryption (KP-ABE)**

In KP-ABE, data is encrypted with sets of attributes, and secret keys are associated with access structures that specify which ciphertexts a user can decrypt. In the KP-ABE approach, data owners have no control over data because the trust authority will be the one who assigns the access policies to the user's secret key. The data owners can specify as many attributes as possible for the ciphertext but cannot enforce the authorization policy on the users and the ciphertexts.

#### **2.3.2 Ciphertext Policy Attribute-based Encryption (CP-ABE)**

In CP-ABE, the user's secret key contains the attributes used to identify each individual in the system, and the ciphertext contains the access policies. With the CP-ABE approach, data owners have control over their data because they can specify who can access the data of their choice based on the access policy used for encryption. The



CP-ABE access policy can be expressed by “*And*,” “*Or*,” and “*M-of-N*” operations in combination with a set of attributes laid on the leaf nodes of the access policy tree. Therefore, CP-ABE allows the data owners to enforce the authorization policy through logical rules. As defined by A. Sahai, J. Bettencourt, and B. Waters (2007), the CP-ABE consists of four major steps: Setup, Key Generation, Encryption, and Decryption.

**Setup** ( $\lambda$ )  $\rightarrow$  ( $PK, MK$ ). The setup algorithm takes the security parameter  $\lambda$  as the only input. The algorithm then generates a public keys  $PK$  and master key  $MK$  as an output.

**Key Generation** ( $MK, S$ )  $\rightarrow$  ( $SK_{DU}$ ). The algorithm takes the master key  $MK$  and a set of attributes  $S$  to define the user in the system as an input. Its outputs as a user secret key  $SK_{DU}$ . The algorithm uses a bilinear mapping between  $MK$  and  $S$  to get  $SK_{DU}$ .

**Encryption** ( $PK, M, T$ )  $\rightarrow$  ( $CT$ ). The encryption algorithm inputs message  $M$ , public key  $PK$ , and access policy  $T$ . And generate a ciphertext  $CT$  that contains the access policy as an output.

**Decryption** ( $CT, SK_{DU}$ )  $\rightarrow$   $M$  or  $\perp$ . The algorithm takes a ciphertext  $CT$ , and a user secret key  $SK_{DU}$  as input. The algorithm then recursively checks whether the attribute set  $S$  in the user secret key  $SK_{DU}$  satisfied the access policy  $T$  in the ciphertext. If satisfied, the algorithm will return message  $M$  as an output. Otherwise, return  $\perp$ .

## 2.4 Non-Blockchain Cloud-based Access Control

This section reviews the cloud-based access control literature that does not use the blockchain in its schema.

S. Fugkeaw and S. Sato (2017) proposed a scalable CP-ABE protocol with an attribute revocation functionality. In their proposed scheme, the proxy is responsible

for re-encrypting ciphertexts stored in the cloud server when the attribute revocation request occurs. This scheme proposed two-layer encryption consisting of CP-ABE encryption and symmetric key encryption. For the encryption process, the message is first encrypted by the CP-ABE method, and the intermediate ciphertext is encrypted with symmetric encryption. If any attribute is revoked, the proxy must perform both the symmetric and the CP-ABE decryption. Then, the proxy will re-encrypt the affected ciphertexts with a new policy.

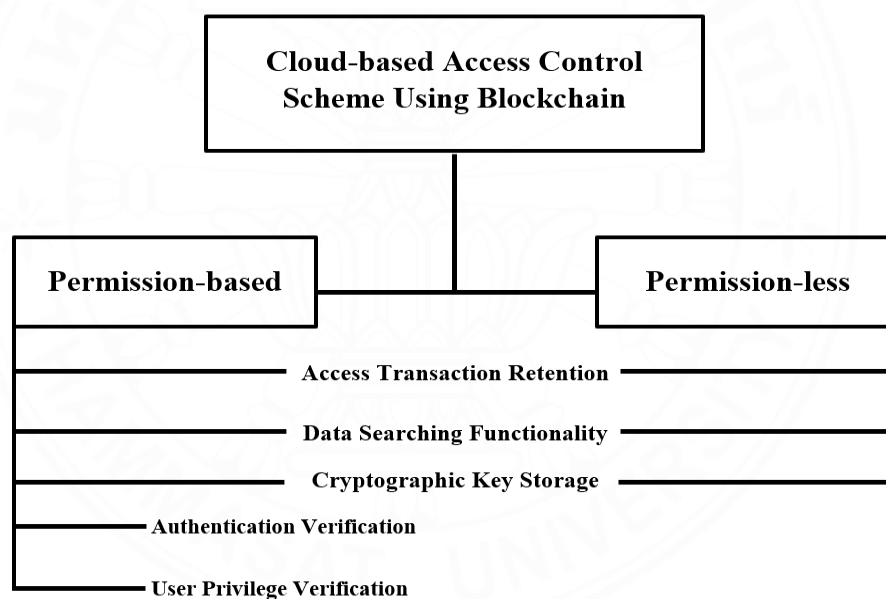
D. Sethia, A. Shakya, R. Aggarwal and S. Bhayana (2019) proposed a constant-size CP-ABE protocol with scalable revocation for resource-constrained IoT devices. In their work, the proxy is employed to perform partial decryption. The proxy holds a revocation list (*RL*) used to generate a decryption component for the data user. To decrypt the data, the data user must submit her secret key to the proxy server and let the proxy generate the complete decryption component using its secret key and *RL*. The correct decryption component will not be created if the data user is in *RL*. Later on, X. Wang, Y. Chi and Y. Zhang (2020) applied a similar approach by focusing on re-encrypting the *RL*. Only a part of the policy will be updated and re-encrypted when the revocation occurs.

S. Maiti and S. Misra (2020) proposed a privacy-preserving Identity-based proxy re-encryption scheme with user revocation. In their scheme, the data owner encrypts data with identity-based encryption. Then, the owner generates a re-encryption key containing all users' identities in the system, and the key will be sent to the proxy. The proxy re-encrypts the ciphertexts with the re-encryption key. If any user is revoked, the process of re-encryption key generation and re-encryption is done by the proxy.

## **2.5 Cloud-based Access Control with Blockchain Integration**

In this section, we introduce a taxonomy of a cloud-based access control scheme using blockchain, which can be classified into two types: Permission-based and Permission-less models. Figure 2.1 presents the taxonomy of the cloud-based access control with blockchain integration. Permission-based and Permission-less models

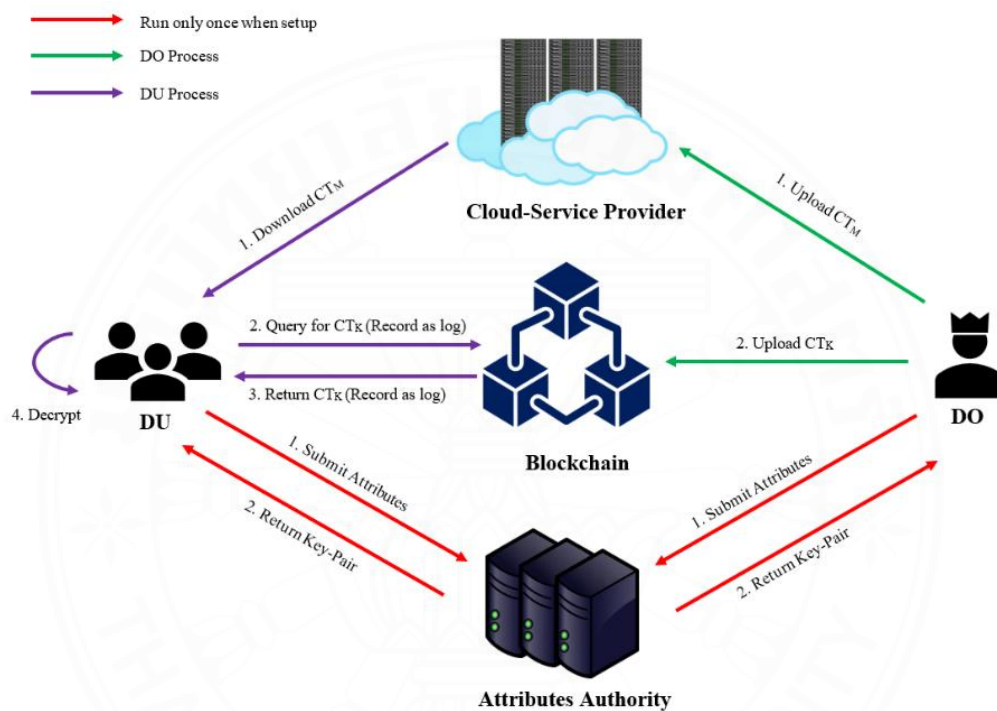
leverage the blockchain to provide three functions: access transaction retention, data search capability, and cryptographic key storage. These three features enhance the access control requirement and its usability as the data owners do not rely on the log files from the cloud provider. They do not need to implement their indexing mechanism and how ciphertext is stored on the cloud. Regarding cryptographic key storage, blockchain can retain a part of the cryptographic key, such as a hash value of ciphertext, an encrypted symmetric key associated with the ciphertext located on the cloud. This helps support key usage accountability by reducing the complexity of key retrieval generally invoked from the user or the cloud.



**Figure 2.1** Taxonomy of A Cloud-based Access Control with Blockchain Integration

In addition to the three common features, the blockchain primarily supports basic access control functions in the Permission-based model, including authentication and user privilege verification. Here, smart contracts are generally used to automate the authentication and authorization function when the users request to access shared data in the cloud. In addition, blockchain is used to store the access transactions in an immutable manner for auditing purposes and to support the data search function as the index or metadata of the ciphertexts can be obtained from the blockchain. In the

Permission-less model, the blockchain is not responsible for authenticating and enforcing the authorization. These functions are based on the application or cloud service provider and cryptographic-based method.



**Figure 2.2** A permission-less blockchain in cloud

### 2.5.1 Permission-less Cloud-based Access Control with Blockchain Integration

Figure 2.2 illustrates a permission-less blockchain system model that supports data storage and logs storage in cloud computing. In the permission-less model, blockchain generally provides a storage service that can be used to store transaction logs, ciphertext, and encrypted keys. The users in the system, such as *DO* and *DU*, must have a secret key issued by the *Attributes Authority* (*AA*). Typically, *DO* encrypts the data with the symmetric key and then uploads  $CT_M$  to store on the cloud server. After that, *DO* invokes CP-ABE or IBE encryption to encrypt the symmetric key and produce the ciphertext  $CT_K$  which will be generally stored on the blockchain. To access the data housed in the cloud, the *DU* must request to access  $CT_M$  in the cloud server. Before *DU* is able to decrypt  $CT_M$ , he/she needs to query for the appropriate  $CT_K$  from

the blockchain. When  $DU$  gets the appropriate  $CT_K$ , they can decrypt  $CT_K$  to get a symmetric key if his/her secret key contains attributes that satisfy the policy used in the encryption process. Upon receiving the symmetric key,  $CT_M$  can be decrypted.

R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) proposed a cloud-assisted revocable CP-ABE by integrating a chameleon hash function and the blockchain system as an integrity authority for the  $CT$  stored on cloud storage. A chameleon hash function is used in the key generation and update protocol for non-revoked users in their system. In this scheme, *the secret key of the data user* contains the version attribute, which is used to determine the version of the key when the revocation occurs. If any user is revoked, the attribute authority will generate the key attributes update for all non-revoked *users*' keys and send them to users. The users then update their keys respectively. As for the ciphertext policy, in their approach, the proxy will retrieve the updated secret key from the user and check the *Version* parameter based on the chameleon hash function. Suppose the *Version* parameter is up to date. In that case, the proxy performs partial decryption based on user attributes, sends the partial decrypt ciphertext to the user, and lets them finalize the decryption themselves. Otherwise, the proxy denied the policy update to the revoked user. With this approach, all non-revoked users must generate a new transformation key by themselves.

X. Liu, Y. Zheng and X. Li (2021) proposed a revocable attribute-based access control system by implementing an additional binary tree as the attributes tree called *KEK* for each user and storing them on the blockchain system. They introduce *Revocation Authority* as an authority that generates the path key for each user in the system based on the built *KEK*. A *path key* is a key that is used to check the user status in the system. When the revocation occurs, RA will update the *KEK*, re-generate the path key for each unrevoked user in the system, and redistribute them via a trusted cloud service. For the old encrypted ciphertext, RA will update the *KEK* that attaches with the ciphertext and re-upload it. Their approach does not require re-encrypting the entire ciphertext but only updating the *KEK* that attaches to the ciphertext, reducing the cost of data encryption. However, the revocation deals with several subsequent operations, imposing expensive overheads. X. Wang et al. (2021) takes a similar

approach to handle the revocation but with a twist that the *KEK* tree is used only for a header, which is used to generate a search token for the data user. In comparison, the original ciphertext will be updated according to the attributes that get removed. Thus, they suffer from the same expensive overhead problems.

L. Guo, X. Yang and W. -C. Yau (2021) proposed efficient traceable attribute encryption with a dynamic access control scheme (TABE-DAC) integrating the blockchain system. They introduced two additional algorithms to the traditional TABE-DAC: *Update policy* and *Verify policy*; these two algorithms enable DO to update the policy with less cost. *The update policy* is designed on top of the CP-ABE algorithm, while *the Verify Policy* operates on the blockchain system. In their solution, DOs must request their signature from the attribute authority by submitting their identities to the attribute authority. This signature is used to update and verify the policy. This process allows the DU to get a secret key associated with their attribute instead of digital signatures. Each ciphertext stored on the blockchain system is divided into two smaller ciphertexts: policy and key. Policy ciphertext contains *the DO* signature as an additional parameter for checking the authenticity when there is an access policy update. However, their proposed protocol did not cover the revocation of users or attributes.

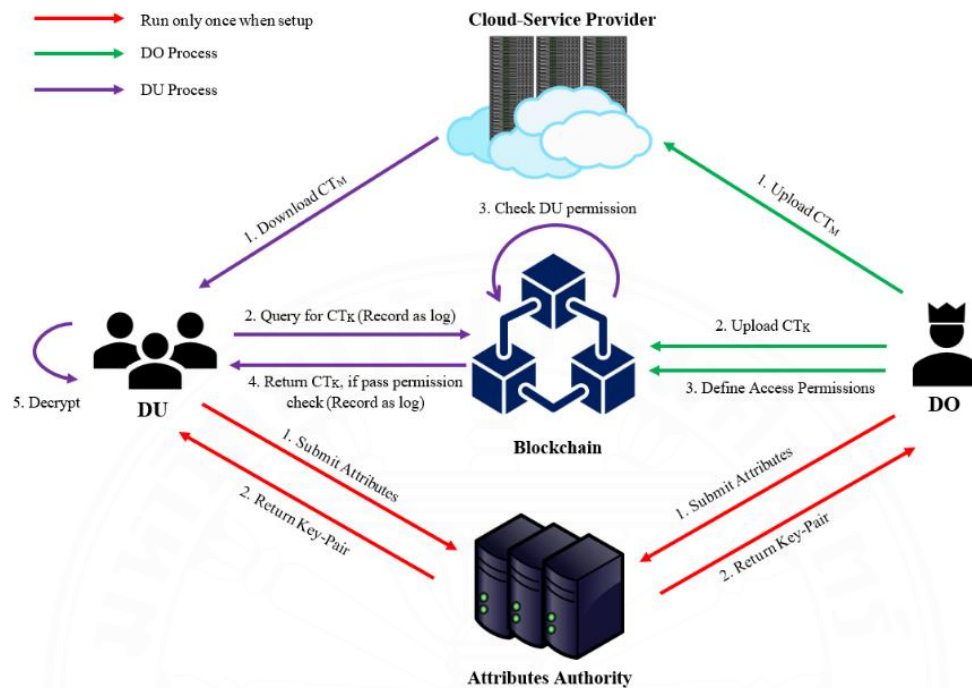
M. Jemel and A. Serhrouchni (2021) proposed a time-based access control protocol combining the CP-ABE and the blockchain. In this scheme, the blockchain is responsible for generating a key to the data users in the system. *The key contains a set of attributes*, including the timestamps that are used for decryption. The timestamps are generated via the blockchain network based on block timestamps. In the ciphertext policy, it contains a time constraint, which is used to check the validity of *the user key*. If any user is revoked, the data owner needs to re-encrypt the *ciphertexts* with a new time constraint and update the policy on the blockchain system. However, there is a problem with the time-duration policy. If the revoked *user's* key contains a time duration specified in any ciphertext, they can still access it.

Y. Jiang, X. Xu and F. Xiao (2022) proposed a CP-ABE-based model where the user secret key is divided into two smaller components: a transformation key (TK) and

a secret key (SK)—the blockchain system as an integrity authority for the *CT* that is stored on cloud storage. The cloud server is responsible for performing partial decryption by taking the TK from the *user* and the ciphertext from data storage and returning a partial decryption result to *the user*. Then, *the user* can use its own SK to decrypt the ciphertext. As for revocation, first, the attribute authority generates an upgrade parameter *UP* that contains the revocation attributes. Then, *UP* is sent to the data owner to let them perform policy updates on the affected ciphertexts. After the data owner performs the policy update protocol, they must re-upload the ciphertexts and re-generate the hash parameter stored on the blockchain system. Lastly, the attribute authority generates a new upgrade key for the remaining users and sends it to users to update their *TK* and *SK* anytime.

In 2022, S. Fugkeaw (2022) proposed a novel *e-KYCs* framework that offers a trust and privacy-preserving system with policy updating functionality based on fine-grained encryption and the blockchain system. This approach uses smart contracts to execute e-KYC registration, consent enforcement, and e-KYC verification. The registration process registers clients to the *e-KYCs* services. The client must provide credentials to the host financial institution (FI). The FI then invokes the registration contract to generate the AES session key and lets the client encrypt credential data. The encrypted session key and credential data are then uploaded to the IPFS using the hash value of the client citizen ID as the indexing. FI then generates an e-consent form and forces the client to sign—the e-KYC verification deals with the decryption of client credential information stored in the IPFS system. In addition to the privacy of credential data, all sensitive transactions stored in the blockchain are encrypted by the transaction key. The CP-ABE algorithm encrypts that key for security and privacy reasons. If the new FIs are added to the system, the policy of the transaction key ciphertext can be easily updated by updating the policy and re-encrypting the transaction key. With the fine-grained access control from the CP-ABE algorithm, only authorized FIs can access this data.





