Spring 2021

Secure data sharing in cloud and IoT by leveraging attribute-based encryption and blockchain

MD Azharul Islam

Department: Computer Science

SECURE DATA SHARING IN CLOUD AND IOT BY LEVERAGING ATTRIBUTE-BASED ENCRYPTION AND BLOCKCHAIN

by

MD AZHARUL ISLAM

A DISSERTATION

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2021

Approved by

PUBLICATION DISSERTATION OPTION

This dissertation consists of the following four articles which have been either published or submitted for publication as follows:

Paper I: Pages 28 - 61, "A collusion-resistant revocable attribute-based encryption scheme for secure data sharing in cloud", was published in 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, Australia.

Paper II: Pages 62 - 106, "Attribute-based encryption scheme for secure data sharing in cloud with fine-grained revocation", has been submitted to IEEE Transactions on Cloud Computing.

Paper III: Pages 107 - 161, "Attribute-Based Encryption Scheme for Secure Multi-group Data Sharing in Cloud", has been accepted for publication in an upcoming issue of IEEE Transactions on Services Computing.

Paper IV: Pages 162 - 189, "A Permissioned Blockchain-based Access Control System for IoT", was published in 2019 IEEE International Conference on Blockchain (Blockchain), Georgia, Atlanta.

ABSTRACT

Data sharing is very important to enable different types of cloud and IoT-based services. For example, organizations migrate their data to the cloud and share it with employees and customers in order to enjoy better fault-tolerance, high-availability, and scalability offered by the cloud. Wearable devices such as smart watch share user's activity, location, and health data (e.g., heart rate, ECG) with the service provider for smart analytic. However, data can be sensitive, and the cloud and IoT service providers cannot be fully trusted with maintaining the security, privacy, and confidentiality of the data. Hence, new schemes and protocols are required to enable secure data sharing in the cloud and IoT. This work outlines our research contribution towards secure data sharing in the cloud and IoT. For secure data sharing in the cloud, this work proposes several novel attribute-based encryption schemes. The core contributions to this end are efficient revocation, prevention of collusion attacks, and multi-group support. On the other hand, for secure data sharing in IoT, a permissioned blockchain-based access control system has been proposed. The system can be used to enforce fine-grained access control on IoT data where the access control decision is made by the blockchain-based on the consensus of the participating nodes.

ACKNOWLEDGMENTS

All praise to almighty ALLAH for giving me the strength to stay put in this long journey. I want to take a moment to thank all the important individuals who helped me along the way.

First, I would love to thank my PhD supervisor, Dr. Sanjay Madria, for his constant guidance and support to help me achieve my goal. Also, my sincere gratitude goes to the rest of my Ph.D. advisory committee members: Dr. Luo, Dr. Siddhardh, Dr. Taylor, and Dr. Zawodniok for taking the interest in my work and serving in my dissertation committee. I would like to thank Professor Price, the Department of Computer Science, ISC, and NSF for supporting my graduate studies and research through GRA and GTA.

I want to thank my friends: Saif, Jisan, Junior, and Noman; my labmates: Shudip, Yasin, Xiaofei, San, Ayon; the entire Bangladesh community here in Rolla; and some particular individuals like Arefin, Ashik, Rysul, Galib. Thank you for all the fun and good memories. I cannot list all the names here, but know that you are always on my mind.

To my parents, I express my heartfelt gratefulness for your sacrifice, and unconditional love, support, and care. My sister, Lucky, and her husband, my brother, Pavel, and his wife, my little brother, Pallab, my nephew Adiyat, deserve my wholehearted thanks as well.

Finally, to my wife, Brinty: your love and understanding helped me through the dark times. Without you believing in me, I never would have made it. It is time to celebrate; you earned this degree right along with me.

I dedicate this dissertation to my parents for their constant support, prayers, and unconditional love.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

Data sharing has become a very common practice amongst individuals, educational organizations, scientific communities, and medical industry. Industries and financial institutions have to heavily rely on data sharing within the organization as well as outside the organization. For example, banks need to share documents internally among the employees as well as externally with the account holders. Educational institutions share data with students, faculties and staffs. Scientific communities share data among themselves for collaboration and knowledge sharing purposes. Medical industry also shares data among patients, different healthcare professionals, pharmacies, and insurance payers for better interoperability. People are constantly sharing their photos, videos, life-events, stories, views, opinions, etc. in different social media platforms such as Facebook, Instagram, Youtube, Vine, TikTok, and Twitter to stay connected with each other regardless of their geographical distance. Moreover, with the recent wide adoption of IoT devices, knowingly or unknowingly people are sharing data everyday with IoT service providers. For example, smart wearables like apple watch and fitbit send activity data to the service provider's server to perform analytic on the data. Smart home assistant like Google home or Amazon Alexa continuously exchange data with the respective service provider.

Cloud and IoT service provider cannot be fully trusted with the sensitive user data because they are prone to insider and outsider attacks. For example, sensitive information (including date of birth and SSN) of nearly 143 million users was allegedly compromised due to the 2017 Equifax data breach [1]. Moreover, they can misuse data or sell it to third-party for financial benefit. Such unwanted incidents can be avoided and the benefits of cloud and IoT can be enjoyed to their full potential by designing secure data sharing schemes.

1.1. SECURE DATA SHARING IN CLOUD

A natural way of achieving secure data sharing in cloud is to encrypt the data before outsourcing it to the cloud. While encryption preserves the data confidentiality in the cloud, sharing the encrypted data brings a new set of challenges. The most fundamental challenge of sharing encrypted data is how to distribute the decryption keys among the users. There are two extreme solutions to this problem discussed as follows:

- The first approach is to encrypt all data with the same key and share the common decryption key with everyone. This approach makes the key distribution process much simpler and it also requires the lowest storage and communication cost since only single encrypted file is created. However, it has the highest security vulnerability as everything is exposed if the shared decryption key is compromised from anyone.
- In the second extreme approach, for each file, the data owner would choose different keys for different users and create a different encrypted copy for each key. While this approach achieves the highest level of security, the storage and communication cost grows linearly with the number of files and number of total users in the system.

While neither of the solutions discussed above seem practical, researchers across the world have worked relentlessly to come up with better solutions. Boneh et al. [1] have proposed a RSA-based [2] solution where users are grouped together according to their privilege, and each group is assigned with a RSA public-private key pair. The file is encrypted using symmetric encryption algorithms (e.g., AES) and a decryption token is created for each group that is allowed to access the file by encrypting the symmetric key

[1]http://money.cnn.com/2017/09/07/technology/business/equifax-data-breach/index.html

with the group's RSA public key. The decryption tokens are added with the encrypted file as a header and uploaded in the cloud. Later, a legitimate user who belongs to one of the groups present in the header, can download the encrypted file and decrypt it with the corresponding RSA private key. Kim et al. proposed a secure data sharing scheme based on identity-based encryption (IBE) [3] in [4] where files are encrypted with a user's unique identity such as email address and shared through cloud. Schemes [5, 6] proposed a secure cloud-based data sharing scheme based on broadcast encryption scheme where secret keys are updated on a periodic basis so that a user with the most updated key can only decrypt. Scheme proposed in [7] is based on Diffie-Hellman protocol. While some schemes being better than the others, all schemes share a common limitation of not offering fine-grained access control. This limitation has been overcome by Shahai et al. when they proposed attribute-based encryption (ABE) scheme in [8]. We discuss in the following different aspects of ABE related to secure data sharing in cloud.

1.1.1. Attribute-Based Encryption (ABE). In ABE, the access policy is expressed in terms of a set $\mathbb{A}$ of descriptive attributes, and decryption is possible by another set $\mathbb{B}$ if attributes in $\mathbb{B}$ satisfy the access policy. The trusted entity known as attribute authority, generates the system parameters and public key; and assigns decryption keys to the user. After the proposal of the original ABE, two main variants of ABE have been proposed:

1. Key Policy Attribute-Based Encryption (KP-ABE): In KP-ABE, the access policy composed of attribute set $\mathbb{A}$ is associated with the user's decryption key and the attribute set $\mathbb{B}$ is associated with the ciphertext.

2. Ciphertext Policy Attribute-Based Encryption (CP-ABE): In CP-ABE, the access policy composed of attribute set $\mathbb{A}$ is associated with the ciphertext and the attribute set $\mathbb{B}$ is associated with the user's decryption key.

Since the access policy is associated with the decryption key and the attribute authority is responsible for generating the decryption keys, attribute authority has more control on which keys can decrypt a particular ciphertxt in KP-ABE [9, 10, 11, 12, 13, 14]. On the other hand, the data owner has more control on which keys can decrypt a particular ciphertext in CP-ABE [15, 16, 17, 18, 19] since the access policy in this case is associated with the ciphertext and the data owner gets to choose the access policy during encryption.

1.1.2. Revocation in ABE. ABE has emerged as a very promising cryptographic tool for secure data sharing in cloud because it is a highly expressive encryption scheme and it allows fine-grained access control that is required for secure data sharing in cloud [20, 21, 22, 23, 24, 25]. However, one of the main challenging issue with ABE is the key revocation problem. Key revocation is a very important security feature for any secure data sharing scheme because it allows one to selectively revoke the decryption ability of users after secret keys have been assigned. Typically, when data is shared with a group of users, some existing group members may leave the group or new members may join in. To support such dynamic changes in the data sharing group, revocation is necessary. Revocation is particularly challenging in ABE because the same attribute can be shared among multiple users and revoking one user's attribute key may affect the attribute key of another non-revoked user who has the same attribute. In the context of ABE, there can be two types of revocations: user-level revocation and attribute-level revocation. User-level revocation gives the ability to revoke a user entirely, while attribute-level revocation allows to revoke particular attribute(s) from a user.

Initially proposed ABE schemes such as [8, 10, 11, 12, 15, 16, 17] do not support revocation. As a result, it was not possible to use them as a practical tool for secure data sharing in cloud where revocation was necessary. There are two possible ways of achieving revocation property in ABE: indirect revocation [26, 22, 27] and direct revocation [28, 29, 30, 31, 32, 33, 34]. A revocation list is maintained in both methods that specifies all the revoked users. In indirect revocation, the attribute authority (the trusted party responsible for creating and distributing attribute keys) has to perform a periodic update of attribute keys according to the revocation list, and send them out to every nonrevoked user. On the other hand, in direct revocation, the owner or the encryptor directly embeds the revocation list in the ciphertext so that only nonrevoked users can decrypt. ABE schemes that relies on indirect revocation have much higher communication and runtime overhead since attribute authority periodically has to do system wide key update in indirect revocation. Though directly revocable ABE schemes are more efficient, they are vulnerable a serious security threat called collusion attack as pointed out in [34]. The collusion attack in ABE is discussed in the following section.

1.1.3. Collusion Attacks. In a collusion attack, multiple malicious users, who individually do not have the ability to decrypt a particular ciphertext, combines their keys together to successfully decrypt the ciphertext. To better understand the collusion attack, let us consider the following example. Suppose CryptoFlix is a Netflix-like streaming service that keeps all its media contents in a public cloud run by a third-party cloud service provider. Before outsourcing media files to the public cloud, it encrypts them using attribute-based encryption scheme. CryptoFlix offers two types of subscription plans: basic and premium. Basic and premium plans allow a subscriber to choose three and five attributes, respectively. Let us assume that two users, Alice and Bob, have a subscription for the basic plan, and another user, Eve has a subscription for the premium plan. Alice loves science fiction movies and documentaries, she gets decryption keys for attribute *movie, scifi,* and *documentary* from the attribute authority (a trusted entity responsible for generating and distributing attribute secret keys). Conversely, Bob loves tv shows and anime, and gets decryption keys for *new_release, tv_show,* and *anime* attributes. Let Eve has decryption keys for attribute *movie, tv_show, documentary, scifi,* and *new_release*. Attribute-level revocation would allow CryptoFlix to downgrade Eve's subscription plan from premium to basic by revoking attribute *movie* and *scifi* such that Eve can decrypt newly released documentary (encrypted under policy *new_release AND documentary*) but not newly released science fiction movie (encrypted under the policy *new_release AND movie AND scifi*). On the other hand, user-level revocation only allows CryptoFlix to revoke Eve entirely but not a subset of attributes from her. So, the user-level revocation is more suitable when Eve's subscription plan expires such that she cannot decrypt either newly released documentary or newly released science fiction movie.

Let us assume that Eve's premium subscription plan recently expired or she downgraded to the basic plan by unsubscribing attribute *movie* and *scifi*. In either case, when CryptoFlix encrypts the newly released science fiction movie XFiction under the policy *(new_release AND movie AND scifi)* and uploads it to a public cloud, Eve cannot decrypt it. Note that the other two users cannot decrypt XFiction either. However, the users can cooperate with each other and try to decrypt it by launching the following collusion attacks:

- *Type I attack:* Multiple users who individually do not have enough attributes to satisfy a policy cooperate with each other so that their collective attribute keys may satisfy the policy.

- *Type II attack:* A revoked user who cannot decrypt a file despite having enough attributes to satisfy the policy cooperates with a nonrevoked user to restore his or her decryption ability in order to decrypt the file.

In a *type I* attack, Alice and Bob would combine their attribute keys and try to decrypt XFiction. On the otherhand, in a *type II* attack, Eve would combine her attribute keys with a nonrevoked user (such as Bob) and try to decrypt XFiction. Not to mention CryptoFlix faces financial loss if any of the attacks becomes successful. Therefore, CryptoFlix must be resilient to both type of collusion attacks.

ABE schemes such as [8, 10, 11, 12, 15, 16, 17] are resistant to *type I* collusion attacks. However, *type II* collusion attacks do not apply to them as those schemes do not support revocation. Hur et al. proposed a solution to the revocation problem in [35, 36, 29]. The proposed solution is based on the idea of attribute group. The user's secret key consists of two parts. One is associated with the user's attributes, and the other is associated with the attribute group. They are called decryption secret key (DSK) and key encryption key (KEK), respectively. The revocation is dictated by KEK and is independent of the user's DSK. Consequently, the KEK of one user works with the DSK of another user. Hence, the proposed revocable ABE scheme is not resistant against *type II* collusion attacks as a revoked user by colluding with a nonrevoked user can get the valid KEK and restore his or her decryption ability. This vulnerability was first pointed out by Li et al. [34]. Schemes such as [31, 37] also have the same vulnerability since these solutions are also based on the same idea. Li et al. refined their initial solution [34] in [38]. To revoke a user from the attribute group, the attribute manager (AM) updates the existing user's KEK keys. They bind a user's DSK with his or her KEK so that the KEK of one user does not work with the DSK of another user. This ensures that a revoked user cannot collude with a nonrevoked user to restore his or her decryption right. However, the problem is that each time a user is revoked, all nonrevoked users' keys (KEK) are affected. This is because DSK and KEK keys are tied together by a common secret exponent that is only known to a semi-trusted party called the attribute manager (AM). This exponent is common across attribute secret keys of all the users. To revoke a user, this exponent needs to be updated in the secret keys for all the nonrevoked users. As a result, revocation of a single user requires the attribute manager to send new KEK secret keys to all non-revoked users so that they can update their KEKs. Transmitting secret keys to nonrevoked users after each revocation is very expensive and hence it is important to formulate a revocable ABE scheme that is resistant to both *type I* and *type II* collusion attacks and does not require transmitting secret keys to non-revoked users after any revocation.

1.1.4. Multi-group. An organization may have multiple internal groups and each group may have its own data sharing policy. One group may need to collaborate with another group which requires cross-group data sharing feature so that members from one group can share data with the members of another group. Moreover, a group can split or multiple groups can merge together. Further more, users can move between different groups. So a multi-group scenario introduces additional challenges to the already challenging problem of secure data sharing. Previously proposed ABE-based data sharing schemes such as [39, 40, 41, 42, 43] only address the challenges that are relevant to single group setting and fail to effectively solve challenges related to multi group setting as discussed earlier. However, many organizations that are potential users of cloud data sharing service are multi-group in nature. For example, in an university setting, there are multiple departments- each representing a different group. There could be dedicated groups for faculties, graduate students, undergraduate students, and administrative staffs. If the university decides to move its data to the cloud while preserving the security of the data from the cloud service provider, it will require a secure data sharing service that supports multi-group setting.

1.2. SECURE DATA SHARING IN IOT

The popularity of IoT device has sky rocketed more recently. Experts have predicted that there will be 250 billion devices connected to the internet worldwide by the end of year 2020 [44]. IoT has the potential to revolutionize manufacturing, healthcare, hospitality, and retail industry. The massive popularity and wide adoption of IoT devices have made the idea of smart home already a reality. If this trend continues, many believe that ideas like smart cities, once considered too futuristic, may not be too far ahead from present day. The main benefits of IoT are that they are low maintenance, easy to operate, mostly automated, remotely configurable, and the can save time and bring convenience by automating many tasks effectively and efficiently. For example, a smart home surveillance system may send alerts to the home owner in realtime if it detects the presence of an intruder while the owner is away. Then the owner can see the live video feed from the camera to verify it and notify the police or disable the alert in case of a false alarm. The owner may also timely check on his pet, interact with it or even feed it with smart feeder. Smart home assistant like Google home or Amazon Alexa can do a quick web search, play a particular video on youtube, set a timer or reminder, read news, schedule a meeting, make a phone call, adjust the room temperature, switch on or off lights, etc. with a simple voice command. The possibilities here are endless and the convenience it brings is unparalleled. However, as IoT is becoming more integral part of our life, the data generated by different IoT devices has become more personal and contains sensitive information in some cases. IoT data is often shared with the service provider. For example, wearable devices like applewatch periodically sends activity and healthcare data to the server that is used to build better machine learning models that can perform smarter analytic. Due to the sensitive nature of the data, secure sharing of the IoT data has become a significantly important issue.

1.2.1. Traditional Approaches. Secure data sharing in IoT has been addressed by the research community, and many interesting directions have been put forward [45, 46]. It is pointed out in [47] that secure protocols like TLS, DTLS, or even TinyTLS-like light-weight protocols may prove to be impractical for IoT because of its unique characteristics and constraints such as heterogeneity, lack of standardization, low computation and memory resource, etc. However, researchers have tried to work around it by designing new protocols and architectures. Majority of the solutions have a cryptography focus. For example, schemes such as [48, 49, 50] took a mix of both public and private key cryptography to solve this problem. Approaches like hardware-based ciphers has also been considered [51]. The focus of these approaches mostly centers around securing the communication channel between IoT device and service providers, protecting the confidentiality of the data, or ensuring user privacy. Moreover, traditional cryptography-based approaches rely on a centralized trusted entity such as certificate authority in PKI or attribute authority in ABE for the implementation of the security element. Also, traditional systems are not well designed for transparency, accountability, and dispute resolution. As a result, if the service provider misuses user's IoT data or the data falls into the wrong hand because of the service provider's mistake, it is hard to held them accountable. To this end, secure IoT data

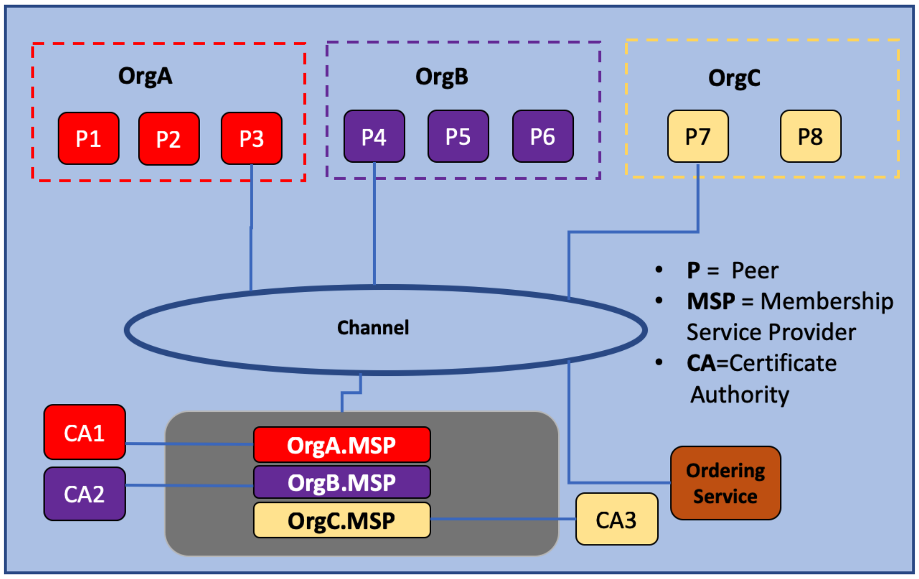


Figure 1.1. Hyperledger Fabric Architecture

sharing solutions not only need to ensure the user's privacy and data confidentiality, but also need to incorporate transparency, accountability, and dispute resolution in a distributed and trust-less fashion.

1.2.2. The Potential of Blockchain. Blockchain is a distributed immutable ledger maintained by a network of peers where all the peers in the network at any given point of time agrees on a single identical version of the ledger through some consensus protocol. The first practical application of blockchain was seen in the digital cryptocurrency called Bitcoin [52] where blockchain was used to publicly keep track of all the transactions and prevent double spending. Cryptocurrency like Ethereum [53] soon emerged to enhance the capability of blockchain by incorporating smartcontract - a program that lives and runs in the blockchain and can automate the asset (cryptocurrency) transfer according to the provided logic. Blockchain offers a great platform to build distributed applications for mutually untrusted parties by eliminating the need of a trusted central authority. Since data, once written on the blockchain cannot be deleted, it naturally enables transparency, accountability that can help in any kind of dispute resolution.

Depending on whether permission is required for a node to join the blockchain network, there are two types of blockchains: public or permissionless blockchain and private or permissioned blockchain. Popular examples of public blockchain are Bitcoin blockchain [52], Ethereum [53], etc. and examples of private blockchain are Hyperledger Fabric [54], Ripple (XDR)[55], etc. While public blockchain is well-suited for cryptocurrency, it has a scalability issue that limits the number of transactions the network can process referred to as blockchain bloat [56]. For example, bitcoin can process only a maximum of seven transactions per minute. This is due to the fact that the block creation frequency (1 block per 10 minutes) and size (1MB) is limited [52]. The security of the public blockchain relies on the proof of work (PoW) where all the peers in the network validate all the transactions and try to solve a computationally intensive cryptographic puzzle. The hardness of the puzzle is set so that a new block is created every 10 minutes. Due to the network latency, there exist multiple forks of the blockchain and it can take up to six hours to eventually reach a consensus. That is why transaction wait time is very high in pubic blockchain (sometimes up to six hours). Though, consensus protocols like proof of stake (PoS) are there, the transaction wait time is still high in public blockchain [54]. It makes public blockchain less feasible for any IoT application where faster transaction is required, e.g. IoT devices sending data to emergency response team. However, in the private or permissioned blockchain, the transactions are much faster. This is because it does not rely on PoW or PoS. Rather, it incorporates much faster consensus protocols like Byzantine Fault Tolerance (BFT) or proof of authority (PoA) and yet provides a way to secure the transactions among a group of participants with verified identities who have a common goal but do not fully trust each other [54]. The architecture of hyperledger fabric showing its important components depicted in Figure 1.1. To form a blockchain, different organizations (e.g., OrgA, OrgB, OrgC) come together and creates a private channel between them. Each organization has its own set of peers that runs the blockchain. The organizations also have their own membership service providers (MSP) responsible for creating and distributing membership credentials required to join the blockchain network. The membership service provider has certificate authority (CA) to generate the required cryptographic materials (e.g., certifiates, keys, etc). Finally, there is an ordering service, responsible for managing the temporal order of the transactions to be written in the blockchain. Because of the faster transaction rate, hyperledger fabric is more suitable for applications that demand low latency.

1.3. DISSERTATION SUMMARY

This dissertation is composed of three papers presented in publication format of the conference or the journal wherein they were published (or submitted to) addressing the aforementioned objectives in the previous sections.

Paper I titled "A collusion-resistant revocable attribute-based encryption scheme for secure data sharing in cloud" presents a novel revocable attribute-based encryption that is resilient against both *type-I* and *type-II* attacks. Besides being resilient to both types of collusion attacks, the novelty of our proposed ABE scheme is that it does not require any trusted entity (e.g., manager) to achieve revocation. Rather, it gives data owner the complete control on the revocation. The revocation does not affect the secret keys of non-revoked users. As a result, non-revoked users do not require to update their secret keys. However, this scheme supports revocation at user-level.

Paper II titled "Attribute-based encryption scheme for secure data sharing in cloud with fine-grained revocation" extends our scheme proposed in paper I, and proposes a new scheme that supports attribute-level revocation. Additionally, it inherits other properties from the previous scheme such as collusion resistance, revocation without the aid of any trusted entity, etc.

Paper III titled "Attribute-Based Encryption Scheme for Secure Multi-group Data Sharing in Cloud" proposes an attribute-based encryption scheme that is suitable for multi-group setting and supports multi-group operations such as group split, group merge, and cross-group data sharing. It is can also prevent *type-I* and *type-II* collusion attack discussed earlier. It reduces the decryption cost from the user end by outsourcing the expensive operations to the cloud. The scheme is based on a dual-cloud architecture and it is secure as long as both cloud do not collude with each other.

Paper IV titled "A Permissioned Blockchain-based Access Control System for IoT" puts forward an access control system for secure data sharing in IoT. It implements attribute-based access control (ABAC) by leveraging smartcontract of permissioned blockchain (hyperledger fabric). It enables the data owner to define access policy of their data and the blockchain ensures that the policy is enforced while data sharing. Since the policy is enforced in the blockchain in a distributed manner, the dispute resolution becomes much easier and transparent.

2. LITERATURE REVIEW

The scope of this dissertation primarily deals with secure data sharing in cloud and IoT. Likewise, we classify the related work into two different categories: 1) Secure data sharing in cloud and 2) Secure data sharing in IoT. For the first category, we discuss different cryptography-based approach to deal with the problem of secure data sharing in cloud. We also discuss their limitation in terms of fine-grained access control. Then we discuss how attribute-based encryption has revolutionized the field of secure data sharing in cloud. Later, we discuss different ABE-based protocols designed for secure data sharing in cloud along with their limitations. For the second category of related work, we discuss various existing approaches for secure data sharing in IoT. Then we discuss how blockchain has been leveraged towards secure data sharing in IoT.

2.1. SECURE DATA SHARING IN CLOUD

The idea of securing data in remote untrusted storage has been around well before the term "cloud computing" was introduced. Eventually, it has become a more urgent problem to solve as cloud computing has gained in more popularity [57]. Researchers from both industry and academia have actively tried to solve this problem that has led to various solution approaches [58]. The related work in this field can be divided into following sub-categories based on the underlying cryptographic primitives:

2.1.1. Schemes Based on Public Key Cryptography and Symmetric Key Cryptography. Boneh et al. proposed Sirius - a scheme for securing and sharing data in remote untrusted server in [1]. The proposed scheme has made use of RSA and AES encryption scheme as the main cryptographic tool. In the bootstrap process, users are divided into multiple groups based on their access privilege and all users in the same group are provided with the same RSA private (decryption) key. Then, the file is symmetrically encrypted with an AES key. Later, decryption token is created for a user group by encrypting the AES key with the public RSA key of the respective group. If the file is to be shared with multiple groups, a separate token is created for each of those groups. Decryption tokens are attached to the encrypted file in the header and uploaded in the remote server. User can later download the encrypted file and decrypt with his or her RSA decryption key. An unauthorized user is unable to decrypt because no decryption token is present for that user's group in the header. Plutus [59] is another approach designed for securing data in remote untrusted server that heavily relies on public key cryptography. Plutus offers a much greater scalability over Sirius by incorporating a clever key rotation technique that minimizes the key management cost associated with key revocation. To eliminate the overhead caused by public key cryptography, Naor et al. proposed a scheme in [60] that does not use public key cryptography and only relies on symmetric-key cryptography. However, the proposed solution works well under certain constrained scenarios such as when there are only a few read-only users or only one publisher with many read-only users.

Zhao et al. put forward a scheme in [61] that relies on a cryptographic primitive called progressive elliptic curve encryption scheme (PECE). In PECE, a piece of data is encrypted multiple times using different keys and the final ciphertext is decryptable with a single key in a single run. In a typical workflow, the cloud storage service provider has a shared public-private key pair with the data owner and a consumer has his or her own public-private key pair. The data owner encrypts the data with the shared public key and a random secret such that cloud service provider cannot decrypt it without knowing the random secret despite having the corresponding shared private key. Then the consumer requests access by providing the data owner with his public key. The data owner computes an intermediate with the consumer's public key and sends it to the cloud service provider. The cloud service provider re-encrypts the ciphertext with the intermediate key and sends it to the consumer. Finally, the consumer decrypts the ciphertext with his or her private key. While this scheme provides data confidentiality from the cloud service provider, it requires the data owner to remain online in order to grant access to any consumer or requester.

2.1.2. Schemes Based on Proxy Re-encryption and Identity-based Encryption. Proxy re-encryption (PRE) [62] allows a delegator (data owner) to encrypt a data with his or her public key (pk_i) and share it with a delegatee (user) by having the ciphertext transformed (re-encrypted) by a proxy with a re-encryption key ($rk_{i \rightarrow j}$) such that the user can decrypt the transformed ciphertext with his or her decryption key (sk_j). The mechanism of proxy re-encryption is shown in Figure 2.1 due to [63]. Proxy re-encryption has emerged as a promising cryptographic tool since the proxy does the transformation (re-encryption) in the encrypted domain without actually decrypting it and hence the actual data remains secure from the proxy. This technique has been adopted for secure data sharing in cloud by delegating the role of proxy to the cloud [63]. Initially proposed proxy re-encryption scheme [62] is based on ElGamal public key cryptography [64] and prone to collusion attack where the proxy can compute the delegator's private key by colluding with the delegatee. Jackboson et al. has fixed this serious problem in [65] by incorporating *k out of n* secret sharing scheme. The re-encryption key is divided into n pieces and given to n proxies such that re-encryption is possible only if at least t number of proxies work together. The proposed scheme is resilient to collusion attack as long as the number of dishonest proxies is less than t.

The security model of proxy re-encryption was not formalized until Ateniese at al. provided the formal definition along with the formal security model of proxy re-encryption in [66]. They also introduced in this work the first bilinear pairing-based proxy re-encryption scheme. It allows the delegator to periodically update the delegation relationship by assigning short-lived re-encryption keys without having to change the delegator's public key. The limitation of the proposed scheme is that it has single-use characteristic which only allows the re-encryption of the original ciphertext but not the output ciphertext of

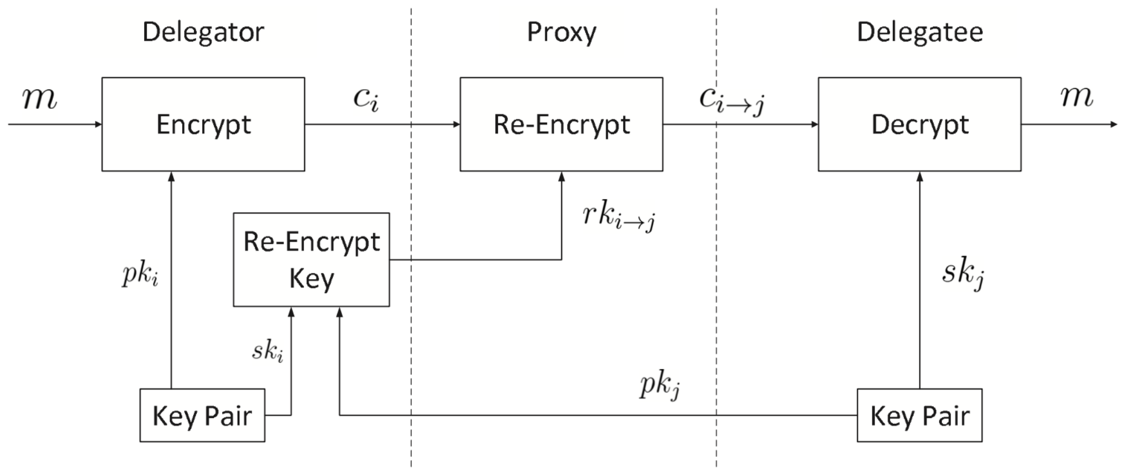


Figure 2.1. The mechanism of proxy re-encryption

re-encryption algorithm. Hohenberger et al. bridged this gap by proposing a multi-use PRE scheme in [67] which is also secure against chosen ciphertext attack (CCA). However, since this scheme is based on bilinear pairing, it is relatively expensive like [66]. Later Deng at al. proposed a pairing-free scheme in [68] that was more efficient. The proposed scheme is also bi-directional in nature that allows proxy to re-encrypt delegator's ciphertext into delegatee's ciphertext and vice versa. It has more potential where bi-directional communication is required.