**Figure 2.3** A permission-based blockchain in cloud

### 2.5.2 Permission-based Cloud-based Access Control with Blockchain Integration

Figure 2.3 illustrates the system model of a permission-based blockchain that supports access control in cloud computing. In the permission-based, the blockchain acts as a policy enforcement point that controls the access permission of the users to the data stored on the cloud system. A data owner (DO) encrypts the data with the symmetric key and then uploads ciphertexts  $CT_M$  to store on the cloud server. Then, the DO typically invokes CP-ABE or IBE method to encrypt the symmetric key, and the ciphertext of the key  $CT_K$  is produced before it is stored in the blockchain. In CP-ABE, it is assumed that the user's secret key is generated and sent by the attribute authority (AA). The DO also specifies the assessment rule to validate the user's permission. The data user (DU) requests the blockchain to validate the authenticity and permission to access the ciphertexts stored in the cloud. If user authentication and authorization are successful, the system returns  $CT_K$  together with the ciphertext address stored in the cloud to the user. The DU can then download  $CT_M$  from the cloud server. Lastly, DU can decrypt the  $CT_K$  if their secret key contains a set of attributes



that satisfies the policy used to encrypt the symmetric key. Upon the decryption of the symmetric key, DU uses the symmetric key to decrypt the  $CT_M$  and get the data. All-access transactions are recorded as an audit log in the blockchain.

In 2017, one of the pioneering approaches of cloud-based access control schemes integrating blockchain systems was proposed by X. Liang, J. Zhao, S. Shetty, J. Liu and D. Li (2017). Their proposed solution focuses on securely sharing health data generated from patients' wearable devices. Data from wearable devices are recorded in a cloud database and blockchain network. Blockchain network stores distributed transaction logs, the hash value of the medical data, and certificate authorities (CA) who issue membership services for each entity in the system to identify who has access to the data. However, the data stored on the cloud server is not encrypted. They chose to hash those data and store them on the blockchain as the digital signature instead. They use attribute-based access control as their access control model. When the user requests to access outsourced data, the DO must verify the request and check the user's permission via the access control list stored in the blockchain. Finally, the access decision is made, either granting or denying. The significant limitations of this scheme are the need for more encryption and the dependency on DO availability to support data access.

In 2019, D. C. Nguyen, P. N. Pathirana, M. Ding and A. Seneviratne (2019) propose a trusted authority to perform electronic health records (EHRs) encryption and decryption. The proposed scheme aims to minimize the user's workload to make the scheme as lightweight as possible. Their work focuses on EHR data access over mobile devices. In their scheme, the data will be encrypted with a trusted authority RSA key pair and stored on IPFS, a cloud storage service. When the user requests the data, the trust authority will access the requested ciphertext stored in the IPFS. Then, the ciphertexts are decrypted before they are sent to the user via a secure channel. In this scheme, ABAC policy stored in the blockchain enforces authorization control. The authors also applied the smart contract to manage and execute the ABAC access control system. However, this scheme requires the trust authority to perform data encryption

and decryption. Hence, the compromise of the authority causes a complete security failure.

S. Wang, Y. Zhang and Y. Zhang (2018) proposed a new secure cloud data-sharing framework based on decentralized IPFS, Ethereum blockchain, and CP-ABE. In this scheme, blockchain stores a CP-ABE secret key, supports keyword search, and provides essential user account management. Their solution offers *DO* complete control over data with no single point of failure in the system due to the integration of decentralized IPFS and blockchain services used to store the cryptographic key. Later, S. Wang, X. Wang and Y. Zhang (2019), they combine the ABAC access control protocol with the blockchain system together by implementing the access interval of each user in the system. As long as the duration is valid, they can access the data in the blockchain. However, users who try to access off-limit data will be permanently banned from the system. Therefore, their scheme offers complete control over data to *DO* while maintaining the integrity and auditability of the data. Nevertheless, this scheme still requires the availability of data owners to handle access requests and computes the hash value upon user access.

Recently, blockchain and CP-ABE have also integrated into industrial internet-of-things (IIOT) to help generate more secure data access with traceability features to track down the culprit who intentionally shares their CP-ABE key (K. Yu, L. Tan, M. Aloqaily, H. Yang, and Y. Jararweh, 2021). They presented a user revocation feature to remove malicious users from the system when they tried to share their private key or their right to access the system. When the system detects a user who abuses the key or key leakage, the system automatically invokes the revocation protocol defined in the IIOT system. The culprit key will be revoked instantly while updating the policy of all the ciphertext in the system. The blockchain is used to store cryptographic components for *DO* and Proxy. In addition, their blockchain acts as an ABAC access control point to verify the requestor to the data in the system.

## CHAPTER 3

### PRELIMINARIES

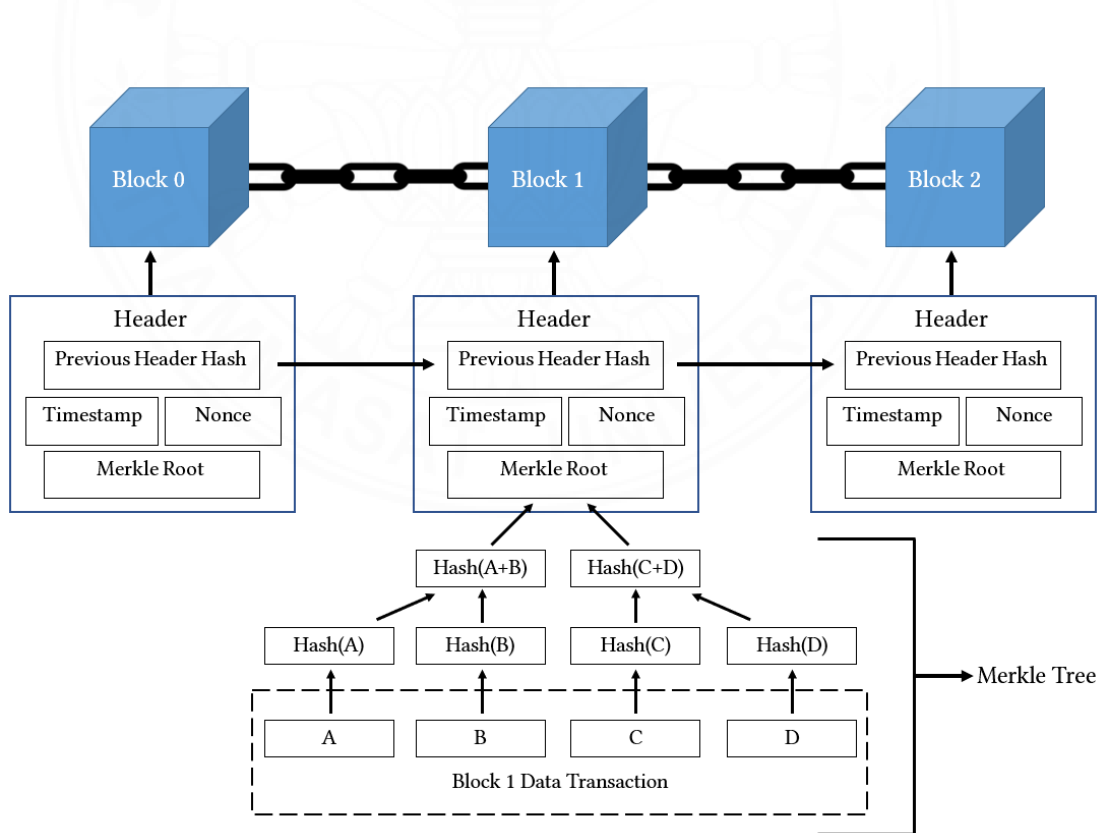
This chapter discusses the theoretical background of theories and technologies used in our proposed system. This includes the background information of blockchain systems, smart contracts, bilinear mapping, access tree structure, Advanced Encryption Standard (AES), RSA algorithm, and hidden vector encryption (HVE).

#### 3.1 Blockchain

*Blockchain technology* is an immutable, distributed, transparent, and traceable ledger that records the provenance of digital data. It is constructed and implemented through decentralization and cryptographic hashing. The digital asset or data stored in each block is immutable since the completed block is hashed and linked to each other in the blockchain network. Typically, each block contains a cryptographic hash of the previous block, a timestamp of when the transaction occurred, nonce, and transaction data. It is usually organized in a *Merkle Tree* structure. The *Merkle Tree* (H. Liu, X. Luo, and X. Xia, 2021) is a data collection where the leaf node contains the transaction data while its parent node contains its hash. The higher parent nodes have a combination of the hash value of the predecessor hash value. The topmost node of the *Merkle Tree* can be treated as the digital signature of each block, and this prevents the modification to the data that has been transacted. With this approach, a slight change to the data in the leaf node will change all the corresponding Merkle Tree hash values. The blockchain block structure is shown in Figure 3.1. The nonce is the number that can be only used once in each blockchain system. Nonce serves as a unique identifier for each block. When the blocks are linked, they are impossible to modify based on the tamper-proof property of hashed value and *Merkle Tree*.

Blockchains act as a distributed database with a growing list of blocks operating on a peer-to-peer network where nodes act as the distributed ledger. Each node in the blockchain system contains the replication of the synchronized data transaction. With these properties, the intervention of third-party entities is eliminated. These blocks can

be accessed anytime as long as they stay on the blockchain system. Blockchain adopts a consensus algorithm to make all the nodes in the system agree on the data when the data are being updated by using either the Proof-Of-Work protocol or the Proof-Of-Stake protocol. As for transparency, each action in the blockchain system is recorded as a transaction, and the blockchain network's legitimate members can access this information. In addition, the blockchain system provides the ability for the verification and traceability of all access transactions or activities that occur as they are all recorded systematically in the blockchain network. Thus, the applications that require the integrity of the data to be able to operate on the blockchain, for example, supply chain applications (S. Malik, V. Dedeoglu, S. S. Kanhere, and R. Jurdak, 2019), electronic health applications (H. Wu, L. Li, H. -y Paik, and S. S. Kanhere, 2021), and financial applications (Z. Su, H. Wang, and X. Shi, 2020; S. Nakamoto, 2009).



**Figure 3.1** Blockchain Block Structure

### 3.2 Smart Contract

A smart contract (or chain code) is a self-runnable program that runs on a Turing complete architecture system and operates on a blockchain network. The smart contracts concept was first introduced in 1997 by Nick Szabo (1997) as a computerized transaction protocol that executes in the manner of a contract. The code will be activated automatically when the predefined conditions are met. The developer predefined these conditions generally by simple “If/when...then” statements. After the execution is finished, the details of the execution will be recorded on the blockchain network. Smart contracts can execute and send transactions over the network. Users interact with a smart contract through transactions that can be executed based on a specific function constructed from rules or code.

### 3.3 Bilinear Mapping

Let  $G_0$  and  $G_1$  be two multiplicative cyclic groups of prime order  $p$ . Let  $g$  be a generator of  $G_0$  and  $e$  be a bilinear map,  $e: G_0 \times G_0 \rightarrow G_1$ . The bilinear map  $e$  has the following properties.

- a) **Bilinearity:**  $\forall u, v \in G_0$  and  $a, b \in \mathbb{Z}_p$ ,  $e(u^a, v^b) = e(u, v)^{ab} = e(u^b, v^a)$
- b) **Non-degeneracy:**  $e(g, g) \neq 1$
- c) **Computability:**  $\forall u, v \in G_0$ , an efficiently computation of  $e(u, v)$  exist

**Definition 1: Access Structure** Let a set  $\{P_1, P_2, \dots, P_n\}$  be given attribute. A collection  $\mathbb{A} \subset 2^{\{P_1, P_2, \dots, P_n\}}$  is monotone if  $\forall B, C : \text{if } B \in \mathbb{A} \text{ and } B \subset C \rightarrow C \in \mathbb{A}$ . An access structure is respectively be a monotone collection  $\mathbb{A}$  of non-empty subsets of  $\{P_1, P_2, \dots, P_n\}$ , i.e.  $\mathbb{A} \subset 2^{\{P_1, P_2, \dots, P_n\}} / \{\emptyset\}$ .

**Definition 2: Access Tree  $T$ .** Let  $T$  be a tree representing an access structure. Each non-leaf node of the tree represents a 0-threshold gate, described by its children, and a threshold value. If  $num_x$  is the number of children of a node  $x$  and  $k_x$  is its threshold value, then  $0 < k_x \leq num_x$ . When  $k_x = 1$ , the threshold gate is an *OR* gate, and when  $k_x =$

$num_x$ , it is an *AND* gate. Each leaf node  $x$  of the tree is described by an attribute and a threshold value  $k_x = 1$ . If the *k-of-n* gate is allowed in  $T$ , in this case,  $k_x = k$  where  $k$  is the threshold value determined in the *k-of-n* gate.

### 3.4 Advance Encryption Standard (AES)

Advance Encryption Standard, or AES, is also known as *Rijndael* cryptosystem. AES is a type of block cipher derived from the Rijndael cipher. It is a symmetric block cipher algorithm that securely encrypts and decrypts digital data. AES is based on substitution-permutation networks with a block size of 128 bits. The operation converts each block using the symmetric key. The difference in key size is used to determine the number of operations of rounds that the AES algorithm needs to perform during the encryption. The key size can be 128, 192, or 256 bits, with 10, 12, and 14 rounds performed for each respective key size. The process of encrypting these blocks involves combining all the blocks using the XOR operation to form the final ciphertext. A set of reverse rounds is applied using the same encryption key to decrypt the ciphertext back into the original plaintext.

For example, a 16-byte data block can be represented as a 4 x 4 two-dimensional state array as follows.

$$\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

#### High-Level Algorithmic Detail

1. **Initial Add Round Key:** This step in only runs once at the start of the algorithm. In this step, the data is passed into the state array via XOR operation with the respective key. Each byte in the 2-dimension data state array is combined with the round key using bitwise XOR operation.

$$\begin{array}{cccc}
 [b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3}] \\
 [b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3}] \\
 [b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3}] \\
 [b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3}]
 \end{array}
 \oplus
 \begin{array}{cccc}
 [k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3}] \\
 [k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3}] \\
 [k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3}] \\
 [k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3}]
 \end{array}$$

2. **Sub-Bytes:** During this step, the state array undergoes a conversion process where each byte is represented in hexadecimal format and split into two halves for rows and columns. These values are then subjected to a substitution box (S-Box) mapping to generate updated values for the final state array. Each location in the state array will be substituted with the data from the same location from the S-Box.

$$\begin{array}{cccc}
 [i_{0,0} & i_{0,1} & i_{0,2} & i_{0,3}] \\
 [i_{1,0} & i_{1,1} & i_{1,2} & i_{1,3}] \\
 [i_{2,0} & i_{2,1} & i_{2,2} & i_{2,3}] \\
 [i_{3,0} & i_{3,1} & i_{3,2} & i_{3,3}]
 \end{array}
 \xrightarrow{\text{Substitute with [4 x 4 S-Box]}}
 \begin{array}{cccc}
 [f_{0,0} & f_{0,1} & f_{0,2} & f_{0,3}] \\
 [f_{1,0} & f_{1,1} & f_{1,2} & f_{1,3}] \\
 [f_{2,0} & f_{2,1} & f_{2,2} & f_{2,3}] \\
 [f_{3,0} & f_{3,1} & f_{3,2} & f_{3,3}]
 \end{array}$$

3. **Shift-Rows:** In this step, the AES algorithm manipulates the rows of the state array by cyclically shifting the bytes in each row by a fixed offset. Specifically, the first row remains unchanged, while each byte of the second row is shifted one position to the left. The third and fourth rows are shifted by offsets of two and three positions, respectively.

$$\begin{array}{cccc}
 [b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3}] \\
 [b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3}] \\
 [b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3}] \\
 [b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3}]
 \end{array}
 \rightarrow
 \begin{array}{cccc}
 [b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3}] \\
 [b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0}] \\
 [b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1}] \\
 [b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2}]
 \end{array}$$

4. **Mix-Columns:** In this step, a fixed matrix  $[c]$  is multiplied with each column of the state array, generating a new column for the next state array. The same fixed matrix is used to multiply all the columns. The resulting state array is then produced, which will be utilized in the next Add Round Key step.

$$\begin{array}{cccc}
 [b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3}] \\
 [b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3}] \\
 [b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3}] \\
 [b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3}]
 \end{array}
 \times
 \begin{array}{c}
 [c_0] \\
 [c_1] \\
 [c_2] \\
 [c_3]
 \end{array}
 =
 \begin{array}{cccc}
 [bc_{0,0} & bc_{0,1} & bc_{0,2} & bc_{0,3}] \\
 [bc_{1,1} & bc_{1,2} & bc_{1,3} & bc_{1,0}] \\
 [bc_{2,2} & bc_{2,3} & bc_{2,0} & bc_{2,1}] \\
 [bc_{3,3} & bc_{3,0} & bc_{3,1} & bc_{3,2}]
 \end{array}$$

5. **Add Round Key:** In this step, the state array involves combining the subkey with the state by performing a bitwise XOR operation between each byte of the state and the corresponding byte of the round key. If this is the last round of the operation, the result state array will serve as an output ciphertext for the specific block. Otherwise, the result in this step will be passed as the new state array for the next round.

$$\begin{array}{cccc}
 [bc_{0,0} & bc_{0,1} & bc_{0,2} & bc_{0,3}] \\
 [bc_{1,0} & bc_{1,1} & bc_{1,2} & bc_{1,3}] \\
 [bc_{2,0} & bc_{2,1} & bc_{2,2} & bc_{2,3}] \\
 [bc_{3,0} & bc_{3,1} & bc_{3,2} & bc_{3,3}]
 \end{array}
 \oplus
 \begin{array}{cccc}
 [k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3}] \\
 [k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3}] \\
 [k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3}] \\
 [k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3}]
 \end{array}$$

When the algorithm finishes computing step 2 to step 5, this is considered as completing one round of the AES algorithm. Then, the algorithm starts the next round, starting at step 2, and continues the algorithm until the final round. The number of rounds required to compute is indicated by the key size used to perform an encryption. In the final round, step 4 (Mix-Columns) of the algorithm will be excluded, producing the final ciphertext.



### 3.5 Rivest-Shamir-Adleman Cryptosystem (RSA)

Rivest-Shamir-Adleman Cryptosystem, or RSA, is a public-key cryptography algorithm that is commonly used for secure data transmission. Its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman first publicly described it in 1977. In RSA, the encryption key is public, meaning anyone can use it. At the same time, the decryption key is kept secret (private). An RSA user generates a public key based on two large prime numbers and an auxiliary value. Anyone can use the public key to encrypt messages, but only the person who knows the prime numbers that are used to generate private keys can decrypt them. The public key comprises the public exponent  $e$  and the modulus  $n$  used for encryption. In contrast, the private key contains the private exponent  $d$ , which must be kept secret as it is used for decryption.

Additionally,  $p$ ,  $q$ , and  $\lambda(n)$  must be kept secure, as they can be used to compute  $d$ . After computing  $d$ , these values can be discarded. RSA contains three major steps: Key Generation, Encryption, and Decryption.

#### **Key Generation (Large prime number $p$ & $q$ ) $\rightarrow$ ( $RSA_{PubKey}$ , $RSA_{PrivKey}$ )**

The RSA key generation function inputs two large prime numbers,  $p$  and  $q$ . And output an RSA key pair of a public and private key. The algorithm first calculates ' $n = p * q$ ' and stores  $n$  as one of the key components. Then the algorithm process to compute *Carmichael's Totient Function* where ' $\lambda(n) = (p-1) * (q-1)$ ' and store  $\lambda(n)$  as one of the private key calculation components. Afterward, the algorithm will randomly select the integer  $e$  that is relatively prime to  $\lambda(n)$  and store them as the final public key components. Finally, the algorithm computes the private key  $d$  where  $d$  is the modular multiplicative inverse of  $e \bmod (\lambda(n))$  as follows ' $d \equiv e^{-1} \bmod (\lambda(n))$ '. The result RSA key-pair can be seen as:

$$RSA_{PubKey} = (e, n), RSA_{PrivKey} = (d, n)$$

#### **Encryption ( $M$ , $RSA_{PubKey}$ ) $\rightarrow$ ( $CT_{RSA}$ )**

The encryption algorithm takes message  $M$  and recipient  $RSA_{PubKey}$  as input. And outputs the encrypted ciphertext  $CT_{RSA}$ . To calculate the  $CT_{RSA}$ , the algorithm first

converts message  $M$  into an integer  $m$ , which can be seen as a padded plaintext with a condition of  $0 \leq m < n$ . Then the algorithm computes  $CT_{RSA}$  using the recipient  $RSA_{PubKey}$  as:

$$CT_{RSA} \equiv m^e \pmod{(n)}.$$

### **Decryption ( $CT_{RSA}, RSA_{PrivKey}$ ) $\rightarrow M$ or $\perp$**

The decryption algorithm takes ciphertext  $CT_{RSA}$  and recipient private key  $RSA_{PrivKey}$  as input. The algorithm either output message  $M$  if the private key component meets the computation restriction or  $Null \perp$ . To perform the decryption, the algorithm takes private key component  $d$ , raises the ciphertext  $CT_{RSA}$  to the power of  $d$ . The computation is as follow:

$$M \equiv CT_{RSA}^d \equiv (m^e)^d \pmod{(n)} \equiv m \pmod{(n)}.$$

Nevertheless, the RSA cryptosystem is relatively slow when compared to symmetric-key encryption algorithms like AES. RSA is generally used for key exchange and digital signatures rather than directly encrypting user data. As a result, it is more common to use RSA in combination with symmetric-key encryption, where the RSA algorithm is used to encrypt and exchange a shared key for symmetric-key encryption. This allows for more efficient encryption and decryption of user data while maintaining RSA's security benefits.

### **3.6 Hidden Vector Encryption (HVE)**

Hidden Vector Encryption (HVE) is predicate encryption designed to provide secure and efficient data encryption with searchability without decryption. HVE (D. V. Veen, 2011; J. H. Park, 2011; Z. Zhang, J. Zhang, Y. Yuan, and Z. Li, 2022) is a form of homomorphic encryption that allows computations to be performed on encrypted data without decrypting it first. The basic idea behind HVE is to represent each plaintext message as a high-dimensional vector and encrypt the plaintext vector by adding a policy random vector. The resulting encrypted vector is then transformed using matrix multiplication, and the result is sent over the network. To decrypt the ciphertext, the

receiver performs a matrix multiplication on the encrypted vector with a secret matrix and subtracts the random vector added during encryption. This results in the original plaintext vector. There are two major types of HVE: Binary-HVE and Non-Binary-HVE.

Binary-HVE is a type of HVE where the policy used to encrypt the data is typically a yes or no question and cannot contain a complex policy. In this type of HVE, the attributes policy vector consists of only '0', '1', or '\*.' '0' in Binary-HVE represents no, while '1' means yes. '\*' in HVE represents 'ignoring' those attributes in the attribute policy vector. For example, a set of attributes in the system contains five attributes: *Father*, *Mother*, *Son*, *Daughter*, and *Pet*.

In this case, the policy vector must contain five attribute values, including *Father*, *Mother*, *Son*, *Daughter*, and *Pet*. For example, suppose the policy specifies that *the father* can decrypt the data while not caring about *the pet* being able to perform decryption. In that case, the construction of the policy will be [1, 0, 0, 0, \*]. This remains true to the secret key for each user in the system. The key length is propositional to the number of attributes in the system.

Non-binary-HVE is a type of HVE where the policy can be defined more flexibly. Each vector's space can represent different attribute values. In this case, we can define the attribute's value in the system based on their attributes index. We assign each attribute value to each position in the index array based on their respective attribute index. For ease of understanding, TABLE 3.1 provides an example of the attribute index and attribute value mapping for non-binary-HVE. As shown in TABLE 3.1, the attribute value usually starts at 0, as well as the index position. As for the policy construction, if we want to define that HR department, the CEO can access the data while gender and level marked as '\*' in HVE are not concerned. The policy will be as follows: [1, \*, 2, \*]. We used Non-Binary-HVE as our attribute hiding protocol in our scheme. We represent them in a vector space where each space represents each attribute. With Non-Binary-HVE we can integrate them with the CP-ABE protocol by indicating the index value as an alphanumeric value with the corresponding numeric value as their corresponding attribute value.

**TABLE 3.1** ATTRIBUTES VECTOR TABLE EXAMPLE

Index Position	Index Value	Value	Attribute Value
0	Department	0	ICT
		1	HR
		2	R&D
1	Gender	0	Male
		1	Female
		2	Non-Binary
2	Status	0	Internship
		1	Junior
		2	CEO
3	Level	0	Low Level
		1	High Level

One of the advantages of HVE is that it enables efficient searching and matching of encrypted data. For example, if the system contains an extensive database of encrypted vectors, HVE can be used for a specific vector without decrypting the entire database. This makes HVE a valuable technique for secure information retrieval and privacy-preserving data mining applications.

## CHAPTER 4

### OUR PROPOSED SCHEME

This section presents an overview of our proposed scheme, smart contracts design, and the details of our proposed cryptographic construct.

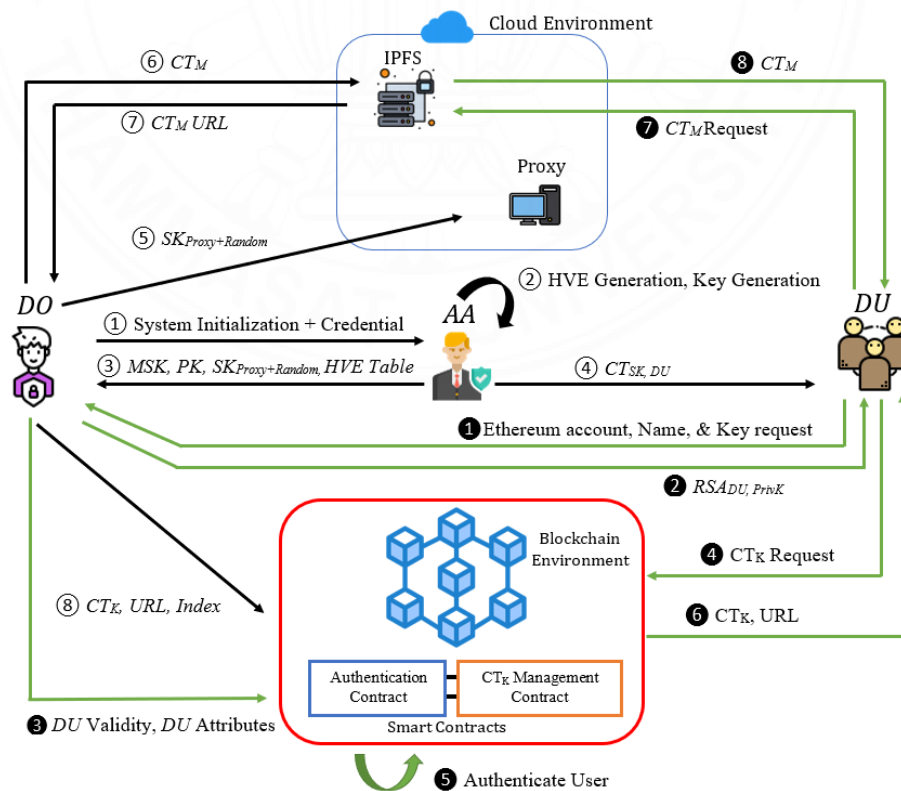
#### 4.1 System Overview

We proposed a secure, fine-grained, traceable, and revocable CP-ABE scheme based on proxy re-encryption and our blockchain-assisted protocols. Figure 4.1 illustrates the system overview of our proposed system model and the basic workflows of each entity in the system. The system model consists of the following entities.

- 1) **DOs (Data Owners)** are the owners of the data responsible for creating and deploying smart contracts on the blockchain, generating symmetric keys for data encryption, and creating the ciphertexts and uploading them to the cloud. DOs need to have an Ethereum account to interact with the blockchain system.
- 2) **DUs (Data Users)** are authorized users with decryption capability to access shared data and view transactions in the blockchain system. Each user needs to have an Ethereum account to access the blockchain system.
- 3) **AA (Attribute Authority)** generates the public parameter  $PK$  and the master secret key  $MSK$  for the data owner and proxy. AA generates the user's secret key based on their attributes. Then, the key is broadcast based on the public key encryption to users in the system.
- 4) **IPFS (Interplanetary File System)** stores encrypted data. It communicates directly with the proxy when the revocation request occurs. It maintains a distributed hash table (DHT), keeping the addresses of the ciphertexts' hash values, which are also returned to store in the blockchain.
- 5) **Blockchain** stores encrypted symmetric keys, indexes, URLs of the data, and all transactions that occur in the system. Blockchain also contains the validity status of each user for authentication purposes. In order to interact with external

entities, blockchain contains *smart contracts* that interact with *DO*, *DU*, *AA*, and *proxy*.

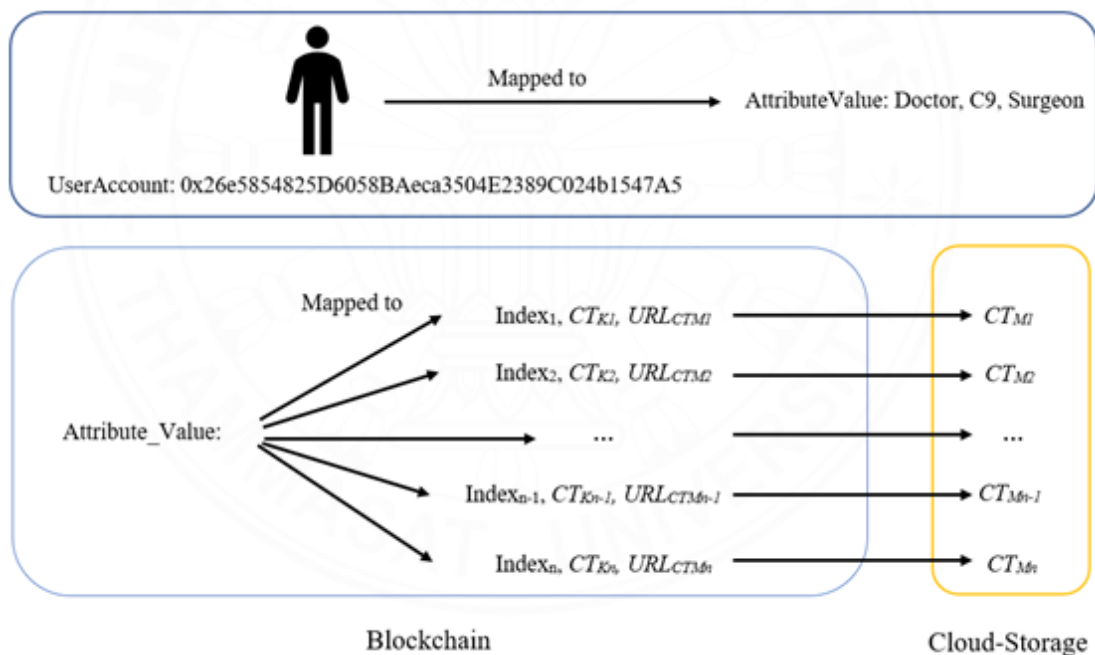
- 6) **Smart contracts** are the programmable objects that operate on the blockchain. In our system, there are two major contracts: *The Authentication contract* and *the  $CT_K$  Management contract*. *Authentication Contract* is used to authenticate *DU* when they request access to the blockchain system.  *$CT_K$  Management contract* is used to store and fetch the ciphertexts when there is an access request.
- 7) **Proxy** is a semi-trusted server located on the cloud. It is responsible for updating the policy that encrypts the symmetric key stored on the blockchain. The proxy also supports the revocation process through the ciphertext re-encryption technique. Proxy has its secret key and the unique Ethereum account used to access the ciphertexts on the blockchain system. Proxy also handles the key update request, user status update, and user secret key update.



**Figure 4.1** System Model

## 4.2 Ciphertext-Attribute-User Ethereum Account Mapping

To enable the fast retrieval of the affected ciphertexts when there is a case of revocation, we introduced the ciphertext, attributes, and user's Ethereum account mapping models shown in Figure 4.2. The model specifies the data mapping between attributes that belong to each user together with the associated  $CT_K$  and their related component in the blockchain. Precisely, the attribute value is mapped with the index of  $CT_K$  and the  $URL$  of  $CT_M$ . Based on the mapping scheme, the set of affected ciphertexts stored in the cloud can be invoked efficiently when the proxy needs to be retrieved for re-encryption.



**Figure 4.2** Ciphertext-Attributes-User Ethereum Account Mapping Model

## 4.3 Attributes Hiding

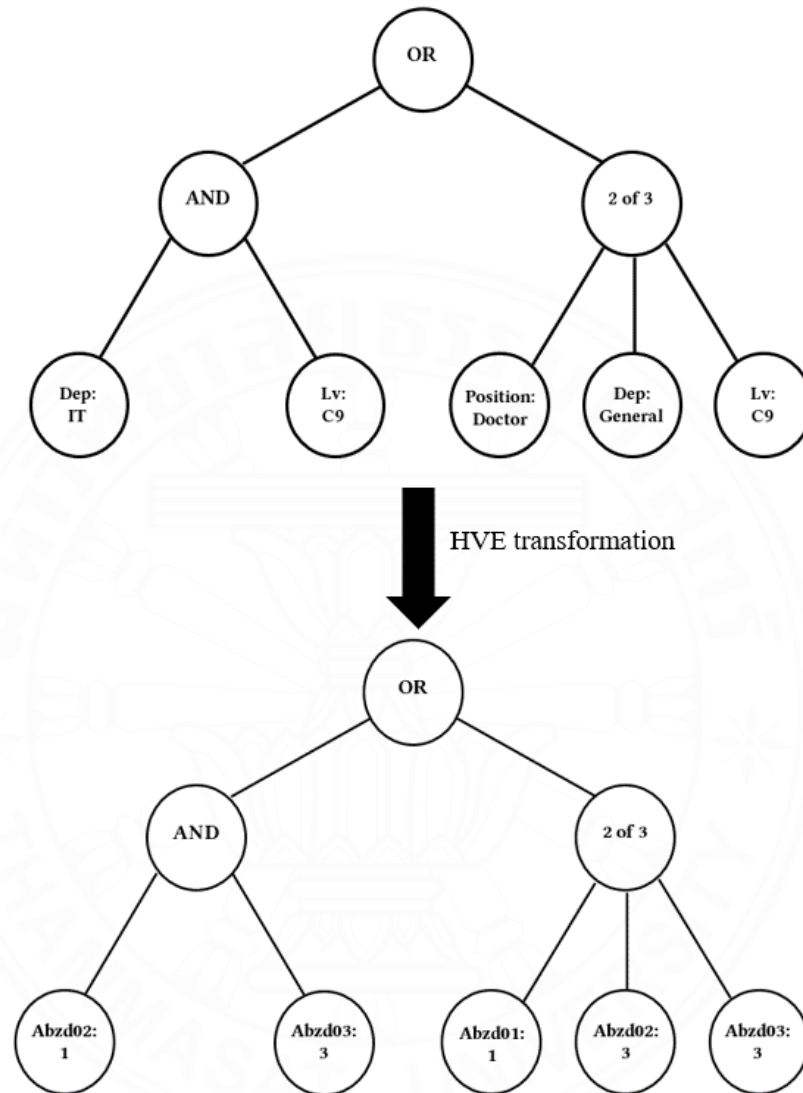
To enable the secure invocation of the access policies to be used by the proxy for the re-encryption process, we proposed the attribute hiding method to support secure outsourced re-encryption. Our attribute hiding scheme is based on Hidden Vector Encryption (HVE) of attributes vector. In our scheme, an attribute vector is an ordered pair of attribute indexes and their appropriate value. The attribute vector table is

managed and maintained by the DO. The attribute vector index is randomly generated for the first four alphabets, and then the system will pad the number to it. Attributes in the system are assigned with the attribute index and mapped to their associated value. Based on the anonymous attributes contained in the policy, the content of the access policies is hidden while they are used to support the encryption and re-encryption process. Figure 4.3 presents an example of policy transformation based on our proposed model. TABLE 4.1 presents an example of attribute index and value mapping. As presented in TABLE 4.1, for example, Alice's secret key contains the following attributes: {Doctor, Neurology, C9, Bangkok}, her private key is then composed of the set of mapped values in the vector: {Abzd01: 1, Abzd02: 2, Abzd03: 3, Abzd04: 1}. Technically, this attribute name and value are used for the data encryption as naturally done in the CP-ABE method.