The capabilities of PRE has been enhanced by coupling it up with other schemes. One prominent example of that is identity-based proxy re-encryption put forward by Green el al. in [69]. It combines identity-based encryption [3] with proxy re-encryption to eliminate the costly certificate management overhead of public keys. However, the scheme proposed in [69] is not collusion-resistant. Nevertheless, idea of eliminating certificate management overhead by incorporating identity-based encryption with PRE inspired many similar subsequent works such as [70, 71, 72, 73, 74]. Although identity-based PRE eliminates the necessity of certificate management, the formal notion of certificateless-based PRE (CL-PRE) scheme was proposed by Youngho et al. in [75]. In this work, PRE was introduced into certificateless public key encryption [76] to achieve the best of both public key encryption and identity-based encryption in PRE while eliminating their individual limitations. Xu et al. proposed a CL-PRE scheme in [77] that is suitable for secure data sharing in cloud environment. Later, Guo et al. improved upon the security of [77] by constructing a RCCA (replyable CCA) secure CL-PRE scheme in [78]. In contrast, Yang et al. proposed a PRE scheme in [79] that despite being certificate-based, has some distinct advantages making it more suitable for cloud. Firstly, it allows the delegator to delegate the partial decryption right to a proxy such that the ciphertext can be partially decrypted by the proxy in a way that greatly reduced the delegatee's decryption cost. This makes perfect sense where delegatees are low resource devices like sensors and the proxy is more resource heavy like cloud. Secondly, it avoids more costly pairing-heavy operations making it even more efficient for low resources devices.

2.1.3. Schemes Based on Attribute-based Encryption. Progress made in various fields of cryptography such as different public key encryption schemes, proxy re-encryption, and identity-based encryption have laid a solid foundation for secure data sharing in cloud. However, none of them is expressive enough to enforce a flexible and fine-grained cryptographic access control on data. Fine-grained access control is necessary when a data sharing group has users with various types of access privilege. The first encryption scheme capable of fine-grained access control was put forward by Sahai and Waters et al. in [8]. The proposed scheme is called fuzzy identity-based encryption because a data is encrypted with a set of attributes and an identity of a user, represented as a collection of attributes can decrypt it if the identity has enough attributes in it. The proposed scheme is also regarded as the first realization of attribute-based encryption (ABE) scheme since encryption and decryption keys are represented in terms of attributes.

The original ABE [8] was realized as a fuzzy identity-based encryption scheme. Soon after that, full fledged ABE has been proposed in two different flavours: key policy attribute-based encryption (KP-ABE) and ciphertext policy attribute-based encrypion (CP-ABE). In KP-ABE, the access policy is associated with the decryption key and attributes are associated with the ciphertext while in CP-ABE, the access policy is associated with the ciphertext and the attributes are associated with the decryption key. The first KP-ABE scheme was proposed by Goyal et al. in [9]. Several other KP-ABE schemes such as [10, 11, 12, 13, 14] has been proposed afterwards. One particular issue with KP-ABE is that the access policy is associated with the secret key. Since the attribute authority is responsible for creating and assigning secret keys, the data owner has little control over the access policy. This may be undesirable for certain data sharing scenarios where data owner wants to control the access policy. To address this issue, Bethencourt et al. proposed CP-ABE in [15] that allows the data owner to encrypt data with an access policy that determines which combination of attributes can decrypt the ciphertext. Various attempts have been made afterward to improve the original CP-ABE scheme. For example, Emura et al. proposed a CP-ABE scheme in [17] keeps the ciphertext size constant regardless of the size of access policy associated with the ciphertext. Ibraimi et al. improved the security model of CP-ABE and showed that their CP-ABE scheme is provably secure under the given model. Waters et al. in [16], not only improved the security, but also enabled encryption with more expressive policies. In the proposed method, they put forward a way to transform any kind of expressive policy into a boolean expression, and use it to encrypt ciphertext.

Apart from KP-ABE and CP-ABE, several other types of ABE schemes have also been proposed. The most notable one is multi- authority ABE (MA-ABE) scheme. Chase et al. first introduced MA-ABE in an attempt to address the key-escrow problem that exists in all previously proposed ABE schemes [80]. In this work, the single-authority has been replaced by multiple authorities, and each authority manages a different set of attributes such that no single authority can create a key capable of decrypting a ciphertext alone. The most challenging aspect in multi-authority setting is to prevent collusion attack from users. In single-authority, collusion prevention is achieved by having the authority rerandomizing the secret sharing appropriately with users such that keys generated for different users cannot be combined. In order to achieve that, the secret needs to be be split up in different way for each user by dividing it among multiple authorities. Additionally, this has to be done without any communication between the authorities. Chase is able to accomplish this goal with two main techniques: Global Identifier (GID) and central authority (CA). Each GID uniquely identified a user in the system and allows the authorities to distinguish users to prevent collusion. The fully trusted CA will hold the master secret for the system and will know all of the other authorities' pseudorandom functions (PRFs), which is used by each authority to randomize the secret key it gives out to a user. For each user, the CA will compute an extra value which, when combined with the user's constructed secret, will result in a GID-independent system decryption value that allows the user to decrypt. This approach has two major concerns: protecting the users' privacy and removing the trusted central authority. Due to the use of GID between users and authority to prevent users collusion attack, the disadvantage is that users' privacy is no longer guaranteed. Imagine a scenario that multiple authorities collude together to pool their information and build a complete profile of all the attributes corresponding to each GID. The use of a single trusted CA is the primitive work of Chase is responsible for issuing each user a unique key. In order to do so in a way to prevent collusion, the CA knows the master secret of the entire system and the secret PRF of each authority. Such approach schematic inevitably gives the CA the power to decrypt any ciphertext. These two concerns were then answered by the subsequent work done by Chase and Chow to improve the privacy and security in MA-ABE [81]. They proposed a solution which removes the trusted CA and protects the users' privacy by preventing the authorities from pooling their information on particular users, thus making ABE more usable in practice. The idea was suggested by Waters, formalized by Chase and Chow, in which each pair of attribute authorities (AA) would share a secret key. The proposed schemes proved that it is secure as long as at least two of the AAs are honest. An anonymous key issuing protocol was also presented in the paper which allows MA-ABE with enhanced user privacy by (1) allowing users to communicate with AAs via pseudonyms instead of GIDs and (2) preventing the AAs from pooling their data and linking multiple attributes belonging to the same user. Lin et al. proposed a different approach for MA-ABE also without a CA that achieves *m*-resilience. Their construction requires scheme designers to fix a constant m for the system, such that any group of $m + 1$ colluding users will be able to break the security of the encryption [82, 83]. Their threshold based scheme requires a set of authorities to be fixed ahead of time (similar to [80]), and they must interact during the system setup. Chase and Chow claimed that their work has two advantages over the work by Lin et al. First, it is difficult and inconvenient for the designers to fix m appropriately since it directly determines system efficiency. For large-scale systems, m need to be set reasonably high in order to guarantee security; this imposes burdens among all the authorities, and on their secure storage. Secondly, Chase and Chow's multi-authority scheme is secure no matter how many users collude, which is more practical. It is worth noting that the improved version of MA-ABE by Chase and Chow does not extend to non-monotonic access structure and the CP-ABE scheme. Li et al. later put forward the first multi-authority-CP-ABE (MA-CP-ABE) scheme in [18] that only allows small universe attributes, meaning that any arbitrary string can't be used as an attribute. Yannis and Waters overcame this limitation by proposing a large universe MA-CP-ABE scheme in [84] that allows the use of any arbitrary string as an attribute.

ABE offers a very convenient way of achieving fine-grained access control for secure data sharing in cloud. However, data sharing groups are often not static and group members frequently leave and join the group. To be able to use in such dynamic data sharing group, ABE needs to have revocation feature. Revocation has become a very active research topic in the field of ABE. Wang et al.[20] proposed a revocation technique using proxy re-encrytpion. It requires updating not only the attribute secret keys of all the nonrevoked users, but also the attribute public keys by a proxy (cloud) that results in a lot of overhead. Additionally, their technique only applies to KP-ABE scheme. Later, Sahai et al. proposed the idea of a revocable ABE using time-based proxy re-encryption in [26], and then realized by Qin et al. in [22]. All attributes are associated with an expiration time. In order to achieve revocation, the cloud periodically updates access structure of ciphertext such that users with expired attribute keys cannot decrypt the ciphertext. The improvements made in this work over [20] is that the revocation task is fully delegated to the cloud and the technique is compatible with CP-ABE. However, the major disadvantage is that it is not resilient against cloud-user collusion attack as cloud can restore the decryption ability of a revoked user even after his or her attributes expire. Proxy re-encryption-based technique proposed in PIRATTE [28] is not free from issues either. PIRRATE only allows *t* our of *n* revocations where *t* needs to be fixed beforehand.

Revocable ABE is further divided into two categories: indirectly revocable ABE and directly revocable ABE based on whether interaction is required between the attribute authority and the nonrevoked users. Interaction between the attribute authority and non-revoked users is required in indirect revocable ABE while no such interaction is required in directly revocable ABE. Revocation schemes such as [20, 26, 22, 28] are examples of indirectly revocable ABE and all of them require nonrevoked users to update their keys by interacting with the attribute authority. This key update process introduces a huge overhead on the user-end. To mitigate this overhead from the user-end, Cui et al. proposed an indirectly revocable ABE scheme in [85]. It minimizes the key update overhead at the user-end by delegating the key-update task to an aide-server. However, this scheme can't detect the malicious event where the aide-server performs the key-update task incorrectly. Yu et al. solved this problem in [32] by introducing a public auditor that can publicly verify the correctness of computation performed by the aide-server. Based on the RouselakisâĂŞWaters CP-ABE scheme [84], Qin et al. proposed another server-aided indirectly revocable CP-ABE scheme [86]. This scheme allows a user to delegate his or her decryption capacity to others while preventing decryption key exposure (DKE) attack that was left as an open problem in [85]. In directly revocable ABE scheme, no interaction is required between the attribute authority and the nonrevoked users. Directly revocable ABE was put forward by Hur et al. in [35] where revocation is realized by incorporating the concept of attribute groups with each attribute. A user's secret key consists of two parts: decryption secret key (DSK) and key encryption key (KEK), respectively. The DSK is associated with the user's attributes, while the KEK is associated with the user's attribute groups. Revocation is achieved by encrypting certain ciphertext components with KEK keys so that a user cannot decrypt the ciphertext without the appropriate KEK key despite having enough attributes in the DSK key. In their subsequent work [36, 29], Hur et al. improved the security by removing the key-escrow problem such that attribute authority alone cannot generate any secret key. This was achieved by splitting the master secret between the attribute authority and the group manager and employing a secure multi-party computation protocol between them during the secret key generation time. The same attribute-group-based revocation technique has been adopted by many other works such as [87, 31, 37]. Li et al. pointed out in [34, 88] that this attribute-group-based revocation technique has a serious flaw where a revoked user, by colluding with a nonrevoked user can restore his or her decryption ability. This is possible because KEK and DSK are independent of each other and one user's KEK is compatible with another user's DSK. A revoked user simply colludes with a nonrevoked user to get the nonrevoked user's KEK and combines it with his or her own DSK to restore the decryption ability. Li et al. proposed a solution to this problem by binding a user's KEK with his or her DSK such that one user's KEK does not work with another user's DSK. The limitation of the proposed scheme is that a semi-trusted attribute manager has to update all nonrevoked users' KEK keys for revocation, which not only adds a lot of overhead, but also adds additional security vulnerability as the attribute manager can collude with revoked users to restroe their decryption ability. Schemes such as [38, 27] further improved the security model of Li et al.'s scheme. However, they also rely on a semi-trusted entity for updating secret key for revocation. Hence, achieving revocation without the help of any semi-trusted entity was left as an open problem.

ABE schemes are relatively expensive as they rely on costly group exponentiation and bilinear pairing operations. Typically, the number of group exponentiation and bilinear operations grows linearly with the size of the access policy. Green et al. found a way to keep the decryption cost constant at the user end in [89]. This is achieved by securely outsourcing those expensive operations to the cloud. The idea is to blind user's attribute secret key with a blinding key and give it to the cloud while keeping the blinding key secret. The cloud can use blinded attribute keys to partially decrypt the ciphertext so that the user can later fully decrypt it using his or her secret blinding key at a constant cost. Later, [90, 91, 92, 93, 94] added verifiability to the outsourced decryption to ensure that the cloud correctly performs the computation. Among them, Qin et al. [90] was able to achieve verifiability with a short and constant overhead, while others ended up adding a much larger overhead to the ciphertext.

2.2. SECURE DATA SHARING IN IOT

With the wide adoption of IoT, its security has gained a lot of focus recently [45]. One important security aspect of IoT is how to securely share IoT data [46]. Jose and Hernandez et al. proposed a secure and privacy preserving framework intended for secure data sharing of IoT devices [48]. The proposed framework was built by taking into account both user privacy and data confidentiality. They argue that a compromised IoT device can potentially hamper user privacy since IoT device carries user's personalized information such name, address, ID, etc. As a mitigation strategy, they propose that a centralized identity management system should be put in place that instead of sharing the whole user credential, shares only a subset of required attributes of the whole credential. To this end, their recommendation is to use anonymous credential systems such as Idemix [95] or Uprove [96]. Such anonymous credential systems allow one to cryptographically proof the possession of certain attributes of his or her credential instead of sending the whole credential. To ensure data confidentiality, they recommend the use of ABE that can work in conjunction with anonymous user credential systems. A similar privacy preserving secure IoT data sharing scheme was proposed in [50] that relies on CP-ABE. Motivated by the establishment of e-Health Record (EHR) and the future potential of IoT application in healthcare [49], Yang et al. proposed an lightweight secure data management framework for healthcare IoT in [97]. This scheme is also based on ABE. However, compared to the previously proposed ABE-based schemes [48, 50], the advantage of this scheme is two-fold: Firstly, this scheme takes into account the distributed nature of the healthcare domain. It offers distributed access control by employing multi-authority ABE where each authority independently manages a distinct set of attributes. Secondly, they have managed to keep the scheme light-weight such that it is usable for low-resource IoT devices. Hossein and Hithnawi et al. proposed a cloud-based secure IoT data sharing platform in [98]. This scheme mainly uses homomorphic encryption to facilitate search query on encrypted data while it is stored in the cloud. However, the limitation of this scheme is that it is too expensive in terms of both storage and running time. Besides, it can't efficiently support certain types of range queries. A cloud-assisted IoT system was put forward in [99] that uses conditional identity-based broadcast proxy re-encryption as the main cryptographic construct for data security. It allows users to store and delegate their IoT data collection task to the cloud at the same time. Tao and Bhuiyan et al. took a hardware-based security approach in [51] to address the security concerns in healthcare IoT such as patient health monitoring sensors. Their proposed secure data collection scheme is composed of four layers: IoT network and sensor devices, FOG layer, cloud computing layer, and healthcare provider layer. To ensure the data security in IoT and FOG layers, they use light-weight field programmable gate array (FPGA) hardware-based cipher algorithm and secret cipher share algorithm, respectively. To ensure patient's privacy at the cloud computing layer, they distribute the database among multiple cloud servers such that no single cloud server have the entire view of the patient database. They also validate the performance of their scheme through proper simulation and show that their scheme is practical.

Most of the traditional cryptography-based solutions are centralized in one way or other and do not match well with the distributed nature of IoT. In an IoT ecosystem different parties are involved who may not trust each other. This requires better accountability and transparency to resolve any kind of dispute which is hard to address in traditional cryptography-based approach. It turns out that many of these issues can be addressed by leveraging blockchain. Hossein and Lukas et al. proposed a blockchain-based secure data sharing platform for IoT in [100]. In the proposed architecture, IoT data is viewed as chunked data streams. Each stream is divided into chunks of data, encrypted with a symmetric key and stored in a distributed file system. Data chunks are cryptographically linked together by making one chunk point to the hash of the next chunk. The symmetric key is encrypted with the owner's public key by following proxy re-encryption scheme. To share a data stream with a particular application, the data owner creates a blockchain transaction containing the stream identifier, proxy re-encryption key, and the blockchain public address of the application. The storage node, upon receiving data access request from a particular application, grants access if there exists such a transaction in the blockchain. Note that, a malicious storage node may still grant access even if no such blockchain transaction exists. However, that does not compromise the data since data is encrypted and each node stores a small portion of the data stream. While this encompasses user user-centric access control, it has several drawbacks. The access control decision is mainly made by the data owner and not in a distributed manner. Since their approach is based on the public blockchain, the transactions are much slower and cannot support real-time IoT application. Xueping and Sachin et al. proposed a data sharing and collaboration platform for mobile healthcare application in [101] where a private blockchain is the centerpiece of the system and connects patient with other stakeholders like hospitals, pharmacy, insurance companies etc. Scheme [102] proposes a secure data sharing technique for IoT that combines proxy re-encryption with blockchain. The proposed solution is intended for a secure IoT marketplace where sellers can sell their IoT measurement data to the buyers and the financial transaction automatically takes place via a blockchain smartcontract. This blockchain-based transaction help in any kind of dispute resolution in the future. Wei and Mingdong et al. proposed a blockchain-based secure data transmission technique for industrial IoT in [103].

PAPER

I. A COLLUSION-RESISTANT REVOCABLE ATTRIBUTE-BASED ENCRYPTION SCHEME FOR SECURE DATA SHARING IN CLOUD

MD Azharul Islam and Sanjay Madria

Department of Computer Science

Missouri University of Science and Technology

Rolla, Missouri 65401

Email: mdazharul.islam@mst.edu and madrias@mst.edu

ABSTRACT

Attribute-based encryption (ABE) is a prominent cryptographic tool for secure data sharing in the cloud because it can be used to enforce very expressive and fine-grained access control on outsourced data. The revocation in ABE remains a challenging problem as most of the revocation techniques available today, suffer from the collusion attack. The revocable ABE schemes which are collusion resistant require the aid of a semi-trusted manager to achieve revocation. More specifically, the semi-trusted manager needs to update the secret keys of nonrevoked users followed by a revocation. This introduces computation and communication overhead, and also increases the overall security vulnerability. In this work, we propose a revocable ABE scheme that is collusion resistant and does not require any semi-trusted entity. In our scheme, the secret keys of the nonrevoked users are never affected. Our decryption requires only an additional pairing operation compared to the baseline ABE scheme. We are able to achieve these at the cost of a little increase (compared to the baseline scheme) in the size of the secret key and the ciphertext. Theoretical performance analysis and experimental results show that our scheme outperforms the relatable existing schemes.

Keywords: secure cloud data sharing, attribute based encryption, revocation

1. INTRODUCTION

Many companies and organizations often outsource their data to a public cloud to enjoy advantages such as availability, scalability, and lower maintenance cost offered by the cloud. However, confidentiality and access control of the outsourced data remains a concern since the data owner loses control over the data once it is uploaded to the cloud. Recently, attribute-based encryption (ABE) has become a very promising tool [1] to achieve confidentiality and fine-grained access control for data outsourcing in the cloud. ABE allows a data owner to encrypt his or her data using a policy expressed in terms of a set of attributes so that it can be decrypted only if the secret key has enough attributes to satisfy the policy. Let us consider the motivational example in the following section.

1.1. A MOTIVATIONAL EXAMPLE

Suppose CryptoFlix is a Netflix-like streaming service that keeps all its media contents in a public cloud run by a third-party cloud service provider. Before outsourcing media files to the public cloud, it encrypts them using attribute-based encryption scheme. CryptoFlix offers two types of subscription plans: basic and premium. Basic and premium plans allow a subscriber to choose three and five attributes, respectively. Let us assume that two users, Alice and Bob, have a subscription for the basic plan, and another user, Eve has a subscription for the premium plan. Alice loves science fiction movies and documentaries, she gets decryption keys for attributes *movie, scifi,* and *documentary* from the attribute authority (a trusted entity responsible for generating and distributing attribute secret keys). Conversely, Bob loves newly released tv shows and gets decryption keys for *new_release, tv_show,* and *documentary* attributes from attribute authority. Let Eve has decryption keys for attribute *movie, tv_show, documentary, scifi,* and *new_release*, but her premium subscription plan recently expired. CryptoFlix encrypts the newly released science fiction movie XFiction under the policy *(new_release AND movie AND scifi)* and uploads it to a public cloud. Note that none of the three users can decrypt XFiction under normal circumstance. However, they can cooperate with each other and try to decrypt it by launching the following attacks:

- *Type I attack:* Multiple users who individually do not have enough attributes to satisfy a policy cooperate with each other so that their collective attribute keys may satisfy the policy.
- *Type II attack:* A revoked user who cannot decrypt a file despite having enough attributes to satisfy the policy cooperates with a nonrevoked user to restore his or her decryption ability in order to decrypt the file.

These attacks are called collusion attack. In a *type I* attack, Alice and Bob would combine their attribute keys and try to decrypt XFiction. On the otherhand, in a *type II* attack, Eve would combine her attribute keys with a nonrevoked user (such as Bob) and try to decrypt XFiction. Not to mention CryptoFlix faces financial loss if any of the attacks becomes successful. This motivational example will be referred to repeatedly in the upcoming sections.

1.2. LIMITATIONS OF THE EXISTING SCHEMES AND OUR NOVELTY

Collusion resistance is a fundamental security requirement of any ABE scheme, as stated in the original ABE scheme proposed by Sahai et al. [2]. Initially proposed ABE schemes such as [2, 3, 1] do not support revocation. If CryptoFlix were to use such an

ABE scheme, it could not revoke Eve even if her subscription expired. As a result, such schemes are not suitable for a practical application like secure cloud data sharing. They are resistant to *type I* collusion attacks. However, *type II* collusion attacks do not apply to them as those schemes do not support revocation. The revocation is a challenging problem in ABE since the same attribute may be shared among different users. Hur et al. proposed a solution to the revocation problem in [4, 5]. The proposed solution is based on the idea of attribute group. The user's secret key consists of two parts. One is associated with the user's attributes, and the other is associated with the attribute group. They are called decryption secret key (DSK) and key encryption key (KEK), respectively. The revocation is dictated by KEK and is independent of the user's DSK. Consequently, the KEK of one user works with the DSK of another user. Hence, the proposed revocable ABE scheme is not resistant against *type II* collusion attacks as a revoked user by colluding with a nonrevoked user can get the valid KEK and restore his or her decryption ability. This vulnerability was first pointed out by Li et al. [6]. Schemes such as [7, 8] also have the same vulnerability since these solutions are also based on the same idea. Li et al. refined their initial solution [6] in [9]. To revoke a user from the attribute group, the attribute manager (AM) updates the existing user's KEK keys. They bind a user's DSK with his or her KEK so that the KEK of one user does not work with the DSK of another user. This ensures that a revoked user cannot collude with a nonrevoked user to restore his or her decryption right. However, this scheme has the following limitations:

- Each time a user is revoked, all nonrevoked users' keys (KEK) are affected because DSK and KEK keys are tied together by a common secret exponent that is only known to a semi-trusted party called the attribute manager (AM). This exponent is common across attribute secret keys of all the users. To revoke a user, this exponent needs to be updated in the secret keys for all nonrevoked users.

- It requires the additional semi-trusted AM for updating all nonrevoked users' secret keys (KEK) and distributing them to the respective users. This not only adds a lot of overhead, but also increases the security vulnerability by adding an additional semi-trusted party to the system.
- AM also becomes the performance bottleneck as it needs to participate in the key generation, key update, encryption, and re-encryption stages. Key generation is an one-time operation. However, other operations occur very frequently, which can be a huge burden for a centralized entity like AM to handle.

By following the footstep of [9], CryptCloud+[10] also proposes a collusion-resistant revocable ABE scheme. In this scheme, the attribute authority has to periodically update the attribute secret keys of all the users according to a revocation list, which is very inefficient. Clearly, recent research leaves significant gaps. Our proposed revocable ABE scheme fills these research gaps for the first time. Like Li et al.'s schemes ([6] and [9]), our revocable ABE scheme is also collusion resistant against both *type I* and *type II* attacks. However, in our scheme, revocation does not affect the secret key of any nonrevoked user. Moreover, our scheme does not need the aid of any additional trusted entity, which minimizes the attack surface. This is made possible since we achieve revocation by modifying the core ABE secret key and ciphertext elements rather than achieving revocation by attribute group keys (KEK). We discuss our technique in the following section.

1.3. OUR TECHNIQUE AND CONTRIBUTION

Our revocation technique can be applied to any ABE scheme that does not support revocation (e.g., [3, 1, 11]). However, in this paper we choose the scheme proposed in [1] for several reasons. This scheme has a large universe construction (meaning that any arbitrary string can be used as an attribute) and it has been proven to be selectively secure in standard model (as opposed to the artificial random oracle model) under decisional q-Parallel

Bilinear Diffie-Hellman exponent (decisional q-BDHE) assumption. These properties are more desirable in real-life application from both functional and security standpoints. Our ABE scheme also inherits these properties. We achieve revocation by modifying the core ABE secret key and ciphertext components and distributing them according to a binary tree. A data owner can revoke any user while encrypting a message using the minimum cover algorithm (Section 3.6) on the binary tree. This required us to design new setup, encryption, keygen, and decryption algorithms for our proposed revocable ABE scheme. We briefly summarize our contribution as follows:

- We propose a collusion-resistant revocable ABE scheme that is resistant against both *type I* and *type II* collusion attacks. We achieve revocation property through modification of the core ABE secret key and ciphertext components. Hence, we eliminate the requirement of any semi-trusted entity for key updating.
- Revocation in our scheme never affects the secret keys of any nonrevoked user. A data owner can revoke any user's decryption ability from a particular file by including him/her in the revocation list during encryption.
- To show the effectiveness of our proposed collusion-resistant revocable ABE scheme, we build a secure cloud data sharing scheme based on it.
- Through proper security analysis, we prove that our scheme is collusion resistant.
- Through extensive theoretical and experimental performance analysis, we show that our scheme outperforms recently proposed similar schemes.

2. RELATED WORKS

Revocable ABE was first addressed by Sahai et al. in [12], and then realized by Qin et al. in [13]. The attributes in both user's secret key and the access structure of a ciphertext are associated with different expiration time. The time specifies how long a secret key is allowed to decrypt a ciphertext. The cloud has to periodically update the access structure of a ciphertext using proxy re-encryption to enforce revocation. However, the problem is that, a colluding cloud can re-encrypt a ciphertext to an expiration time so that it can be decrypted by a user's expired secret key. The scheme proposed in [14] also incorporates the same idea of achieving revocation by updating the ciphertext. Instead of updating the ciphertext in a timely fashion, an aide server updates the ciphertext according to the data owner's provided revocation list. However, this scheme also suffers from a similar kind of collusion attack (server-revoked user) as [13]. Recently, [15] and [16] also proposed a revocable ABE scheme where the revocation is aided by an untrusted server. The untrusted server uses the transformation key and periodic key updates provided by the attribute authority to partially decrypt ciphertext for a specific time period. Nonrevoked users at the specific time period can fully decrypt the partially decrypted ciphertext.

Hur et al. followed a different approach for revocation of ABE in [4] and later improved its security in [5]. According to their proposed solution, a user belongs to various attribute groups and the authorized set of attributes is determined by the attribute groups the user belongs to. A user's secret key consists of two parts: decryption secret key (DSK) and key encryption key (KEK), respectively. The DSK is associated with the user's attributes, while the KEK is associated with the user's attribute groups. Revocation is achieved by encrypting certain ciphertext components with KEK keys so that a user cannot decrypt the ciphertext without the appropriate KEK key despite having enough attributes in the DSK key. Li et al. first pointed out in [6] that DSK and KEK keys in Hur et al.'s proposed solution are independent of each other. As a result, a revoked user can collude with a nonrevoked user, and combine the nonrevoked user's KEK key with his or her own DSK keys to restore the decryption ability. Schemes like [7, 8] also suffer from the same revoked-nonrevoked user collusion attack. Li et al. proposed an initial solution to solve this

Table 1. Comparison with related schemes in terms of security and functionality.

Scheme	Security assumption	Security Model	Collusion resistant	Revocation affects others' key
Hur-I [4]	Generic Group	RO	✗	✓
Hur-II [5]	Generic Group	RO	✗	✓
CryptCloud+ [10]	l-SDH	Standard	✓	✓
Flexible [6]	Generic Group	RO	✓	✓
UserCol [9]	Generic Group	RO	✓	✓
Ours	decisional q-BDHE	Standard	✓	✗

collusion problem in [6, 17] and later improved the security in [9]. The revoked and nonrevoked user collusion problem was solved by binding the DSK key with the KEK key so that one user's DSK key does not work with another user's KEK key. The limitation of the proposed scheme was that a semi-trusted attribute manager has to update all nonrevoked users' KEK keys for revocation, which not only adds a lot of overhead, but also adds additional security vulnerability as the attribute manager can collude with revoked users to restroe their decryption ability. The attribute manager also needs to process every ciphertext, which can be a performance bottleneck. The solution proposed in CryptCloud+ [10] also relies on a semi-trusted entity for key update in revocation. In Table 1, we compare our scheme with the most relatable ones in terms of security and functionalities. Our scheme is selectively secure in the standard model under the decisional q-Parallel Bilinear Diffie-Hellman Exponent (decisional q-BDHE) assumption [1]. CryptCloud+ is proven to be secure in the standard model based on the hardness of l-Strong Diffie-Hellman (l-SDH) assumption. Hur-I [4], Hur-II [5], Flexible [6], and UserCol [9] are secure in the random oracle (RO) model with generic group assumption. According to [1] the random oracle model is an artificial model and not desirable for real-life applications. Hur-I and Hur-II are not resistant against a revoked-nonrevoked user collusion attack (*type II*), while the rest are secure against this attack. Note that only in our scheme does the revocation not affect the secret keys of the existing nonrevoked users.

3. PRELIMINARIES

3.1. SYMBOLS AND NOTATIONS USED

In this paper, we use $\mathbb{G}$ and $\mathbb{G}_T$ to represent two multiplicative cyclic groups of prime order p, while g is a generator of $\mathbb{G}$. The symbol $\mathbb{Z}_p$ is used to denote the group of integers modulo p. We also make use of a *randomness extractor function*[18] defined as $\mathcal{F} : \mathbb{G}_T \rightarrow \mathbb{K}$, where $\mathbb{K}$ is the symmetric key space. The encryption and decryption functions of the symmetric encryption scheme are denoted as *Enc* and *Dec*, respectively.

3.2. BILINEAR MAP

A bilinear map is a function e defined as $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and must have the following properties:

1. *Bilinearity:* For $\forall g_1, g_2 \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}_p$, the following relationship must always hold: $e\left(g_1^a, g_2^b\right) = e\left(g_1, g_2\right)^{ab}$
2. *Non-degeneracy:* $e\left(g_1, g_2\right) \neq 1$
3. *Computability:* group operations in $\mathbb{G}$ and e should be efficiently computable.