**TABLE 4.1** ATTRIBUTES VECTOR TABLE EXAMPLE

Index	Index Value	Value	Attribute Value
Abzd01	Position	1	Doctor
		2	Network Engineer
		3	HR
Abzd02	Dep	1	IT
		2	Neurology
		3	General
Abzd03	Lv	1	Internship
		2	Junior
		3	C9
Abzd04	Location	1	Bangkok
		2	Pattaya





**Figure 4.3** Attribute-Tree with Attribute Hiding

#### 4.4 Smart Contract Design

This section describes the details of the smart contracts' functions used in our scheme. Our smart contracts are developed in the Solidity (C. Dann en, 2017) while operating on the Ethereum blockchain on the Ganache Truffle Suite. Two major smart contracts consist of an *Authentication Contract* triggered by user enrollment and authentication requests and a *CT<sub>K</sub> Management Contract* triggered by the revocation request initiated by the data owners.

Precisely, the *Authentication Contract* consists of three primary functions: User enrollment, User authentication, and User attribute queries. The details of each function are described below.

**1. *UserEnrollment(DUEthereumAccount, DUAttribute[ ])*:** This function enrolls DU into the system with its appropriate attributes. It takes two inputs: *DU public Ethereum account* and their associate *attributes array*. It maps *DU's validity status* and associated attributes to the *DU's public Ethereum account*. In our scheme, DU's attributes are used as a query index for ciphertext retrieval when there is a revocation case. Its procedure is detailed in Algorithm 1.

**2. *Authentication( )*:** This function takes no input. Instead, when *DU* requests access to the blockchain, their public Ethereum account will be set as *msg.sender*. This value will be used as a validation variable in the mapping mechanism to retrieve their validity status. The function will terminate the requestor's connection and record their access attempt on the blockchain if their validity status is invalid. The algorithmic function of the authentication is presented in Algorithm 2.

**3. *DUAttributeQuery(DUEthereumAccount)*:** This function is run by the proxy when the user revocation occurs. It is used to query DU attributes from the blockchain system. It takes *the DU public Ethereum account* as input and returns an array of DU attributes or *NULL*. The algorithmic details of the DU attribute query are presented in Algorithm 3.

**4. *UserStatusUpdate(DUEthereumAccount, ValidityStatus, UpdatedAttributes)*:** It takes as inputs the DU public Ethereum account, Validity Status of DU, and Updated Attributes. It is run when any DU has been revoked from the system. The *UserStatusUpdate* function is presented in Algorithm 4.

---

**Algorithm 1: UserEnrollment**(*DUethereumAccount*, *DUAttribute*[ ]) 

---

**Input:** *DUethereumAccount*, *DUAttributes*[ ]**Output:** Bool

```

1 If msg.sender is not (dataOwner or Proxy) then
2   Return false and terminate the connection;
3 End if
4 If DUethereumAccount is existed in
5   User[DU Ethereum Public Account] then
6   Return false
7 Else
8   Set User[DU Ethereum Public Account].Validity = Valid;
9   Set User[DU Ethereum Public Account].Attr[ ] = DUAttributes[];
10 Return true

```

---



---

**Algorithm 2: Authentication**( ) 

---

**Input:** *null***Output:** Bool

```

1 If User[msg.sender].Validity != Valid then
2   Return false and terminate the connection;
3 Else
4 Return true

```

---



---

**Algorithm 3: DUAttribute Query**(*DUethereumAccount*) 

---

**Input:** *DUethereumAccount***Output:** *Attributes*[ ] or *Null*

```

1 If msg.sender is not (dataOwner or Proxy) then
2   Return false and terminate the connection;
3 End if
4 If DUethereumAccount is not exist then
5   Return NULL;
6 Else
7   Set Attribute[ ] = User[DUethereumAccount].Attr;
8 Return Attribute[ ];

```

---



---

**Algorithm 4: UserStatusUpdate**(*DUethereumAccount*, *ValidityStatus*, *UpdatedAttributes*) 

---

**Input:** *DUethereumAccount*, *ValidityStatus*, *UpdatedAttributes***Output:** Bool

```

1 If msg.sender is not (dataOwner or proxy) then
2   Return false and terminate the connection;
3 End if
4 If DUethereumAccount is not existed then
5   Return false
6 Else
7   Set Users[DUethereumAccount].Validity = Validity Status
8   Set Users[DUethereumAccount].Attr.push(Updated Attributes)
9 Return true

```

---

For the *CT<sub>K</sub> Management Contract*, it is designed to perform three primary functions: Key Ciphertext upload, Key Ciphertext query, and Key Ciphertext attributes query. This contract consists of four functions: *CT<sub>K</sub>Upload*, *SetCT<sub>K</sub>Attributes*, *CT<sub>K</sub>AttributeQuery*, and *CT<sub>K</sub>Query*. The details of each function are described below.

**5. *CT<sub>K</sub>Upload( CT<sub>K</sub>, URL, Index)*:** The function takes as inputs *AES Key* ciphertext *CT<sub>K</sub>*, *CT<sub>M</sub> URL* stored on IPFS, and *index*. It is used to upload the *CT<sub>K</sub>* and *URL* to the blockchain system by enabling Solidity to map the *CT<sub>K</sub>* and *URL* to the respective *index*. The *CT<sub>K</sub> upload* function is presented in the Algorithm 5 as follows.

**6. *SetCT<sub>K</sub>Attribute(AttributeValue[ ], Index)*:** The function takes as inputs the attribute Value array and the *CT<sub>K</sub>* index. It maps the input attributes to the corresponding *CT<sub>K</sub> index* to support ciphertext queries. First, it counts the length of the *Attribute Value* array and uses it as a maximum loop count. Then, the function will perform a *for loop* to map the index to the attribute value and store it on the blockchain. The detail of the function is presented in Algorithm 6.

**7. *CT<sub>K</sub>AttributeQuery(AttributeValue)*:** This function is run when there is a revocation request. It takes the attribute value to be queried and returns the array of the index of the ciphertext corresponding to the attribute value stored on the blockchain. First, the function checks if an attribute value exists in the blockchain; it returns the array of indexes mapped to the attribute value. The algorithmic function of the *CT<sub>K</sub> Attribute Query* is presented in Algorithm 7.

**8. *CT<sub>K</sub>Query(Index)*:** This function is used to query the index of *CT<sub>k</sub>*. It takes the index as an input and returns its corresponding *CT<sub>K</sub>* and URL, or *NULL*, to the executor. The function first checks whether the index of the requested *CT<sub>k</sub>* is available. If the index exists in the system, the function will return the *CT<sub>K</sub>* and the *URL* that is mapped to the

Index from *Algorithm 5* to the executor. Otherwise, this function returns *NULL*. The procedure of *CT<sub>K</sub> Query* is presented in Algorithm 8.

---

**Algorithm 5: CT<sub>K</sub> Upload** (*CT<sub>K</sub>, URL, Index*)

---

**Input:** *CT<sub>K</sub>, URL, Index*

**Output:** Bool

```

1 If msg.sender is not (dataOwner or Proxy) then
2   Return false and terminate the connection;
3 End if
4 Set CTKIndex[Index].CTK = CTK
5 Set CTKIndex[Index].CTK_URL = URL
6 Return true

```

---



---

**Algorithm 6: Set CT<sub>K</sub> Attribute**(*AttributeValue[ ], Index*)

---

**Input:** *AttributeValue[ ], Index*

**Output:** Bool

```

1 If msg.sender is not (dataOwner or Proxy) then
2   Return false and terminate the connection;
3 End if
4 Set j = AttributeValue.length
5 For (i = 0; i <= j; i++)
6   Set attr = AttributeValue[i]
7   Set x = CTKAttributes[attr].No
8   x = x + 1;
9   Set CTKAttributes[attr].No = x
10  Set CTKAttributes[attr].Index.push(Index)
11 Return true

```

---



---

**Algorithm 7: CT<sub>K</sub> Attribute Query**(*AttributeValue*)

---

**Input:** *AttributeValue*

**Output:** *Indexes[ ]* or Null

```

1 If msg.sender is not (dataOwner or Proxy) then
2   Return false and terminate the connection;
3 End if
4 If AttributeValue is not exist then
5   Return NULL
6 Else
7   Set Indexes = CTKAttributes[AttributeValue].Index[ ]
8   Return Indexes[ ]

```

---

**Algorithm 8:  $CT_K$  Query( $Index$ )****Input:**  $Index$ **Output:** ( $CT_K$  and URL) or NULL1 **If**  $CT_KIndex[Index].CT_K$  is not exist **then**2     **Return** NULL3 **Else**4     **Return**  $CT_KIndex[Index].CT_K$  and  $CT_KIndex[Index].CT_K\_URL$ **4.5 Cryptographic Constructs**

This section describes the cryptographic construct of our proposed system. The notations used in our model are shown in TABLE 4.2.

**TABLE 4.2** NOTATION USED IN OUR MODEL

Notation	Description
$AA$	The attribute authority
$S$	A set of attributes issued to data users and data owners in the system. In this case, the attributes are hidden under HVE method
$SK_{DU}$	A user's secret key issued by the AA.
$SK_{Proxy+random}$	A Proxy's secret key bound with a random value issued by the AA.
$PK$	Public attribute key issued by AA.
$MSK$	Master attribute key issued by AA for $SK_{DU}$ and $SK_{DO}$ generating.
$AES\_Key$	Symmetric AES key (256-bit) used for encrypting the data.
$RSA_{DU, PubK}$	RSA public key of the DU
$RSA_{DU, PrivK}$	RSA private key of the DU
$M$	A message that the data owner needs to encryption and distribute on the cloud storage.
$CT_M$	Ciphertext of a message.
$CT_K$	The ciphertext of AES_key
$CT_{SK, DU}$	RSA encrypted $SK_{DU}$
$AP$	An access policy used for CP-ABE encryption.

Our cryptographic process consists of five major phases as follows: *System Initialization, Key Generation, Encryption, Decryption, and Revocation.*

**Phase 1: System Initialization**

$CreateAttributeAuthority(\lambda) \rightarrow PK, MSK$ . The algorithm takes security parameter  $\lambda$  as an input and returns public key  $PK$  and master secret keys  $MSK$ . The algorithm selects a bilinear group  $G_0$  of prime order  $p$  with generator  $g$ . After that, the

algorithm then chooses two random  $\alpha, \beta \in Z_p$  and compute a public key and master secret key as:

$$PK = \{G_0, g, h = g^\beta, f = g^{\frac{1}{\beta}}, e(g, g)^\alpha\},$$

$$MSK = \{\beta, g^\alpha\}.$$

### Phase 2: Key Generation

In our model, we define three key types used by *DO*, *Proxy*, and *DU*. The crypto process of each key generation is described as follows:

#### 1) *AESKeyGen(RandomString)* $\rightarrow$ *AES\_Key*

The algorithm takes random string as an input to generate a 256-bit *AES\_key*. *DO* uses *AES\_key* to encrypt the data before uploading them to cloud storage.

#### 2) *RSASKeyGen( 2 RandomLargePrimeNumber )* $\rightarrow$ *RSADU, PubK, RSADU, PrivK*

Initially, the user runs the *RSASKeyGeneration* algorithm, which inputs two random large prime numbers to generate an RSA key pair. The Certification Authority (CA) then signs each user's public key. The CA also publishes the certificate containing the public key in the public directory.

#### 3) *UserKeyGeneration (PK, MSK, S<sub>DU</sub>, Ver)* $\rightarrow$ *SK<sub>DU</sub>*.

This algorithm is run by the AA. The algorithm takes *PK*, *MSK*, a set of *DU*'s attributes *S<sub>DU</sub>*, and a Version parameter *Ver* used to specify the key version of the *SK<sub>DU</sub>*. The algorithm generates a user secret key *SK<sub>DU</sub>* containing the key version and user's attributes.

The algorithm first chooses a random  $r$  and  $r_j \in Z_p$  for each attributes  $j \in S$ . Then the algorithm compute *SK<sub>DU</sub>* as:

$$SK_{DU} = (D = g^{(\alpha+r)/\beta}, \forall j \in S: Dj = g^r \cdot H(j)^{r_j}, D'j = g^{r_j}).$$

Then, AA encrypts the  $SK_{DU}$  based on public key encryption by using the user's public key  $RSA_{DU, PubK}$ . The encryption is computed as:

$$ENC_{RSA}(RSA_{DU, PubK}, SK_{DU}) \rightarrow CT_{SK, DU}$$

After that, AA sends *the*  $CT_{SK, DU}$  to the user. The user then uses its *RSA private key* to decrypt the  $CT_{SK, DU}$  and gets the  $SK_{DU}$ . DO then enrolls DU to the blockchain system by running **Algorithm 1**.

For the proxy's key, after the key generation algorithm is finished, the key will be appended with a 256-bit random number and encrypted by the proxy's public key. Then, the encrypted key will be sent to the proxy.

### Phase 3: Encryption

In our proposed scheme, we introduce a dual encryption method comprising symmetric key encryption and CP-ABE Encryption. The encryption consists of two following steps.

1) *Encrypt Message*( $AES\_Key, M$ )  $\rightarrow CT_M$

The algorithm is run by DO. It takes a symmetric key  $AES\_Key$  to encrypt data  $M$ . The algorithm produces ciphertext  $CT_M$  and DO stores it on the IPFS.

2) *Encrypt AES\_Key*( $PK, AP, AES\_Key$ )  $\rightarrow CT_K$ .

The algorithm takes as inputs  $PK$ , access policy  $AP$ , and  $AES\_Key$ . Then, it outputs  $CT_K$ . To compute  $CT_K$ , the encryption algorithm encrypts a  $AES\_Key$  under the access structure  $AP$ . The algorithm then chooses a polynomial  $q_x$  for each node  $x$  (including leaves node) in  $AP$ . The polynomials are chosen in the top-down manner, starting from root node  $R$ . For each node  $x$  in the tree, set the degree  $d_x$  of the polynomial  $q_x$  to be one less than the threshold value  $k_x$  of the node,  $d_x = k_x - 1$ . After that, starting



from root node  $R$  the algorithm chooses a random  $s \in \mathbb{Z}_p$  and set  $q_x(0) = q_{\text{parent}(x)(\text{index}(x))}$  and choose  $d_x$  randomly to completely define  $q_x$ . Let  $Y$  be a set of leaf nodes in  $AP$ . The result ciphertext is computed as follows:

$$CT_K = (AP, C^{\sim} = (AES\_Key) e(g, g)^{a^S}, C = h^S, \forall_y \in Y : C_y = g^{q_y(0)}, C'_y = att(y)^{q_y(0)})$$

The  $CT_K$  is then stored on the blockchain together with its index value by initiating **Algorithm 5** and **Algorithm 6** respectively.

#### Phase 4: Decryption

The decryption is done by the DU after the successful authentication via the smart contract (**Algorithm 2**) and retrieves appropriate key ciphertext and their corresponding URL (**Algorithm 7**) via the blockchain system. DU then downloads  $CT_M$  from the IPFS and performs the decryption. This phase includes two algorithms: *Decrypt  $CT_K$*  and *Decrypt  $CT_M$* .

##### 1) *Decrypt $CT_K(SK_{DU}, CT_K) \rightarrow AES\_Key$*

The algorithm takes as inputs DU's secret key  $SK_{DU}$  and  $CT_K$ . The algorithm outputs an  $AES\_Key$  which will be used in the final decryption step. The decryption algorithm can be specified as a recursive algorithm as follows:

$DEC_{CP-ABE}(SK_{DU}, CT_K, x)$  is the algorithm that takes  $CT_K = (AP, C^{\sim}, C, \forall_y \in Y : C_y, C'_y)$ , a secret key  $SK_{DU}$  which associate with a set  $S$  of attributes, and a node  $x$  from access policy  $AP$ . If  $SK_{DU}$  contains attributes that belong to  $AP$ , the algorithm will return  $AES\_Key$ , otherwise it returns  $Null$ . The computation is as follows: If the node  $x$  is a leaf node, then we set  $i = att(x)$ . If  $i \in S$ , then

$$\begin{aligned} DEC_{CP-ABE}(SK_{DU}, CT_K, x) &= \frac{e(D_{i, C_x})}{e(D'_{i, C'_x})} \\ &= \frac{e(g^r \cdot H(i)^r i, h^{q_x(0)})}{e(g^r i, H(i)^{q_x(0)})} \\ &= e(g, g)^{r \cdot q_x(0)} \end{aligned}$$

If  $i \notin S$ , then we set  $DEC_{CP-ABE}(SK_{DU}, CT_K, x) = Null$ .

As for the recursive case where  $x$  is not a leaf node. The algorithm  $DEC_{CP-ABE}(SK_{DU}, CT_K, x)$  will proceed as follows: For all node  $z$  that are the children of node  $x$ , it then executes  $DEC_{CP-ABE}(SK_{DU}, CT_K, z)$  and stores the output as  $F_z$ . Then we define  $S_x$  as an arbitrary  $k_x$ -sized set of child nodes  $z$  with the condition that  $F_z \neq Null$ . If the set  $S_x$  cannot fulfill the condition before then the node was not satisfied with the policy and returns  $Null$ . Otherwise, we compute as follows:

$$\begin{aligned}
 F_x &= \prod_{z \in S_x} F_z^{\Delta_{i, S'_x(0)}}, \text{ where } e_{S'_x = \{index(z): z \in S_x\}}^{i=index(z)} \\
 &= \prod_{z \in S_x} (e(g, g)^{r \cdot q_z(0)})^{\Delta_{i, S'_x(0)}} \\
 &= \prod_{z \in S_x} (e(g, g)^{r \cdot q_{parent(z)}(index(z))})^{\Delta_{i, S'_x(0)}} \\
 &= \prod_{z \in S_x} e(g, g)^{r \cdot q_x(i) \cdot \Delta_{i, S'_x(0)}} \\
 &= e(g, g)^{r \cdot q_x(0)}
 \end{aligned}$$

And return the result.