3.3. DECISIONAL *Q*-PARALLEL BILINEAR DIFFIE-HELLMAN EXPONENT ASSUMPTION

We review the definition of decisional q-BDHE assumption from [1]. Following the notations from Section 3.1, assume that a, s, and q exponents (e.g., $b_1, b_2, \ldots, b_q$) are randomly chosen from $\mathbb{Z}_p$. Then, according to the decisional q-BDHE assumption it is hard for any probabilistic polynomial time adversary to distinguish $e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element $e(g,g)^r \in \mathbb{G}_T$ if the adversary is provided with the vector $\vec{v} = g, g^s, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, g^{2q}, \forall 1 \leq i \leq q : g^{sb_i}, g^{a/b_i}, \ldots, g^{a^q/b_i}, g^{a^{q+2}/b_i}, \ldots, g^{a^{2q}/b_i}, \forall 1 \leq$

$i, j \leq q, i \neq j : g^{asb_j/b_i}, \ldots, g^{a^q sb_j/b_i}$. The advantage ϵ of a probabilistic polynomial time algorithm $\mathcal{B}$ in solving the decisional q-BDHE problem is defined as $\epsilon \leq |\Pr[\mathcal{B}(\vec{v}, A = e(g,g)^{a^{q+1}s}) = 0] - \Pr[\mathcal{B}(\vec{v}, A = e(g,g)^{a^r}) = 0]|$.

3.4. ACCESS STRUCTURE

Let $\mathbb{P} = \{P_1, P_2, \cdot, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathbb{P}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure is a collection $\mathbb{A}$ of non-empty subsets of $\mathbb{P}$ (i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdot, P_n\}} \backslash \{\emptyset\}$). The sets in $\mathbb{A}$ are called authorized sets, and sets not in $\mathbb{A}$ are called unauthorized sets. In our context, the role of the parties is defined by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes.

3.5. LINEAR SECRET SHARING SCHEME (LSSS)

Let p be a large prime. Then, a secret sharing scheme $\prod$ over a set of parties P is called linear (over $\mathbb{Z}_p$) if

- The shares of each party form a vector over $\mathbb{Z}_p$.
- There exists a matrix M with l rows and n columns called the share-generating matrix for $\prod$. For all $i = 1, 2, \ldots, l$, the i^{th} row of M, we define a function ρ such that $\rho(i)$ maps row i of matrix M to an associated party. When we consider the column vector $\vec{v} = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of l shares of the secret s according to $\prod$. The share $(Mv)_i$ belongs to party $\rho(i)$.

Any linear secret sharing scheme defined above has the following *linear reconstruction* property: Let $\prod$ be an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \ldots, l\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to $\prod$, then $\sum_{i \in I} w_i \lambda_i = s$. These constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix M.

3.6. MINIMUM COVER

Let $\mathcal{T}$ be a full binary tree with m leaf nodes. Nodes in $\mathcal{T}$ are labelled as y_j, and each leaf node is associated with a user labelled as $u_1, u_2, \ldots, u_m$. We use *path(u_k)* to denote all the nodes in the path from the root to the associated leaf node of u_k. Let the set $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and *RL* represent the set of all users and revoked users, respectively. Then, we can apply subset cover to get the minimum number of tree nodes (we call it *cover(RL)*) that cover $\mathcal{U} - RL$ (i.e., all nonrevoked users). The algorithm to find *cover(RL)* is as follows:

- $\forall y_j \in RL$, color all the nodes of *path(y_j)*.
- *cover(RL)* is the set of all uncolored nodes that are direct children of the colored nodes.

Figure 1 is an example of a binary tree $\mathcal{T}$ with 8 leaf nodes and 8 associated users (i.e., $\mathcal{U} = \{u_1, u_2, \ldots, u_8\}$). If $RL = \{y_{13}, y_{15}\}$, then *Cover(RL)* $= \{y_2, y_{12}, y_{14}\}$. Node that six nonrevoked users (i.e., $\{u_1, u_2, u_3, u_4, u_5, u_7\}$) are covered with just three nodes.

4. SYSTEM ARCHITECTURE AND ADVERSARIAL MODEL

4.1. SYSTEM ARCHITECTURE

A high-level system architecture of our secure data sharing scheme has been presented in Figure 2. Our architecture has four main entities: the attribute authority (AA), the cloud service provider (CSP), the data owner, and the data user. The attribute authority

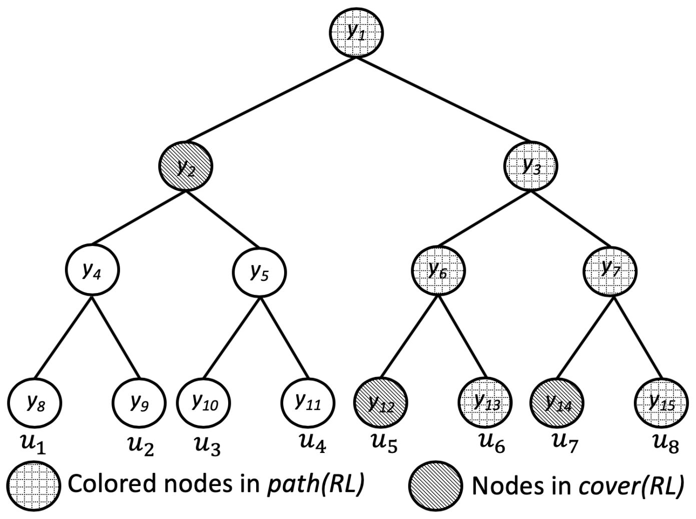


Figure 1. Finding minimum cover

is responsible for managing all the attributes, and it creates and distributes attribute secret keys to the users according to their authorized attribute set. It also creates and publishes the public parameters. The CSP manages the public cloud where the encrypted data is outsourced and stored. Data owners encrypt their files using an attribute-based encryption scheme and uploads the files to the public cloud so that they are always available for the data users. Once the data is uploaded in the cloud, the data users can download and decrypt it anytime if they have enough attributes in their attribute secret keys. Note that a user may have the dual role of a data owner and a data user.

4.2. ADVERSARIAL MODEL

4.2.1. Security Assumptions. We consider the CSP to be an honest but curious entity that is a standard practice in revocable ABE literature [5, 6, 9, 7, 19]. This implies that the CSP properly follows the protocols of our scheme (i.e., it honestly performs tasks like storing and updating encrypted files as per the data owners' request and letting the data users download encrypted files upon request). The CSP does not tamper with any stored information in the cloud. However, the CSP is open to deduce any plaintext information

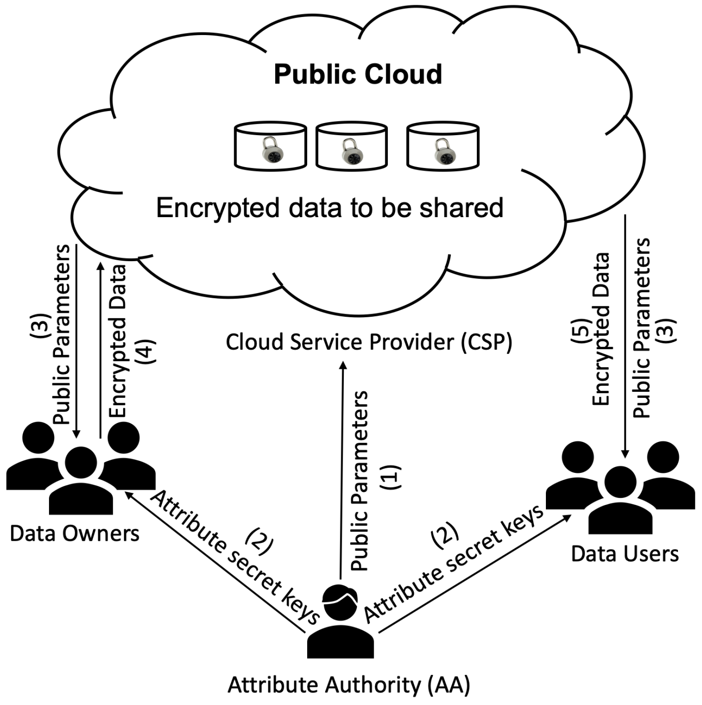


Figure 2. System Architecture

from the stored encrypted files and public parameters on its own. In previously proposed revocation schemes such as [4, 5, 6, 9], a semi-trusted entity (manager) needs to update users' secret keys and ciphertext in order to achieve revocation. This design introduces additional security vulnerabilities, as a compromised manager can update secret keys and ciphertext in a way that restores a revoked user's decryption right. Hence, user and manager collusion is not allowed. However, we do not need any semi-trusted manager to achieve revocation since the revocation right is given to the data owner, who can decide whom to revoke during encryption. We assume that the attribute authority is a trusted entity, and it distributes attribute secret keys to users via a secure channel such as SSL.

4.2.2. Adversaries and Attacks. A dishonest data user is the main adversary of our system. A dishonest data user can be either revoked or nonrevoked. The goal of such adversaries is to decrypt a ciphertext that cannot be decrypted individually, either because they do not have enough attributes in their attribute secret keys or because they are revoked users (but may have enough attributes). To achieve this goal, a dishonest data user colludes with other dishonest data user(s) and launches *type I* and *type II* attacks.

5. OUR PROPOSED REVOCABLE ABE SCHEME

There are two types of ABE schemes: ciphertext policy attribute-based encryption (CP-ABE) and key policy attribute-based encryption (KP-ABE). The access polity is embedded in the ciphertext in CP-ABE, while the access policy is embedded in the keys in KP-ABE. We use ABE to denote CP-ABE in the rest of the paper unless otherwise specified. In this section, we first give the definition of our proposed revocable ABE scheme, followed by its security model. Finally, we give the detailed construction of our scheme.

5.1. DEFINITION OF OUR PROPOSED REVOCABLE ABE

The revocable ABE scheme is consisted of four algorithms defined as follows:

• (PK, MK) $\longleftarrow$ `Setup` $(Att_{\max}, l_{\max}, m)$: The setup algorithm takes as input the maximum number of attributes allowed in a secret key, the maximum number of columns possible in a LSSS matrix, and the total number of users in the system denoted as $Att_{\max}$, $l_{\max}$, and m, respectively. It outputs the public key PK, and the master secret key MK.

• CT $\longleftarrow$ `Encrypt` (PK, (M, ρ), $\mathcal{M}$, RL): The inputs to the encryption algorithm are the public key PK, the LSSS access structure (M, ρ), the message to be encrypted $\mathcal{M}$, and the revocation list RL. The algorithm outputs a ciphertext CT so that no user in RL can decrypt CT even if the attribute set S satisfies the access structure.

• SK $\longleftarrow$ `Keygen` (PK, MK, S, u_k): The key generation algorithm takes as input the public key PK, the master secret key MK, the user's authorized attribute set S, and the user's identifier u_k. It outputs the user's secret key SK.

• $\mathcal{M}/\perp \longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public key PK, the user's secret key SK, and the ciphertext CT. It outputs the plaintext message $\mathcal{M}$ if the user does not belong to the corresponding revocation list RL of CT and his or her authorized attribute set S satisfies the access structure. The decryption algorithm outputs $\perp$ otherwise.

5.2. SECURITY MODEL

We formalize the security model of our proposed revocable ABE scheme by the following IND-CPA (indistinguishable chosen plaintext attack) game.

Init. The adversary $\mathcal{A}$ commits to an access structure (M^*, ρ) by giving it to the challenger.

Setup. The challenger runs the `Setup` ($Att_{\max}, l_{\max}, m$) algorithm to generate PK, SK, and sends PK to $\mathcal{A}$.

Phase I Query. The adversary $\mathcal{A}$ repeatedly makes $q1$ private key queries for the user-authorized attribute set tuples as in $Q_1 = (u_1, S_1), Q_2 = (u_2, S_2), \ldots, Q_{q1} = (u_{q1}, S_{q1})$. The challenger calls `Keygen` (PK, MK, S_k, u_k) for each query $Q_k = (u_k, S_k)$, and sends the secret key SK$_k$ to $\mathcal{A}$.

Challenge. $\mathcal{A}$ selects two equal size messages $\mathcal{M}_0, \mathcal{M}_1$, a revocation list RL^*, and sends them to the challenger. Additionally, $\mathcal{A}$ also sends to the challenger the committed access structure (M^*, ρ). The challenger chooses a bit $b \in \{0, 1\}$ by flipping a random coin, and runs `Encrypt` (PK, (M^*, ρ), $\mathcal{M}_b$, RL^*). The challenger then sends the output ciphertext CT* to $\mathcal{A}$. The constraint is that none of the attribute sets (e.g., $S_1, S_2, \ldots, S_{q1}$) in phase I query satisfy the access structure (M^*, ρ).

Phase II Query. $\mathcal{A}$ adaptively makes private key queries for tuples $Q_{q1+1} = (u_{q1+1}, S_{q1+1}), Q_{q1+2} = (u_{q1+2}, S_{q1+2}), \ldots, Q_q = (u_q, S_q)$ with the restriction that none of the attribute sets in these tuples (e.g., $S_{q1+1}, S_{q1+2}, \ldots, S_q$) satisfy the committed access structure (M^*, ρ).

Guess. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for b.

The advantage of the adversary $\mathcal{A}$ in the above game is defined as $Adv = |\Pr[b' = b] - 1/2|$. By allowing $\mathcal{A}$ to do decryption queries in phase I and phase II query stage, this security model can be easily extended to chosen-ciphertext attack (CCA).

Definition 1 *Our proposed revocable ABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the IND-CPA game.*

5.3. CONSTRUCTION OF OUR PROPOSED REVOCABLE ABE SCHEME

The detailed construction of our proposed collusion-resistant revocable ABE scheme is given as follows:

• (PK, MK) $\longleftarrow$ `Setup` $(Att_{\max}, l_{\max}, m)$: The setup algorithm takes as input $Att_{\max}$, the maximum number of attributes any user's secret key may have; $l_{\max}$, the maximum number of columns any LSSS matrix M may have; and m, the total number of users. It outputs the public and the master secret key PK and SK, respectively. The setup algorithm first chooses a group $\mathbb{G}$ of prime order p with a generator g and defines a bilinear map as $e : \mathbb{G}\times\mathbb{G} \rightarrow \mathbb{G}_T$. We assume that attributes can be represented in $\mathbb{Z}_p$. In practice, a collision-resistant hash function can be used to transform any string attribute into $\mathbb{Z}_p$. The setup algorithm then randomly chooses $a, \alpha, \beta \in \mathbb{Z}_p$. It also utilizes a hash function defined as $\mathcal{H} : \mathbb{Z}_p \rightarrow \mathbb{G}$. The hash function is realized by choosing a polynomial $\mathcal{L}(x) \in \mathbb{Z}_p$ of degree $N = Att_{\max} + l_{\max} - 1$ and computing $h_0 = \mathcal{H}(0) = g^{\mathcal{L}(0)}, h_1 = \mathcal{H}(1) = g^{\mathcal{L}(1)}, \ldots, h_N = \mathcal{H}(N) = g^{\mathcal{L}(N)}$. With these $N + 1$ values, one can compute $h_x = \mathcal{H}(x) = g^{\mathcal{L}(x)}$ for any $x \in \mathbb{Z}_p$ by using interpolation.

The AA then creates a full binary tree $\mathcal{T}$ of m leaves and associates each user u_j to a different leaf node. For each node y_i in $\mathcal{T}$, AA randomly chooses $g_{y_i} \in \mathbb{G}$. Finally, AA publishes the public key as PK $= \left(\mathcal{T}, \mathbb{G}, \mathbb{G}_T, e(g, g)^{\alpha}, g^{a}, g^{\beta}, h_0, h_1, \ldots, h_N\right)$ and sets the master secret key as MK $= (g^{\alpha}, \beta)$ and keeps it secret.

• CT $\longleftarrow$ `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*): The encrypt algorithm takes as input the public parameters PK, LSSS access structure *(M, ρ)*, the plaintext message $\mathcal{M} \in \mathbb{G}_T$, and a revocation list *RL*. It outputs the ciphertext (CT). In the LSSS access structure *(M, ρ)*, M is an $l \times n$ matrix and ρ is a function that associates rows of M to attributes. In this construction, ρ is limited to be an injective function (i.e., an attribute can be associated with at most one row of M). The algorithm chooses a random vector $\vec{v} = (s, r_2, \ldots, r_n) \in \mathbb{Z}_p^n$. These values are used to share the random encryption exponent s. For $\forall i \in \{1, 2, \ldots, l\}$ it computes $\lambda_i = \vec{v}.M_i$, where M_i is the vector corresponding to the i^{th} row of M. Finally, the algorithm finds *cover(RL)*, randomly chooses an exponent $r \in \mathbb{Z}_p$, and computes the ciphertext as CT $= (C = \mathcal{M}.e(g,g)^{\alpha s}, C' = g^{s\beta}, D = g^r, \forall y_j \in cover(RL) : C_{y_j} = g_{y_j}^s, \forall i \in \{1, 2, \ldots, l\} : C_i = g^{a\lambda_i}\mathcal{H}(\rho(i))^{-r})$. We assume that the LSSS access structure is implicitely included in CT.

• SK $\longleftarrow$ `Keygen` (PK, MK, S, u_k): The keygen algorithm takes as input PK, MK, a set S of attributes the user is authorized for, and the user's identifier u_k and outputs the attribute secret key SK. Let u_k's associated leaf node in $\mathcal{T}$ be y. The algorithm finds *path(y)*, chooses a random $t \in \mathbb{Z}_p$, and creates the private key as SK $= (\forall y_j \in path(y) : K_{y_j} = \left(g^{\alpha+at}g_{y_j}\right)^{1/\beta}, L = g^t, \forall x \in S : K_x = \mathcal{H}(x)^t)$.

• $\mathcal{M}/\perp$ $\longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public paprameters PK, the user's attribute secret key SK for an attribute set S, and the ciphertext CT for a LSSS access structure *(M, ρ)*. Assume that S satisfies the access structure and $I \subset \{1, 2, 3, \ldots, l\}$ is defined as $I = \{i : \rho(i) \in S\}$. The algorithm finds the constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of a secret s according to M, then $\sum_{i \in I} w_i \lambda_i = s$. Note that these constants $\{w_i\}$ can be found in polynomial time in the size of M and there could potentially be different ways of choosing such $\{w_i\}$. The algorithm then computes the following:

$$
\begin{aligned}
P &= e(K_{y_j}, C') = e\left(\left(g^{\alpha+at} g_{y_j}\right)^{1/\beta}, g^{\beta s}\right) \\
&= e(g,g)^{\alpha s + ats}.e(g_{y_j}, g)^s \\
Q &= e(C_{y_j}, g) = e(g_{y_j}, g)^s \\
W &= P/\left(Q \prod_{i \in I} \left(e(C_i, L) e(D, K_{\rho(i)})\right)^{w_i}\right) \\
&= P/\left(Q \prod_{i \in I} e(g,g)^{at\lambda_i w_i}\right) \\
&= e(g,g)^{\alpha s + ats}.e(g_{y_j}, g)^s / \left(e(g_{y_j}, g)^s e(g,g)^{ats}\right) \\
&= e(g,g)^{\alpha s}.
\end{aligned}
$$

Then, the decryption algorithm retrieves the plaintext message $\mathcal{M}$ as in $C/W = \mathcal{M}$, and outputs it. If S does not satisfy the access structure, or the owner of SK belongs to the revocation list RL, then the decryption algorithm outputs $\perp$.

6. A SECURE CLOUD DATA-SHARING SCHEME BASED ON OUR PROPOSED ABE SCHEME

In this section, we propose a secure data-sharing scheme for the cloud based on our proposed collusion-resistant revocable ABE scheme. We discuss the details in the following.

6.1. SYSTEM INITIALIZATION

The attribute authority (AA) runs the `Setup` algorithm to initialize the system. It publishes the public key PK in the cloud and keeps the master key MK secret.

6.2. KEY DISTRIBUTION

For each user in the system, the AA runs the `Keygen` algorithm and generates the attribute secret key SK for his or her authorized attribute set *S*. The AA then sends SK to the corresponding user via a secure channel.

6.3. FILE OUTSOURCING

The data owner downloads the public parameter PK from the cloud. It randomly chooses $\mathcal{M} \in \mathbb{G}_T$ and extracts the symmetric key using the *randomness extractor function* $\mathcal{F}$ as in $\mathcal{K} = \mathcal{F}(\mathcal{M})$. Using $\mathcal{K}$, the data owner encrypts the actual file F using the symmetric encryption scheme as in CT′ =*Enc(*$\mathcal{K}$*, F)*. Then, the data owner includes any of the users in the revocation list *RL* to revoke from the ciphertext, runs `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*) algorithm, and receives CT as output. The data owner then uploads the final ciphertext (CT, CT′) as an encrypted file in the cloud.

6.4. FILE RETRIEVING

The data user downloads an encrypted file (CT, CT′) from the cloud and calls `Decrypt` (PK, SK, CT) algorithm with his or her attribute secret key SK. If SK has enough attributes to satisfy the access policy of CT, the algorithm returns $\mathcal{M}$ as an output. Then, the user extracts the symmetric key using the *randomness extractor function* $\mathcal{F}$ as in $\mathcal{K} = \mathcal{F}(\mathcal{M})$. Using $\mathcal{K}$, the data user retrieves the actual file F by running the decryption function of the symmetric encryption scheme as in F =*Dec(*$\mathcal{K}$*, CT′)*.

7. SECURITY ANALYSIS

In this section, we analyze the security of our proposed revocable ABE scheme in terms of semantic security and collusion attacks.

7.1. SEMANTIC SECURITY

Our IND-CPA game (in Section 5.2) is similar to that of [1] except in our game, each secret key query in query phase I and II also includes a user identifier u_i, and $\mathcal{A}$ provides with a revocation list RL^* in the challenge stage. It was proven in [1] that if the decisional q-BDHE assumption holds, then no polynomial-time adversary $\mathcal{A}$ can selectively win the IND-CPA game with a challenge matrix M^* of size $l^* \times n^*$ corresponding to the access structure (M^*, ρ), and maximum number of attributes per key of $Att_{\max}$ where $n^* + Att_{\max} \leq q$.

Assume there exists an adversary $\mathcal{A}$ who has a non-negligible advantage $\epsilon = \text{Adv}_{\mathcal{A}}$ in the our IND-CPA game and it chooses a challenge access structure (M^*, ρ) with the matrix M^* of at most q columns. Then using the same technique as in [1], we can build a simulator $\mathcal{B}$ that plays the decisional q-BDHE problem. The simulator basically programs all the IND-CPA game parameters from the decisional q-BDHE parameters so that the challenger cannot distinguish whether it is playing our IND-CPA game or the decisional q-BDHE problem. Since $\mathcal{A}$ has a non-negligible advantage in our IND-CPA game (according to our prior assumption), $\mathcal{A}$ also has a non-negligible advantage in the decisional q-BDHE problem. However, according to the definition (of decisional q-BDHE problem), no polynomial-time adversary has a non-negligible advantage in solving the decisional q-BDHE problem. This implies that $\mathcal{A}$ does not have a non-negligible advantage in our IND-CPA game. So, the DEFINITION 1 holds true.

7.2. COLLUSION ATTACK

The revocable ABE needs to be secure against both *type I* and *type II* collusion attack. The semantic security discussed earlier also guarantees that our scheme is secure against *type I* collusion attack since the secret key queries made by the adversary $\mathcal{A}$ in query phase I and phase II cannot individually satisfy the committed access structure (M^*, ρ). In this section, we formalize the security of our scheme against *type II* collusion attack by the following theorem.

Theorem 1 *A revoked user with enough attributes in his or her attribute secret key cannot decrypt a ciphertext even if the user colludes with a nonrevoked user who does not have enough attributes in his or her attribute secret key to decrypt the particular ciphertext.*

$$\begin{aligned} P &= e(K'_{y_j}, C') = e\left(\left(g^{\alpha+at'} g_{y_j}\right)^{1/\beta}, g^{\beta s}\right) \\ &= e(g,g)^{\alpha s+at's}.e(g_{y_j},g)^s \qquad (1) \\ Q &= e(C_{y_j}, g) = e(g_{y_j}, g)^s \qquad (2) \\ W' &= P/\left(Q\prod_{i\in I}\left(e(C_i,L)e(D,K_{\rho(i)})\right)^{w_i}\right) \\ &= P/\left(Q\prod_{i\in I} e(g,g)^{at\lambda_i w_i}\right) \\ &= e(g,g)^{\alpha s+at's}.e(g_{y_j},g)^s/\left(e(g_{y_j},g)^s e(g,g)^{ats}\right) \\ &= e(g,g)^{\alpha s+at's-ats}. \qquad (3) \end{aligned}$$

Proof: Assume that u_i, u_k are two users and y, y' are their associated leaf nodes in the binary tree. SK $= (\forall y_j \in path(y) : K_{y_j} = \left(g^{\alpha+at} g_{y_j}\right)^{1/\beta}, L = g^t, \forall x \in S : K_x = \mathcal{H}(x)^t))$ and SK$'$ $= (\forall y_j \in path(y') : K'_{y_j} = \left(g^{\alpha+at'} g_{y_j}\right)^{1/\beta}, L' = g^{t'}, \forall x \in S' : K'_x = \mathcal{H}(x)^{t'})$ are their attribute secret keys, respectively. A message $\mathcal{M}$ is encrypted so that $u_i \in RL$. This results in a ciphertext CT $= (C = \mathcal{M}.e(g,g)^{\alpha s}, C' = g^{s\beta}, D = g^r, \forall y_j \in cover(RL) : C_{y_j} = g^s_{y_j}, \forall i \in \{1,2,\ldots,l\} : C_i = g^{a\lambda_i}\mathcal{H}(\rho(i))^{-r})$. Note that CT does not have any C_{y_j} corresponding to the attribute secret key component K_{y_j} of u_i since $path(y) \cap cover(RL) = \emptyset$. However, there exists a component K'_{y_j} in the attribute secret key of the nonrevoked user u_k that corresponds to a C_{y_j} since $path(y') \cap cover(RL) \neq \emptyset$. Without the lose of generality,

Table 2. Symbols used in performance analysis and experiment

Symbol	**Meaning**
u	Total number of attributes in the system
a	Number of attributes in access structure $\mathbb{A}$
b	Number of attributes in user secret key
s	Required minimum number of attributes to satisfy policy
$\lvert\mathbb{G}_i\rvert$	Size of a single element in group $\mathbb{G}_i$
$\lvert\mathbb{K}\rvert$	Size of symmetric key
$\lvert p\rvert$	Size of a single element in $\mathbb{Z}_p$
C_i	Single exponentiation time in group $\mathbb{G}_i$
P	Computation time of a pairing operation
m	Total number of users in the group
r	Total nodes in *cover(RL)*

let us assume that S satisfies the access structure (M, ρ) but S' does not satisfy (M, ρ). This implies that u_i has enough attributes in his or her attribute secret keys to decrypt CT but u_k does not have enough attributes to do so. In order to successfully decrypt CT, u_i also needs K_{y_j} in his or her attribute secret keys that corresponds to C_{y_j}. As a result, u_i alone cannot decrypt CT with SK despite having enough attributes in it. However, u_i can collude with the nonrevoked user u_k to get K'_{y_j} from u_k's attribute secret key SK$'$ that corresponds to a C_{y_j} in CT. Then, u_i can try to decrypt CT by performing a series of computation as in equation 1, 2, and 3. However, it is apparent that u_i cannot successfully compute $W = e(g, g)^{\alpha s}$ and hence is unable to retrieve $\mathcal{M}$ from $C = \mathcal{M}e(g, g)^{\alpha s}$. This proves THEOREM 1.

8. THEORETICAL PERFORMANCE ANALYSIS

In this section, we compare our proposed ABE scheme with other related schemes in terms of storage, communication, and computational efficiency from the theoretical aspect. We also include [1] (referred to as BW) in the comparison as a baseline since our scheme is based on this. Table 4 illustrates different symbols that have been used for this purpose.

Table 3. Comparison of storage and communication efficiency with other schemes

Scheme	**Ciphertext size**	**Secret key size**	**Public key size**
Hur-I [4]	$(2a+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$(2b+1)\|\mathbb{G}_1\| + (\log m)\|\mathbb{K}\|$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Hur-II [5]	$(2a+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$2(b+1)\|\mathbb{G}_1$	$3\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
CryptCloud+ [10]	$(2a+5)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$(b+6)\|\mathbb{G}_1\| + 2\|p\|$	$(u+6)\|\mathbb{G}_1\| + 3\|p\|$
Flexible [6]	$(2a+6)\|\mathbb{G}_1\| + \|\mathbb{G}_T\| + 2\|p\|$	$(b+4)\|\mathbb{G}_1\| + 2\|p\|$	$3\|\mathbb{G}_1\| + 2\|\mathbb{G}_T\| + \|p\|$
UserCol [9]	$(2a+ra+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$4b\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$2(u+3)\|\mathbb{G}_1\| + 2\|\mathbb{G}_T\| + (2m-1)\|p\|$
BW [1]	$(a+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$(b+2)\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$
Ours	$(a+r+2)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$	$(b+1+\log m)\|\mathbb{G}_1\|$	$(2m+1)\|\mathbb{G}_1\| + \|\mathbb{G}_T\|$

8.1. STORAGE AND COMMUNICATION EFFICIENCY

The space efficiency comparison in terms of ciphertext, secret key, and public key size has been summarized in Table 5. The ciphertext size, secret key size, and public key size represent the storage cost required by the cloud, each user, and the attribute authority to store them, respectively. Additionally, they represent the communication cost when these are sent from one party to another. However, we do not consider here the communication cost that is associated with any intermediate step during the preparation of the ciphertext, secret key, or public key. For example, in order to achieve revocation, the data owner sends the whole ciphertext to the manager for re-encryption in UserCol[9]. As a result, the communication cost for sending the ciphertext to the cloud would be twice as much as what is shown in Table 5 for [9]. Similar intermediate steps are necessary in [6, 9, 5] during the secret key generation phase. Thus, the actual communication cost can be higher than what is shown in Table 5. However, there is no intermediate step in our proposed scheme, so the cost shown in Table 5 for our scheme is much closer to the real cost.

Compared to the baseline scheme [1], our scheme requires $r + 1$ and $\log m - 1$ additional group ($\mathbb{G}_1$) elements for the ciphertext and secret key, respectively. This is because the data owner has to create $r + 1$ additional group elements in the ciphertext out of which r elements are for *cover(RL)*. On the other hand, the AA has to create $\log m - 1$ additional group elements in the user's secret key along the *path* in the binary tree. Our scheme also needs $2m - 1$ additional group elements in the public parameter compared to the baseline scheme because a group element ($\mathbb{G}_1$) associated with every node in the binary tree is needed in the public parameter.

Among all collusion-resistant revocable schemes (ours and [10, 6, 9]), [9] has the biggest ciphertext because for each attribute present in the policy of the ciphertext, r additional group elements are needed. This results in a total of ra additional group elements in the ciphertext. How our scheme compares against [10, 6] in terms of ciphertext size, depends on the value of total number of attributes in the ciphertext (a) and the number of nodes in *cover(RL)* (r). If there are few members to revoke or if revoked members are not very sparsely distributed in the binary tree, then r will be much smaller, and hence our scheme will have smaller ciphertext. For bigger policies (i.e., bigger a), our ciphertext size will be relatively smaller. In terms of the secret key size, our scheme requires at most $\log m$ additional group elements compared to [10, 6, 9]. However, if the attribute secret key holds a lot of attributes, then [9] will have larger secret key size. For example, if there are 1000 users in the group and a user has 10 attributes in his or her attribute secret key, then there will be only 21 group elements in our secret key as opposed to 41 in [9]. Public key size increases proportionally with the number of total users for both our scheme and [9]. However, the total public parameter size of [9] is larger than ours.

Compared to [4] and [5] (not resistant against *type II* collusion attacks), our scheme has a larger public key size, as the public key includes an additional group element for each node in the binary tree. While our scheme requires $\log m$ additional group elements, [4] requires $\log m$ additional symmetric keys in the secret key because it includes $\log m$ KEK keys to each user's secret key along the *path* of the associated leaf node in the KEK key tree. However, [5] improves this by adding only a single group element per attribute in the secret key. How [4] and [5] compare against our scheme in terms of ciphertext size is similar to the logic we presented earlier while comparing our scheme with [10, 6].