Then we define the final decryption algorithm. The algorithm initiates by calling the  $DEC_{CP-ABE}(SK_{DU}, CT_K, x)$  on the root node  $R$  of the access tree  $AP$ . If the access tree is satisfied by  $S$ , then we set  $A = DEC_{CP-ABE}(SK_{DU}, CT_K, r) = e(g, g)^{r \cdot q_x(0)} = e(g, g)^{r \cdot s}$ . The algorithm then performs the decryption by computing as follows:

$$C \sim / (e(C, D) / A) = C \sim / (e(h^s, g^{(\alpha+r)/\beta}) / e(g, g)^{r \cdot s}) = AES\_Key$$

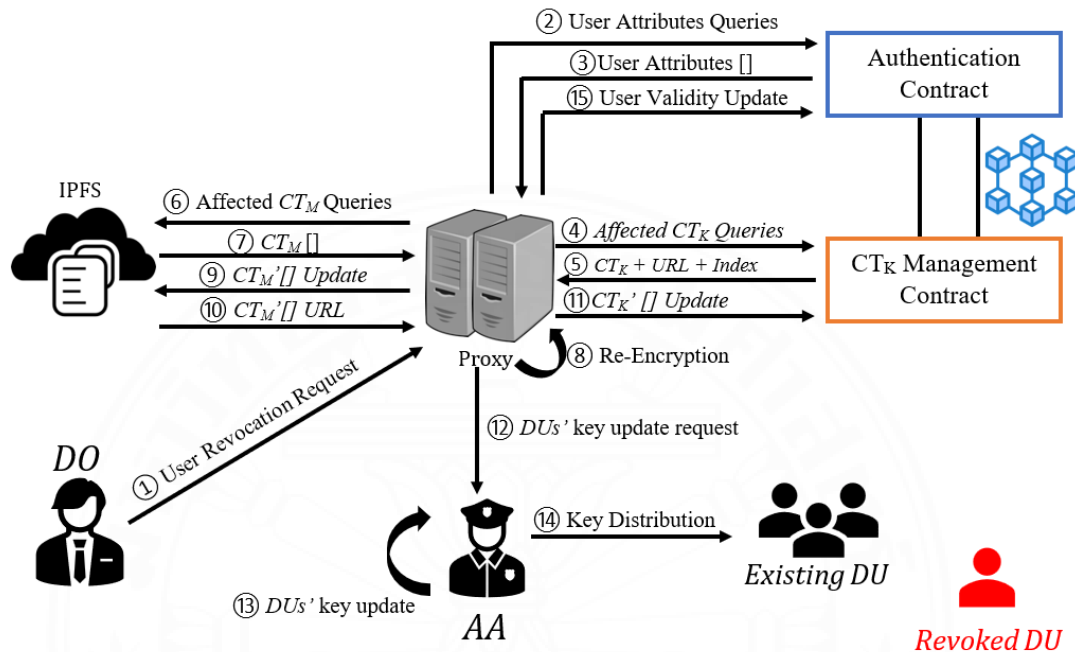
2) Decrypt  $CT_M(AES\_Key, CT_M) \rightarrow M$

The algorithm takes as inputs  $AES\_Key$  and  $CT_M$ . The algorithm produces data  $M$  by decrypting the  $CT_M$  with the  $AES\_Key$ .

### Phase 5: Revocation

This phase consists of two cases: user revocation and attribute revocation. In our scheme, the revocation task is executed by the proxy server located in the cloud. At the same time, the DO only generates a new access policy  $AP'$  and sends the revocation

request to the proxy. The algorithmic details of each case are described as follows.



**Figure 4.4** User Revocation Process Diagram

### Case 1: User Revocation

If any user is revoked from the system, the DO sends a revocation request containing new access Policy  $AP'$ , revoked DU's Ethereum Account  $ACC_{DU}$ , revoked DU information, and new version parameter  $Ver'$  to the proxy. Figure 4.4 presents the workflow between each entity in the system. User revocation cases contain four major steps as follows.

#### Step 1: Query for affected $CT_K$

The proxy queries for attributes of the revoked DU and then queries for the ciphertexts that can be accessed by the revoked user as the *affectedCT<sub>k</sub>* array by executing the *attributeCTqueryFunction*. The affected ciphertexts are stored in the IPFS, while their corresponding keys are stored in the blockchain system. The function takes as input a revoked DU's Ethereum Account  $ACC_{DU}$ . It outputs an array of key ciphertexts  $CT_K$  that contains the corresponding  $URL$  path to their respective  $CT_M$  that needs to be re-encrypted with a new symmetric key. The processes for invoking the

DU's attributes and  $CT_K$  are done through *Algorithm 3*, *Algorithm 7*, and *Algorithm 8*, respectively. The function is defined as follows:

```

attributeCTqueryFunction(ACCDU) return (Array[CTKi, URLCTMi, Indexi]){
    string DUattr[ ];
    string affectedCTkIndex[ ];
    affectedCTk[ ];
    string temporaryIndex;
    DUattr = DUAttributeQuery(ACCDU);
    For ( i = 0; i <= len(DUattr) ; i++) {
        temporaryIndex = CTKAttributeQuery(DUattr[ i ]);
        affectedCTkIndex.push(temporaryIndex);
        temporaryIndex = “ ”;
    }
    removeDuplicate(affectedCTkIndex);
    For ( j = 0; j <= len(affectedCTkIndex); j++){
        affectedCTk.push(CTKQuery(affectedCTkIndex[ j ]), affectedCTkIndex[ j ]);
    }
    Return affectedCTk;
}

```

### Step 2: Re-generate a Symmetric Key

The proxy generates a new symmetric key corresponding to the affected ciphertext. The function is defined as follows:

$$Re-GenSymKey (Rs, r) \rightarrow (AES\_Key_R)$$

The algorithm takes as inputs a random string  $Rs$  and a random parameter  $r$ . Then, the algorithm generates a new  $AES\_Key_R$ , a symmetric key perturbed by a random parameter.

### Step 3: Re-encrypt ciphertexts

The proxy then runs the *Re-Encryption* function to update the policy of the affected ciphertexts. The function takes as inputs an array of  $affectedCT_k$ , a proxy secret key  $SK_{Proxy+Random}$ ,  $AES\_Key_R$ , the public key  $PK$ , and a new access policy  $AP'$ . Then, it returns the re-encrypted ciphertexts array to the proxy. The function is defined as follows:

```

Re-Encryption(affectedCTk[ ], SKProxy+Random, AES_KeyR, PK, AP') return (Array[CTKi,
URLCTMi, Indexi]) {
    SKProxy = ExtractRandom(KProxy+Random);
    For ( i = 0 ; i <= len(affectedCTk) ; i++){
        CTK = affectedCTk [i][0];
        CTM = Download(affectedCTk [i][1]);
        AES_Key = DECCP-ABE(SKProxy, CTKi);
        M = DECAES(CTM, AES);
        AES' = AES_KeyR - r;
        CTM' = ENCAES(M, AES');
        CTM'URL = Upload(CTM');
        CT'Ki = ENCCP-ABE(PK, AP', AES');
        affectedCTk [i][0] = CT'Ki;
        affectedCTk [i][1] = CTM'URL;
    }
    SKProxy = "0";
    Return affectedCTk;
}

```

After the algorithm returns the array of *affectedCT<sub>k</sub>* to the proxy, the proxy then runs a recursive function based on the number of key ciphertexts in the *affectedCT<sub>k</sub>* array. The proxy then runs **Algorithm 5** to update their respective value.

#### Step 4: Key Update

In this step, the proxy sends the key update request, the Revoked DU information, and the new *Ver'* to the AA. AA then invokes the key update function to update all non-revoked users' keys. The key update steps are as follows:

1) AA generates an update parameter *UP* based on *Ver'* parameter.

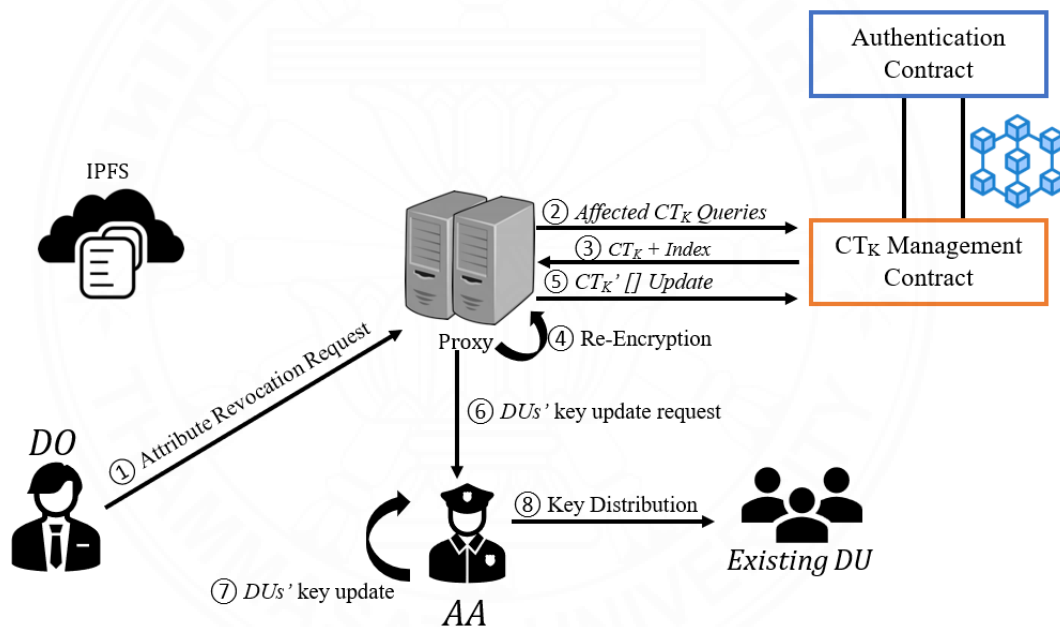
$$UP = g^{(rj' - rj)/\beta} \text{ where } rj = Ver, rj' = Ver'$$

2) AA applies an *UP* to all non-revoked users in the system. The update secret key function is as follows:

$$SK'_{DU} = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'j = g^{rj}, D'j' = D'j \cdot UP \text{ where } rj = Ver, rj' = Ver')$$

3) AA then encrypts the new  $SK'_{DU}$  with user's public key and sends to the users.

After finishing the key update, the proxy updates the access permission list (APL) on the blockchain system with the revoked DU's Ethereum account to prevent further access from the revoked user by invoking **Algorithm 4** from the *Authentication Contract*. The reason behind this is that the blockchain will check the validity status of the user account before they can access it. If their account is *valid*, they can access the blockchain system. Otherwise, the blockchain will terminate the connection to the requestor.



**Figure 4.5** Attribute Revocation Process Diagram

### Case 2: Attribute Revocation

To revoke the attribute, DO sends the revocation requests with parameters: a new access Policy  $AP'$ , and revoked Attribute information  $Attr$  to the proxy. Figure 4.5 presents the attribute revocation process.

#### Step 1: Query for affected key ciphertext $CT_K$

In this case, the proxy queries the key ciphertexts that contain the revocation attribute by executing the *attributeCTqueryFunction*. In this case, the algorithm only

re-encrypts the key ciphertexts that contain the revoked attribute. The function takes as input a revoked attribute  $Attr$ . It outputs an array of key ciphertexts that need to be re-encrypted with a new policy  $AP'$ . The function is defined as follows:

```

attributeCTqueryFunction(Attr) return (Array[CTKi, URLCTMi, Indexi]) {
    string affectedCTkIndex[ ];
    affectedCTk[ ];
    affectedCTkIndex = CTkAttributeQuery(Attr);
    For ( i = 0; i <= len(affectedCTkIndex); i++){
        affectedCTk.push(CTkQuery(affectedCTkIndex[i]),affectedCTkIndex[i]);
    }
    Return affectedCTk;
}

```

#### Step 2: Re-encrypt key ciphertexts

The function takes as inputs an array of  $affectedCT_k$ , a proxy secret key  $SK_{Proxy+Random}$ , public key  $PK$ , and a new access policy  $AP'$ . Then, it returns the re-encrypted ciphertexts array to the proxy. The function is defined as follows:

```

Re-Encryption(affectedCTk[ ], SKProxy+Random, PK, AP') return (Array[CTKi, URLCTMi, Indexi]) {
    SKProxy = ExtractRandom(SKProxy+Random);
    For ( i = 0 ; i <= len(affectedCTk) ; i++){
        CTK = affectedCTk [i][0];
        AES_Key = DECCP-ABE(SKProxy, CTKi);
        CT'Ki = ENCCP-ABE(PK, AP', AES_Key);
        affectedCTk [i][0] = CT'Ki;
    }
    SKProxy = "0";
    Return affectedCTk;
}

```

After the algorithm returns the resulting array of ciphertext re-encryption  $affectedCT_k$  to the proxy. The proxy then runs a recursive function based on the number of key ciphertexts in the array and runs the **Algorithm 5** function from the *CT<sub>k</sub>Management Contract* to update their respective value.

### Step 3: Key Update

In this step, the proxy sends the key update request to the AA. The AA runs the key update function as follows:

- 1) AA generates an update parameter  $UP$  based on the  $Attr$  parameter.

$$UP = g^{(rj' - rj)/\beta} \text{ where } rj = Attr$$

- 2) AA runs  $UP$  to update a set of attributes for all active DUs in the system. The update secret key is done through the following function:

$$SK'_{DU} = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'j = g^{rj}, D'j' = D'j \cdot UP \text{ where } rj = Attr)$$

- 3) AA then encrypts the new  $SK'_{DU}$  with user's public key and sends to the users.



## CHAPTER 5

### SECURITY ANALYSIS

This section explains the security analysis of our scheme based on security assumptions, security games, and cryptographic constructs given in Chapters 3 and 4.

#### 5.1 Security Model of our proposed scheme

In our proposed scheme, the security is proven in a game-based theory. Our scheme is based on CP-ABE (J. Bethencourt, A. Sahai, and B. Waters, 2007), AES symmetric key encryption, and RSA public key encryption scheme. Detailed proof of its security can be referred to the original CP-ABE paper, the AES (Announcing the ADVANCED ENCRYPTION STANDARD (AES), 2001), and the Rivest-Shamir-Adleman cryptosystem (P. Meelu & S. Malik, 2010).

Our security assumption is based on *the IND-CPA* security assumption. The security model of our proposed scheme is defined by the security game with the assumption that only the data owner is fully trusted. At the same time, the data users are assumed to be dishonest. Also, the adversary may corrupt the authorities. However, the key queries can be adaptive. In our proposed model, there are two types of adversaries:

- *Type-A adversary* is the one who has no permission to access the data from the beginning.
- *Type-B adversary* is the data user previously revoked from the system.

Consequently, the implementation of a key update algorithm is essential. Furthermore, it is essential to note that the *Type-B adversary* faces challenges in accurately updating the key version.

The security game is conducted between *adversary A* and *challenger B* is defined as follows:

**Setup:** First, challenger  $B$  runs System Initialization algorithm to generate master secret key  $MSK$ , public key  $PK$ . After that,  $B$  sent  $PK$  to  $A$ .

**Phase 1:** This phase contains three cases based on the type of adversary. Adversary  $A$  repeatedly generates secret key queries  $SK$  corresponding to sets of attributes  $S$ . Challenger  $B$  then performs *Key Generation Algorithm* and *Key Update Algorithm* to the recently generated  $SK$  before returning  $SK$  to  $A$ .

**Challenge Phase:** Adversary  $A$  generates two messages,  $M_1$  and  $M_2$ , with an equal length such that  $|M_1|=|M_2|$ . In addition,  $A$  generates two random 256-bit symmetric keys,  $AES\_Key_1$  and  $AES\_Key_2$ , and access policy  $AP^*$ . Access policy  $AP^*$  must not contain the attributes that appear in set  $S$  of adversary  $A$ 's secret key. Then adversary  $A$  submits the randomly generate symmetric key to challenger  $B$ . Challenger  $B$  randomly selected  $b$ , where  $b \in \{1,2\}$  with an equal chance to get both values.  $B$  computes  $AESEncrypt(AES\_Key_b, M_b) \rightarrow CT_M^*$  and  $Encrypt(PK, AP^*, AES\_Key_b) \rightarrow CT_K^*$ . After that,  $CT_M^*$  and  $CT_K^*$  are given to  $A$ .

**Phase 2:** Repeat step 1 with the condition that each attribute set  $S$  that use to construct the secret key cannot contain the attributes in the access policy  $AP^*$ .

**Guess Phase:** Adversary  $A$  outputs a guess  $b'$  of  $b$ . The adversary wins if  $b' = b$  and the advantage of the adversary  $A$  is equal to  $|\Pr[b' = b] - 1/2|$

**Definition 3** Our scheme is secure against polynomial time adversaries in an *IND-CPA* security assumption who have, at most, negligible advantages in the above game with the probabilistic advantage value of  $|\Pr[b' = b] - 1/2|$ .

## 5.2 Security Proof of our proposed scheme

**Theorem 1:** There are no polynomial-time advantages for an adversary, suppose that *IND-CPA* security assumption holds, who can break the security of *AES* symmetric encryption/decryption and CP-ABE with non-negligible advantage.

*Proof:* Suppose the *Adversary A* possesses probabilistic polynomial time advantages and can break the security scheme with a non-negligible advantage against our proposed scheme. In that case, we can simulate the following game, enabling *A* to break our scheme with non-negligible advantage.

**Initialization:** Suppose *Adversary A* has non-negligible advantage against our scheme.

**Setup:** *Adversary A* submits system initialization request to *Challenger B*. *Challenger B* then runs the System Initialization algorithm. This algorithm selects a bilinear group  $G_0$  of prime order  $p$  with generator  $g$ , chooses two random values  $\alpha$  and  $\beta$  from  $Z_p$  to compute a public key, and sends the resulting *PK* to *A*.

$$PK = \left\{ G_0, g, h = g^\beta, f = g^{\frac{1}{\beta}}, e(g, g)^\alpha \right\}, MSK = \{\beta, g^\alpha\}.$$

**Phase 1:** *A* then requests secret key generation queries to *B*.

*Type-A Adversary:* Suppose *B* generates a set of attributes  $S$  with the condition that  $S$  cannot satisfy access policy  $AP^*$ . *B* then generate a secret key  $SK_A$ . The  $SK_A$  is constructed as follows:

$$SK_A = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'_j = g^{rj})$$

After constructing a  $SK_A$ , *Challenger B* has to perform a key update mechanism to update the key version parameter in the  $SK_A$ . In this case, *B* generates the update

parameter  $UP_A$  that contains the latest key version parameter and applies the  $UP_A$  to  $SK_A^*$ . Then  $B$  sent the newly updated  $SK_A^*$  to  $A$ . The construction of  $UP_A$  and newly updated  $SK_A^*$  is as follows:

$$UP_A = g^{(rj' - rj)/\beta} \text{ where } rj = Ver, rj' = Ver'$$

$$SK_A^* = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'j = g^{rj}, D'j' = D'j \cdot UP \text{ where } rj = Ver, rj' = Ver')$$

*Type-B Adversary:* Suppose  $B$  generates a set of attributes  $S$  with the condition that  $S$  does not contain the right key Version parameter  $Ver'$  thus they cannot satisfy access policy  $AP^*$ .  $B$  then generates a secret key  $SK_A$ . The  $SK_A$  is constructed as follows:

$$SK_A = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'j = g^{rj})$$

After constructing a  $SK_A$ , *Challenger B* has to perform a key update mechanism to update the key version parameter in the  $SK_A$ . In this case,  $B$  generates the update parameter  $UP_A$  that does not contain the latest key version parameter and applies the  $UP_A$  to  $SK_A$ . Then  $B$  sent the newly updated  $SK_A^*$  to  $A$ . The construction of  $UP_A$  and newly updated  $SK_A^*$  is as follows:

$$UP_A = g^{(rj' - rj)/\beta} \text{ where } rj = Ver, rj' \neq Ver'$$

$$SK_A^* = (D = g^{(\alpha+r)/\beta}, \forall j \in S : Dj = g^r \cdot H(j)^{rj}, D'j = g^{rj}, D'j' = D'j \cdot UP \text{ where } rj = Ver, rj' \neq Ver')$$

**Challenge Phase:**  $A$  generates two messages,  $M_1$  and  $M_2$ , with equal length such that  $|M_1| = |M_2|$ . In addition,  $A$  generates two random 256-bit symmetric keys,  $AES\_Key_1$

and  $AES\_Key_2$ , and access policy  $AP^*$ . Then submit those to *challenger B*. *Challenger B* randomly selected  $b$ , where  $b \in \{1,2\}$  with an equal chance to get both values. *B* then encrypts  $M_b$  with symmetric key encryption  $AES\_Key_b$ . Afterward, *B* encrypts  $AES\_Key_b$  with the CP-ABE algorithm. Furthermore, *B* sends the result  $CT_M^*$  and  $CT_K^*$  to *A*. The  $CT_M^*$  and  $CT_K^*$  is constructed as follows:

$$CT_M^* = AESEncrypt(AES\_Key_b, M_b)$$

$$CT_K^* = (AP^*, C^{\sim} = (AES\_Key_b)e(g,g)^{a^S}, C = h^S, \forall_y \in Y : C_y = g^{q_y^{(0)}}, C'_y = att(y)^{q_y^{(0)}})$$

**Phase 2:** *A* continuously generates secret key queries. The process is the same as phase 1, with the same restriction for different adversary types. For *Type-A Adversary*, each attribute set  $S$  cannot contain the attributes that satisfy access policy  $AP^*$ . As for *Type-B Adversary*, their secret key must not contain the right Version parameter.

**Guess Phase:** *A* output a guess statement  $b'$ , where  $b' \in \{1,2\}$ . *A* win the game by outputting the guess statement where  $b' = b$ . Hence, the advantage of *Adversary A* against our proposed scheme is as follows:

$$ADV_A = | \Pr[b' = b] - 1/2 |$$

Since *A* has a nonnegligible advantage against our scheme, we have successfully proven the theorem.

The full proof of CP-ABE can be referred to the original paper (J. Bethencourt, A. Sahai, and B. Waters, 2007).

**Theorem 2:**  $SK_{DU}$  from corrupted attribute authorities alone cannot be used to decrypt all the components of the ciphertexts stored on the cloud storage.

*Proof:* In our scheme, the ciphertexts stored on the cloud storage are encrypted by symmetric key encryption. Moreover, the CP-ABE mechanism encrypts the symmetric key and stores it securely on the blockchain system.

To perform full decryption on the ciphertext stored on the cloud storage, the adversary needs to know the key ciphertext's index and pass the authentication process from the blockchain system to retrieve the key ciphertext. With this method,  $SK_{DU}$  from the corrupted authorities alone cannot be used to decrypt the ciphertext stored on the cloud.

**Theorem 3:** After the DUs have been revoked from the system, access to the data stored on the blockchain and IPFS is no longer available to revoked DUs. In the event that revoked DUs find a way to access the data stored on both blockchain and IPFS, they will not be able to utilize their secret key and the old AES key to access the data.

*Proof:* In our proposed approach, when the DO initiates the user revocation protocol, a new secret key version parameter,  $Ver'$ , and a new Access Policy,  $AP'$ , which includes  $Ver'$  as a mandatory policy, are generated. This means that DUs lacking the specific  $Ver'$  attribute in their  $SK_{DU}$  cannot decrypt the  $CT_K$ ' store on the blockchain. Additionally, for each key ciphertext  $CT_K$  accessible by the revoked DU, the proxy generates a new AES key. It performs symmetric key re-encryption for the data ciphertext component,  $CT_M$ , residing on the IPFS. The updated  $CT_M'$ , resulting from the re-encryption, replaces the old  $CT_M$  on the IPFS. Alongside this, the newly generated AES key is re-encrypted using the new access policy  $AP'$  to form a new key ciphertext,  $CT_K'$ . The proxy then updates the URL of the newly updated  $CT_M$  and its corresponding  $CT_K'$  on the blockchain. Subsequently, the proxy employs Algorithm 4 from the smart contract to revoke the DU's access privileges within the blockchain system. This ensures that the revoked DU can no longer query or retrieve data stored on the blockchain system. Through the ciphertext and symmetric key re-encryption

process, the revoked user is effectively blocked from using their holding keys to access the data.

### **5.3 Forward Security**

Forward security refers to the concept that the revoked user cannot access the data subsequently. In our scheme, this security property is assured by the key update mechanism. Essentially, the key update done by the AA will not be issued to the revoked user as of the verification of the access permission list and the update parameter verification.

### **5.4 Backward Security**

Backward security refers to the concept that the revoked user cannot decrypt previously encrypted shared data. Our scheme guarantees this security property based on the ciphertext re-encryption. In our scheme, when the user is revoked, all the ciphertexts ever accessed by the revoked user will be re-encrypted by a new *AES key*, which is encrypted by a new access policy.

### **5.5 Confidentiality of Ciphertexts on Cloud and Blockchain Storage**

The ciphertexts stored on a cloud storage are encrypted by symmetric key encryption algorithms with 256-bit key length, while their *AES key* is encrypted by CP-ABE encryption and stored on the blockchain system. With both encryptions, by their security protocol, the ciphertexts cannot be cracked in polynomial time-space. In addition, without valid blockchain credentials, the key ciphertexts cannot be retrieved.

### **5.6 Proxy's Key Security**

In our scheme, we allow the proxy to keep its key in a secure manner. During the key generation method, we added a 256-bit random number to obfuscate the proxy's secret key after generating the key. Without the knowledge of the padding number policy, the attacker cannot use the proxy's secret key. In addition to padded random numbers, all attributes in our scheme are hidden via an HVE method. Even if the attacker can gain the proxy's secret key, they cannot use it to decrypt the ciphertext or have knowledge of what attributes are used to construct the key.

## CHAPTER 6

### COMPARATIVE ANALYSIS AND EVALUATION

In this section, we evaluate our proposed scheme by presenting the comparative functional analysis and computation cost analysis of our scheme and related works. In addition, we conducted experiments to measure the performance of the encryption, decryption, re-encryption, and ciphertext querying processes of our scheme and related works. For ease of understanding, we provide the notation used in the comparative analysis, as shown in TABLE 6.1.

**TABLE 6.1** NOTATION FOR COMPARATIVE ANALYSIS SECTION

Notation	Description
$PRX$	Proxy Server
$BC$	Blockchain
$CS$	Cloud Storage
$G_0$	Exponential operation in group $G_0$
$G_1$	Exponential operation in group $G_1$
$E$	Bilinear pairing operation
$ G_0 $	Size of element in $G_0$
$ G_1 $	Size of element in $G_1$
$ E $	Size of element that use in bilinear pairing
$ AP $	Number of attributes in access policy
$ UA $	Number of attributes in user secret key
$AES_{Enc}$	AES encryption operation
$AES_{Dec}$	AES decryption operation
$ AES_{Key} $	AES key size
$ M $	Data size
$ CT_M $	Encrypted data size
$ PK $	CP-ABE Public key size
$ RSA_{DU} $	RSA Private key size
$ APL $	Access Permission list size

#### 6.1 Functionality Analysis

TABLE 6.2 illustrates the functionality comparison of our proposed scheme and three related literature, including R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), Y. Jiang, X. Xu and F. Xiao (2022), and K. Yu, L. Tan, M. Aloqaily, H. Yang



and Y. Jararweh (2021). The functionality comparison is analyzed based on the aspect of the attribute hiding functionality, revocation capability, and ciphertext querying functionality. In general, only our scheme supports attribute-hiding functionality. As for the revocation aspect, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) only support user revocation, while Y. Jiang, X. Xu and F. Xiao (2022) only supports attribute revocation. In our proposed scheme, we support both users and attribute revocation levels. Lastly, our proposed scheme is the only scheme that provides the formal method for querying the affected ciphertexts when the revocation occurs.

**TABLE 6.2** FUNCTIONAL COMPARISON

	Attributes Hiding	Revocation		Ciphertext Querying Functionality
		User.	Attr.	
R. Guo et al.	$x$	✓	$x$	$x$
Y. Jiang et al.	$x$	$x$	✓	$x$
K. Yu et al.	$x$	✓	$x$	$x$
Our	✓	✓	✓	✓

**TABLE 6.3** COMPUTATION COST COMPARISON

	Computation Cost	
	Encryption Cost	Decryption Cost
R. Guo et al.	$(3 AP  + 1)G_0$	$(2 AP  +  UA )G_1 + 2 AP E$
Y. Jiang et al.	$(4 AP  + 2)G_0 + E$	$( UA  + 2)E + (2 AP )G_1$
K. Yu et al.	$(2 AP  + 4)G_0 + 3G_1$	$(2 AP  +  UA )E + ( UA  + 2)G_1$
Our	$(2 AP  + 1)G_0 + 2G_1 + AES_{Enc}$	$(2 UA  + 1)E + (2 AP  + 2)G_1 + AES_{Dec}$

## 6.2 Computation Cost Analysis

The computation cost of cryptographic operations is crucial to evaluate the access control system's efficiency, scalability, and practicality. As shown in TABLE 6.3, the encryption cost of all schemes is subject to the number of attributes in the policy

that need to be encrypted with the data together with the exponential operation of the prime order group  $G_0$ . In schemes R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), Y. Jiang, X. Xu and F. Xiao (2022), and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) their data encryption methods are done through the CP-ABE mechanism. Y. Jiang, X. Xu and F. Xiao (2022) scheme deals with a bilinear pairing operation, which results in higher computation costs than the other schemes. While in our scheme, we utilized a 2-step encryption operation that contains AES and CP-ABE algorithms. Our AES cryptosystem is used to encrypt/decrypt the desired data, while CP-ABE is used to encrypt/decrypt the AES key. The computation cost of the AES algorithm is relatively small compared to the CP-ABE method due to the smaller key size and lighter crypto operation costs. As a result, our encryption cost yields the least execution time compared to the others. For the decryption cases, the computation cost is subject to the number of attributes in the policy and the number of attributes contained in the user secret key, together with the exponential operation of prime order group  $G_1$  and bilinear pairing operation, are the major costs. In R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) scheme, the computation cost is based on both pairing and exponential operation of prime order group  $G_1$ , which yields more computation cost than Y. Jiang, X. Xu and F. Xiao (2022), K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021), and ours. Y. Jiang, X. Xu and F. Xiao (2022) and our scheme only deals with the exponential operation on group  $G_1$ , while K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) relies on pairing operation. Specifically, our scheme yielded a smaller decryption cost than other schemes because the decryption was done over the encrypted *AES key*, while other schemes worked directly with the encrypted data.

### 6.3 Communication Cost Analysis

*Communication cost* is the cost that occurs when there is data communication between different entities in the system. This section will focus on the ciphertext or data communication between entities, and the access request will not be accounted for in

this comparison. Here, we consider the size of security parameters sent to and forth between the entities in the systems.

**TABLE 6.4** COMMUNICATION COST COMPARISON

	Scheme			
	R. Guo et al.	Y. Jiang et al.	K. Yu et al.	Our
DO & DU	-	$ M $ , Signature	-	$ RSA_{DU} $
DO & BC	-	URL + Hash( $CT_M$ ) + Signature	$ PK $	$[(2 AP +1) G_0  + 2 G_I  +  AES_{Key} ]$ , Index, $CT_{kAttr}$ & URL
DO & CS	-	$[(4 AP +2) G_0  +  E  +  M ]$	$[(2 AP +4) G_0  + 3 G_I  +  M ]$	$ CT_M $
DO & PRX	$[(3 AP +1) G_0  +  M ]$	-	-	$[(2 UA ) G_0  +  G_I ]$
PRX & CS	$[(3 AP +1) G_0  +  M ]$	$[(4 AP +2) G_0  +  E  +  M ]$	$[(2 AP +4) G_0  + 3 G_I  +  M ]$	$ CT_M $
PRX & BC	Hash( $CT_M$ )	-	$ PK $ , Signature( $SK_{DU}$ )	$[(2 AP +1) G_0  + 2 G_I  +  AES_{Key} ]$ , Index, $CT_{kAttr}$ & URL
DU & BC	-	-	-	$[(2 AP +1) G_0  + 2 G_I  +  AES_{Key} ]$ , Index, URL
DU & CS	-	-	-	$ CT_M $
DU & PRX	$[( UA +2) G_0 ,  G_I  +  M ]$	$[( UA +3) G_0 ,  G_I  +  M ]$	$[2 G_I  +  M ]$	-

As shown in TABLE 6.4, in R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), the communication cost occurs at the proxy server and other entities such as data owners and users in the system. In their system, after the DO performs CP-ABE encryption, the DO needs to upload the ciphertext to the proxy before it is uploaded to the cloud storage. Then, the proxy hashes the ciphertext and uses its values as the index to be stored on the blockchain system. In Y. Jiang, X. Xu and F. Xiao (2022), the doctor has to send EHR data, the hash value, and their digital signature to the patient to let

them perform CP-ABE encryption on the EHR data and upload the encrypted data to the cloud storage. With this approach, the communication cost between DO and DU tends to be larger than the other approach. The data's signature and hash value are then uploaded to the blockchain by DO. For R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and Y. Jiang, X. Xu and F. Xiao (2022) when DU wants to access the ciphertext, they need to send the request with the transformation key to the proxy. With this, each time DU wants to access the data, the communication cost of sending the transformation key occurs. The proxy then fetches the ciphertext from the cloud storage and performs partial decryption before the intermediate ciphertext is sent to the DU.

In K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021), there are no communication costs between DO and DU because, in their scheme, the cryptographic component is generated by trusted authorities and a trusted proxy and stored those components on the blockchain and on the proxy itself. When DO wants to encrypt the data, they need to make an access request to the blockchain system to receive the *PK*. This renders the same communication overhead as in R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and Y. Jiang, X. Xu and F. Xiao (2022) when the DU wants to access the data. The proxy invokes the ciphertext from the cloud storage, performs partial decryption, and sends the result to the DU.

In our scheme, the communication cost between DO and DU is only the DU *RSA key pair* transmission. In contrast, the communication between DO and Proxy involves transferring the proxy's CP-ABE key. For the blockchain system, the communication between it and the DO and Proxy is the key ciphertext, its respective index, URL, and its attributes. Even though blockchain and other entities communicate by transmitting such cryptographic elements, their size is relatively small. In our scheme, DU can access the data stored on the cloud and blockchain directly, compared to other schemes where DU must send their data access request to a proxy and let the proxy fetch the data for them.

#### 6.4 Storage Cost Analysis

*Storage cost* refers to the cost of storing cryptographic keys, ciphertexts, hash value of the ciphertexts, digital signature, index value, and, in some cases, ciphertexts attributes at their respective entities.

**TABLE 6.5** STORAGE COST COMPARISON

	Scheme			
	R. Guo et al.	Y. Jiang et al.	K. Yu et al.	Our
DO	$ PK $	$ PK $	-	$ PK $
DU	$[( UA +2) G_0 + G_I ]$	$[( UA +3) G_0 + G_I ]$	$[(2 UA +1) G_0 + G_I ]$	$[(2 UA ) G_0 + G_I ],  RSA_{DU} $
PRX	-	-	$[(2 UA +1) G_0 ]$	$[(2 UA ) G_0 + G_I ]$
BC	$\text{Hash}(CT_M)$	$\text{Hash}(CT_M),$ Signature	$PK,$ Signature( $SK_{DU}$ )	$[(2 AP +1) G_0 +2 G_I + AES_{Key} ],$ Index, Attributes[], $ APL $
CS	$[(3 AP +1) G_0 + M ]$	$[(4 AP +2) G_0 + E + M ]$	$[(2 AP +4) G_0 +3 G_I + M ]$	$ CT_M $

As shown in TABLE 6.5, most works share the exact storage cost for storing public keys at DO, CP-ABE secret key at DU, and CP-ABE encrypted ciphertexts at cloud storage. For our scheme, cloud storage stored the symmetric encryption ciphertext instead. For proxy, in our scheme, the proxy holds its secret key, while in K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) it holds the DU transformation key. On the blockchain side, schemes R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and Y. Jiang, X. Xu and F. Xiao (2022) are closely similar; the blockchain stores the ciphertexts' hash values as the integrity tampered-proof certificates. If their ciphertexts have been altered, their hash value will be different from the hash value of their respective ciphertexts in the blockchain system. In Y. Jiang, X. Xu and F. Xiao (2022), they also stored the digital signature of the DU with the hash value of the ciphertext. For K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh

(2021), their blockchain system holds the public key of the system and the digital signature of DU's secret key. With their approach, when DO wants to encrypt the data, they need to request the blockchain system for the PK. In our scheme, blockchain stores CP-ABE encrypted AES keys with their index and attributes. With our approach, the proxy can retrieve the corresponding cryptographic component directly from the blockchain system and cloud storage without external interference when the revocation occurs. Moreover, DU can access the data directly with the blockchain system as long as their account status is still valid on the blockchain. In addition to storing the ciphertext, our scheme also stored the access permission list of the DU in the system for authentication.

## 6.5 Experimental Analysis

To evaluate the performance of our proposed system, we conducted experiments to compare the encryption time, decryption time, revocation time, and query time of our scheme and the related works, including schemes R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), Y. Jiang, X. Xu and F. Xiao (2022), and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021). These three works were chosen because they all deployed blockchain as their immutable record for data integrity, implemented the revocation protocol, and used CP-ABE as their core cryptography.