Table 4. Comparison with other schemes in terms of computation cost.

Scheme	**Key generation**	**Encryption**	**Decryption**	**Key update**
Hur-I [4]	$2(b+1)C_1$	$(3a+1)C_1+C_T$	$C_1+(2s-1)C_T+(2s+1)P$	bC_1
Hur-II [5]	$(3b+5)C_1$	$(3a+2m+3)C_1+C_T$	$s(m+1)C_1+(2s-1)C_T+(3s+1)P$	bmC_1
CryptCloud+ [10]	$(b+13)C_1+C_T+(2b+7)P$	$(a+5)C_1+C_T$	$2C_1+sC_T+(2s+5)P$	$3mC_1$
Flexible [6]	$(2b+9)C_1+2P$	$2(a+3)C_1+2C_T$	$(2s+3)C_T+(2s+4)P$	$(2m+b+1)C_1+P$
UserCol [9]	$(4b+2)C_1$	$(3a+ra+1)C_1+C_T$	$(2s-1)C_T+(3s+1)P$	$(2m-1)C_1$
BW [1]	$(b+2)C_1$	$(2a+1)C_1+C_T$	$sC_T+(2s+1)P$	N/A
Ours	$(b+2+\log m)C_1$	$(2a+r+2)C_1+C_T$	$sC_T+2(s+1)P$	0

8.2. COMPUTATION COST ANALYSIS

We show the computation cost of our scheme and compare it with other schemes in Table 6. The computation cost has been expressed in terms of group exponentiation and pairing operation in a similar manner as in [18, 5, 4]. This is a reasonable consideration since these two operations dominate relatively lightweight hash, multiplication, division, and addition operations.

Compared to the baseline scheme [1], our scheme requires just one additional pairing operation (in $\mathbb{G}_1$) for decryption. For encryption, our scheme requires $r+1$ additional group exponentiation operations (in $\mathbb{G}_1$), out of which r is for creating r additional group elements for *cover(RL)*. On the other hand, our scheme requires $\log m$ additional group exponentiation operations (in $\mathbb{G}_1$) for the secret key generation. Among $\log m$ additional group operations (in $\mathbb{G}_1$), $\log m-1$ is for creating $\log m-1$ additional group elements in the user's secret key along the *path* of the associated leaf node in the binary tree.

Our key update cost is zero since we achieve revocation without affecting the secret key of other nonrevoked users. In contrast, all other revocation schemes require a significant amount of computation for key updating to achieve revocation. The number of group exponentiation operation (in $\mathbb{G}_1$) required for key update is proportional to the number of users in the system (except [4]).

Table 5. Parameter details for different pairing groups

Curves	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	p	k	*Security*
SS512	512	512	1024	160	2	80
MNT224	224	672	1344	224	6	100

In terms of decryption speed, our scheme outperforms all the revocation scheme, thanks to only one additional pairing operation for decryption compared to the baseline scheme [3]. We can see that [5] has the slowest decryption speed as the required number of group exponentiation operation (in $\mathbb{G}_1$) is proportional to the total number of users (m).

The number of group exponentiation operation (in $\mathbb{G}_1$) for our key generation algorithm increases logarithmically with the number of users (m). However, our key generation algorithm is still faster than that of [10]. This is because in addition to the group exponentiation operation, CryptCloud+ also requires $(2b + 7)P$ pairing operations, which is more expensive than the group exponentiation operation. Our key generation cost may even be lower than that of [4, 5, 6, 9] if the number of attributes in the secret key (b) is relatively high since $\log m$ increases very slowly.

Our encryption speed is much faster than that of [5, 9] since the required number of group exponentiation operation (in $\mathbb{G}_1$) is proportional to r and ra for [5] and [9], respectively. Among all the revocation schemes, [10] has the fastest encryption time.

9. EXPERIMENT

Implementation: We have implemented our scheme in `Charm` [20]. It is a Python-based framework developed for rapid prototyping of advanced cryptographic protocols. `Charm` uses PBC library [21] (written in C language) for low-level system calls including most expensive group exponentiation and pairing operations. As a result, cryptographic protocols written in `Charm` performs very close to the one written C language [22]. All hash functions were implemented using SHA224.

A detailed parameter description for our experimental setup is given in Table 3. SS512 is a super singular EC curve (with symmetric Type 1 pairing), and MNT224 is the Miyaji, Nakabayashi, Takano curves (with asymmetric Type 3 pairing). In Table 4, p = bit length of prime order p, k = embedding degree, *Security* is the security level in bits with respect to the discrete log problem, and the numbers associated with the curve name represent the base field size in bits (i.e., SS512 has a base field size of 512 bits). Though our ReVO-ABE construction is based on a symmetric pairing group ($\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$), we have tested our implementation in both symmetric and asymmetric group settings. `Charm` treats groups as asymmetric, though the actual setting depends on the type of underlying chosen curve. More specifically, there are three different groups ($\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$), and pairing is defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. We keep most of the terms in $\mathbb{G}_1$ while implementing our scheme in the asymmetric setting since operations in $\mathbb{G}_1$ are generally much faster than those in $\mathbb{G}_2$.

Testbed setup: We have conducted all the experiments on a Macbook Pro laptop with Intel® Core i7@2.2 GHz quad-core processor and 16 GB RAM running MacOS 10.14.6. We have used Python 3.7 and the PBC-0.5.14 library.

Results: We have mainly compared our key generation, encryption, decryptioin, and key update running time with CryptoCloud+ [10], Flexible [6], and UserCol [9]. We have also reported the key generation, encryption, and decryption running time of BW [1] scheme as a baseline comparison since our scheme is based on the BW scheme. However, we have not compared our experimental results with Hur I [4] and Hur II [5] since they are not collusion resistant (against *type II* attacks) and the encryption, decryption, and key update of Hur II takes much longer than the rest. The running time of each algorithm (key generation, encryption, decryption, and key update) were measured in SS512 and MNT224 curves. Each result reported here has been averaged over five individual runs.

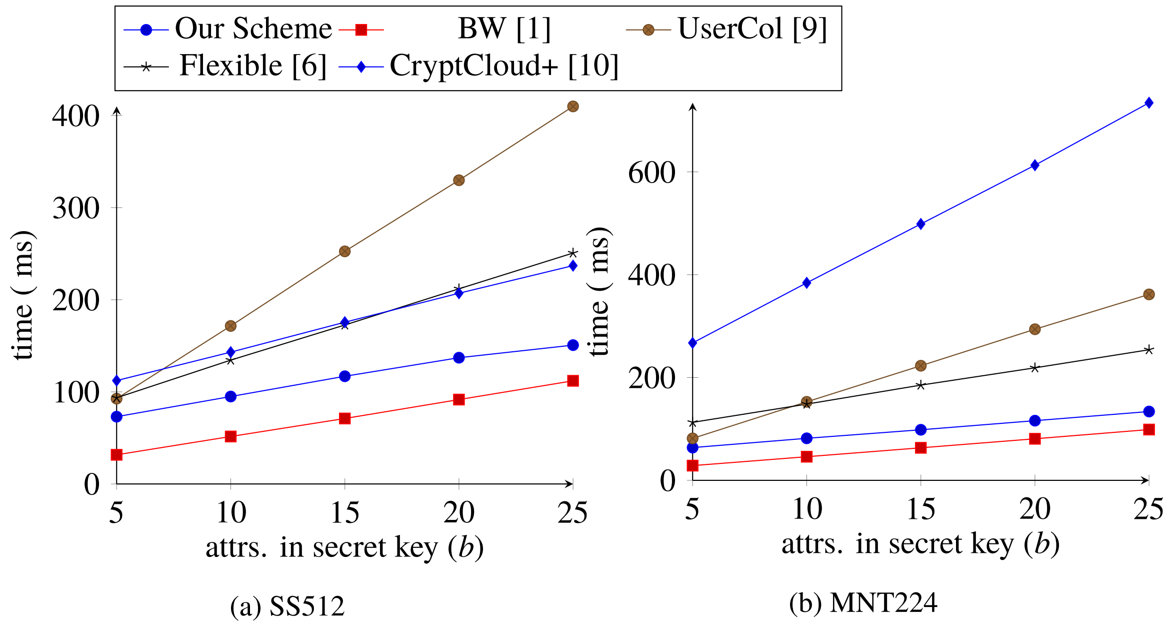


Figure 3. Key generation time

The running time of the key generation algorithm linearly increases with the total number of attributes in the secret key (b) (Figure 3). However, our scheme also has a logarithmic relationship with the total number of users (m). We have set the value of m to be 1000 and measured the running time of key generation algorithm by varying the value of b between 5 and 25. For all four curves, our running time is close to that of BW. Even for the value of b being as little as 5, our running time is lower than that of others. The performance of our key generation algorithm is even better compared to others for a higher value of b because with the higher value of b, the number of group exponentiation operation (in $\mathbb{G}_1$) increases at a slower rate compared to others (refer to Table 6). Interestingly, the running time of UserCol is the highest for the SS512 curve (Figure 3a). However, for MNT224 curve, its running time is much faster than that of CryptCloud+ because the pairing operations take more time in MNT224 curve compared to the SS512 curve, and CryptCloud+ requires $(2b + 7)$ pairing operations while UserCol requires only two pairing operations for the key generation.

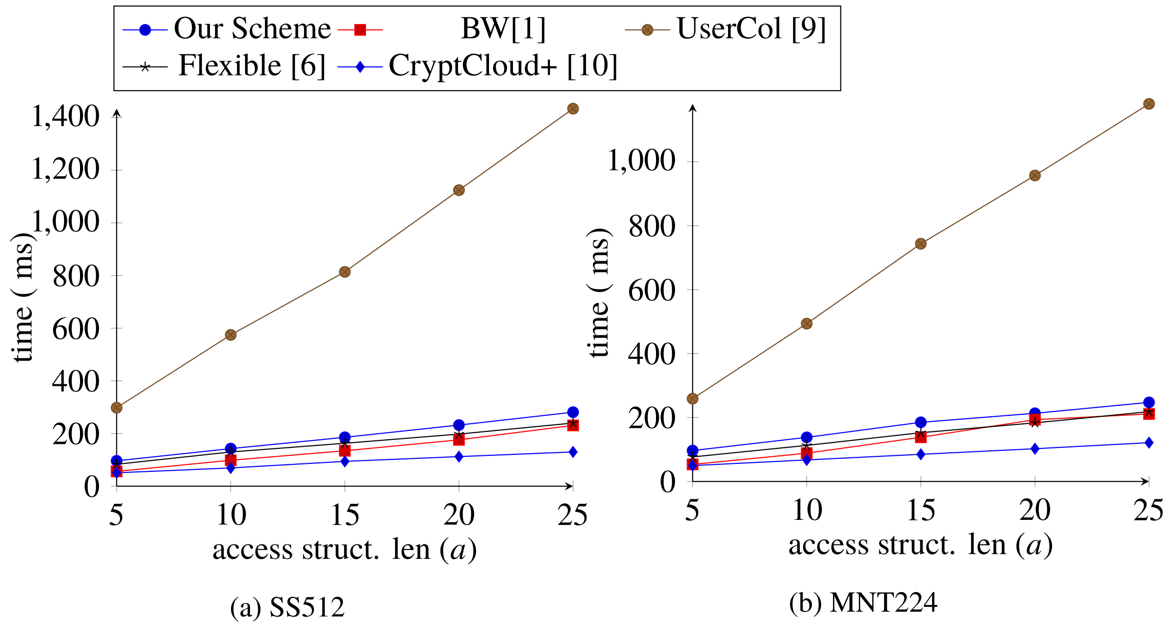
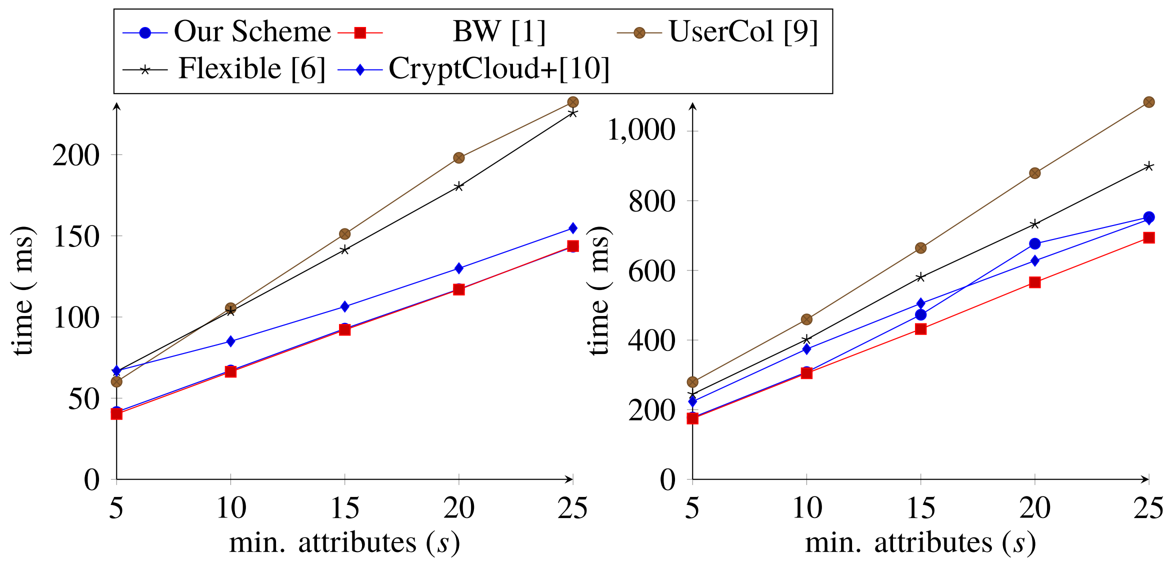


Figure 4. Encryption time

The encryption running time has been shown in Figure 5. UserCol has the fastest running time and CryptCloud+ has the slowest running time. The running time of encryption algorithm mainly depends on the access structure length or the number of attributes in the access structure (a). This is because the number of group exponentiation operations (in $\mathbb{G}_1$) has a linear relationship with a. However, the number of group exponentiation operations of our scheme and UserCol is also proportional to r and ar, respectively. As a result, the running time of UserCol is significantly faster than the rest. For our experiment, we have set the value of r to be 10 and varied the value of a between 5 and 25.

Figure 6 shows that the running time of the decryption algorithm has a linear relationship with the number of minimum required attributes to satisfy the access structure (s). We have varied the value of s from 5 to 25 at an interval of 5. Both the number of group exponentiation (in $\mathbb{G}_T$) and pairing operation has a linear relationship with s for all schemes. The pairing operation is much slower in MNT224 curve compared to the SS512 curve. Consequently, the decryption running time is higher in the MNT224 curve



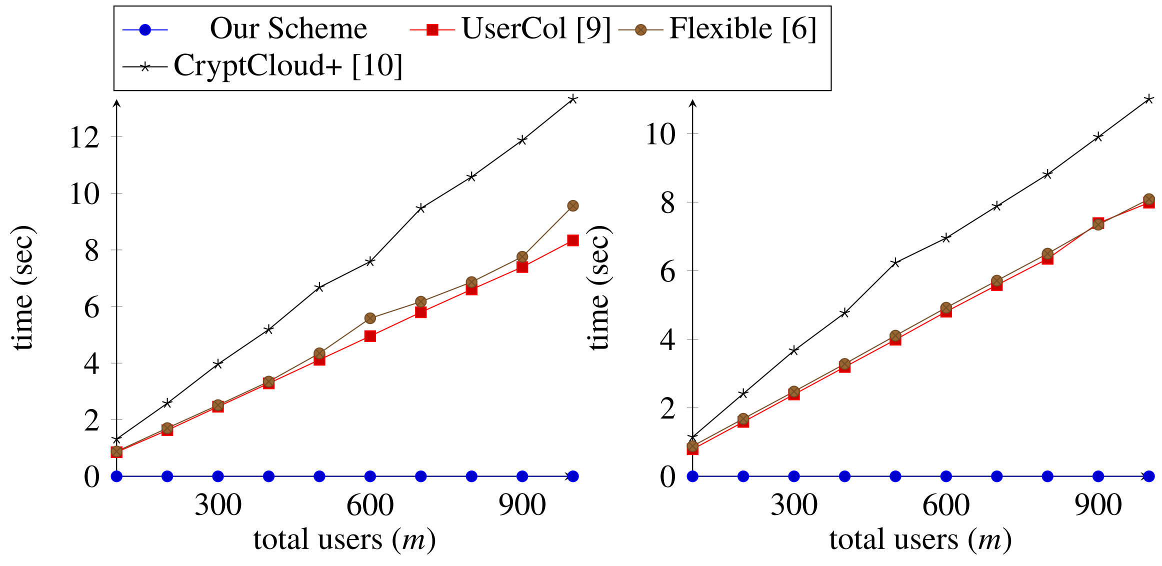
(a) SS512 (b) MNT224

Figure 5. Decryption time

for all schemes. Note that our decryption time is very close to the baseline scheme, as our decryption requires only one additional pairing operation compared to that of the baseline scheme.

The revocation in our scheme does not affect the secret key of any nonrevoked user. As a result, our revocation does not require any key update and hence the key update cost is zero for our scheme. However, other revocable schemes need to update the secret keys of existing nonrevoked users. In Figure 6, we have shown how the running time of the key update algorithm increases with the number of users (m) by varying the value of m from 100 to 1000. The running time of the key update algorithm increases linearly with m for all schemes except ours. In this comparison we exclude [1] as it does not support revocation, and hence the key update time is irrelevant.

Note that, the performance of all the algorithms do not have equal importance. For instance, key generation is normally a one time task. A file may be encrypted by the owner only once but is potentially decrypted many times by different users. If users are revoked frequently, then key update cost can be very critical. As a result, the performance



(a) SS512 (b) MNT224

Figure 6. Key update time

of decryption and key update are more important than that of other algorithms (e.g., key generation and encryption). From Figure 6 and 6 we can see that our decryption and key update algorithms outperform other schemes.

10. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed a revocable ABE scheme that is resistant against both *type I* and *type II* collusion attacks. Our scheme does not require any semi-trusted entity to achieve revocation. Moreover, the revocation does not affect the secret key of any non-revoked user, and hence the key update cost for revocation is zero in our scheme. We have also proposed a cloud-based secure data sharing scheme based on our proposed revocable ABE. Through security analysis, we have shown that our scheme is collusion (both *type I* and *type II*) resistant. It is evident from both theoretical performance analysis and experimental results that our scheme outperforms the most relatable ABE schemes. However, in this work we have only considered user-level revocation, but not attribute-level revocation. More specifically, using our proposed revocation scheme, a user (e.g., Eve) can be revoked from a ciphertext as a whole, but not some of his or her specific attributes (e.g., revoke only *new_release* and *movie* while keeping *tv_show, documentary,* and *scifi* intact). In the future, we want to extend our work to support attribute-level revocation. We have also limited the scope of this work to static access structures only. It will be an interesting extension if we could incorporate dynamic access policies in our scheme in the future.

REFERENCES

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

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, 2005.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007.

[4] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, 2011.

[5] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2013.

[6] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2018.

[7] H. Wang, Z. Zheng, L. Wu, and P. Li, "New directly revocable attribute-based encryption scheme and its application in cloud storage environment," *Cluster Computing*, vol. 20, no. 3, pp. 2385–2392, 2017.

[8] X. Wang and J. Fang, "A revocable outsourcing attribute-based encryption scheme," in *Cloud Computing, Security, Privacy in New Computing Environments: 7th International Conference, CloudComp 2016, and First International Conference, SPNCE 2016, Guangzhou, China, November 25–26, and December 15–16, 2016, Proceedings*, vol. 197, p. 145, Springer, 2017.

[9] J. Li, W. Yao, J. Han, Y. Zhang, and J. Shen, "User collusion avoidance cp-abe with efficient attribute revocation for cloud storage," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1767–1777, 2018.

[10] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K.-K. R. Choo, "Cryptcloud+: secure and expressive data access control for cloud storage," *IEEE Transactions on Services Computing*, 2018.

[11] Y. Jiang, W. Susilo, Y. Mu, and F. Guo, "Ciphertext-policy attribute-based encryption with key-delegation abuse resistance," in *Australasian Conference on Information Security and Privacy*, pp. 477–494, Springer, 2016.

[12] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *Advances in Cryptology–CRYPTO 2012*, pp. 199–217, Springer, 2012.

[13] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355–370, 2014.

[14] G. Zeng, "Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation," in *Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6-8, 2017, Proceedings*, vol. 10631, p. 172, Springer, 2018.

[15] B. Qin, Q. Zhao, D. Zheng, and H. Cui, "(dual) server-aided revocable attribute-based encryption with decryption key exposure resistance," *Information Sciences*, vol. 490, pp. 74–92, 2019.

[16] H. Cui, T. Hon Yuen, R. H. Deng, and G. Wang, "Server-aided revocable attribute-based encryption for cloud computing services," *Concurrency and Computation: Practice and Experience*, p. e5680, 2020.

[17] J. Li, Y. Shi, and Y. Zhang, "Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage," *International Journal of Communication Systems*, vol. 30, no. 1, p. e2942, 2017.

[18] B. Qin, R. H. Deng, S. Liu, and S. Ma, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1384–1393, 2015.

[19] R. Zhang, L. Hui, S. Yiu, X. Yu, Z. Liu, and Z. L. Jiang, "A traceable outsourcing cp-abe scheme with attribute revocation," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pp. 363–370, IEEE, 2017.

[20] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[21] B. Lynn, “The stanford pairing based crypto library,” *Privacy Preservation Scheme for Multicast Communications in Smart Buildings of the Smart Grid*, vol. 324, 2013.

[22] Y. Rouselakis and B. Waters, “Practical constructions and new proof methods for large universe attribute-based encryption,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 463–474, ACM, 2013.

II. ATTRIBUTE-BASED ENCRYPTION SCHEME FOR SECURE DATA SHARING IN CLOUD WITH FINE-GRAINED REVOCATION

MD Azharul Islam and Sanjay Madria

Department of Computer Science

Missouri University of Science and Technology

Rolla, Missouri 65401

Email: m.islam@mst.edu and madrias@mst.edu

ABSTRACT

Attribute-based encryption (ABE) is a prominent cryptographic tool for secure data sharing in the cloud because it can be used to enforce very expressive and fine-grained access control on outsourced data. The revocation in ABE remains a challenging problem as most of the revocation techniques available today, suffer from the collusion attack. The revocable ABE schemes which are collusion resistant require a semi-trusted manager to update the secret keys of nonrevoked users in order to achieve revocation. This introduces computation and communication overhead, and also increases the overall security vulnerability. In this paper, we propose two collusion resistant revocable ABE schemes that do not require any semi-trusted entity. Our first scheme supports revocation at the user-level that is equivalent to revoking all the attributes from a user. Our second scheme supports revocation at the attribute-level that enables more fine-grained revocation by allowing selective attribute(s) revocation from a user. We call them user-level revocable ABE (ULR-ABE) and attribute-level revocable ABE (ALR-ABE), respectively. For both the schemes, the secret keys of the nonrevoked users are never affected and the decryption algorithm has the same performance as the baseline ABE scheme. We are able to achieve these at the cost of some increase (compared to the baseline scheme) in the size of the secret key and the ciphertext.

Keywords: secure cloud data sharing, attribute based encryption, revocation

1. INTRODUCTION

Many companies and organizations often outsource their data to a public cloud to enjoy advantages such as availability, scalability, and lower maintenance cost offered by the cloud. However, confidentiality and access control of the outsourced data remains a concern since the data owner loses control over the data once it is uploaded to the cloud. Recently, attribute-based encryption (ABE) has become a very promising tool [1] to achieve confidentiality and fine-grained access control for data outsourcing in the cloud. ABE allows a data owner to encrypt his or her data using a policy expressed in terms of a set of attributes so that it can be decrypted only if the secret key has enough attributes to satisfy the policy. Let us consider the following motivational example.

1.1. A MOTIVATIONAL EXAMPLE

Suppose CryptoFlix is a Netflix-like streaming service that keeps all its media contents in a public cloud run by a third-party cloud service provider. Before outsourcing media files to the public cloud, it encrypts them using attribute-based encryption scheme. CryptoFlix offers two types of subscription plans: basic and premium. Basic and premium plans allow a subscriber to choose three and five attributes, respectively. Let us assume that two users, Alice and Bob, have a subscription for the basic plan, and another user, Eve has a subscription for the premium plan. Alice loves science fiction movies and documentaries, she gets decryption keys for attribute *movie, scifi,* and *documentary* from the attribute authority (a trusted entity responsible for generating and distributing attribute secret keys). Conversely, Bob loves tv shows and anime, and gets decryption keys for *new_release, tv_show,* and *anime* attributes. Let Eve has decryption keys for attribute *movie, tv_show, documentary, scifi,* and *new_release*. Attribute-level revocation would allow CryptoFlix to downgrade Eve's subscription plan from premium to basic by revoking attribute *movie* and *scifi* such that Eve can decrypt newly released documentary (encrypted under policy *new_release AND documentary*) but not newly released science fiction movie (encrypted under the policy *new_release AND movie AND scifi*). On the other hand, user-level revocation only allows CryptoFlix to revoke Eve entirely but not a subset of attributes from her. So, the user-level revocation is more suitable when Eve's subscription plan expires such that she cannot decrypt either newly released documentary or newly released science fiction movie.

Let us assume that Eve's premium subscription plan recently expired or she downgraded to the basic plan by unsubscribing attribute *movie* and *scifi*. In either case, when CryptoFlix encrypts the newly released science fiction movie XFiction under the policy *(new_release AND movie AND scifi)* and uploads it to a public cloud, Eve cannot decrypt it. Note that the other two users cannot decrypt XFiction either. However, the users can cooperate with each other and try to decrypt it by launching the following collusion attacks:

- *Type I attack:* Multiple users who individually do not have enough attributes to satisfy a policy cooperate with each other so that their collective attribute keys may satisfy the policy.
- *Type II attack:* A revoked user who cannot decrypt a file despite having enough attributes to satisfy the policy cooperates with a nonrevoked user to restore his or her decryption ability in order to decrypt the file.

In a *type I* attack, Alice and Bob would combine their attribute keys and try to decrypt XFiction. On the otherhand, in a *type II* attack, Eve would combine her attribute keys with a nonrevoked user (such as Bob) and try to decrypt XFiction. Not to mention CryptoFlix faces financial loss if any of the attacks becomes successful. Therefore, CryptoFlix must be resilient to both type of collusion attacks. This motivational example will be referred to repeatedly in the upcoming sections.

1.2. LIMITATIONS OF THE EXISTING SCHEMES

Collusion resistance is a fundamental security requirement of any ABE scheme as stated in [2]. Initially proposed ABE schemes such as [2, 3, 1] do not support revocation. If CryptoFlix were to use such an ABE scheme, it could not revoke Eve even if her subscription expired. As a result, such schemes are not suitable for a practical application like secure cloud data sharing. They are resistant to *type I* collusion attacks. However, *type II* collusion attacks do not apply to them as those schemes do not support revocation. The revocation is a challenging problem in ABE since the same attribute may be shared among different users. Hur et al. proposed a solution to the revocation problem in [4, 5] where they achieved attribute-level revocation. The proposed solution is based on the idea of attribute group. The user's secret key consists of two parts. One is associated with the user's attributes, and the other is associated with the attribute group. They are called decryption secret key (DSK) and key encryption key (KEK), respectively. The revocation is dictated by KEK and is independent of the user's DSK. Consequently, the KEK of one user works with the DSK of another user. Hence, the proposed revocable ABE scheme is not resistant against *type II* collusion attacks as a revoked user by colluding with a nonrevoked user can get the valid KEK and restore his or her decryption ability. This vulnerability was first pointed out by Li et al. [6]. Schemes such as [7, 8] also have the same vulnerability since these solutions are also based on the same idea. Li et al. refined their initial solution [6] in [9]. To revoke a user from the attribute group, the attribute manager (AM) updates the existing user's KEK keys. They bind a user's DSK with his or her KEK so that the KEK of one user does not work with the DSK of another user. This ensures that a revoked user cannot collude with a nonrevoked user to restore his or her decryption right. However, this scheme has the following limitations:

- Each time a user is revoked, all nonrevoked users' keys (KEK) are affected because DSK and KEK keys are tied together by a common secret exponent that is only known to a semi-trusted party called the attribute manager (AM). This exponent is common across attribute secret keys of all the users. To revoke a user, this exponent needs to be updated in the secret keys for all nonrevoked users.
- It requires the additional semi-trusted AM for updating all nonrevoked users' secret keys (KEK) and distributing them to the respective users. This not only adds a lot of overhead, but also increases the security vulnerability by adding an additional semi-trusted party to the system.
- AM also becomes the performance bottleneck as it needs to participate in the key generation, key update, encryption, and re-encryption stages. Key generation is an one-time operation. However, other operations occur very frequently, which can be a huge burden for a centralized entity like AM to handle.

By following the footstep of [9], CryptCloud+[10] also proposes a collusion-resistant revocable ABE scheme. In this scheme, the attribute authority has to periodically update the attribute secret keys of all the users according to a revocation list and send it to nonrevoked users, which is very inefficient. Cui et al. proposed a more secure and efficient method to update nonrevoked users' keys in server-aided revocable ABE (SR-ABE) scheme [11]. In SR-ABE, users have two types of keys: transformation key and secret decryption key. Transformation key is given to an untrusted server and the secret decryption key is given to the user. Attribute authority generates key updates for time period t which is sent to the untrusted server. Then, the untrusted server uses the key update and the user's transformation key to partially decrypt the ciphertext encrypted for the time period t. Finally, a user can fully decrypt it with the secret decryption key if he is not revoked for time period t. The updates are created in a way such that if a user is revoked by the attribute authority for time period t, he cannot decrypt any ciphertext created for that time period even if he colludes with the untrusted server. Additionally, it can resist both *type-I* and *type-II* collusion attack. However, SR-ABE does not allow attribute-level revocation. Besides, revocation is done centrally by the attribute authority. Hence, independent data owners do not have any control over revocation.

1.3. OUR CONTRIBUTION

In this paper, we propose two revocable ABE schemes that are resistant to both *type I* and *type II* collusion attacks. Our first scheme [12] supports user-level revocation, and called user-level revocable ABE (ULR-ABE). Our second scheme supports attribute-level revocation, and called, attribute-level revocable ABE (ALR-ABE). Unlike Li et al.'s schemes ([6] and [9]), our revocation does not affect the secret key of any nonrevoked user. Moreover, our schemes do not need the aid of any additional trusted entity, which minimizes the attack surface. For revocation, periodic key update like SR-ABE [11] is not required since data owners can independently revoke users according to their own revocation list.

Our revocation techniques can be applied to any ABE scheme that does not support revocation (e.g., [3, 1, 13]). However, in this paper, we choose the scheme proposed in [1] for several reasons. This scheme has a large universe construction (meaning that any arbitrary string can be used as an attribute) and it has been proven to be selectively secure in standard model (as opposed to the artificial random oracle model) under decisional q-Parallel Bilinear Diffie-Hellman exponent (decisional q-BDHE) assumption. These properties are more desirable in real-life application from both functional and security standpoints. Our ABE schemes also inherit these properties. We achieve revocation by modifying the core ABE secret key and ciphertext components and distributing them according to a binary tree. A data owner can revoke any user or attributes(s) from a user while encrypting a message using the minimum cover algorithm (Section 3.6) on the binary tree. More specifically, ULR-ABE scheme generates some redundant components in both ciphertext and user's secret key such that a non-revoked user has exactly one redundant key component that can decrypt one of the ciphertext components, while revoked user do not have any such component. On the other hand, in order to achieve attribute-level revocation, ALR-ABE scheme generates such redundant components in ciphertext and secret key for each attribute. Note that it is not trivial to generate such redundant components as we have to cryptographically bind the components together such that multiple users cannot combine their keys and be successful with any collusion attack. Hence, both ULR-ABE and ALR-ABE scheme require us to design new setup, encryption, keygen, and decryption algorithms. We briefly summarize our contribution as follows:

- We propose two revocable ABE schemes called ULR-ABE and ALR-ABE to support user-level revocation and attribute-level revocation, respectively while maintaining the important collusion resistance property (against both *type I* and *type II* collusion attacks).
- The schemes enable the data owner to revoke any user (ULR-ABE) or any number of attribute from any particular user (ALR-ABE) during encryption. Hence, we eliminate the requirement of any semi-trusted entity to achieve revocation by key updating.
- Revocation of our schemes never affect the secret keys of any nonrevoked user.
- Through proper security analysis, we prove that our proposed ULR-ABE and ALR-ABE schemes are collusion resistant.
- Through extensive theoretical and experimental performance analysis, we show that our schemes outperform recently proposed similar schemes in terms of decryption speed and key update cost.