For the experiment setting, we utilized Open SSL as a core PKI system for generating key pairs to users and the proxy in our system. The CP-ABE Toolkit, Java-Pairing based Cryptography (PBC Library, 2022; A. De Caro & V. Iovino, 2011), and AES Toolkits (Packetizer, 2023) are used to simulate the cryptographic operation of our scheme, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), Y. Jiang, X. Xu and F. Xiao (2022), and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021). As for blockchain simulation, we used Ethereum on Ganache Truffle Suite (2023) with the help of Web3.js as a communication interface. For decentralized storage services, we used the IPFS application. The experiment environment for DO is as follows: AMD Ryzen 5 5600G (3.90 GHz), 16 GB of RAM, and a 64-bit Windows 11 Operating System. As for the proxy environment, we use the Google Cloud platform

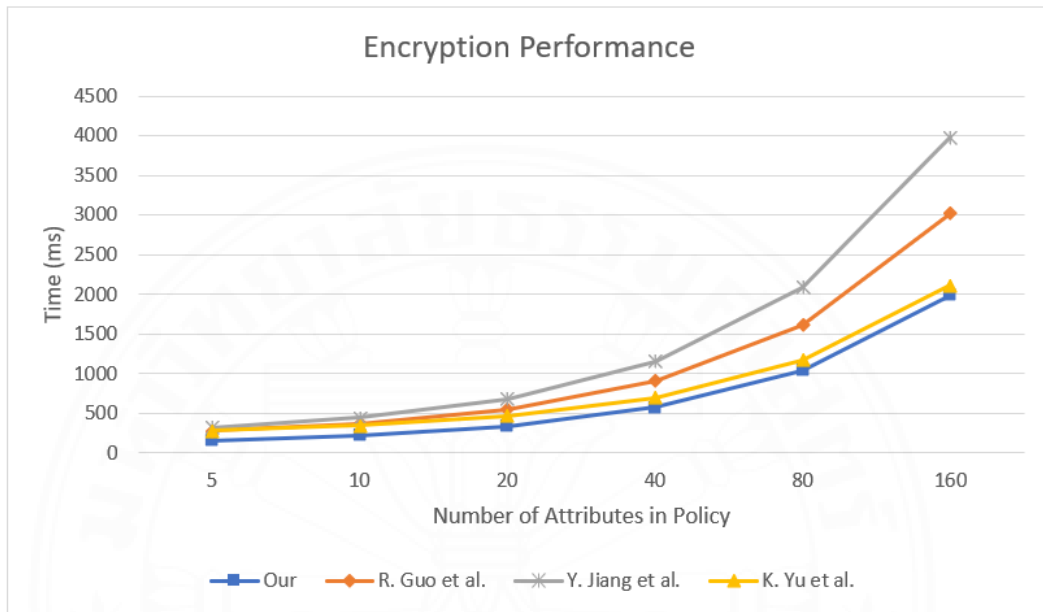
with computer engine 'E2-Micro' with Intel Xeon 2.20 GHz, 1 GB of RAM, and Ubuntu 20.04.5 LTS OS. The proxy server runs the IPFS and the essential cryptographic protocol to support the revocation, such as *AES* symmetric key encryption and CP-ABE encryption.

In our experiment, the user secret key contains five attributes, and the 300-KB file was used to test the encryption, decryption, and revocation operation.

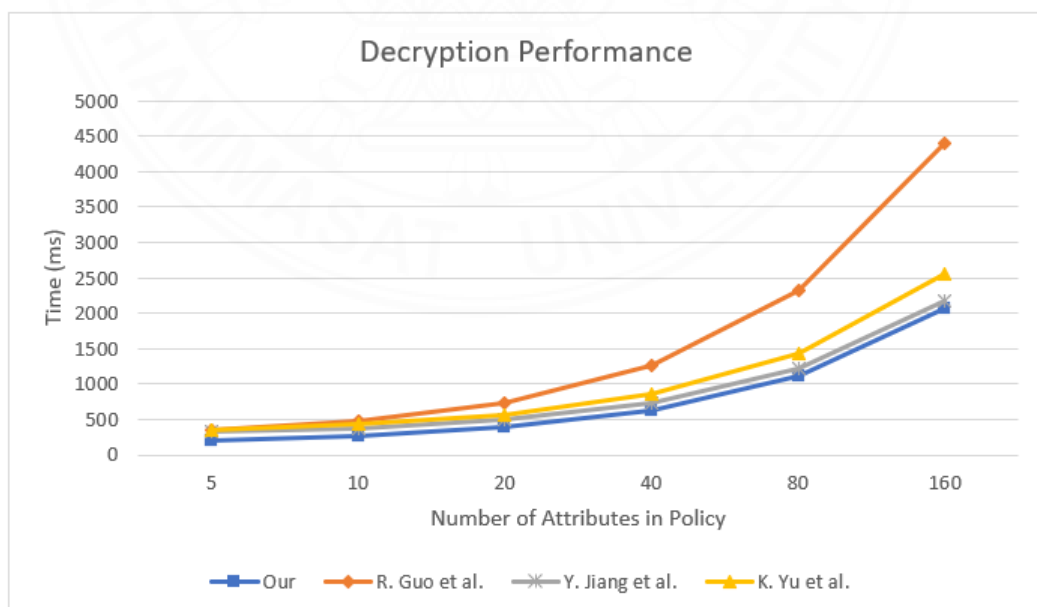
### 6.5.1 Encryption and Decryption Performance

We measured the encryption and decryption time by varying the size of the access policy. The experiments were done to measure the processing time used for data encryption and decryption between our scheme, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) scheme, Y. Jiang, X. Xu and F. Xiao (2022) scheme, and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) scheme. Figure 6.1 and Figure 6.2 present the encryption and decryption performance, respectively. For the encryption performance, our scheme and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) grew linearly with the size of the access policy, while the graphs of scheme R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and scheme Y. Jiang, X. Xu and F. Xiao (2022) tend to increase sharply when the higher number of attributes in the policy was applied. This is because R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and Y. Jiang, X. Xu and F. Xiao (2022) schemes deal with more complexity of computation that relates to the number of attributes in the policy than ours and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021). Their scheme requires more policy instances to be computed with a prime order group. Specifically, in the scheme Y. Jiang, X. Xu and F. Xiao (2022), the encryption cost was also subject to additional bilinear pairing operation. As for the decryption performance, the performance of all schemes was subject to the size of the access policy. R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) yielded the highest decryption cost since its decryption function requires both exponential and bilinear pairing operations on the number of attributes in the policy. For Y. Jiang, X. Xu and F. Xiao (2022) scheme and our scheme, the

decryption process deals with the exponential operation on group  $G_1$ , while K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) relies on pairing operation.



**Figure 6.1** Encryption Performance



**Figure 6.2** Decryption Performance



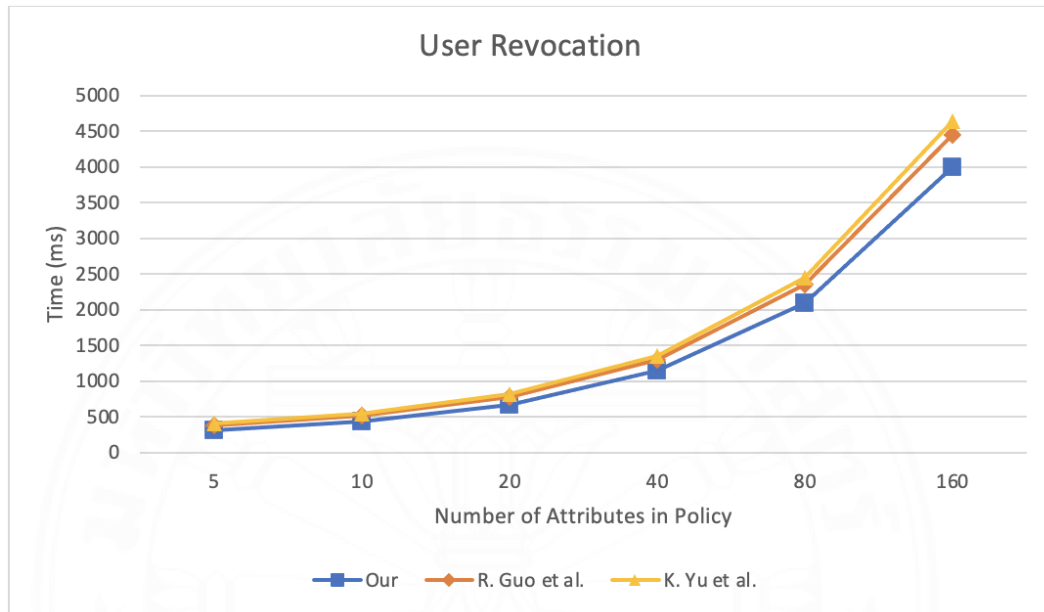
### 6.5.2 Revocation Performance

To evaluate the revocation performance, we measured the revocation time by varying the size of the access policy used for re-encryption and/or ciphertext update. The experiments were done to measure the processing time used for user revocation between our scheme, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) scheme, and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) scheme. For the attribute revocation, we measured the revocation time between our scheme and Y. Jiang, X. Xu and F. Xiao (2022) scheme.

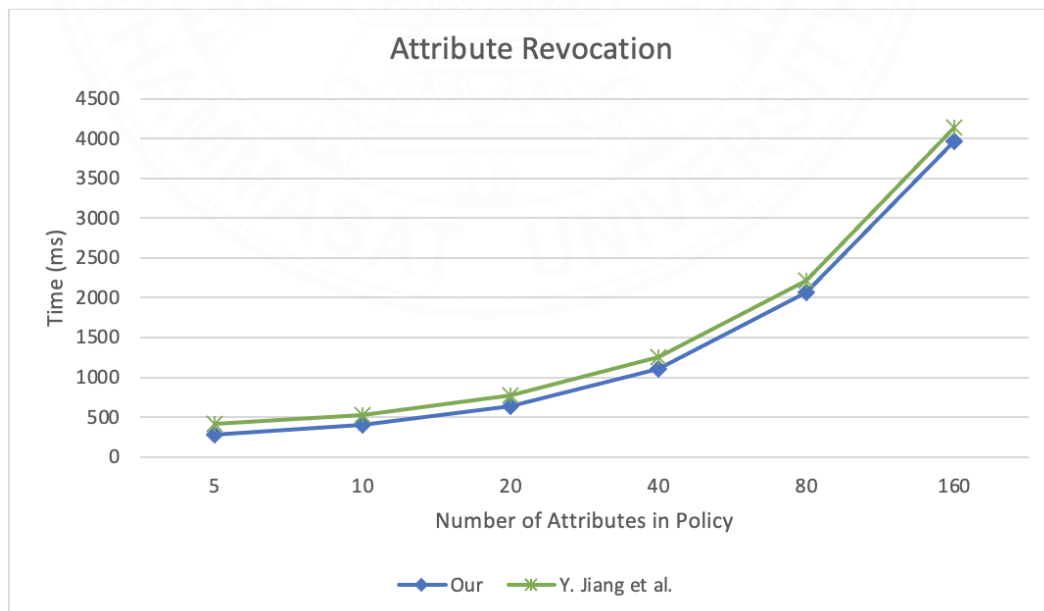
For the user revocation case, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) require the transformation of the ciphertext based on the policy update. The operations related to exponential and bilinear pairing operation of CP-ABE and user key transformation generation were major overheads. In our scheme, the number of attributes that need to be updated is subject to an exponential with constant pairing and AES encryption. In R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021), the proxy needed to perform partial CP-ABE decryption for each user in the system to update the ciphertexts to suit each user's attributes fully. Moreover, transform them into smaller ciphertext. Then, the user uses their secret key to complete the decryption. Their decryption processes were subject to an exponential pairing of both user attributes and policy attributes. As shown in Figure 6.3, our scheme delivered the processing time used for the user revocation on par with schemes R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021).

In the attribute revocation case, Y. Jiang, X. Xu and F. Xiao (2022) relied on the policy update protocol, reducing the cost of re-encryption of the entire ciphertexts. However, the proxy also needs to perform partial decryption for each user whose attributes were removed. As shown in Figure 6.4, the performance of Y. Jiang, X. Xu and F. Xiao (2022) contains both policy update protocol and partial decryption, which are used to update the revocation attributes. Our scheme only requires the re-encryption

of the AES key, which is much smaller than the ciphertext produced from the data encryption.



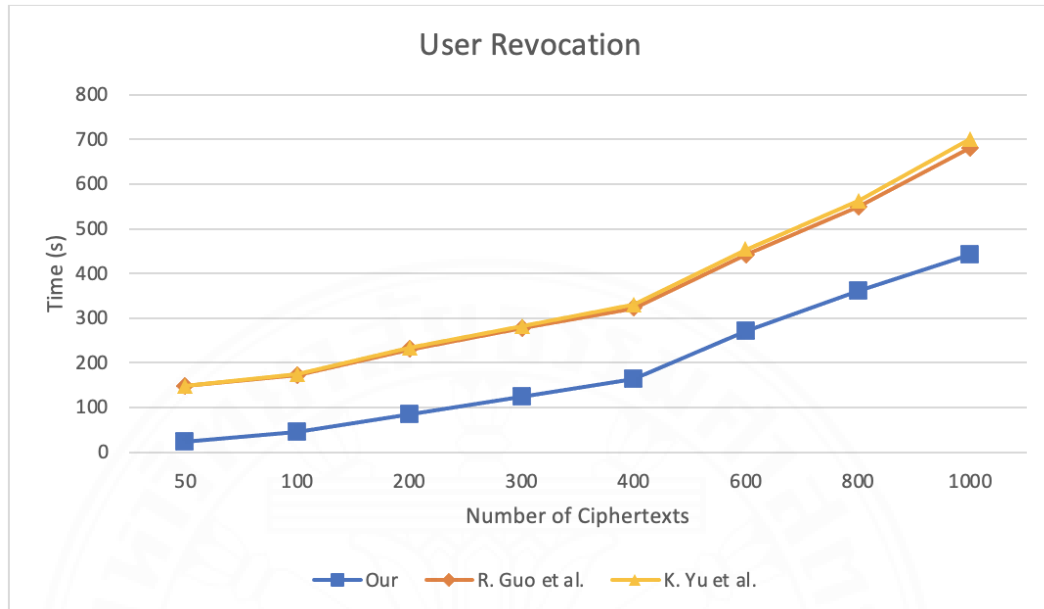
**Figure 6.3** User Revocation Performance based on number of attributes in policy



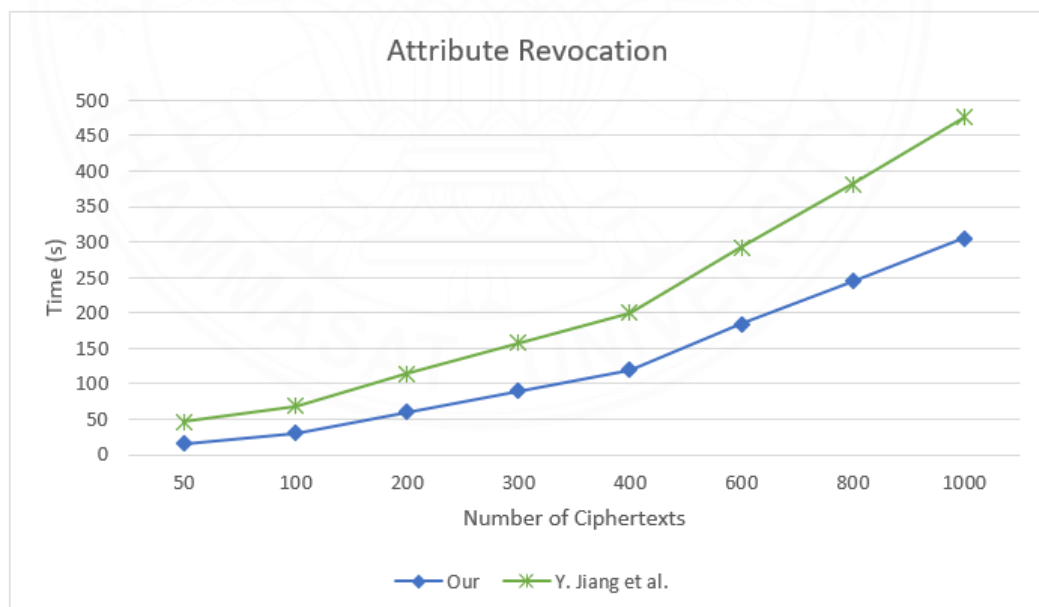
**Figure 6.4** Attribute Revocation Performance based on number of attributes in policy

In addition to measuring the performance based on the number of attributes, we conducted experiments to measure the user revocation and attribute revocation time based on the number of ciphertexts that need to be re-encrypted when there is a case of revocation. The experiments were done to measure the processing time used for user revocation between our scheme, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) scheme, and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) scheme. For the attribute revocation, we compared the performance between our scheme and Y. Jiang, X. Xu and F. Xiao (2022) scheme. In the experiments, the access policy size containing five attributes was used. To measure the revocation time, we took the ciphertexts query time into the final revocation cost. The total number of ciphertexts in the experiment setting varied up to 1000 ciphertexts. Figure 6.5 and Figure 6.6 represent the total processing time used to re-encrypt the ciphertexts for user and attribute revocation, respectively.

When any user is revoked from the system, we need to ensure that the revoked user cannot access each ciphertext in the system that contains the user-revoked attributes. As shown in Figure 6.5, our scheme took less time to complete the user revocation process. R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) yielded the same performance since they had the common process for policy update and partial decryption of the ciphertexts affected by the policy update. The cost of searching for the affected also contributed to the significant overhead of the overall processing time. In our scheme, the re-encryption cost was mainly subject to the cost of AES re-encryption and AES key re-encryption, which yields a relatively small computation cost compared to ciphertext re-encryption. Significantly, the cost of retrieving the affected ciphertexts was optimized based on our proposed ciphertext attributes and user mapping mechanism. When the number of ciphertexts was increased, the efficiency of our proposed mechanism obviously outperformed the related works.



**Figure 6.5** User Revocation Performance based on number of ciphertexts



**Figure 6.6** Attribute Revocation Performance based on number of ciphertexts

For attribute revocation cases, it generally took less cost than the user revocation because only ciphertexts affected with the revoked attribute will be re-encrypted. As shown in Figure 6.6, our scheme experienced a lower execution time than Y. Jiang, X.