Table 1. Comparison with related schemes in terms of security and functionality.

Scheme	Security assumption	Security Model	Collusion resistant	Revocation affects others' key	Fine-grained revocation
Hur-I [4]	Generic Group	RO	✗	✓	✓
Hur-II [5]	Generic Group	RO	✗	✓	✓
CryptCloud+ [10]	l-SDH	Standard	✓	✓	✗
Flexible [6]	Generic Group	RO	✓	✓	✗
UserCol [9]	Generic Group	RO	✓	✓	✓
SR-ABE [11]	decisional q-BDHE	Standard	✓	✓	✗
ULR-ABE [12]	decisional q-BDHE	Standard	✓	✗	✗
ALR-ABE	decisional q-BDHE	Standard	✓	✗	✓

2. RELATED WORKS

According to recent surveys such as [14] and [15], revocation is still considered as one of the most challenging problems in ABE. Revocable ABE was first addressed by Sahai et al. in [16], and then realized by Qin et al. in [17]. The attributes in both user's secret key and the access structure of a ciphertext are associated with different expiration time. The time specifies how long a secret key is allowed to decrypt a ciphertext. The cloud has to periodically update the access structure of a ciphertext using proxy re-encryption to enforce revocation. However, the problem is that, a colluding cloud can re-encrypt a ciphertext to an expiration time so that it can be decrypted by a user's expired secret key. The scheme proposed in [18] also incorporates the same idea of achieving revocation by updating the ciphertext. Instead of updating the ciphertext in a timely fashion, an aide server updates the ciphertext according to the data owner's provided revocation list. However, this scheme also suffers from a similar kind of collusion attack (server-revoked user) as [17]. Recently, [11] and [19] also proposed a revocable ABE scheme where the revocation is aided by an untrusted server. The untrusted server uses the transformation key and periodic key updates provided by the attribute authority to partially decrypt ciphertext for a specific time period. Nonrevoked users at the specific time period can fully decrypt the partially decrypted ciphertext. This scheme can prevent both type of collusion attacks. However, it still needs the aid of an additional entity (the aide server) to achieve revocation, and it does not support attribute-level revocation. More recently, Azhar et al. proposed a revocable ABE scheme that supports a multiple-group environment efficiently where users can move between different groups. However, this scheme also does not support attribute-level revocation, and they need the aid of two semi-trusted servers to achieve revocation. A recently proposed traceable ABE scheme [20, 21] focuses on revoking a user by finding the leaked keys rather than preventing a revoked-nonrevoked user collusion attack.

Hur et al. followed a different approach for revocation of ABE in [4] and later improved its security in [5]. According to their proposed solution, a user belongs to various attribute groups and the authorized set of attributes is determined by the attribute groups the user belongs to. A user's secret key consists of two parts: decryption secret key (DSK) and key encryption key (KEK), respectively. The DSK is associated with the user's attributes, while the KEK is associated with the user's attribute groups. Revocation is achieved by encrypting certain ciphertext components with KEK keys so that a user cannot decrypt the ciphertext without the appropriate KEK key despite having enough attributes in the DSK key. Li et al. first pointed out in [6] that DSK and KEK keys in Hur et al.'s proposed solution are independent of each other. As a result, a revoked user can collude with a nonrevoked user, and combine the nonrevoked user's KEK key with his or her own DSK keys to restore the decryption ability.

Schemes like [7, 8] also suffer from the same revoked-nonrevoked user collusion attack. Li et al. proposed an initial solution to solve this collusion problem in [6, 22] and later improved the security in [9]. The revoked and nonrevoked user collusion problem was solved by binding the DSK key with the KEK key so that one user's DSK key does not work with another user's KEK key. The limitation of the proposed scheme was that a semi-trusted attribute manager has to update all nonrevoked users' KEK keys for revocation, which not only adds a lot of overhead, but also adds additional security vulnerability as the attribute manager can collude with revoked users to restore their decryption ability. The attribute manager also needs to process every ciphertext, which can be a performance bottleneck. The solution proposed in CryptCloud+ [10] also relies on a semi-trusted entity for key update in revocation.

In Table 1, we compare our schemes with the most relatable ones in terms of security and functionalities. Both our schemes (ULR-ABE and ALR-ABE) and SR-ABE [11] are selectively secure in the standard model under the decisional q-Parallel Bilinear Diffie-Hellman Exponent (decisional q-BDHE) assumption [1]. CryptCloud+ is proven to be secure in the standard model based on the hardness of l-Strong Diffie-Hellman (l-SDH) assumption. Hur-I [4], Hur-II [5], Flexible [6], and UserCol [9] are secure in the random oracle (RO) model with generic group assumption. According to [1] the random oracle model is an artificial model and not desirable for real-life applications. Hur-I and Hur-II are not resistant against a revoked-nonrevoked user collusion attack (*type II*), while the rest are secure against this attack. Note that only in our scheme does the revocation not affect the secret keys of the existing nonrevoked users.

3. PRELIMINARIES

3.1. SYMBOLS AND NOTATIONS USED

In this paper, we use $\mathbb{G}$ and $\mathbb{G}_T$ to represent two multiplicative cyclic groups of prime order p, while g is a generator of $\mathbb{G}$. The symbol $\mathbb{Z}_p$ is used to denote the group of integers modulo p. We also make use of a *randomness extractor function*[23] defined as $\mathcal{F} : \mathbb{G}_T \to \mathbb{K}$, where $\mathbb{K}$ is the symmetric key space. The encryption and decryption functions of the symmetric encryption scheme are denoted as *Enc* and *Dec*, respectively.

3.2. BILINEAR MAP

A bilinear map is a function e defined as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and must have the following properties:

1. *Bilinearity:* For $\forall g_1, g_2 \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}_p$, the following relationship must always hold: $e\left(g_1^a, g_2^b\right) = e\left(g_1, g_2\right)^{ab}$

2. *Non-degeneracy:* $e(g_1, g_2) \neq 1$

3. *Computability:* group operations in $\mathbb{G}$ and e should be efficiently computable.

3.3. DECISIONAL Q-PARALLEL BILINEAR DIFFIE-HELLMAN EXPONENT ASSUMPTION

We review the definition of decisional q-BDHE assumption from [1]. Following the notations from Section 3.1, assume that a, s, and q exponents (e.g., $b_1, b_2, \ldots, b_q$) are randomly chosen from $\mathbb{Z}_p$. Then, according to the decisional q-BDHE assumption it is hard for any probabilistic polynomial time adversary to distinguish $e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element $e(g,g)^r \in \mathbb{G}_T$ if the adversary is provided with the vector $\vec{v} = g, g^s, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, g^{2q}, \forall 1 \leq i \leq q : g^{sb_i}, g^{a/b_i}, \ldots, g^{a^q/b_i}, g^{a^{q+2}/b_i}, \ldots, g^{a^{2q}/b_i}, \forall 1 \leq i, j \leq q, i \neq j : g^{asb_j/b_i}, \ldots, g^{a^q sb_j/b_i}$. The advantage ϵ of a probabilistic polynomial time algorithm $\mathcal{B}$ in solving the decisional q-BDHE problem is defined as $\epsilon \leq |\Pr[\mathcal{B}(\vec{v}, A = e(g,g)^{a^{q+1}s}) = 0] - \Pr[\mathcal{B}(\vec{v}, A = e(g,g)^{a^r}) = 0]|$.

3.4. ACCESS STRUCTURE

Let $\mathbb{P} = \{P_1, P_2, \cdot, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathbb{P}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure is a collection $\mathbb{A}$ of non-empty subsets of $\mathbb{P}$ (i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdot, P_n\}} \backslash \{\emptyset\}$). The sets in $\mathbb{A}$ are called authorized sets, and sets not in $\mathbb{A}$ are called unauthorized sets. In our context, the role of the parties is defined by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes.

3.5. LINEAR SECRET SHARING SCHEME (LSSS)

Let p be a large prime. Then, a secret sharing scheme $\prod$ over a set of parties P is called linear (over $\mathbb{Z}_p$) if

- The shares of each party form a vector over $\mathbb{Z}_p$.
- There exists a matrix M with l rows and n columns called the share-generating matrix for $\prod$. For all $i = 1, 2, \ldots, l$, the i^{th} row of M, we define a function ρ such that $\rho(i)$ maps row i of matrix M to an associated party. When we consider the column vector $\vec{v} = (s, R_2, \ldots, R_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $R_2, \ldots, R_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of l shares of the secret s according to $\prod$. The share $(Mv)_i$ belongs to party $\rho(i)$.

Any linear secret sharing scheme defined above has the following *linear reconstruction* property: Let $\prod$ be an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \ldots, l\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to $\prod$, then $\sum_{i \in I} w_i \lambda_i = s$. These constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix M.

3.6. MINIMUM COVER

Let $\mathcal{T}$ be a full binary tree with m leaf nodes. Nodes in $\mathcal{T}$ are labelled as y_j, and each leaf node is associated with a user labelled as $u_1, u_2, \ldots, u_m$. We use *path(u_k)* to denote all the nodes in the path from the root to the associated leaf node of u_k. Let the set $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and *RL* represent the set of all users and revoked users, respectively. Then, we can apply subset cover to get the minimum number of tree nodes (we call it *cover(RL)*) that cover $\mathcal{U} - RL$ (i.e., all nonrevoked users). The algorithm to find *cover(RL)* is as follows:

- $\forall u_j \in RL$, color all the nodes of *path(u_j).*
- *cover(RL)* is the set of all uncolored nodes that are direct children of the colored nodes.

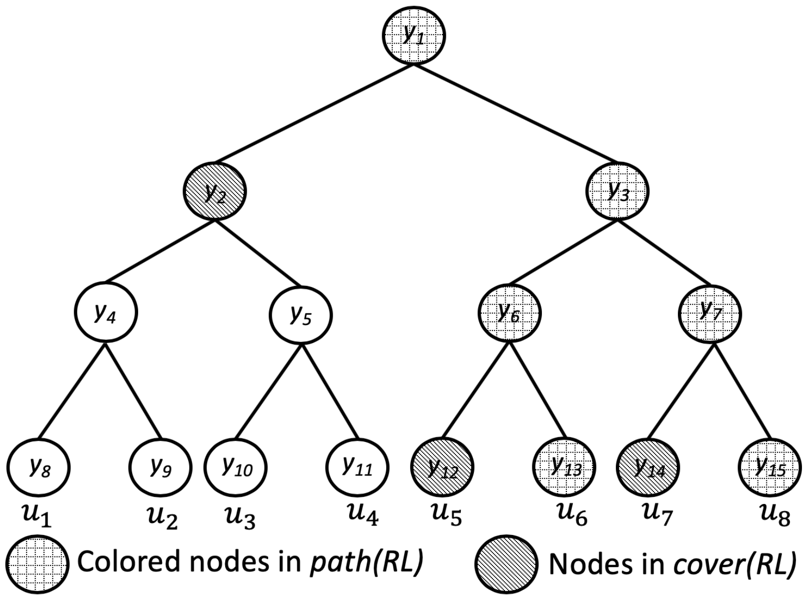


Figure 1. Finding minimum cover

Figure 1 is an example of a binary tree $\mathcal{T}$ with 8 leaf nodes and 8 associated users (i.e., $\mathcal{U} = \{u_1, u_2, \ldots, u_8\}$). If $RL = \{u_6, u_8\}$, then *Cover(RL)* $= \{y_2, y_{12}, y_{14}\}$. Node that six nonrevoked users (i.e., $\{u_1, u_2, u_3, u_4, u_5, u_7\}$) are covered with just three nodes.

4. SYSTEM ARCHITECTURE AND ADVERSARIAL MODEL

4.1. SYSTEM ARCHITECTURE

A high-level system architecture of our secure data sharing scheme has been presented in Figure 2. Our architecture has four main entities: the attribute authority (AA), the cloud service provider (CSP), the data owner, and the data user. The attribute authority is responsible for managing all the attributes, and it creates and distributes attribute secret keys to the users according to their authorized attribute set. It also creates and publishes the public parameters. The CSP manages the public cloud where the encrypted data is outsourced and stored. Data owners encrypt their files using an our ULR-ABE or ALR-ABE scheme and uploads the files to the public cloud so that they are always available for the

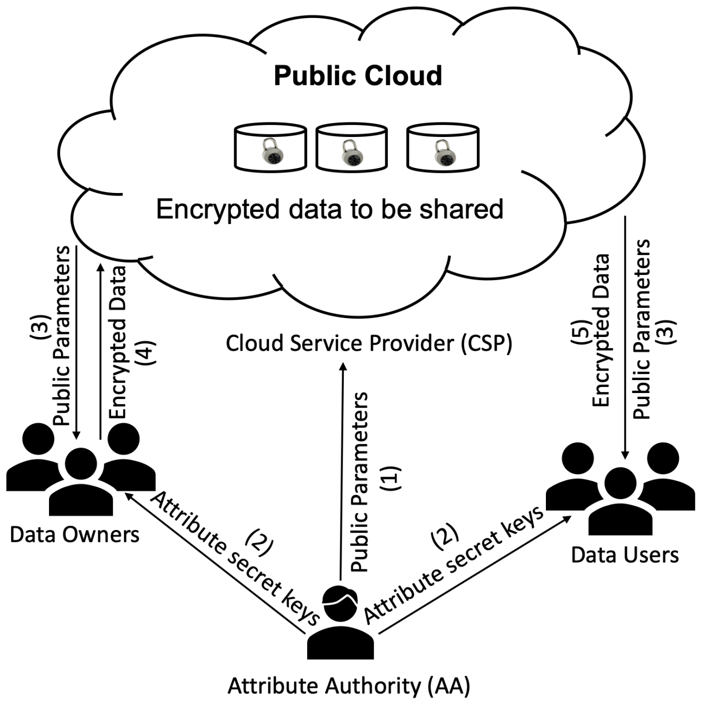


Figure 2. System Architecture

data users. Once the data is uploaded in the cloud, the data users can download and decrypt it anytime if they have enough attributes in their attribute secret keys. Note that a user may have the dual role of a data owner and a data user.

4.2. ADVERSARIAL MODEL

4.2.1. Security Assumptions. We consider the CSP to be an honest but curious entity that is a standard practice in revocable ABE literature [5, 6, 9, 7, 24]. This implies that the CSP properly follows the protocols of our scheme (i.e., it honestly performs tasks like storing and updating encrypted files as per the data owners' request and letting the data users download encrypted files upon request). The CSP does not tamper with any stored information in the cloud. However, the CSP is open to deduce any plaintext information from the stored encrypted files and public parameters on its own. In previously proposed revocation schemes such as [4, 5, 6, 9], a semi-trusted entity (manager) needs to update users' secret keys and ciphertext in order to achieve revocation. This design introduces additional security vulnerabilities, as a compromised manager can update secret keys and ciphertext in a way that restores a revoked user's decryption right. Hence, user and manager collusion is not allowed. On the other hand in SR-ABE [11], the attribute authority revokes users by generating periodic key update for the nonrevoked users such that the revocation process can be carried out by an untrusted server with the key update. However, we do not need any semi-trusted manager or periodic key update to achieve revocation since the revocation ability is given to the data owner, who can decide whom to revoke during encryption. We assume that the attribute authority is a trusted entity, and it distributes attribute secret keys to users via a secure channel such as SSL.

4.2.2. Adversaries and Attacks. A dishonest data user is the main adversary of our system. A dishonest data user can be either revoked or nonrevoked. The goal of such adversaries is to decrypt a ciphertext that cannot be decrypted individually, either because they do not have enough attributes in their attribute secret keys or because some of their mandatory attribute(s) (attribute(s) mandatory to satisfy the access structure) are revoked or they are entirely revoked as a user (but may have enough attributes). To achieve this goal, a dishonest data user colludes with other dishonest data user(s) and launches *type I* and *type II* attacks.

5. USER-LEVEL REVOCABLE ABE (ULR-ABE)

In this section we give details of our user-level revocable ABE scheme. We first give the definition of ULR-ABE, followed by its security model. Finally, we give the detailed construction of the scheme.

5.1. DEFINITION OF ULR-ABE

Our ULR-ABE scheme is consisted of four algorithms defined as follows:

• (PK, MK) $\longleftarrow$ `Setup` $(Att_{\max}, l_{\max}, m)$: The setup algorithm takes as input the maximum number of attributes allowed in a secret key, the maximum number of columns possible in a LSSS matrix, and the total number of users in the system denoted as $Att_{\max}$, $l_{\max}$, and m, respectively. It outputs the public key PK, and the master secret key MK.

• CT $\longleftarrow$ `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*): The inputs to the encryption algorithm are the public key PK, the LSSS access structure *(M, ρ)*, the message to be encrypted $\mathcal{M}$, and the revocation list *RL*. The algorithm outputs a ciphertext CT so that no user in *RL* can decrypt CT even if the attribute set S satisfies the access structure.

• SK $\longleftarrow$ `Keygen` (PK, MK, S, u_k): The key generation algorithm takes as input the public key PK, the master secret key MK, the user's authorized attribute set S, and the user's identifier u_k. It outputs the user's secret key SK.

• $\mathcal{M}/\perp$ $\longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public key PK, the user's secret key SK, and the ciphertext CT. It outputs the plaintext message $\mathcal{M}$ if the user does not belong to the corresponding revocation list *RL* of CT and his or her authorized attribute set S satisfies the access structure. The decryption algorithm outputs $\perp$ otherwise.

5.2. SECURITY MODEL

We formalize the security model of ULR-ABE scheme by the following IND-CPA (indistinguishable chosen plaintext attack) game.

Init. The adversary $\mathcal{A}$ commits to an access structure (M^*, ρ) by giving it to the challenger.

Setup. The challenger runs the `Setup` $(Att_{\max}, l_{\max}, m)$ algorithm to generate PK, SK, and sends PK to $\mathcal{A}$.

Phase I Query. The adversary $\mathcal{A}$ repeatedly makes $q1$ private key queries for the user-authorized attribute set tuples as in $Q_1 = (u_1, S_1), Q_2 = (u_2, S_2), \ldots, Q_{q1} = (u_{q1}, S_{q1})$. The challenger calls `Keygen` (PK, MK, S_k, u_k) for each query $Q_k = (u_k, S_k)$, and sends the secret key SK$_k$ to $\mathcal{A}$.

Challenge. $\mathcal{A}$ selects two equal size messages $\mathcal{M}_0, \mathcal{M}_1$, a revocation list RL^*, and sends them to the challenger. Additionally, $\mathcal{A}$ also sends to the challenger the committed access structure (M^*, ρ). The challenger chooses a bit $b \in \{0, 1\}$ by flipping a random coin, and runs `Encrypt` (PK, (M^*, ρ), $\mathcal{M}_b$, RL^*). The challenger then sends the output ciphertext CT* to $\mathcal{A}$. The constraint is that none of the attribute sets (e.g., $S_1, S_2, \ldots, S_{q1}$) in phase I query satisfy the access structure (M^*, ρ).

Phase II Query. $\mathcal{A}$ adaptively makes private key queries for tuples $Q_{q1+1} = (u_{q1+1}, S_{q1+1}), Q_{q1+2} = (u_{q1+2}, S_{q1+2}), \ldots, Q_q = (u_q, S_q)$ with the restriction that none of the attribute sets in these tuples (e.g., $S_{q1+1}, S_{q1+2}, \ldots, S_q$) satisfy the committed access structure (M^*, ρ).

Guess. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for b.

The advantage of the adversary $\mathcal{A}$ in the above game is defined as $Adv = |\Pr[b' = b] - 1/2|$. By allowing $\mathcal{A}$ to do decryption queries in phase I and phase II query stage, this security model can be easily extended to chosen-ciphertext attack (CCA).

Definition 2 *ULR-ABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the IND-CPA game.*

5.3. CONSTRUCTION OF ULR-ABE SCHEME

The detailed construction of ULR-ABE scheme is given as follows:

• (PK, MK) $\longleftarrow$ `Setup` ($Att_{\max}, l_{\max}, m$): The setup algorithm takes as input $Att_{\max}$, the maximum number of attributes any user's secret key may have; $l_{\max}$, the maximum number of columns any LSSS matrix M may have; and m, the total number of users. It outputs the public and the master secret key PK and SK, respectively. The setup algorithm first chooses a group $\mathbb{G}$ of prime order p with a generator g and defines a bilinear map as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We assume that attributes can be represented in $\mathbb{Z}_p$. In practice, a collision-resistant hash function can be used to transform any string attribute into $\mathbb{Z}_p$. The setup algorithm then randomly chooses $a, \alpha, \beta \in \mathbb{Z}_p$. It also utilizes a hash function defined as $\mathcal{H} : \mathbb{Z}_p \to \mathbb{G}$. The hash function is realized by choosing a polynomial $\mathcal{L}(x) \in \mathbb{Z}_p$ of degree $N = Att_{\max} + l_{\max} - 1$ and computing $h_0 = \mathcal{H}(0) = g^{\mathcal{L}(0)}, h_1 = \mathcal{H}(1) = g^{\mathcal{L}(1)}, \ldots, h_N = \mathcal{H}(N) = g^{\mathcal{L}(N)}$. With these $N + 1$ values, one can compute $h_x = \mathcal{H}(x) = g^{\mathcal{L}(x)}$ for any $x \in \mathbb{Z}_p$ by using interpolation.

The AA then creates a full binary tree $\mathcal{T}$ of m leaves and associates each user u_j to a different leaf node. For each node y_i in $\mathcal{T}$, AA randomly chooses $g_{y_i} \in \mathbb{G}$. Finally, AA publishes the public key as PK $= \left(\mathcal{T}, \mathbb{G}, \mathbb{G}_T, e(g, g)^{\alpha}, g^{a}, g^{\beta}, h_0, h_1, \ldots, h_N\right)$ and sets the master secret key as MK $= (g^{\alpha}, \beta)$ and keeps it secret.

• CT $\longleftarrow$ `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*): The encrypt algorithm takes as input the public parameters PK, LSSS access structure *(M, ρ)*, the plaintext message $\mathcal{M} \in \mathbb{G}_T$, and a revocation list *RL*. It outputs the ciphertext (CT). In the LSSS access structure *(M, ρ)*, *M* is an $l \times n$ matrix and ρ is a function that associates rows of *M* to attributes. In this construction, ρ is limited to be an injective function (i.e., an attribute can be associated with at most one row of *M*). The algorithm chooses a random vector $\vec{v} = (s, r_2, \ldots, r_n) \in \mathbb{Z}_p^n$. These values are used to share the random encryption exponent s. For $\forall i \in \{1, 2, \ldots, l\}$ it computes $\lambda_i = \vec{v}.M_i$, where M_i is the vector corresponding to the i^{th} row of M. Finally, the algorithm finds *cover(RL)*, randomly chooses an exponent $r \in \mathbb{Z}_p$, and computes the ciphertext as CT $= (C = \mathcal{M}.e(g, g)^{\alpha s}, C' = g^{s\beta}, D = g^{r}, \forall y_j \in$ *cover(RL)* $: C_{y_j} = g_{y_j}^{s}, \forall i \in \{1, 2, \ldots, l\} : C_i = g^{a\lambda_i}\mathcal{H}(\rho(i))^{-r})$. We assume that the LSSS access structure is implicitely included in CT.

• SK $\longleftarrow$ `Keygen` (PK, MK, S, u_k): The keygen algorithm takes as input PK, MK, a set S of attributes the user is authorized for, and the user's identifier u_k and outputs the attribute secret key SK. Let u_k's associated leaf node in $\mathcal{T}$ be y. The algorithm finds *path(y)*, chooses a random $t \in \mathbb{Z}_p$, and creates the private key as SK $= (\forall y_j \in$ *path(y)* $:$ $K_{y_j} = \left(g^{\alpha+at} g_{y_j}\right)^{1/\beta}, L = g^t, \forall x \in S : K_x = \mathcal{H}(x)^t)$.

• $\mathcal{M}/\perp \longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public paprameters PK, the user's attribute secret key SK for an attribute set S, and the ciphertext CT for a LSSS access structure *(M, ρ)*. Assume that S satisfies the access structure and $I \subset \{1, 2, 3, \ldots, l\}$ is defined as $I = \{i : \rho(i) \in S\}$. The algorithm finds the constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of a secret s according to M, then $\sum_{i \in I} w_i \lambda_i = s$. Note that these constants $\{w_i\}$ can be found in polynomial time in the size of M and there could potentially be different ways of choosing such $\{w_i\}$. The algorithm then computes the following:

$$\begin{aligned}
P &= e(K_{y_j}, C') = e\left(\left(g^{\alpha+at} g_{y_j}\right)^{1/\beta}, g^{\beta s}\right) \\
&= e(g,g)^{\alpha s + ats} . e(g_{y_j}, g)^s \\
Q &= e(C_{y_j}, g) = e(g_{y_j}, g)^s \\
W &= P \Big/ \left(Q \prod_{i \in I} \left(e(C_i, L) e(D, K_{\rho(i)}) \right)^{w_i} \right) \\
&= P \Big/ \left(Q \prod_{i \in I} e(g,g)^{at\lambda_i w_i} \right) \\
&= e(g,g)^{\alpha s + ats} . e(g_{y_j}, g)^s / \left(e(g_{y_j}, g)^s e(g,g)^{ats} \right) \\
&= e(g,g)^{\alpha s}.
\end{aligned}$$

Then, the decryption algorithm retrieves the plaintext message $\mathcal{M}$ as in $C/W = \mathcal{M}$, and outputs it. If S does not satisfy the access structure, or the owner of SK belongs to the revocation list *RL*, then the decryption algorithm outputs $\perp$.

6. ATTRIBUTE-LEVEL REVOCABLE ABE (ALR-ABE)

In ULR-ABE scheme, our strategy to achieve revocation without the aid of any semi-trusted entity is to create a ciphertext component C_{y_j} for each node in *cover(RL)* and create a secret key component K_{y_j} for each node in the path from the user's associated leaf node to the root of the binary tree. For any revoked user, there is no common node in the binary tree between the user's associated path and *cover(RL)*. Hence, the revoked user does not have any K_{y_j} in the secret key that works with any C_{y_j} in the ciphertext. For our ALR-ABE scheme, we apply the similar technique on each attribute in order to enable attribute revocation. However, applying this technique is not trivial as we have to cryptographically bind together both secret key components and ciphertext components such that malicious users cannot combine their keys to decrypt unauthorized ciphertext.

In the following, we first give the definition of our proposed ALR-ABE scheme and its security model. Then, we give the detailed construction of the scheme.

6.1. DEFINITION OF ALR-ABE

Our ALR-ABE scheme is consisted of four algorithms defined as follows:

• (PK, MK) $\longleftarrow$ `Setup` $(Att_{\max}, l_{\max}, m)$: The setup algorithm takes as input the maximum number of attributes allowed in a secret key, the maximum number of columns possible in a LSSS matrix, and the total number of users in the system denoted as $Att_{\max}$, $l_{\max}$, and m, respectively. It outputs the public key PK, and the master secret key MK.

• CT $\longleftarrow$ `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*): The inputs to the encryption algorithm are the public key PK, the LSSS access structure *(M, ρ)*, the message to be encrypted $\mathcal{M}$, and a set of revocation list $RL = \{RL_1, RL_2, \ldots, RL_l\}$, where where RL_k is the revocation list for the attribute $\rho(k)$. The algorithm outputs a ciphertext CT. If a user belongs to RL_k, he or she cannot decrypt attribute $\rho(k)$ of the LSSS access structure associated with CT.

• SK $\longleftarrow$ `Keygen` (PK, MK, S, u_k): The key generation algorithm takes as input the public key PK, the master secret key MK, the user's authorized attribute set S, and the user's identifier u_k. It outputs the user's secret key SK.

• $\mathcal{M}/\perp \longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public key PK, the user's secret key SK, and the ciphertext CT. It outputs the plaintext message $\mathcal{M}$ if the user has enough nonrevoked attributes in his or her authorized attribute set S to satisfy the access structure of CT. The decryption algorithm outputs $\perp$ otherwise.

6.2. SECURITY MODEL

The security model of ALR-ABE scheme can be formalized by the following IND-CPA game.

Init. The adversary $\mathcal{A}$ commits to an access structure (M^*, ρ) by giving it to the challenger.

Setup. The challenger runs the `Setup` ($Att_{\max}, l_{\max}, m$) algorithm to generate PK, SK, and sends PK to $\mathcal{A}$.

Phase I Query. The adversary $\mathcal{A}$ repeatedly makes $q1$ private key queries for the user-authorized attribute set tuples as in $Q_1 = (u_1, S_1), Q_2 = (u_2, S_2), \ldots, Q_{q1} = (u_{q1}, S_{q1})$. The challenger calls `Keygen` (PK, MK, S_k, u_k) for each query $Q_k = (u_k, S_k)$, and sends the secret key SK$_k$ to $\mathcal{A}$.

Challenge. $\mathcal{A}$ selects two equal size messages $\mathcal{M}_0, \mathcal{M}_1$, a set of revocation lists RL^*, and sends them to the challenger. Additionally, $\mathcal{A}$ also sends to the challenger the committed access structure (M^*, ρ). The challenger chooses a bit $b \in \{0, 1\}$ by flipping a random coin, and runs `Encrypt` (PK, (M^*, ρ), $\mathcal{M}_b$, RL^*). The challenger then sends the output ciphertext CT* to $\mathcal{A}$. The constraint is that none of the attribute sets (e.g., $S_1, S_2, \ldots, S_{q1}$) in phase I query satisfy the access structure (M^*, ρ).

Phase II Query. $\mathcal{A}$ adaptively makes private key queries for tuples $Q_{q1+1} = (u_{q1+1}, S_{q1+1}), Q_{q1+2} = (u_{q1+2}, S_{q1+2}), \ldots, Q_q = (u_q, S_q)$ with the restriction that none of the attribute sets in these tuples (e.g., $S_{q1+1}, S_{q1+2}, \ldots, S_q$) satisfy the committed access structure (M^*, ρ).

Guess. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for b.

The advantage of the adversary $\mathcal{A}$ in the above game is defined as $Adv = |\Pr[b' = b] - 1/2|$. By allowing $\mathcal{A}$ to do decryption queries in phase I and phase II query stage, this security model can be easily extended to chosen-ciphertext attack (CCA).

Definition 3 *Our proposed ALR-ABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the IND-CPA game.*

6.3. CONSTRUCTION OF ALR-ABE SCHEME

The detailed construction of our proposed ALR-ABE scheme is as follows:

• (PK, MK) $\longleftarrow$ `Setup` ($Att_{\max}, l_{\max}, m$): The setup algorithm takes as input $Att_{\max}$, the maximum number of attributes any user's secret key may have; $l_{\max}$, the maximum number of columns any LSSS matrix M may have; and m, the total number of users. It outputs the public and the master secret key PK and SK, respectively. The setup algorithm first chooses a group $\mathbb{G}$ of prime order p with a generator g and defines a bilinear map as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We assume that attributes can be represented in $\mathbb{Z}_p$. In practice, a collision-resistant hash function can be used to transform any string attribute into $\mathbb{Z}_p$. The setup algorithm then randomly chooses $a, \alpha, \beta \in \mathbb{Z}_p$. It also utilizes a hash function defined as $\mathcal{H} : \mathbb{Z}_p \to \mathbb{G}$ which is similar to that of ULR-ABE scheme.

The AA then creates a full binary tree $\mathcal{T}$ of m leaves and associates each user u_j to a different leaf node. For each node y_i in $\mathcal{T}$, AA randomly chooses $g_{y_i} \in \mathbb{G}$. Finally, AA publishes the public key as PK $= (\mathcal{T}, \mathbb{G}, \mathbb{G}_T, e(g, g)^{\alpha}, g^a, h_0, h_1, \ldots, h_N)$ and sets the master secret key as MK $= g^{\alpha}$ and keeps it secret.

• CT $\longleftarrow$ `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*): The encrypt algorithm takes as input the public parameters PK, LSSS access structure *(M, ρ)*, the plaintext message $\mathcal{M} \in \mathbb{G}_T$, and a set of revocation lists *RL*. It outputs the ciphertext (CT). In the LSSS access structure *(M, ρ)*, *M* is an $l \times n$ matrix and ρ is a function that associates rows of *M* to attributes. In this construction, ρ is limited to be an injective function (i.e., an attribute can be associated with at most one row of *M*). The algorithm chooses a random vector $\vec{v} = (s, R_2, \ldots, R_n) \in \mathbb{Z}_p^n$. These values are used to share the random encryption exponent s. For $\forall i \in \{1, 2, \ldots, l\}$ it computes $\lambda_i = \vec{v}.M_i$, where M_i is the vector corresponding to the i^{th} row of M. Let us assume that $RL = \{RL_1, RL_2, \ldots, RL_l\}$, where RL_k is the revocation list for the attribute $\rho(k)$. The algorithm finds *cover(RL)*$_k$ for each attribute $\rho(k)$, randomly chooses exponents $r_1, r_2, \ldots, r_l \in \mathbb{Z}_p$, and computes the ciphertext as CT $= (C = \mathcal{M}.e(g, g)^{\alpha s}, C' = g^s, ((\forall y_j \in$ *cover(RL$_1$)* $: C_{1,y_j} = g^{a\lambda_1}\mathcal{H}(\rho(1))^{-r_1}(g_{y_j})^{-r_1}, D_1 = g^{r_1}), \ldots, (\forall y_j \in$ *cover(RL$_l$)* $: C_{l,y_j} = g^{a\lambda_l}\mathcal{H}(\rho(l))^{-r_l}(g_{y_j})^{-r_l}, D_l = g^{r_l})$. We assume that the LSSS access structure is implicitely included in CT.

• SK $\longleftarrow$ `Keygen` (PK, MK, *S*, u_k): The keygen algorithm takes as input PK, MK, a set *S* of attributes the user is authorized for, and the user's identifier u_k and outputs the attribute secret key SK. Let u_k's associated leaf node in $\mathcal{T}$ be y. The algorithm finds *path(y)*, chooses a random $t \in \mathbb{Z}_p$, and creates the private key as SK $= (K = g^{\alpha + at}, L = g^t, \forall x \in S, \forall y_j \in$ *path(y)* $: K_{x,y_j} = (H(x)g_{y_j})^t)$.

• $\mathcal{M}/\perp \longleftarrow$ `Decrypt` (PK, SK, CT): The decryption algorithm takes as input the public paprameters PK, the user's attribute secret key SK for an attribute set *S*, and the ciphertext CT for a LSSS access structure *(M, ρ)*. Assume that *S* satisfies the access structure and $I \subset \{1, 2, 3, \ldots, l\}$ is defined as $I = \{i : \rho(i) \in S\}$. The algorithm finds the constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of a secret s according to *M*, then

$\sum_{i\in I} w_i\lambda_i = s$. Note that these constants $\{w_i\}$ can be found in polynomial time in the size of M and there could potentially be different ways of choosing such $\{w_i\}$. The algorithm then computes the following:

$$\begin{aligned}
P &= e(K, C') = e\left(g^{\alpha+at}, g^s\right)\\
&= e(g,g)^{\alpha s} e(g,g)^{ats}.\\
Q_i &= e(C_{i,y_j}, L)e(K_{\rho(i),y_j}, D_i)\\
&= e(g^{a\lambda_i}\mathcal{H}(\rho(i))^{-r_i}(g_{y_j})^{-r_i}, g^t)e((H(\rho(i))g_{y_j})^t, g^{r_i})\\
&= e(g,g)^{at\lambda_i}.\\
W &= \prod_{i\in I} Q_i^{w_i} = \prod_{i\in I} e(g,g)^{atw_i\lambda_i} = e(g,g)^{at\sum_{i\in I} w_i\lambda_i}\\
&= e(g,g)^{ats}.
\end{aligned}$$

Then, the decryption algorithm retrieves the plaintext message $\mathcal{M}$ as in $CW/P = \mathcal{M}$, and outputs it. If S does not satisfy the access structure, or the owner of SK belongs to the revocation list RL, then the decryption algorithm outputs $\perp$.

7. A SECURE CLOUD DATA-SHARING SCHEME BASED ON OUR PROPOSED ABE SCHEME

In this section, we propose a secure data-sharing scheme for the cloud. We keep the data sharing scheme generic such that it works with both ULR-ABE and ALR-ABE scheme. ULR-ABE should be used as the underlying encryption scheme if only user-level revocation is required. However, if more fine-grained revocation is necessary, then ALR-ABE scheme should be used. We discuss the details in the following.

7.1. SYSTEM INITIALIZATION

The attribute authority (AA) runs the `Setup` algorithm to initialize the system. It publishes the public key PK in the cloud and keeps the master key MK secret.

7.2. KEY DISTRIBUTION

For each user in the system, the AA runs the `Keygen` algorithm and generates the attribute secret key SK for his or her authorized attribute set *S*. The AA then sends SK to the corresponding user via a secure channel.

7.3. FILE OUTSOURCING

The data owner downloads the public parameter PK from the cloud. It randomly chooses $\mathcal{M} \in \mathbb{G}_T$ and extracts the symmetric key using the *randomness extractor function* $\mathcal{F}$ as in $\mathcal{K} = \mathcal{F}(\mathcal{M})$. Using $\mathcal{K}$, the data owner encrypts the actual file F using the symmetric encryption scheme as in CT$'$ =*Enc*($\mathcal{K}, F$). For each attribute, the data owner creates a revocation list by adding the users he wants to revoke the attribute from. Let *RL* be the revocation list (or set of all revocation lists if ALR-ABE is chosen). The data owner then runs `Encrypt` (PK, *(M, ρ)*, $\mathcal{M}$, *RL*) algorithm, and generates CT as output. Note that if a user is included in *RL*, then his or her secret key (or revoked attributes for ALR-Scheme) won't work during decryption. Finally, the data owner uploads the ciphertext (CT, CT$'$) as an encrypted file in the cloud.

7.4. FILE RETRIEVING

The data user downloads an encrypted file (CT, CT$'$) from the cloud and calls `Decrypt` (PK, SK, CT) algorithm with his or her attribute secret key SK. If the user is not revoked (when ULR-ABE is used) or SK has enough nonrevoked attributes to satisfy the access policy of CT (when ALR-ABE is used), the algorithm returns $\mathcal{M}$ as an output. Then, the user extracts the symmetric key using the *randomness extractor function* $\mathcal{F}$ as in $\mathcal{K} = \mathcal{F}(\mathcal{M})$. Using $\mathcal{K}$, the data user retrieves the actual file F by running the decryption function of the symmetric encryption scheme as in F =*Dec(*$\mathcal{K}$, *CT′)*.

8. SECURITY ANALYSIS

In this section, we analyze the security of our proposed ABE schemes in terms of semantic security and collusion attacks.

8.1. SEMANTIC SECURITY

Our IND-CPA game for both ULR-ABE (in Section 5.2) and ALR-ABE (in Section 6.2) scheme is similar to that of [1] except in our game, each secret key query in query phase I and II also includes a user identifier u_i, and $\mathcal{A}$ provides with a revocation list (for ULR-ABE) or a set of revocation lists (for ALR-ABE) RL^* in the challenge stage. It was proven in [1] that if the decisional q-BDHE assumption holds, then no polynomial-time adversary $\mathcal{A}$ can selectively win the IND-CPA game with a challenge matrix M^* of size $l^* \times n^*$ corresponding to the access structure (M^*, ρ), and maximum number of attributes per key of $Att_{\max}$ where $n^* + Att_{\max} \leq q$.

Assume there exists an adversary $\mathcal{A}$ who has a non-negligible advantage $\epsilon = \text{Adv}_{\mathcal{A}}$ in the our IND-CPA game and it chooses a challenge access structure (M^*, ρ) with the matrix M^* of at most q columns. Then using the same technique as in [1], we can build a simulator $\mathcal{B}$ that plays the decisional q-BDHE problem. The simulator basically programs all the IND-CPA game parameters from the decisional q-BDHE parameters so that the challenger cannot distinguish whether it is playing our IND-CPA game or the decisional q-BDHE problem. Since $\mathcal{A}$ has a non-negligible advantage in our IND-CPA game (according to our prior assumption), $\mathcal{A}$ also has a non-negligible advantage in the decisional q-BDHE problem.

However, according to the definition (of decisional q-BDHE problem), no polynomial-time adversary has a non-negligible advantage in solving the decisional q-BDHE problem. This implies that $\mathcal{A}$ does not have a non-negligible advantage in our IND-CPA game. So, the DEFINITION 2 and 3 hold true.

8.2. COLLUSION ATTACK

Both ULR-ABE and ALR-ABE scheme can prevent *type I* collusion attack because we bind all attributes of a particular user's secret key with a random exponent t such that attribute of one user's secret key does not work with the attribute of another user's secret key. In ULR-ABE scheme, the attribute components of the secret key SK is $\forall x \in S : K_x = \mathcal{H}(x)^t$. The exponent t is chosen randomly and it is different for different SKs. Since it is not possible for the adversary know t, the adversary cannot make attribute components of multiple secret keys to work together. Hence, *type I* collusion attack is not possible in ULR-ABE scheme. On the other hand in ALR-ABE scheme, the attribute components of secret key SK is represented as $\forall y_j \in$ *path(y)* $: K_{x,y_j} = (H(x)g_{y_j})^t$. Like ULR-ABE scheme, the attribute components of ALR-ABE scheme are also binded together with a random exponent t. So, *type I* collusion attack is not possible in ALR-ABE scheme as well.

Next, we discuss the security analysis of *type II* collusion attack for both ULR-ABE and ALR-ABE schemes.

8.2.1. Security Analysis of ULR-ABE Scheme.

We formalize the security of ULR-ABE scheme against *type II* collusion attack by the following theorem.

Theorem 2 *A revoked user with enough attributes in his or her attribute secret key cannot decrypt a ciphertext even if the user colludes with a nonrevoked user who does not have enough attributes in his or her attribute secret key to decrypt the particular ciphertext.*

Proof: Assume that u_i, u_k are two users and y, y' are their associated leaf nodes in the binary tree. SK $= (\forall y_j \in$ *path(y)* $: K_{y_j} = \left(g^{\alpha+at} g_{y_j}\right)^{1/\beta}, L = g^t, \forall x \in S : K_x = \mathcal{H}(x)^t))$ and SK$'$ $= (\forall y_j \in path(y') : K'_{y_j} = \left(g^{\alpha+at'} g_{y_j}\right)^{1/\beta}, L' = g^{t'}, \forall x \in S' : K'_x = \mathcal{H}(x)^{t'})$ are their attribute secret keys, respectively. A message $\mathcal{M}$ is encrypted so that $u_i \in RL$. This results in a ciphertext CT $= (C = \mathcal{M}.e(g,g)^{\alpha s}, C' = g^{s\beta}, D = g^r, \forall y_j \in$ *cover(RL)* $: C_{y_j} = g^s_{y_j}, \forall i \in \{1, 2, \ldots, l\} : C_i = g^{a\lambda_i} \mathcal{H}(\rho(i))^{-r})$. Note that CT does not have any C_{y_j} corresponding to the attribute secret key component K_{y_j} of u_i since $path(y) \cap$ *cover(RL)* $= \emptyset$. However, there exists a component K'_{y_j} in the attribute secret key of the nonrevoked user u_k that corresponds to a C_{y_j} since $path(y') \cap$ *cover(RL)* $\neq \emptyset$. Without the lose of generality, lets assume that S satisfies the access structure (M, ρ) but S' does not satisfy (M, ρ). This implies that u_i has enough attributes in his or her attribute secret keys to decrypt CT but u_k does not have enough attributes to do so. In order to successfully decrypt CT, u_i also needs K_{y_j} in his or her attribute secret keys that corresponds to C_{y_j}. As a result, u_i alone cannot decrypt CT with SK despite having enough attributes in it. However, u_i can collude with the nonrevoked user u_k to get K'_{y_j} from u_k's attribute secret key SK$'$ that corresponds to a C_{y_j} in CT. Then, u_i can try to decrypt CT as follows:

$$\begin{aligned}
P &= e(K'_{y_j}, C') = e\left(\left(g^{\alpha+at'} g_{y_j}\right)^{1/\beta}, g^{\beta s}\right) \\
&= e(g,g)^{\alpha s + at's}.e(g_{y_j}, g)^s \\
Q &= e(C_{y_j}, g) = e(g_{y_j}, g)^s \\
W' &= P/\left(Q \prod_{i \in I} \left(e(C_i, L) e(D, K_{\rho(i)})\right)^{w_i}\right) \\
&= P/\left(Q \prod_{i \in I} e(g,g)^{at\lambda_i w_i}\right) \\
&= e(g,g)^{\alpha s + at's}.e(g_{y_j}, g)^s / \left(e(g_{y_j}, g)^s e(g,g)^{ats}\right) \\
&= e(g,g)^{\alpha s + at's - ats}.
\end{aligned}$$

It is apparent that u_i cannot successfully compute $W = e(g,g)^{\alpha s}$ and hence is unable to retrieve $\mathcal{M}$ from $C = \mathcal{M}e(g,g)^{\alpha s}$. This proves THEOREM 2.

8.2.2. Security Analysis of ALR-ABE Scheme. The security of ALR-ABE scheme against *type II* collusion attack can be formalized by the following theorem.

Theorem 3 *A user with some mandatory attribute(s) (attribute(s) mandatory to satisfy the access structure) been revoked from his or her attribute secret key cannot decrypt a ciphertext even if the user colludes with another user who has the mandatory attribute(s) but does not have enough attributes in his or her attribute secret key to decrypt the particular ciphertext.*

Proof: Assume that u_i, u_i' are two users and y, y' are their associated leaf nodes in the binary tree. SK = $(K = g^{\alpha+at}, L = g^t, \forall x \in S, \forall y_j \in \mathit{path(y)} : K_{x,y_j} = (H(x)g_{y_j})^t)$ and SK′ = $(K' = g^{\alpha+at'}, L' = g^{t'}, \forall x \in S', \forall y_j \in \mathit{path(y)} : K'_{x,y_j} = (H(x)g_{y_j})^{t'})$ are their attribute secret keys, respectively. A message $\mathcal{M}$ is encrypted that results in a ciphertext CT = $(C = \mathcal{M}.e(g,g)^{\alpha s}, C' = g^s, ((\forall y_j \in \mathit{cover(RL_1)} : C_{1,y_j} = g^{a\lambda_1}\mathcal{H}(\rho(1))^{-r_1}(g_{y_j})^{-r_1}, D_1 = g^{r_1}), \ldots, (\forall y_j \in \mathit{cover(RL_l)} : C_{l,y_j} = g^{a\lambda_l}\mathcal{H}(\rho(l))^{-r_l}(g_{y_j})^{-r_l}, D_l = g^{r_l})$. Let $\rho(k)$ be a mandatory attribute to satisfy the access structure of CT and attribute $\rho(k)$ is revoked from user u_i (e.g., $u_i \in RL_k$) but not from user u_i'. This means that CT does not have any C_{k,y_j} corresponding to the attribute secret key component $K_{\rho(k),y_j}$ of u_i since $\mathit{path}(y) \cap \mathit{cover(RL)} = \emptyset$. However, there exists a component $K'_{\rho(k),y_j}$ in the attribute secret key of user u_i' that corresponds to C_{k,y_j} since $\mathit{path}(y') \cap \mathit{cover(RL)} \neq \emptyset$. Without the lose of generality, lets assume that S satisfies the access structure (M, ρ) but S' does not satisfy (M, ρ). This implies that u_i has enough attributes in his or her attribute secret key to satisfy the access structure, but cannot decrypt CT since the mandatory attribute $\rho(k)$ is revoked. On the other hand, although u_i' have attribute $\rho(k)$ nonrevoked, he or she does not have enough attributes to satisfy the access structure alone. In order to successfully decrypt CT, u_i also needs $K_{\rho(k),y_j}$

in his or her attribute secret keys that corresponds to C_{k,y_j}. As a result, u_i alone cannot decrypt CT with SK despite having enough attributes in it. However, u_i can collude with the nonrevoked user u_i' to get $K'_{\rho(k),y_j}$ from u_i''s attribute secret key SK$'$ that corresponds to a C_{k,y_j} in CT. In order to decrypt CT, u_i first computes

$$\begin{aligned} P = e(K, C') &= e\left(g^{\alpha+at}, g^{s}\right) \\ &= e(g,g)^{\alpha s} e(g,g)^{ats}. \end{aligned}$$

Then, $\forall i \in I - k$, u_i computes Q_i as in

$$\begin{aligned} Q_i &= e(C_{i,y_j}, L) e(K_{\rho(i),y_j}, D_i) \\ &= e(g^{a\lambda_i} \mathcal{H}(\rho(i))^{-r_i} (g_{y_j})^{-r_i}, g^{t}) e((H(\rho(i)) g_{y_j})^{t}, g^{r_i}) \\ &= e(g,g)^{at\lambda_i}. \end{aligned}$$

But for $i = k$, u_i computes

$$\begin{aligned} Q_k &= e(C_{k,y_j}, L') e(K'_{\rho(k),y_j}, D_k) \\ &= e(g^{a\lambda_k} \mathcal{H}(\rho(k))^{-r_k} (g_{y_j})^{-r_k}, g^{t'}) e((H(\rho(k)) g_{y_j})^{t'}, g^{r_k}) \\ &= e(g,g)^{at'\lambda_k}. \end{aligned}$$

Then, u_i computes W' as in

$$\begin{aligned} W' &= \left(\prod_{i \in I-k} Q_i^{w_i}\right) Q_k^{w_k} \\ &= \left(\prod_{i \in I-k} e(g,g)^{atw_i\lambda_i}\right) e(g,g)^{at'w_k\lambda_k} \\ &= e(g,g)^{at \sum_{i \in I-k} w_i\lambda_i + at'w_k\lambda_k}. \end{aligned}$$

Table 2. Symbols used in performance analysis and experiment

Symbol	**Meaning**
u	Total number of attributes in the system
a	Number of attributes in access structure $\mathbb{A}$
b	Number of attributes in user secret key
s	Required minimum number of attributes to satisfy policy
$\|\mathbb{G}_i\|$	Size of a single element in group $\mathbb{G}_i$
$\|\mathbb{K}\|$	Size of symmetric key
$\|p\|$	Size of a single element in $\mathbb{Z}_p$
C_i	Single exponentiation time in group $\mathbb{G}_i$
P	Computation time of a pairing operation
m	Total number of users in the group
r	Total nodes in *cover(RL)*

From the decryption algorithm, we can see that in order to retrieve $\mathcal{M}$ from $C = \mathcal{M}e(g, g)^{\alpha s}$, u_i must recover $W = e(g, g)^{ats}$ first. However, the above computation of W' shows that it is not possible unless t equals to t'. Since t and t' are chosen randomly from a large field $\mathbb{Z}_p$, their chance of being equal is negligible. Hence, u_i is unable to retrieve $\mathcal{M}$ from $C = \mathcal{M}e(g, g)^{\alpha s}$. This proves THEOREM 3.

9. THEORETICAL PERFORMANCE ANALYSIS

In this section, we compare our proposed ABE schemes with other related schemes in terms of storage, communication, and computational efficiency from the theoretical aspect. Note that we also include [1] (referred to as BW) in the comparison as a baseline since both our ULR-ABE and ALR-ABE schemes are based on this. Table 4 illustrates different symbols that have been used for this purpose.

Table 3. Comparison of storage and communication efficiency with other schemes

Scheme	Ciphertext size	Secret key size	Public key size
Hur-I [4]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + ar\lvert\mathbb{K}\rvert$	$(2b + 1)\lvert\mathbb{G}_1\rvert + (\log m)\lvert\mathbb{K}\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
Hur-II [5]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$2(b + 1)\lvert\mathbb{G}_1$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
CryptCloud+ [10]	$(2a + 5)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$(b + 6)\lvert\mathbb{G}_1\rvert + 2\lvert p\rvert$	$(u + 6)\lvert\mathbb{G}_1\rvert + 3\lvert p\rvert$
Flexible [6]	$(2a + 6)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + 2\lvert p\rvert$	$(b + 4)\lvert\mathbb{G}_1\rvert + 2\lvert p\rvert$	$3\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_T\rvert + \lvert p\rvert$
UserCol [9]	$(ar + 2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$4b\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$2(u + 3)\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_T\rvert + (2m - 1)\lvert p\rvert$
SR-ABE [11]	$(3a + 2)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$(2\log m(b + 1) + 1)\lvert\mathbb{G}_1\rvert$	$7\lvert\mathbb{G}_1\rvert$
BW [1]	$(a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$(b + 2)\lvert\mathbb{G}_1\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
ULR-ABE [12]	$(a + r + 2)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$(b + 1 + \log m)\lvert\mathbb{G}_1\rvert$	$(2m + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
ALT-ABE	$(ar + a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$(b\log m + 2)\lvert\mathbb{G}_1\rvert$	$2m\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$

9.1. STORAGE AND COMMUNICATION EFFICIENCY

The space efficiency comparison in terms of ciphertext, secret key, and public key size has been summarized in Table 5. The ciphertext size, secret key size, and public key size represent the storage cost required by the cloud, each user, and the attribute authority to store them, respectively. Additionally, they represent the communication cost when these are sent from one party to another. However, we do not consider here the communication cost that is associated with any intermediate step during the preparation of the ciphertext, secret key, or public key. For example, in order to achieve revocation, the data owner sends the whole ciphertext to the manager for re-encryption in UserCol[9]. As a result, the communication cost for sending the ciphertext to the cloud would be twice as much as what is shown in Table 5 for [9]. Similar intermediate steps are necessary in [6, 9, 11] during the secret key generation phase. Thus, the actual communication cost can be higher than what is shown in Table 5. However, there is no intermediate step in our proposed scheme, so the cost shown in Table 5 for our scheme is much closer to the real cost.

Compared to the baseline scheme [1], our ULR-ABE scheme requires $r + 1$ and $\log m - 1$ additional group ($\mathbb{G}_1$) elements for the ciphertext and secret key, respectively while our ALR-ABE scheme requires ar and $b\log m - b$ additional group ($\mathbb{G}_1$) elements. This is because in ULR-ABE scheme, the data owner has to create $r + 1$ additional group elements in the ciphertext out of which r elements are for *cover(RL)*. On the other hand in

ALR-ABE scheme, for each attribute in the access structure, the data owner creates r group elements (all are for *cover(RL)*). The AA has to create $\log m - 1$ additional group elements in the user's secret key along the *path* in the binary tree in ULR-ABE scheme while in ALR-ABE scheme, AA creates $\log m$ group elements for each attribute in the secret key as opposed to just one in BW. Both ULR-ABE and ALR-ABE schemes need $2m - 1$ additional group elements in the public parameter compared to the baseline scheme because a group element ($\mathbb{G}_1$) associated with every node in the binary tree is needed in the public parameter.

Among all collusion-resistant revocable schemes (ULR-ABE, ALR-ABE, [10, 6, 9], and [11]), [9] and ALR-ABE have the largest sized ciphertext because both require r group elements per attribute in the ciphertext to enable attribute-level revocation. This results in a total of ra additional group elements in the ciphertext. Scheme [10, 6, 11], and ULR-ABE do not require r group elements per attribute as they do not support attribute-level revocation. How our ALR-ABE scheme compares against [10, 6] in terms of ciphertext size, depends on the number of attributes in the ciphertext (a) and the number of nodes in *cover(RL)* (e.g., r). If there are few members to revoke or if revoked members are not very sparsely distributed in the binary tree, then r will be much smaller, and hence the ciphertext size will be relatively small.

The secret key size of ALR-ABE scheme is relatively larger than other schemes because AA generates $\log m$ group elements for each attribute in the secret key that are necessary for attribute-level revocation. Though [4, 5, 9] support attribute-level revocation, they don't require $\log m$ group elements per attribute. This is because their decryption method is different from ours where the ciphertext is partially decrypted by a semi-trusted entity such that it can be later fully decrypted by a smaller sized secret key. Although [11] does not support attribute-level revocation, it requires $2\log m$ group elements per attribute making its secret key size the largest of all.

Table 4. Comparison with other schemes in terms of computation cost.

Scheme	**Key generation**	**Encryption**	**Decryption**	**Key update**
Hur-I [4]	$2(b+1)C_1$	$(3a+1)C_1 + C_T$	$C_1 + (2s-1)C_T + (2s+1)P$	bC_1
Hur-II [5]	$(3b+5)C_1$	$(3a+2m+3)C_1 + C_T$	$s(m+1)C_1 + (2s-1)C_T + (3s+1)P$	bmC_1
CryptCloud+ [10]	$(b+13)C_1 + C_T + (2b+7)P$	$(a+5)C_1 + C_T$	$2C_1 + sC_T + (2s+5)P$	$3mC_1$
Flexible [6]	$(2b+9)C_1 + 2P$	$2(a+3)C_1 + 2C_T$	$(2s+3)C_T + (2s+4)P$	$(2m+b+1)C_1 + P$
UserCol [9]	$(4b+2)C_1$	$(3a+ar+1)C_1 + C_T$	$(2s-1)C_T + (3s+1)P$	$(2m-1)C_1$
SR-ABE [11]	$((2b+3)\log m + b + 1)C_1$	$(5a+2)C_1 + C_T$	$sC_T + (3s+4)P$	$(2r+4)C_1$
BW [1]	$(b+2)C_1$	$(2a+1)C_1 + C_T$	$sC_T + (2s+1)P$	N/A
ULR-ABE [12]	$(b+\log m+2)C_1$	$(2a+r+2)C_1 + C_T$	$sC_T + 2(s+1)P$	0
ALR-ABE	$(b\log m+2)C_1$	$(3a+ar+1)C_1 + C_T$	$sC_T + (2s+1)P$	0

Public key size increases proportionally with the number of total users for [9], ULR-ABE, and ALR-ABE scheme. However, the total public parameter size of [9] is larger than our schemes. Compared to [4, 5] and [11], ULR-ABE and ALR-ABE have a larger sized public key, as the public key includes an additional group element for each node in the binary tree.

9.2. COMPUTATION COST ANALYSIS

We show the computation cost of our scheme and compare it with other schemes in Table 6. The computation cost has been expressed in terms of group exponentiation and pairing operation in a similar manner as in [23, 5, 4]. This is a reasonable consideration since these two operations dominate relatively lightweight hash, multiplication, division, and addition operations.

If we compare [12] and our scheme with the baseline scheme [1], we can see that the revocation does not have any effect on the decryption running time. However, the the running time of key generation and encrytion algorithms depend on the granularity of the revocation. For encryption, [12] requires $r + 1$ additional group exponentiation operations (in $\mathbb{G}_1$), out of which r is for creating r additional group elements for *cover(RL)*. Our scheme on the other hand requires such $r + 1$ additional group exponentiation operations per attribute since our revocation is at attribute-level. Similarly for secret key generation, while [12] requires log$m$ additional group exponentiation operations (in $\mathbb{G}_1$) our scheme requires such log$m$ operations per attribute in the secret key.

Both our scheme and [12] have a zero key update cost since both scheme achieve revocation without affecting the secret key of other nonrevoked users. In contrast, all other revocation schemes require a significant amount of computation for key updating to achieve revocation. The number of group exponentiation operation (in $\mathbb{G}_1$) required for key update is proportional to the number of users in the system (except [4] and [11]).

In terms of decryption speed, our scheme is similar to the baseline scheme [1] and outperforms all the revocation scheme. We can see that [5] has the slowest decryption speed as the required number of group exponentiation operation (in $\mathbb{G}_1$) is proportional to the total number of users (m).

Our key generation algorithm is faster than that of [10]. This is because in addition to the group exponentiation operation, CryptCloud+ also requires $(2b+7)P$ pairing operations, which is more expensive than the group exponentiation operation. Our key generation time is also faster than that of [11] as it requires twice as much additional (2log$m$) group exponentiation operations (in $\mathbb{G}_1$) per attribute compared to ours.

When we compare our scheme in terms of encryption speed with other schemes that support fine-grained revocation (e.g., [4, 5], and [9]), both our scheme and [9] have similar performance as both needs log$m$ exponentiation operations (in $\mathbb{G}_1$). Scheme [5] is much slower than ours since the number of group exponentiation operations for each attribute grows linearly with the total number of users (e.g., m). However, [4] has a faster encryption running time than ours because the cloud is trusted to perform revocation transformation of the ciphertext and partial decryption. Hence, it can avoid multiple group exponentiation operations per attribute like ours. Schemes that do not support attribute-level revocation in general have a faster encryption running time than ours.

Table 5. Parameter details for different pairing groups

Curves	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	p	k	*Security*
SS512	512	512	1024	160	2	80
MNT159	159	477	954	158	6	70
MNT201	201	603	1206	181	6	90
MNT224	224	672	1344	224	6	100

10. EXPERIMENT

Implementation: We have implemented our scheme in `Charm` [25]. It is a Python based framework developed for rapid prototyping of advanced cryptographic protocols. `Charm` uses PBC library [26] (written in C language) for low-level system calls including most expensive group exponentiation and pairing operations. As a result, cryptographic protocols written in `Charm` performs very close to the one written C language [27]. All hash functions were implemented using SHA224.

A detailed parameter description for our experimental setup is given in Table 3. SS512 is a super singular EC curve (with symmetric Type 1 pairing), and MNT (159, 201, 224) are the Miyaji, Nakabayashi, Takano curves (with asymmetric Type 3 pairing). In Table 4, p = bit length of prime order p, k = embedding degree, *Security* is the security level in bits with respect to the discrete log problem, and the numbers associated with the curve name represent the base field size in bits (i.e., SS512 has a base field size of 512 bits). Though our schemes are based on a symmetric pairing group ($\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$), we have tested our implementation in both symmetric and asymmetric group settings. `Charm` treats groups as asymmetric, though the actual setting depends on the type of underlying chosen curve. More specifically, there are three different groups ($\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$), and pairing is defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. We keep most of the terms in $\mathbb{G}_1$ while implementing our scheme in the asymmetric setting since operations in $\mathbb{G}_1$ are generally much faster than that in $\mathbb{G}_2$.