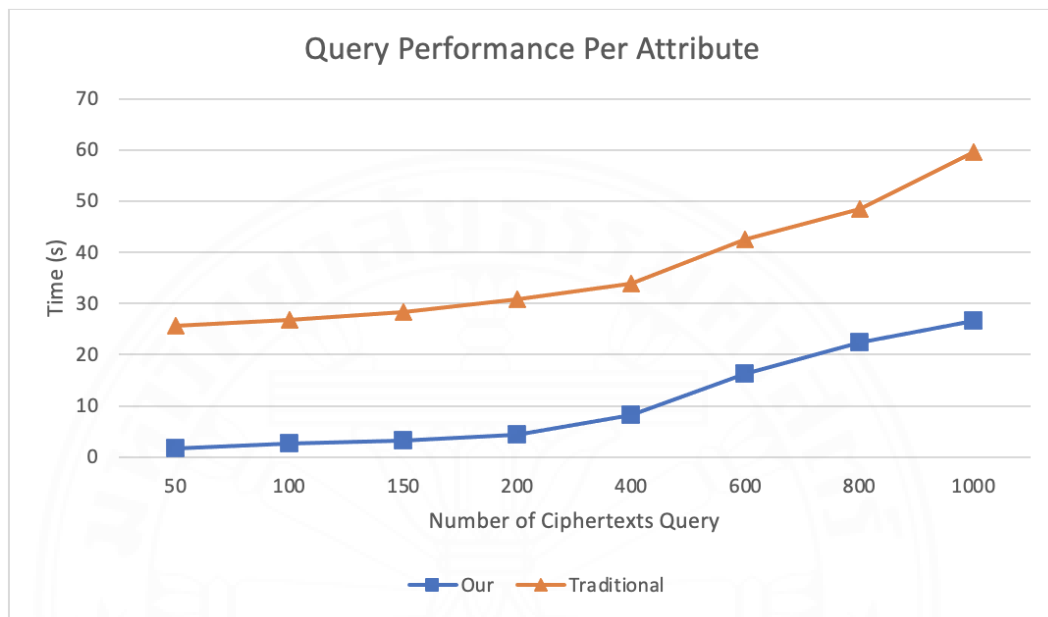
Xu and F. Xiao (2022). The cost of our attribute revocation is similar to the user revocation since our scheme only required the re-encryption of the AES key. Furthermore, our attributes mapping scheme enabled faster retrieval of affected ciphertexts without checking all the ciphertexts in the system. At the same time, Y. Jiang, X. Xu and F. Xiao (2022) contains policy update protocol, partial decryption, and traditional ciphertext query on the ciphertexts, which yield more overhead than ours.

### 6.5.3 Ciphertext Query and Revocation performance

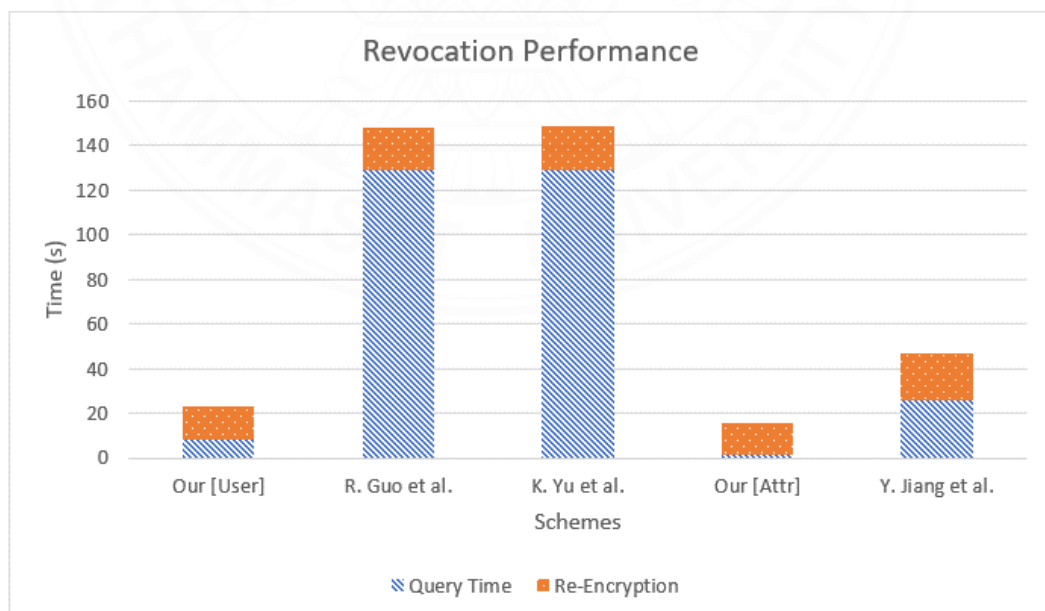
In Figure 6.7, we experimented with ciphertexts querying performance based on the number of ciphertexts in the system. The query is conducted via a smart contract that operates on the blockchain system. The query performance in Figure 6.7 is the average query time per attribute for all affected ciphertexts that require revocation. The traditional query, which is typically based on an exhaustive search, requires checking all ciphertexts where the revoked attribute resides. In our case, our attribute mapping scheme on the blockchain system minimizes the cost of checking all individual ciphertexts. Instead, it directly retrieves the affected ciphertexts based on the execution of the smart contract working over the index source retained in the blockchain mapped to the attributes. As a result, our proposed scheme significantly improves the ciphertext retrieval required in the re-encryption step.

In addition to querying performance results, we provided a detailed revocation cost analysis by measuring the re-encryption time and ciphertext query time for the given revocation case. In this experiment, we fixed the number of attributes in the policy and user's secret key attributes at five, and 50 ciphertexts need to be re-encrypted out of 1,000 ciphertexts in the system. The querying process was done via a smart contract on the blockchain for both the proposed scheme and the traditional query. Figure 6.8 exhibits the performance result of the ciphertext query and re-encryption cost of the user revocation between our proposed scheme, R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021) scheme, and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y.

Jararweh (2021) scheme, and the attribute revocation cost between our scheme and Y. Jiang, X. Xu and F. Xiao (2022) scheme.



**Figure 6.7** Query Performance Per Attribute



**Figure 6.8** Revocation Performance with Query Time

As shown in Figure 6.8, the re-encryption performance of each scheme was not quite much different because the re-encryption process of R. Guo, G. Yang, H. Shi, Y. Zhang and D. Zheng (2021), Y. Jiang, X. Xu and F. Xiao (2022), and K. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh (2021) requires a policy update protocol to update each ciphertext and partial ciphertext decryption to enable all non-revoked users to get their key updated. In our scheme, we need to perform new AES key generation and re-encryption to fully update the affected ciphertexts when user revocation occurs. For the attribute revocation case, our scheme only requires symmetric key re-encryption with a new policy. As for the ciphertext query performance, our scheme significantly outperforms other works for both user and attribute revocation. For the user revocation case, the ciphertext query time was indicated by the amount of the user's secret key attributes. This is because when the user is revoked, we need to ensure that each ciphertext in the system that contains the user-revoked attributes cannot be accessed by the revoked user. In this experiment, the user's secret key contains five attributes. Thus, retrieving the affected ciphertexts must query five attributes based on the number of attributes the revoked user's secret key contains. This is to mitigate the error of not retrieving all the ciphertexts that the revoked user can access directly from IPFS if any revoked users hold the symmetric key for accessing the files. For attribute revocation cases, the cost was also subject to the number of attributes revoked. This is because our proposed attribute mapping mechanism and the ciphertext query function reduce the query time on the blockchain system. Our proposed scheme can retrieve the affected ciphertext without checking all the ciphertexts in the system. In contrast, all related works relied on exhaustive searches over all ciphertext and retrieved the affected ones. As a result, the overall performance of our scheme yields the least execution time than the others.

## **CHAPTER 7**

### **CONCLUSION AND FUTURE WORK**

In this thesis, we have proposed a revocable CP-ABE protocol supporting efficient ciphertexts querying and proxy re-encryption technique with the policy hiding capability. In cloud computing, we integrated a blockchain system to empower the decentralized access control functions such as user enrollment, user authentication, cryptographic elements retention, and ciphertext indexing. Specifically, we proposed a ciphertext querying method by leveraging the smart contract to efficiently minimize the cost of ciphertext re-encryption when there is a revocation case. Moreover, most of the revocation process is offloaded to the proxy server to reduce the execution and communication cost of DO when the revocation occurs. Finally, we conducted a comparative analysis to display the functionality, and computation cost, and to conduct the experiments to measure the performance of our scheme and related works. Based on the comprehensive analysis and performance evaluation, our scheme has been proven for its novelty and practicality of the proposed access control protocol and the revocation algorithms.

For future works, we will tackle the proxy signcryption approach for blockchain-based cloud systems to enable anonymous authentication and data integrity validation. In addition, we will investigate the searchable encryption algorithms and design the mechanism that integrates with blockchain to support the efficient search over a collection of encrypted documents or files in the IPFS.



## REFERENCES

- Anand, D., Khemchandani, V., & Sharma, R. (2013). Identity-Based Cryptography Techniques and Applications (A Review). *Proceedings - 5th International Conference on Computational Intelligence and Communication Networks*. CICN 2013, 343-348. doi: 10.1109/CICN.2013.78.
- Announcing the ADVANCE ENCRYPTION STANDARD (AES). (2001). *Federal Information Processing Standards Publication 197*. United States National Institute of Standards and Technology (NIST).
- Bethencourt, J., Sahai, A., & Waters, B. (2007). Ciphertext-policy attribute-based encryption. *Proc. of IEEE Symposium on Security and Privacy*, 321–334.
- Dann en, C. (2017). *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginner*, New York, NY, USA, Apress.
- De Caro, A., & Iovino, V. (2011). jPBC: Java pairing based cryptography, *2011 IEEE Symposium on Computers and Communications (ISCC)* (pp. 850-855). doi: 10.1109/ISCC.2011.5983948.
- Fugkeaw, S., & Sato, S. (2017). Achieving Scalable and Optimized Attribute Revocation in Cloud Computing. *IEICE Transactions on Information and Systems, E100.D(5)*, 973-983.
- Fugkeaw, S. (2022). Enabling Trust and Privacy-Preserving e-KYC System Using Blockchain. *IEEE Access*, *10*, 49028-49039. doi: 10.1109/ACCESS.2022.3172973.
- Ganache Truffle Suite (2023, Jan 27). *Personal Blockchain Environment* [Online]. Retrieved from <https://trufflesuite.com/ganache/>
- Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2016). Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data. *Proceedings of CCS'06*, Alexandria, Virginia, USA.

- Guo, R., Yang, G., Shi, H., Zhang, Y., & Zheng, D. (2021). O3-R-CP-ABE: An Efficient and Revocable Attribute-Based Encryption Scheme in the Cloud-Assisted IoMT System. *IEEE Internet of Things Journal*, 8(11), 8949-8963. doi: 10.1109/JIOT.2021.3055541.
- Guo, L., Yang, X., & Yau, W. -C. (2021). TABE-DAC: Efficient Traceable Attribute-Based Encryption Scheme With Dynamic Access Control Based on Blockchain. *IEEE Access*, 9, 8479-8490. doi: 10.1109/ACCESS.2021.3049549.
- Hu, V., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandling, K., Miller, R., & Scarfone, K. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *NIST Special Publication*, SP 800-162.
- Jemel, M., & Serhrouchni, A. (2017). Decentralized Access Control Mechanism with Temporal Dimension Based on Blockchain. *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, 177-182. doi: 10.1109/ICEBE.2017.35
- Jiang, Y., Xu, X., & Xiao, F. (2022). Attribute-based Encryption with Blockchain Protection Scheme for Electronic Health Records, *IEEE Transactions on Network and Service Management*. doi: 10.1109/TNSM.2022.3193707.
- Kumar, R., Palanisamy, B., & Sural, S. (2021). BEAAS: Blockchain Enabled Attribute-Based Access Control as a Service. *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney, Australia. doi: 10.1109/ICBC51069.2021.9461151.
- Liang, X., An, N., Li, D., Zhang, Q., & Wang, R. (2022). A Blockchain and ABAC Based Data Access Control Scheme in Smart Grid. *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*, Huaihua City, China. doi: 10.1109/ICBCTIS55569.2022.00023.
- Liang, X., Zhao, J., Shetty, S., Liu, J., & Li, D. (2017). Integrating blockchain for data sharing and collaboration in mobile healthcare applications. *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 1-5. doi: 10.1109/PIMRC.2017.8292361.

- Liu, H., Luo, X., Liu, H., & Xia, X. (2021). Merkle Tree: A Fundamental Component of Blockchains. *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, Changchun, China. doi: 10.1109/EIECS53707.2021.9588047.
- Liu, X., Zheng, Y., & Li, X. (2021). A revocable attribute-based access control system using blockchain. *Journal of Physics: Conference Series*.
- Malik, S., Dedeoglu, V., Kanhere, S. S., & Jurdak, R. (2019). TrustChain: Trust Management in Blockchain and IoT Supported Supply Chains. *2019 IEEE International Conference on Blockchain (Blockchain)*, 184-193. doi: 10.1109/Blockchain.2019.00032.
- Maiti, S., & Misra, S. (2020), P2B: Privacy Preserving Identity-Based Broadcast Proxy Re-Encryption. *IEEE Transactions on Vehicular Technology*, 69(5), 5610-5617. doi: 10.1109/TVT.2020.2982422.
- Meelu, P., & Malik, S. (2010). RSA and its Correctness through Modular Arithmetic. *AIP Conf. Proc.*, 1324, 463–466. doi: 10.1063/1.3526259.
- Nakamoto, S. (2009). *Bitcoin: A Peer-to-Peer Electronic Cash System*.
- Nguyen, D. C., Pathirana, P. N., Ding, M., & Seneviratne, A. (2019). Blockchain for Secure EHRs Sharing of Mobile Cloud Based E-Health Systems. *IEEE Access*, 7, 66792-66806. doi: 10.1109/ACCESS.2019.2917555.
- Packetizer (2023, Jan 19). *AES application and Source code page* [Online]. Retrieved from <https://www.aescrypt.com/download/>
- PBC library. (2022, October 14). *Pairing-Based Cryptography* [Online]. Retrieved from <https://crypto.stanford.edu/pbc/>
- Park, J. H. (2011). Efficient Hidden Vector Encryption for Conjunctive Queries on Encrypted Data. *IEEE Transactions on Knowledge and Data Engineering*, 23(10), 1483-1497. doi: 10.1109/TKDE.2010.206.

- Rouhani, S., Belchior, R., Cruz, R., & Deters, R. (2021). Distributed Attribute-Based Access Control System Using a Permissioned Blockchain. *World Wide Web* 24, 1617–1644. doi: 10.1007/s11280-021-00874-7.
- Sethia, D., Shakya, A., Aggarwal, R., & Bhayana, S. (2019). Constant Size CP-ABE with Scalable Revocation for Resource-Constrained IoT Devices. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 0951-0957). doi: 10.1109/UEMCON47517.2019.8993103.
- Shamir, A. (1985). Identity-Based Cryptosystems and Signature Schemes. In G.R. Blakley & D. Chaum (Eds.), *Advances in Cryptology: CRYPTO 1984 (Lecture Notes in Computer Science, 196, 47-53)*. doi: 10.1007/3-540-39568-7\_5.
- Su, Z., Wang, H., Wang, H., & Shi, X. (2020). A Financial data security sharing solution based on blockchain technology and proxy re-encryption technology. *2020 IEEE 3rd International Conference of Safe Production and Informatization (IICSPI)*, 462-465. doi: 10.1109/IICSPI51290.2020.9332363.
- Szabo, N. (1997). The idea of smart contracts. *Nick Szabos Papers and Concise Tutorials*, 6.
- Veen, D. V. (2011) Secure searching through encrypted data - Creating an efficient Hidden Vector Encryption construction using Inner Product Encryption. *Master's Thesis, University of Twente*. <https://essay.utwente.nl/61033/>.
- Wang, X., Chi, Y., & Zhang, Y. (2020). Traceable Ciphertext Policy Attribute-based Encryption Scheme with User Revocation for Cloud Storage. *2020 International Conference on Computer Engineering and Application (ICCEA)*, Guangzhou, China. doi: 10.1109/ICCEA50009.2020.00026.
- Wang, S., Wang, X., & Zhang, Y. (2019). *A Secure Cloud Storage Framework With Access Control Based on Blockchain*. *IEEE Access*, 7, 112713-112 725. doi: 10.1109/ACCESS.2019.2929205.
- Wang, S., Zhang, Y., & Zhang, Y. (2018). A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. *IEEE Access*, 6, 38437-38450. doi: 10.1109/ACCESS.2018.2851611.

- Wang, X., Zhou, Z., Luo, X., Xu, Y., Bai, Y., & Luo, F. (2021). A Blockchain-Based Fine-Grained Access Data Control Scheme With Attribute Change Function. *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, Atlanta, GA, USA. doi: 10.1109/SWC50871.2021.00054.
- Wu, H., Li, L., Paik, H. -y., & Kanhere, S. S. (2021). MB-EHR: A Multilayer Blockchain-based EHR. *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 1-3. doi: 10.1109/ICBC51069.2021.9461075.
- Xu, M., Chen, X. & Kou, G. (2019). A systematic review of blockchain. *Financ Innov* 5, 27. doi: 10.1186/s40854-019-0147-z.
- Ying, Z., Wei, L., Li, Q., Liu, X., & Cui, J. (2018). A Lightweight Policy Preserving EHR Sharing Scheme in the Cloud. *IEEE Access*, 6, 53698-53708. doi: 10.1109/ACCESS.2018.2871170.
- Yu, K., Tan, L., Aloqaily, M., Yang, H., & Jararweh, Y. (2021). Blockchain-Enhanced Data Sharing With Traceable and Direct Revocation in IIoT. *IEEE Transactions on Industrial Informatics*, 17(11), 7669-7678. doi: 10.1109/TII.2021.3049141.
- Zhang, Z., Zhang, J., Yuan, Y., & Li, Z. (2022). An Expressive Fully Policy-Hidden Ciphertext Policy Attribute-Based Encryption Scheme With Credible Verification Based on Blockchain. *IEEE Internet of Things Journal*, 9(11), 8681-8692. doi: 10.1109/JIOT.2021.3117378.
- Zheng, W., Zheng, Z., Chen, X., Dai, K., Li, P., & Chen, R. (2019). NutBaaS: A Blockchain-as-a-Service Platform. *IEEE Access*, 7, 134422-134433. doi: 10.1109/ACCESS.2019.2941905.

## BIOGRAPHY

Name Khanadech Worapaluk

Education 2021: Bachelor of Engineering (Computer Engineering)  
Sirindhorn International Institute of Technology  
Thammasat University

Publication  
Worapaluk, K. & Fugkeaw, S. (2023). An Efficiently Revocable Cloud-based Access Control Using Proxy Re-encryption and Blockchain. *2023 20th International Joint Conference on Computer Science and Software Engineering (JCSSE), Phitsanulok, Thailand* (pp. 178-183). doi: 10.1109/JCSSE58229.2023.10202130.