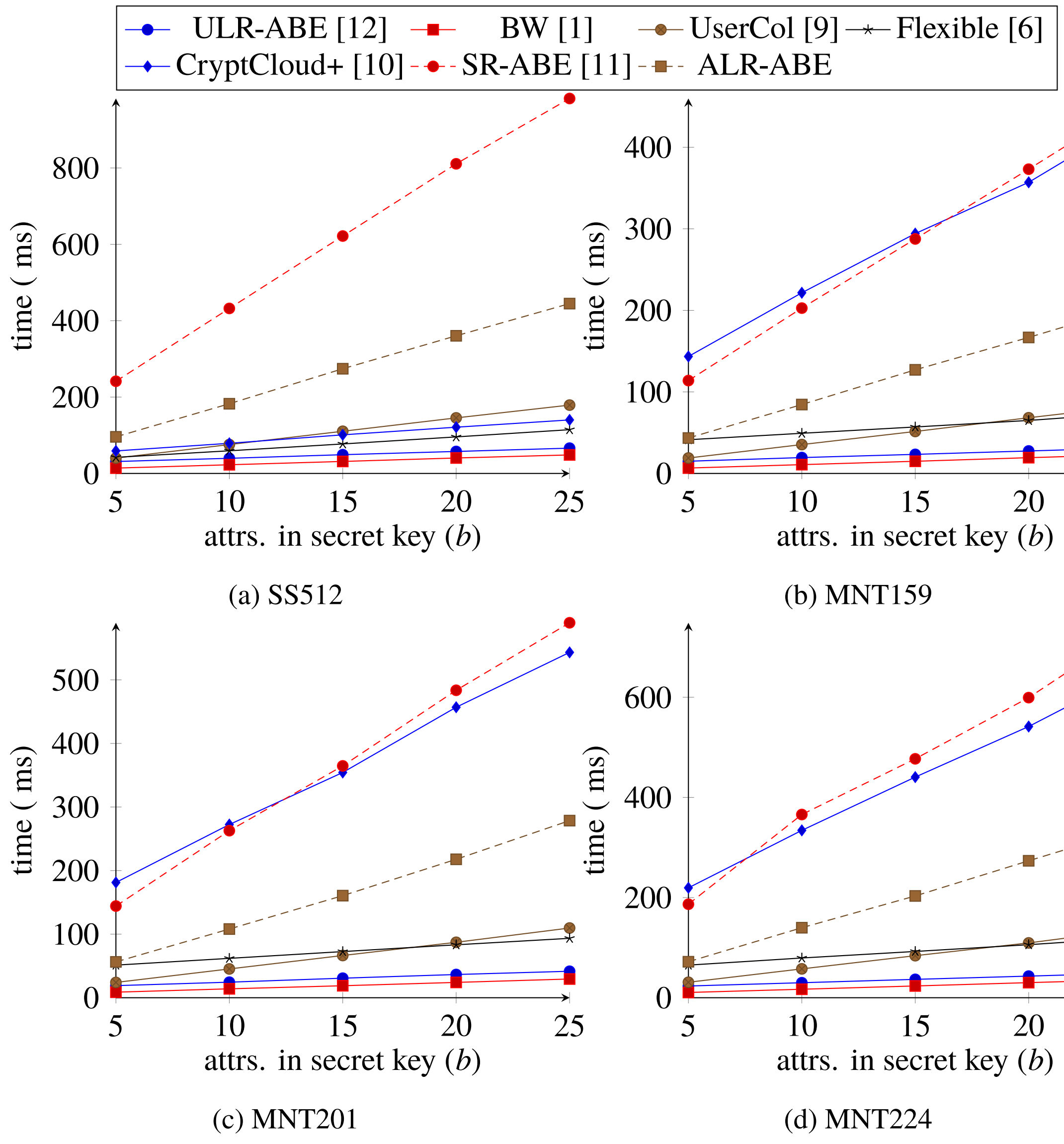
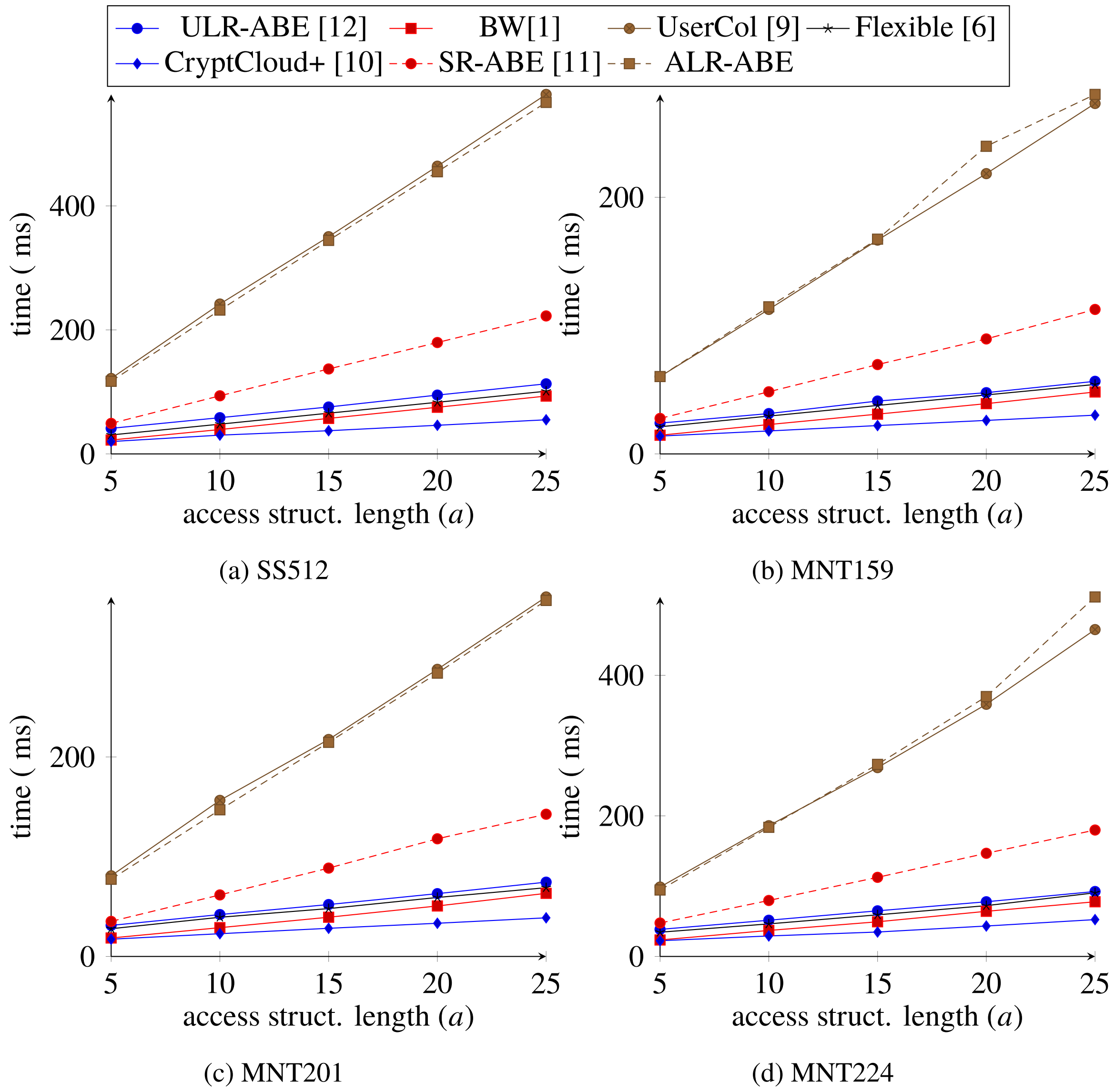


Figure 3. Key generation time

Testbed setup: We have conducted all the experiments on a Macbook Pro laptop with Intel® Core i7@2.2 GHz quad-core processor and 16 GB RAM running MacOS Cataline 10.15.7. We have used Python 3.7 and the PBC-0.5.14 library.

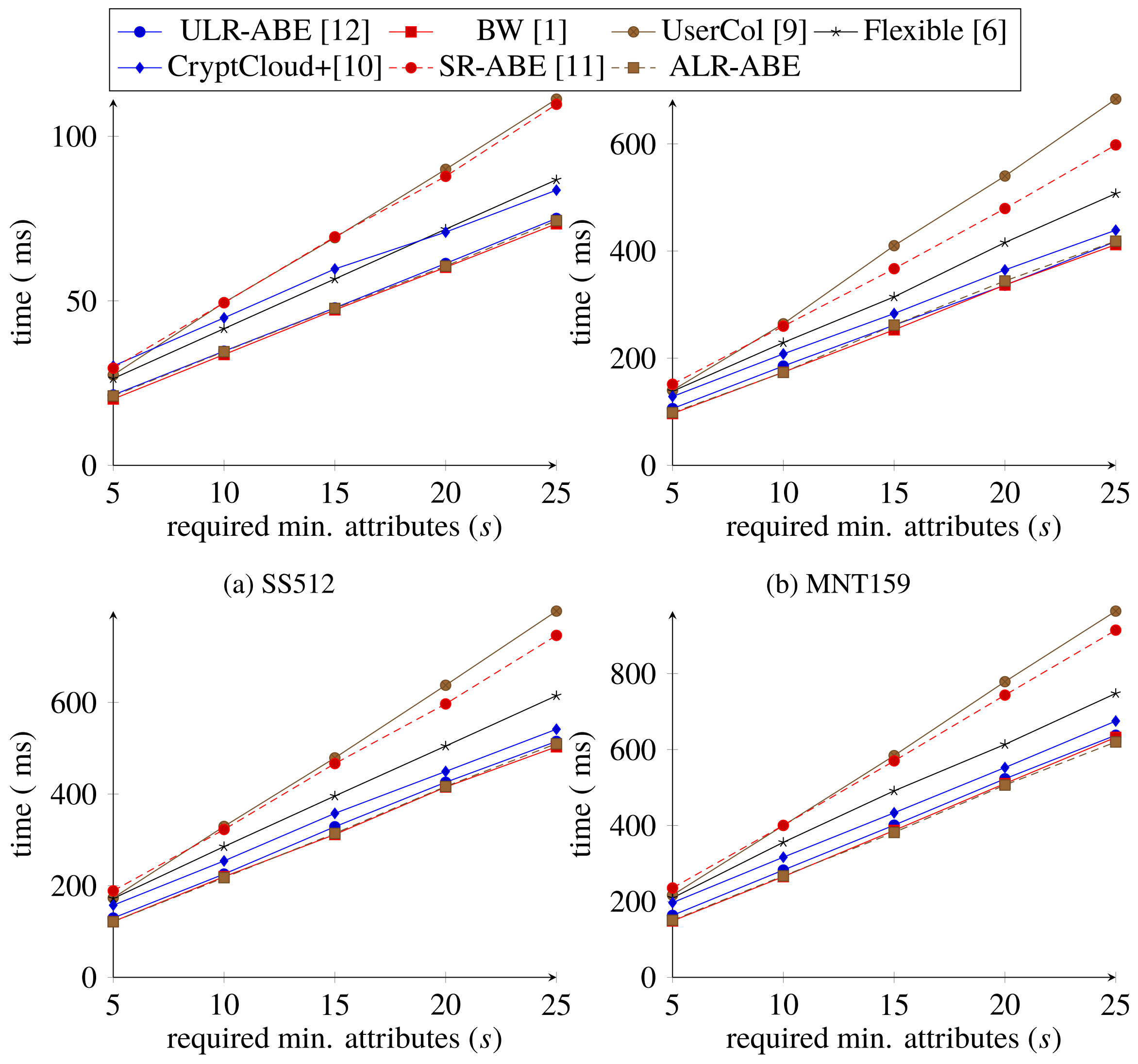
Results: We have mainly compared our key generation, encryption, decryptioin, and key update running time with Flexible [6], UserCol [9], CryptoCloud+ [10], and SR-ABE [11] scheme. We have also reported the key generation, encryption, and decryption running



(a) SS512 (b) MNT159 (c) MNT201 (d) MNT224

Figure 4. Encryption time

time of BW [1] scheme as a baseline comparison since our scheme is based on the BW scheme. However, we have not compared our experimental results with Hur I [4] and Hur II [5] since they are not collusion resistant (against *type II* attacks) and the encryption, decryption, and key update of Hur II takes much longer than the rest. The running time of each algorithm (key generation, encryption, decryption, and key update) were measured for



(a) SS512

(b) MNT159

(c) MNT201

(d) MNT224

Figure 5. Decryption time

SS512, MNT159, MNT201, and MNT224 curves. Each result reported here has been averaged over five individual runs.

From Figure 3 we can see that the key generation running time of all schemes increases linearly with the number of attributes in the secret key (b). The running time of ALR-ABE scheme and [11] is longer than those of [1, 9, 6], and ULR-ABE scheme since the running time of the key generation algorithm is proportional to $b \log m$ where m is the total number of user. However, the key generation running time of both schemes seems to be linearly increasing with b since we have kept the value of m to be fixed at 1000. Compared to ALR-ABE scheme, key generation of [11] is slower as it requires more group exponentiation operations (refer to Table 6). Interestingly, the key generation running time of [10] is faster than ALR-ABE in SS512 curve (Figure 3a) but slower than ALR-ABE scheme in MNT159, MNT201, and MNT224 curves (Figure 3b, 3c, and 3d, respectively). This is because in addition to group exponentiation operation, [10] also requires $(2b + 7)$ pairing operations that take more time in MNT159, MNT201, and MNT224 curves compared to the SS512 curve.

The encryption running time has been shown in Figure 5. We can see that schemes with fine-grained revocation (e.g., ALR-ABE and UserCol) have relatively slower encryption running time for all curves. This is because ALR-ABE and UserCol generate r (r being the number of nodes in *cover(RL)*) number of $\mathbb{G}_1$ group components for each attribute present in the access structure such that a user, not revoked from an attribute, have exactly one matching secret key component to decrypt one of the r group components. Creating a $\mathbb{G}_1$ group component requires one group exponentiation operation and hence the total number of group exponentiation operations for both ALR-ABE and UserCol is proportional to ar (where a is the total number of attributes in the access structure). As a result, the encryption running time increases linearly with ar for both schemes. For other schemes (e.g., [1, 6, 10, 11], and ULR-ABE) the running time increases linearly with a as the number of group exponentiation operation required is proportional to a. In our experiment, we have set the value of r to be 10 and varied the value of a between 5 and 25.

Figure 6 shows that the running time of the decryption algorithm has a linear relationship with the number of minimum required attributes to satisfy the access structure (s). We have varied the value of s from 5 to 25 at an interval of 5. Both the number of group exponentiation (in $\mathbb{G}_T$) and pairing operation has a linear relationship with s for all the schemes. The pairing operation is much slower in MNT159, MNT201 and MNT224 curves compared to the SS512 curve. Consequently, the decryption time is higher in those three

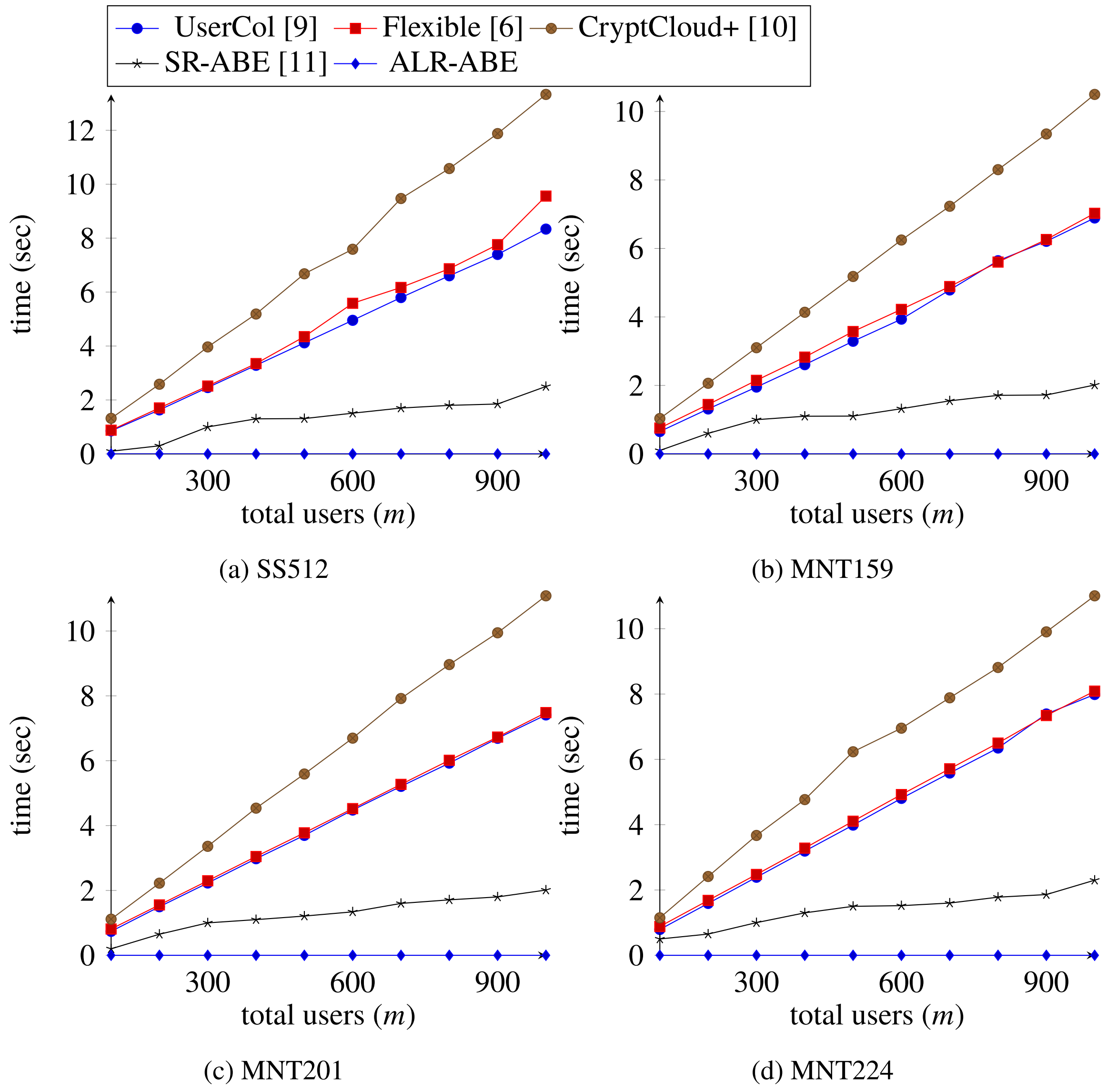


Figure 6. Key update time

curves for all the schemes. Note that the decryption time of ULR-ABE and ALR-ABE is very close to the baseline scheme [1], as our decryption requires similar number of pairing operation compared to that of the baseline scheme.

In Figure 6, we have shown how the running time of the key update algorithm increases with the number of users (m) by varying the value of m from 100 to 1000. The revocation in ULR-ABE and ALR-ABE does not affect the secret key of any nonrevoked user. As a result, the revocation does not require any key update and hence the key update cost is zero for both schemes. Note that in Figure 6 ULR-ABE and ALR-ABE lines overlap with each other and only ALR-ABE line is visible. However, other revocable schemes need to update the secret keys of existing nonrevoked users. The running time of the key update algorithm increases linearly with *m* for scheme [9, 6, 10]. However, scheme SR-ABE performs better than [9, 6, 10] since the running time increases linearly with the number of nodes in *cover(RL)* (e.g., *r*) as opposed to number of users. In general, the value of *r* is significantly smaller than *m* but *r* increases as the value of *m* gets bigger.

Note that, the performance of all the algorithms do not have equal importance. For instance, key generation is normally a one time task. A file may be encrypted by the owner only once but is potentially decrypted many times by different users. If users are revoked frequently, then key update cost can be very critical. As a result, the performance of decryption and key update is more important than that of other algorithms (e.g., key generation and encryption). From Figure 6 and 6 we can see that our decryption and key update algorithms outperform other schemes.

11. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed two revocable ABE schemes called ULR-ABE and ALR-ABE. The first one supports revocation at the user-level while the later one supports revocation at the attribute-level while maintaining the important collusion resistance (against both *type I* and *type II* collusion attacks) property. Our schemes do not require any semi-trusted entity to achieve revocation. Moreover, the revocation does not affect the secret key of any non-revoked user, and hence the key update cost for revocation is zero in our scheme. It is evident from theoretical performance analysis and experimental results that both ULR-ABE and ALR-ABE schemes outperform the most closely related ABE schemes in terms of decryption and key update cost. Compared to ULR-ABE scheme, ALR-ABE scheme has a slower key generation and encryption algorithm, and the secret key and ciphertext sizes are also larger. In the future, we would like to improve the running time of both key generation and encryption algorithms as well as make the secret key and ciphertext sizes shorter. The scope of this work is limited to static access structures only. It will be an interesting extension if we could incorporate dynamic access policies in our schemes in the future.

REFERENCES

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

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, 2005.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007.

[4] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, 2011.

[5] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2013.

[6] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2018.

[7] H. Wang, Z. Zheng, L. Wu, and P. Li, "New directly revocable attribute-based encryption scheme and its application in cloud storage environment," *Cluster Computing*, vol. 20, no. 3, pp. 2385–2392, 2017.

[8] X. Wang and J. Fang, "A revocable outsourcing attribute-based encryption scheme," in *Cloud Computing, Security, Privacy in New Computing Environments: 7th International Conference, CloudComp 2016, and First International Conference, SPNCE 2016, Guangzhou, China, November 25–26, and December 15–16, 2016, Proceedings*, vol. 197, p. 145, Springer, 2017.

[9] J. Li, W. Yao, J. Han, Y. Zhang, and J. Shen, "User collusion avoidance cp-abe with efficient attribute revocation for cloud storage," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1767–1777, 2018.

[10] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K.-K. R. Choo, "Cryptcloud+: secure and expressive data access control for cloud storage," *IEEE Transactions on Services Computing*, 2018.

[11] B. Qin, Q. Zhao, D. Zheng, and H. Cui, "(dual) server-aided revocable attribute-based encryption with decryption key exposure resistance," *Information Sciences*, vol. 490, pp. 74–92, 2019.

[12] M. A. Islam and S. Madria, "A collusion-resistant revocable attribute-based encryption scheme for secure data sharing in cloud," in *The 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May 11-14, 2020, Melbourne, Victoria, Australia*, pp. 21–30, IEEE/ACM, 2020.

[13] Y. Jiang, W. Susilo, Y. Mu, and F. Guo, "Ciphertext-policy attribute-based encryption with key-delegation abuse resistance," in *Australasian Conference on Information Security and Privacy*, pp. 477–494, Springer, 2016.

[14] R. R. Al-Dahhan, Q. Shi, G. M. Lee, and K. Kifayat, "Survey on revocation in ciphertext-policy attribute-based encryption," *Sensors*, vol. 19, no. 7, p. 1695, 2019.

[15] Y. Zhang, R. H. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng, "Attribute-based encryption for cloud computing access control: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–41, 2020.

[16] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *Advances in Cryptology–CRYPTO 2012*, pp. 199–217, Springer, 2012.

[17] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355–370, 2014.

[18] G. Zeng, "Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation," in *Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6-8, 2017, Proceedings*, vol. 10631, p. 172, Springer, 2018.

[19] H. Cui, T. Hon Yuen, R. H. Deng, and G. Wang, "Server-aided revocable attribute-based encryption for cloud computing services," *Concurrency and Computation: Practice and Experience*, p. e5680, 2020.

[20] X. Wang, Y. Chi, and Y. Zhang, "Traceable ciphertext policy attribute-based encryption scheme with user revocation for cloud storage," in *2020 International Conference on Computer Engineering and Application (ICCEA)*, pp. 91–95, IEEE, 2020.

[21] D. Han, N. Pan, and K.-C. Li, "A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[22] J. Li, Y. Shi, and Y. Zhang, "Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage," *International Journal of Communication Systems*, vol. 30, no. 1, p. e2942, 2017.

[23] B. Qin, R. H. Deng, S. Liu, and S. Ma, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1384–1393, 2015.

[24] R. Zhang, L. Hui, S. Yiu, X. Yu, Z. Liu, and Z. L. Jiang, "A traceable outsourcing cp-abe scheme with attribute revocation," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pp. 363–370, IEEE, 2017.

[25] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[26] B. Lynn, "The stanford pairing based crypto library," *Privacy Preservation Scheme for Multicast Communications in Smart Buildings of the Smart Grid*, vol. 324, 2013.

[27] Y. Rouselakis and B. Waters, "Practical constructions and new proof methods for large universe attribute-based encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 463–474, ACM, 2013.

III. ATTRIBUTE-BASED ENCRYPTION SCHEME FOR SECURE MULTI-GROUP DATA SHARING IN CLOUD

MD Azharul Islam and Sanjay Madria

Department of Computer Science

Missouri University of Science and Technology

Rolla, Missouri 65401

Email: mdazharul.islam@mst.edu and madrias@mst.edu

ABSTRACT

Most of the organizations using the cloud-based data sharing platforms are multi-group in nature. The existing directly revocable attribute-based encryption (ABE) schemes though seem to be a good fit, but they fail to provide any effective solution for secure multi-group data sharing scenarios. To bridge this gap, we first propose **Re**vocable **ABE** with **V**erifiable **O**utsourced decryption (ReVO-ABE)- a directly revocable collusion-resistant ABE scheme that allows any number of user revocation and joining without affecting the secret membership keys of the nonrevoked users. Based on ReVO-ABE, we build a **D**ynamic **M**ulti-**G**roup **S**ecure **D**ata **S**haring scheme called DMG-SDS. For operations that are exclusive to multi-groups like group merge and split can be performed without affecting the attribute secret keys or membership keys of the nonrevoked users, which is not possible with any of the existing schemes. Our proposed scheme meets the necessary security requirements, and the performance assessment shows that it has much better performance benefits when compared with the most recent competitive schemes.

Keywords: Secure Data Sharing, Attribute-Based Encryption, Access Control.

1. INTRODUCTION

Attribute-based encryption (ABE) offers convenient encryption–decryption methods with the use of attributes and highly expressive access policies. For example, a file may be encrypted using the access policy *manager AND (marketing OR quality)* for a company **C**. This limits the file access to only the marketing or quality manager. Using ABE, company **C** can migrate all of its data to the cloud and enforce fine-grained access control while sharing it among the employees. As a result, the company **C** may enjoy advantages like availability, scalability, and lower maintenance cost offered by the cloud while safeguarding its data from any potential data breach. Benefits as such have made ABE more popular than other encryption schemes [1, 2]. In practice, data-sharing groups are dynamic, meaning that members are revoked or added any time. Dealing with such dynamic groups poses new challenges to attribute-based group data sharing schemes since the same attributes may be shared among different users. For example, company **C** may fire the current marketing manager and recruit a new one instead. Consequently, the decryption rights from the former marketing manager should be revoked while giving decryption rights to the new one even though both have attributes *manager* and *marketing*. Things get even more challenging in a dynamic multi-group setting as it has some exclusive operations such as groups merge and split in addition to those discussed earlier. This is important because most of the organizations in real life are multi-group in nature. For example, a company **C** may have two different groups for its two separate branches: **A** and **B**. Group merging is necessary when the company decides to merge both branches to cut down the operational cost. Similarly, a group split is required when the company wants to split **B** into branches **B1** and **B2** for better growth opportunity.

1.1. LIMITATIONS OF THE EXISTING SCHEMES

In the ABE, dynamic group is supported by revocable ABE. There are two possible ways of achieving revocation property in ABE: indirect revocation [3, 4, 5] and direct revocation [6, 7, 8, 9, 10, 11, 12]. A revocation list is maintained in both the methods that specifies all the revoked users. In a indirect revocation, the attribute authority (the trusted party responsible for creating and distributing attribute keys) performs a periodic update of the attribute keys according to the revocation list, and distributes them to every nonrevoked user. On the other hand, the direct revocation schemes such as [13, 7] are based on the idea of attribute groups. A user's secret key has two parts. One is associated with the user's authorized attributes, while the other one is associated with the attribute group. They are called attribute secret key (SK_S) and membership key (MbK), respectively. The data owner excludes MbKs of all the users in the revocation list from the attribute group while encrypting. Consequently, the users in the revocation list cannot decrypt using their MbKs. However, these schemes are vulnerable to revoked–nonrevoked user collusion attack, where a nonrevoked user can restore a revoked user's decryption ability by sharing his or her MbK with the nonrevoked user. This is possible because SK_S and MbK are independent of each other. Schemes proposed in [9, 10, 11] also suffer from the same type of collusion attack. Li et al. proposed a solution addressing this issue in [12] by binding SK_S with MbK so that one user's SK_S does not work with another user's MbK. Li et al. later refined the solution by updating the security model in [14]. However, the group admin (the trusted centralized entity that manages the attribute group and membership keys) becomes a performance bottleneck in [12, 14] because in each revocation epoch, the group admin has to update and transmit the new membership keys (MbK) to all the nonrevoked users. The scheme proposed in [5] also suffers from the similar issues as it follows the similar revocation technique as [12, 14]. Despite these issues, revoked-nonrevoked user collusion resistance is a desirable property for an ABE scheme according to [15].

The schemes [13, 9, 10, 11, 14, 5, 8] use a static binary key tree for assigning MbKs to the users. Specifically, a binary tree with a total of $m_{\max}$ leaf nodes is created during the initialization and distinct keys are assigned to the nodes. Each user is associated with a leaf node, and a user's membership key (MbK) consists of all the keys in the path from the leaf to the root node. Adding a new user (after exceeding $m_{\max}$) changes the key tree structure and affects the membership keys of the nonrevoked users. This requires the admin to transmit new MbKs to the nonrevoked users. To avoid this, a static binary tree is created by choosing a large $m_{\max}$ which sets a limit at $m_{\max}$ for the maximum number of users to be added. Scheme [6] does not put a limit on the total number of new user joining. However, [6] generates MbKs using t out of n secret sharing scheme, and hence allows only a t number of maximum revocations.

The existing revocable ABE schemes require the distribution of secret keys to the nonrevoked users during dynamic multi-group operations like group split and merge. For example, when a company **C** splits branch **B** into branch **B1** and **B2**, it may try to handle this in either of the following ways. The attribute authority may create attributes *branchB1* and *branchB2*, and distribute corresponding attribute secret keys to all existing users of branch **B1** and **B2**, respectively. Or, two new binary key trees may be created for two different branches. For the new binary key tree, the new membership keys of a user's corresponding leaf node to the root must be sent out. On the other hand, the merging of branch **B** and **C** can be handled by merging two key trees into one. In this case, the secret key of the new root node has to be sent as a membership key to all the nonrevoked users.

To the best of our knowledge, there is no directly revocable ABE scheme that is resistant to revoked–nonrevoked user collusion attack with the following properties: 1) Does not put any limit on the total number of user revocation or joining. 2) The group admin itself does not need to transmit any new secret membership keys (MbKs) to nonrevoked users followed by revocation or joining. 3) Supports multi-group operations like group merge and split without requiring to transmit attribute secret keys or membership keys to the nonrevoked users.

1.2. OUR TECHNIQUE AND CONTRIBUTION

We propose a collusion-resistant directly revocable ABE scheme that does not have the limitations listed above. To realize our ABE scheme, we introduce a federated cloud-based architecture with two clouds (cloud1 and cloud2). The user has the attribute secret key (SK_S) and the membership key (MbK), while cloud1 has the transformation key (TK_S) and cloud1 master key (C1MK), and cloud2 has the update key (UK). After a revocation or a new user joining, the group admin creates a single proxy update key (Upr) and gives it to cloud2 so that cloud2 can update UKs for all the nonrevoked users. To decrypt a ciphertext (CT), cloud1 and cloud2 jointly create a partially decrypted ciphertext (CT_{part}). Then, a non-revoked user can fully decrypt CT_{part} using SK_S and MbK. We cryptographically bind SK_S, MbK, TK_S, and UK together that helps our scheme to achieve revocation, and prevent the revoked-nonrevoked user and cloud-revoked user collusion attacks as long as one of the two clouds remains honest.

To overcome any limit on the number of revocations or joinings, we propose a data structure called the extended TGDH (or e-TGDH) tree based on the TGDH tree (Section 3.2.1). The difference between the original TGDH and our proposed e-TGDH tree is that when any leaf node is added or removed from the TGDH tree, all the keys along the path from the leaf to the root are overridden while they are efficiently preserved in the e-TGDH tree (Section 5). We provide algorithms for removing (Alg. 1) and adding (Alg. 2) a leaf node in the e-TGDH tree that are used for user revocation or joining, respectively. The e-TGDH tree replaces the role of the key tree and offers the following advantages: 1) users need to keep only a single membership key as opposed to all the keys in the path of the key tree, 2) revocation and joining can happen anytime and as many times as needed, 3) after revocation, joining or any multi-group operation such as group split or merge does not require to directly transmit any secret key to the nonrevoked users. Rather, it gives the user with the ability to compute all updated secret keys using public information available in the cloud. The only disadvantage is that a decryption may require an additional $\log m$ operations (as opposed to constant number of operations in the key tree approach) for a total m of users. We compensate this additional cost by outsourcing the computationally expensive operations of decryption (while creating CT_{part}) to the cloud using the key blinding technique in [16]. To make sure the cloud completes the computation correctly, a short proof for verification is also added in the ciphertext (using the technique in [17]). We call our direct revocation scheme **Re**vocable **ABE** with **V**erifiable **O**utsourced decryption (ReVO-ABE). Finally, using our ReVO-ABE, we build a **D**ynamic **M**ulti-**G**roup **S**ecure **D**ata **S**haring scheme called DMG-SDS. We summarize our key contributions in this paper as follows:

1. We propose ReVo-ABE by utilizing our federated cloud architecture and newly proposed data structure called e-TGDH. It is collusion-resistant, and does not put any limit on the number of user revocations or joining.

2. We propose the first ABE based multi-group data sharing scheme, called the dynamic multi-group secure data sharing scheme (DMG-SDS) that supports the operations like group merge and group split.

3. Our security analysis shows that our scheme is secure against different kind of collusion attacks. We present a detailed performance analysis of our scheme from both the experimental and theoretical standpoint, and the results show that our proposed scheme has better performance benefits than others.

2. RELATED WORKS

Attribute-based encryption was introduced by Waters et al. [18] and the two other variants of ABE called key-policy attribute-based encryption (KP-ABE) [19] and ciphertext-policy attribute-based encryption (CP-ABE) [20] were eventually proposed. In KP-ABE, the access policy is associated with the secret keys, while in CP-ABE, it is associated with the ciphertext. It was until the emergence of revocable ABE [3] that ABE became more popular for secure group data sharing in the cloud since it has the ability to revoke or add a user in the data sharing group. The schemes [3, 4] proposed indirect revocation schemes where the attribute authority realizes the revocation by periodic update and redistribution of secret keys to the nonrevoked users. A more practical method named directly revocable ABE was proposed in PIRRATE [6] and [7], where the revocation list is embedded in the ciphertext by the encryptor. PIRRATE has a limitation that only allows t out of n revocations, where t has to be fixed beforehand. On the other hand, encryption and decryption time increases linearly with the number of nonrevoked users in [7]. A more efficient directly revocable ABE scheme was proposed by embedding a revocation list in the ciphertext using subset cover from a binary key tree [8, 9, 10, 11]. The main limitation was that the total number of users needs to be fixed during the tree creation, and no new user can be added afterwards. Despite these limitations, the directly revocable ABE is the most compatible ABE scheme for secure group data sharing in the cloud. However, it fails to provide an efficient solution for multi-group data sharing because it cannot handle group merge and split operations without affecting the attribute secret keys and membership keys of nonrevoked users.

The schemes [12, 21, 14] put forward a revocable ABE scheme where each user's private key is composed of two parts: one associated with her authorized attribute (SK_S) and the other associated with the group she belongs to (MbK). Unlike previous schemes such as [7, 13, 11, 9, 10], MbK here is collusion resistant. As a result, the system remains secure even if a revoked user obtains the group secret key component of a valid user. But the disadvantage is that after a revocation, the group manager has to update and transmit group

Table 1. Comparison with related schemes in terms of security and functionality.

Scheme	Security assumption	Model	Outsourced Decryption	Verifiability	Revocation	Unlimited joining	Multi-group	Collusion resistant
DASS [8]	Decisional PB-DHE	Standard	✗	✗	✓	✗	✗	✗
Hur-I [13]	Generic Group	RO	✗	✗	✓	✗	✗	✗
Hur-II [7]	Generic Group	RO	✗	✗	✓	✓	✗	✗
PIRATTE [6]	Generic Group	RO	✗	✗	✓	✓	✗	✗
VO-ABE [17]	Decisional q-PBDHE	Standard	✓	✓	✗	✗	✗	N/A
CryptCloud+[5]	l-SDH	Standard	✗	✗	✓	✗	✗	✓
Flexible [12]	Generic Group	RO	✓	✗	✓	✓	✗	✓
UserCol [14]	Generic Group	RO	✗	✗	✓	✗	✗	✓
Ours	CDH	RO	✓	✓	✓	✓	✓	✓

secret key components of all the existing users. Besides, every encryption and re-encryption also requires the group manager's participation, which is inefficient. The scheme [5] also suffers from the same issues. The directly revocable schemes proposed in [22, 23, 24, 25] delegate the revocation task to the cloud, but unable to prevent a collusion attack when a revoked user colludes with a nonrevoked user and the cloud. A recently proposed traceable ABE scheme [26, 27] focuses on revoking a user by finding the leaked keys rather than preventing a revoked-nonrevoked user collusion attack.

ABE is computationally intensive because it has expensive pairing and group exponentiation operations. Green et al. [16] first reduced ABE decryption cost at the user end by securely outsourcing those expensive operations to the cloud. The idea is to blind user's attribute secret key with a blinding key and give it to the cloud while keeping the blinding key secret. The cloud can use blinded attribute keys to partially decrypt the ciphertext so that the user can later fully decrypt it using the secret blinding key. Later, [17, 28, 29, 30, 31] added verifiability to the outsourced decryption to ensure that the cloud correctly performs the computation. Among them, Qin et al. [17] was able to achieve verifiability with a short and constant overhead, while others ended up adding a much larger overhead to the ciphertext. However, none of them supports revocation.

A comparison in terms of different functionalities and security among [8, 13, 7, 6, 17, 5, 12, 14] and our work has been shown in Table 1. Since these schemes are the most closely related to our scheme, we will refer to these schemes throughout this paper for comparison. VO-ABE [17] is the only one that does not support revocation but it is also the only one (except ours) to support verifiable outsourced decryption. The scheme [6, 13, 7, 12], and ours are based on the CP-ABE scheme in [20]. As a result, these four are secure in the random oracle (RO) model like [20]. Only [12, 14, 5] and our scheme are collusion resistant (against a revoked-nonrevoked user collusion attack), and none of the schemes except ours supports multi-group scenario.

3. BACKGROUND

In this section, we first present the symbols and notations used in this paper repeatedly. Then, we discuss the access structure and access tree. Finally, the cryptographic primitives related to our scheme are given, followed by their security assumptions.

3.1. SYMBOLS AND NOTATIONS USED

We use $\mathbb{G}$ and $\mathbb{G}_T$ to represent two multiplicative cyclic groups of prime order p, while g is a generator of $\mathbb{G}$. The symbol $\mathbb{Z}_p$ is used to denote the group of integers modulo p. We utilize four hash functions, defined as $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}$, $\mathcal{H}_1 : \mathbb{G}_T \rightarrow \{0,1\}^{l_1}$, $\mathcal{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}$, and $\mathcal{H}_3 : \mathbb{G} \rightarrow \mathbb{Z}_p$. We also make use of a *randomness extractor function* [17] defined as $F : \mathbb{G}_T \rightarrow \mathbb{K}$, where $\mathbb{K}$ is the symmetric key space. *Enc* and *Dec* are the encryption and decryption functions, respectively for the symmetric encryption scheme used. We denote the CP-ABE scheme proposed in [20] as CP-ABE$_0$.

Access structure ($\mathbb{A}$): Let $\mathcal{U}$ be an attribute universe. An access structure on $\mathcal{U}$ is a collection $\mathbb{A}$ of non-empty sets of attributes (i.e. $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \emptyset$). The sets in $\mathbb{A}$ are called the *authorized sets*, and the sets not in $\mathbb{A}$ are called the *unauthorized sets*.

Access tree (Υ): An access tree Υ is a tree representation of access structure $\mathbb{A}$. Each non-leaf node represents a threshold gate. Each leaf node y is associated with an attribute of $\mathbb{A}$, and $attr(y)$ returns that attribute. For any node y, it is assumed that $0 \leq t_y \leq num_y$, where t_y and num_y stand for the threshold value and the number of children of y, respectively. The parent of y is represented by $parent(y)$. Each child y of a parent node x is given a number from 1 to num_x denoted by $index(y)$. To decrypt a leaf node, the corresponding attribute secret key is required. However, to decrypt a non-leaf node y, at least t_y children must be decrypted. An access structure $\mathbb{A}$ is said to be satisfied if the root of the corresponding Υ can be decrypted successfully.

Lagrange Coefficient ($\mathcal{L}_{i,Q}$): We define $\mathcal{L}_{i,Q}(y) = \prod_{j \in Q, j \neq i} \frac{y-j}{i-j}$ as the *Lagrange Coefficient* for $i \in \mathbb{Z}_p$ and a set $Q = \{x | x \in \mathbb{Z}_p\}$.

3.2. CRYPTOGRAPHIC PRIMITIVES

Definition 4 ***Bilinear Map:*** *A bilinear map is a function e defined as* $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ *where e must have the following properties:*

1. *Bilinearity: for all* $u, v \in \mathbb{G}$, *and* $a, b \in \mathbb{Z}_p$, *the following relationship must always hold:* $e\left(u^a, v^b\right) = e(u, v)^{ab}$
2. *Non-degeneracy:* $e(g, g) \neq 1$
3. *Computability: group operations in* $\mathbb{G}$ *and e should be efficiently computable.*

3.2.1. Tree-based Group Diffie-Hellman (TGDH).

Tree-based Group Diffie - Hellman is a secure and efficient protocol that can be realized by a complete binary tree called the TGDH tree [32]. Each node in the tree is either a leaf node or has two children. The index of a node is represented as $< l, k >$, where l is the level of the node and

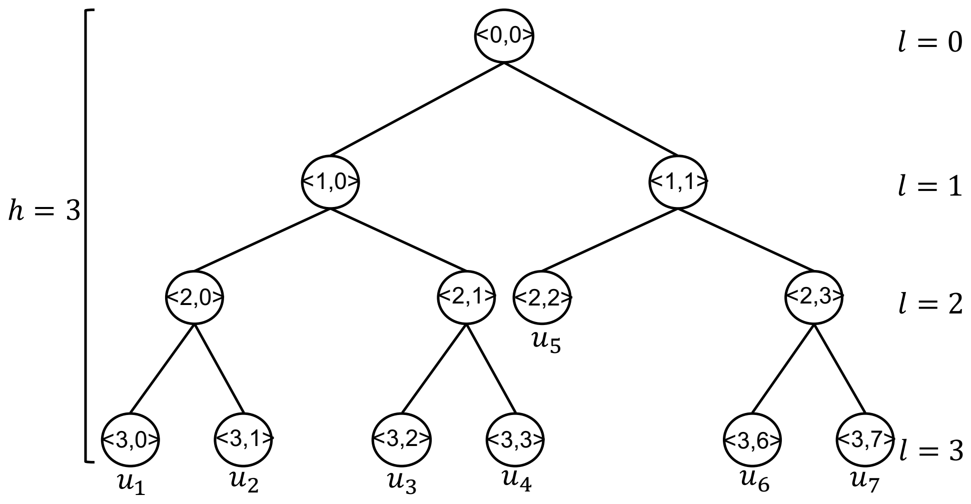


Figure 1. A TGDH *Key Tree* of height 3 consisting of seven members

$0 \leq k \leq 2^l - 1$. Each member of the group is associated with a leaf node. Every node $< l, k >$ in the tree has a secret key $K_{<l,k>}$ and a blinded key $BK_{<l,k>}$ that are calculated as follows:

$$\begin{aligned} BK_{<l,k>} &= g^{K_{<l,k>}} && mod \quad p && (1) \\ K_{<l,k>} &= \mathcal{H}_3\left((BK_{<l+1,2k+1>})^{K_{<l+1,2k>}}\right) && mod \quad p \\ &= \mathcal{H}_3\left((BK_{<l+1,2k>})^{K_{<l+1,2k+1>}}\right) && mod \quad p \\ &= \mathcal{H}_3\left(g^{K_{<l+1,2k>}K_{<l+1,2k+1>}}\right) && mod \quad p && (2) \end{aligned}$$

Equation (2) is a recursive formula and the base case is when node $< l, k >$ is a leaf. Each member is associated with a leaf $< l, k >$ and knows the secret key $K_{<l,k>} = a_i$ of that leaf. This is the user's membership secret key. The set of nodes from a leaf to the root is called the path of that leaf and the set of sibling nodes of all the nodes in the path is called the co-path of that leaf. If an authentic current member of the group at leaf $< l, k >$ knows all the blinded keys of its co-path, it can calculate the secret keys of all the nodes in its path from $< l, k >$ to the root using Equation (2). In TGDH protocol, $K_{<0,0>}$ and $BK_{<0,0>}$ serves as group secret and public key, respectively.

Figure 1 is a TGDH tree consisting of seven members. Member u_4 is associated with leaf $< 3, 3 >$ and his or her co-path is $\{< 3, 2 >, < 2, 0 >, < 1, 1 >\}$. If u_4 knows the blinded key set of its co-path (i.e., $\{BK_{<3,2>}, BK_{<2,0>}, BK_{<1,1>}\}$), he/she can compute the secret key set $\{K_{<2,1>}, K_{<1,0>}, K_{<0,0>}\}$.

3.3. COMPLEXITY ASSUMPTIONS

In the following, we review the complexity assumption of TGDH called DDH (decisional Diffie-Hellmanand) and CDH (computation Diffie-Hellman), as well as the complexity assumption of bilinear map called DBDH (decisional bilinear Diffie-Hellman).

Decisional Diffie-Hellman (DDH) Assumption: Let a, b, r be chosen randomly from $\mathbb{Z}_p$. Then, no probabilistic polynomial time adversary can distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^r) with a non-negligible advantage.

Computation Diffie-Hellman (CDH) Assumption: By following the notations of DDH assumption, given (g, g^a, g^b), no polynomial time adversary can compute g^{ab}.

Decisional Bilinear Diffie-Hellman (DBDH) Assumption: Let g be a generator of a cyclic group $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear pairing. Then, no probabilistic polynomial time adversary can distinguish $\left(g^a, g^b, g^c, e(g, g)^{abc}\right)$ from $\left(g^a, g^b, g^c, e(g, g)^r\right)$ with a non-negligible advantage given that $a, b, c, r \in \mathbb{Z}_p$ are chosen randomly.

4. SYSTEM AND THREAT MODEL

In this section, we present our system model followed by the threat model and its implications.

4.1. SYSTEM MODEL

The system model we consider here is a multi-group cloud storage similar to that of Figure 2. Our system is composed of four basic entities: federated cloud with two different cloud service providers (cloud1 and cloud2), attribute authority (*AA*), group admin (*GA*), and users. Users can be also data owners. There can be multiple user groups, and each group consists of some users and a group admin.

The workflow starts when AA initializes the system by generating the public key (PK), the master secret key (MK), the cloud1 master secret key (C1MK), and the group master secret key (GMK). Then, AA publishes PK and securely sends C1MK and GMK to cloud1 and GAs, respectively while keeps MK secret (not shown in Figure 2). To add a user in the group, GA creates a new leaf node in the TGDH tree and provides him or her with a membership key (MbK) associated with the new leaf. Whereas GA revokes a user by removing the associated leaf node from the tree (Section 6). GA publishes the group public key to the cloud. Then, AA generates attribute secret key (SK_S) for the user's authorized attribute set (S), and securely sends to the user. At the same time, AA secretly sends the transformation key (TK_S) and the update key (UK) corresponding to SK_S to cloud1 and cloud2, respectively. After each revocation or joining epoch, GA sends the proxy update key (UPr) to cloud2, and cloud2 updates the UK of all nonrevoked users. The data owner uploads the ciphertext (CT) to cloud1 for sharing purpose. Eventually, the user requests for the CT from cloud1. If the user is a nonrevoked member and his or her authorized attribute set satisfies the ciphertext policy, then cloud1 and cloud2 jointly carryout outsourced decryption using the user's TK and UK, and returns the partially decrypted ciphertext (CT_{part} to the user. The valid user then computes the group secret key (GSK) from his or her MbK, and fully decrypts CT_{part}. Our model differs from the typical single-group model ([7, 6]) because there can be multiple groups, and operations like group merge and split are allowed.

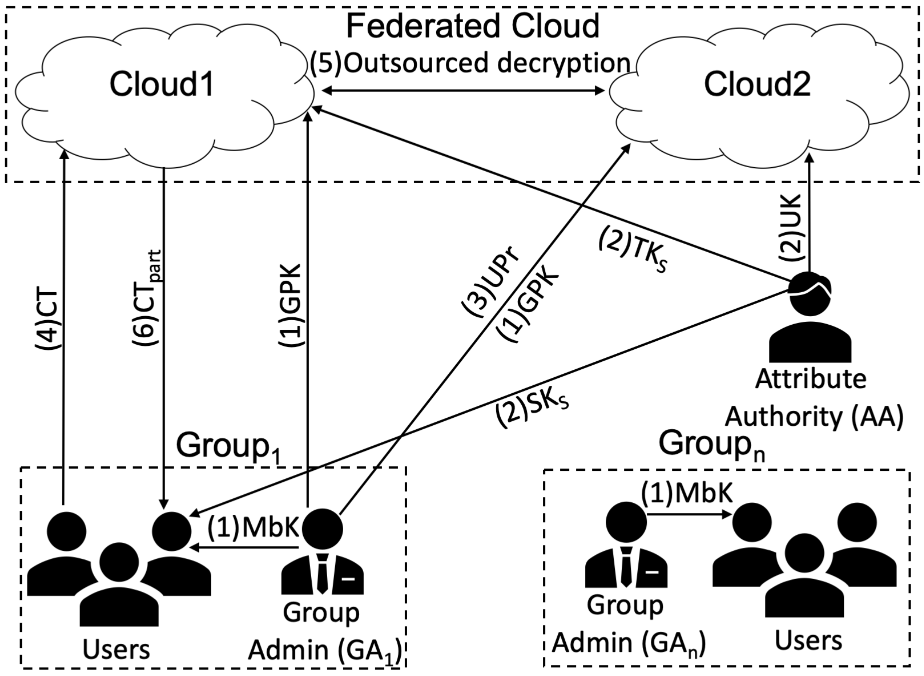


Figure 2. Multi-group cloud storage

4.2. THREAT MODEL

We consider all entities to be preloaded with a public-private key pair and all the secret key distribution happens via a secure channel. We assume the cloud to be a semi-honest entity, a standard practice in revocable ABE literature [7, 4, 8, 9, 11]. Semi-honest means that the cloud is open to deduce any information from the services it provides like storing and updating files and giving them to legitimate users upon request. We also assume that the attribute authority and the group admins are honest entities and do not collude.

In this paper, we consider different types of collusion attacks. To be more specific, we assume that in federation, individual cloud1 or cloud2 (but not both, and clouds don't collude with each other also) may collude with a revoked user to restore his/her decryption ability. We also assume that a revoked user may collude with the nonrevoked user(s) to restore his/her decryption ability. We formally define the possible collusion attacks as follows:

Type-1 collusion attack: Here, cloud1 colludes with a revoked user (while cloud2 follows the protocol) in order to help him/her decrypt a ciphertext. The attribute set (S) in the revoked user's attribute secret key (SK_S) may satisfy the access policy of the ciphertext (CT). To be more specific, cloud1 with TK_S, and the revoked user with SK_S and MbK, tries to decrypt CT without any cooperation from cloud2 given that S may satisfy the access policy of CT.

Type-2 collusion attack: This is same as the type–1 attack except that cloud2 colludes with the revoked user while cloud1 follows the protocol. So in this case, cloud2 with the update key UK, and the revoked user with SK_S and MbK, try to decrypt CT without any cooperation from cloud1 given that S may satisfy the access policy of CT.

Type-3 collusion attack: In this type of collusion attack, a revoked user (who has SK_S and MbK) colludes with a nonrevoked user (who has $\mathrm{SK}'_{S'}$ and MbK′), and one of the clouds (e.g., either cloud1 or cloud2) in order to decrypt a ciphertext CT. However, the constraint is that S may satisfy the access policy of CT but S' does not satisfy it. The reason for this constraint is that if S' satisfies the access policy, then the nonrevoked user can legally decrypt CT (and directly give to the nonrevoked user).

5. EXTENDED TGDH KEY TREE STRUCTURE

Before discussing ReVO-ABE in Section 6, we present its main component extended TGDH (or e-TGDH) tree structure, in this section. As users keep joining or leaving, the TGDH tree discussed in Section 3.2.1 keeps changing. This also results in a new blinded and secret key for each node in the path. For example, revoking u_5 from the key tree in Figure 1 results in the key tree in Figure 3a. Here, blinded and secret keys of node $< 1, 1 >$ and $< 0, 0 >$ are changed. To compute the group secret key ($K_{<0,0>}$), a valid user needs his/her membership secret key and blinded keys of the co-path. If the history of the blinded key is not saved, nonrevoked users cannot compute any previous group secret key unless

Table 2. The e-TGDH index table for group *GID*

UID	(*version*, $< l, k >$)
u_1	**(0,<3,0>)**
u_2	**(0,<3,1>)**
u_3	**(0,<3,2>)**
u_4	**(0,<3,3>)**
u_5	**(0,<2,2>)**
u_6	**(0,<3,6>)**, (1,< 2,2>)
u_7	**(0,<3,7>)**, (1,<2,3>), (2,<3,6>)
u_8	(2,<3,7>)

one has previously been computed and saved. Thus, it is necessary to preserve blinded key history. One solution approach is to create a new tree each time, but this poses a huge overhead as a tree of size 2^m needs to be created every time. To solve this problem efficiently, we propose a new data structure named the e-TGDH tree. It is based on the observation that a revocation or joining only affects the keys of nodes in the path. Therefore, a new version of the tree can be generated by reusing the unaffected nodes and creating new nodes only for those who are affected in the path. This requires only O(log$m$) additional space to create a new version of the tree while keeping the previous one.

Versioning and Indexing: We will use a simple versioning technique to keep track of the changes made in the tree. $CurrVer_j$ (initially 0) will be used to refer to the latest version number for a group, say j. According to our scheme, the group admin will store only a single TGDH tree of the latest version that contains both secret and blinded keys. The clouds will store the e-TGDH tree that contains only blinded keys. the indices of the left and right child of an intermediate node $< l, k >$ are represented as $< l + 1, 2k >$ and $< l + 1, 2k + 1 >$, respectively. The advantage is that the binary representation of k in l digits is actually the traversal path of node $< l, k >$ from the root. For example, the traversal path of leaf $< 3, 2 >$ in Figure 1 is 010, where 0 and 1 denote left and right, respectively. In the e-TGDH tree, a node might be shared among multiple tree versions and its index may vary in different versions. The clouds maintain a table called e-TGDH index to store node indices for different versions. Since users are associated with leaf nodes, only leaf indices are required to be saved in the table. We will use Table 2 to illustrate how indices are saved as e-TGDH tree changes. Bold-faced entries in the second column (having version=0) represent the initial leaf indices for the users of the tree in Figure 1.

Algorithm 1 Revokes *UID* from the group and updates the e-TGDH tree

```
Procedure: Revoke-User ( list_BK, root, GID, version, UID )
 1: new_root = new_ptr = old_ptr = next ← null
 2: path ← Get-Path ( GID, version, UID )
 3: if list_BK is empty then
 4:    new_root ← sibling of UID's associated leaf node
 5:    Record-Index ( version+1, new_root, 0, 0 )
 6: else
 7:    new_root = new_ptr ← New-Node(); old_ptr ← root
 8:    i = l = k ← 0; new_ptr.BK ← list_BK[i]
 9:    while path.length-2 > i do
10:       next ← New-Node()
11:       if path[i]==0 then
12:          new_ptr.right ← old_ptr.right; new_ptr.left ← next
13:          old_ptr ← old_ptr.left; k ← 2k
14:       else
15:          new_ptr.left ← old_ptr.left; new_ptr.right ← next
16:          old_ptr ← old_ptr.right; k ← 2k + 1
17:       end if
18:       i++; l++; next.BK ← list_BK[i]; new_ptr ← next
19:    end while
20:    tmp_ptr ← old_ptr
21:    old_ptr ← (path[i]==0 ? old_ptr.left : old_ptr.right)
22:    old_ptr ← (path[++i]==0 ? old_ptr.right : old_ptr.left)
23:    if path[i − 1]==0 then
24:       new_ptr.right← tmp_ptr.right; new_ptr.left← old_ptr
25:       Record-Index ( version+1, old_ptr, l+1, 2k )
26:    else
27:       new_ptr.left← tmp_ptr.left; new_ptr.right← old_ptr
28:       Record-Index ( version+1, old_ptr, l+1, 2k+1 )
29:    end if
30: end if
31: return ( new_root, ++version )
```

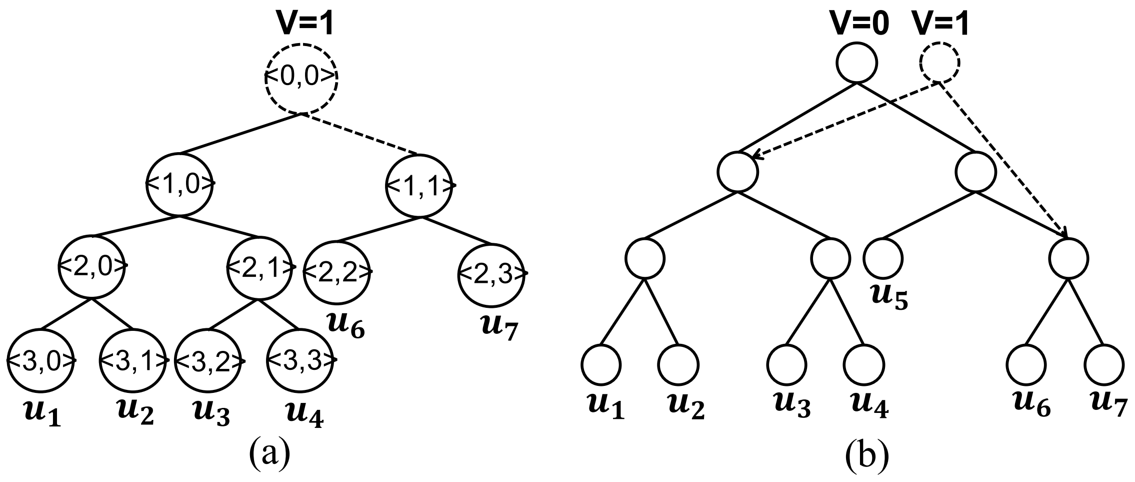


Figure 3. Group admin and cloud's view of the TGDH and e-TGDH tree respectively after removing user u_5 from the group

The changes caused by user revocation or joining is propagated along the e-TGDH tree, which is discussed in Section 5.1 and 5.2, respectively.

5.1. USER REVOCATION

To revoke a user from the group, the group admin deletes the corresponding leaf node from its TGDH tree, updates the blinded and secret keys of the path, and increments $CurrVer_j$. The admin then sends *GID*- the group id, *UID*- the user id, and *list_BK*- the list of updated blinded keys in the path to the cloud. The cloud calls procedure "Revoke-User" in Alg. 1 with parameters *version*, *root*, and received values from the admin, where *version* equals $CurrVer_j$ and *root* is the corresponding e-TGDH tree root for that version. The algorithm first gets the traversal path of the leaf node associated with *UID* by calling the subroutine "Get-Path" (line 2). This subroutine gets the leaf index of *UID* with the highest *version* value from e-TGDH index table and returns k as a l characters binary string. An empty *list_BK* (line 3) means that *UID* is associated with a leaf that is a child of the root. In this case, the other child of the root becomes the new root (line 4). Then, subroutine

Record-Index (Alg. 3) is called on the new root (line 5). This subroutine creates new entries in an e-TGDH index table for the users whose leaf node indices have been changed. If the *list_BK* is not empty, a new node is created for each *BK* in *list_BK* (line 7–19). Each new node will have two children; one being the next new node and the other being the existing unaffected child in the tree. In line 20–29, the sibling node of *UID*'s associated leaf node is added and "Record-Index" is called on that node to create new leaf indices with new a version number in the e-TGDH index table. Finally, *version* is incremented and returned with the *new_root* (line 31). Now, *version* is assigned to $CurrVer_j$ and *new_root* becomes the root for this new *version*.

Example: Let us assume that Figure 1 represents the TGDH tree of *version*=0 for a group of seven users. After revoking a user u_5, Figure 3a becomes the admin's view of the TGDH tree. Admin sends u_5, *GID*, and $\{BK_{<0,0>}\}$ to the cloud. Then, the cloud calls Alg. 2. The value of *path* in line 2 becomes '10' since $(0, <2, 2>)$ is the entry with the greatest *version* value. A new node is created and $BK_{<0,0>}$ becomes its blinded key. "Record-Index" is called on the sibling node of u_5 that results in $(1, <2, 2>)$ and $(1, <2, 3>)$ entries for u_6 and u_7 in Table 2. Finally, the *version* is incremented to 1 and is returned with the new *root*. As a result, $CurrVer_j$ becomes 1. Figure 3b shows the e-TGDH tree after the revocation. Dotted lines have been used to show the changes that happened due to this revocation.

5.2. NEW USER JOINING

To add a new user with user id (*UID*) to the group, the admin picks the shallowest leaf node associated with user id (*_UID*) and replaces the leaf node with a node having two children, the left child being the shallowest leaf and the right child being a new node associated with *UID*. Eventually the cloud gets *UID, _UID*, and *list_BK* from the admin and calls the procedure described in Alg. 2. "Get-Path" subroutine is called (line 1) to get the *path* of *_UID*. A node is added for each *BK* in *list_BK* (line 4–14) similarly to that of Alg. 1. In line 15–17, the leaf of *_UID* becomes the left leaf of its new parent. A new node is created for the new user *UID* and is assigned as the right leaf. "Record-Index" is called on the new parent to record the index with new version for *_UID* and *UID*. Finally, *version* is incremented and returned with *new_root*. $CurrVer_j$ and the corresponding root are also updated accordingly.

Algorithm 2 Adds *UID* to the group and updates the e-TGDH tree

```
Procedure: Add-User ( list_BK, root, GID, version, UID, _UID )
 1: path ← Get-Path ( GID, version, _UID )
 2: new_root = new_ptr ← New-Node(); old_ptr ← root
 3: i = l = k ← 0; new_root.BK ← list_BK[i]
 4: while path.length > i do
 5:     next ← New-Node()
 6:     if path[i]==0 then
 7:         new_ptr.right ← old_ptr.right; new_ptr.left ← next
 8:         old_ptr ← old_ptr.left; k ← 2k
 9:     else
10:         new_ptr.left ← old_ptr.left; new_ptr.right ← next
11:         old_ptr ← old_ptr.right; k ← 2k + 1
12:     end if
13:     i++; l++; next.BK ← list_BK[i]; new_ptr ← next
14: end while
15: new_ptr.left ← old_ptr; new_ptr.right ← New-Node()
16: new_ptr.right.BK ← list_BK[i+1]
17: new_ptr.right.UID ← UID; new_ptr.right.leaf ← True
18: Record-Index ( version+1, new_ptr, l, k)
19: return ( new_root, ++version )
```

Algorithm 3 Records the updated indices in the e-TGDH index table

```
Procedure: Record-Index ( version, node, l, k )
 1: if node is a leaf then
 2:     Write ( node.UID, version, l, k )
 3: else
 4:     Record-Index ( version, node.left, l+1, 2k )
 5:     Record-Index ( version, node.right, l+1, 2k+1 )
 6: end if
```

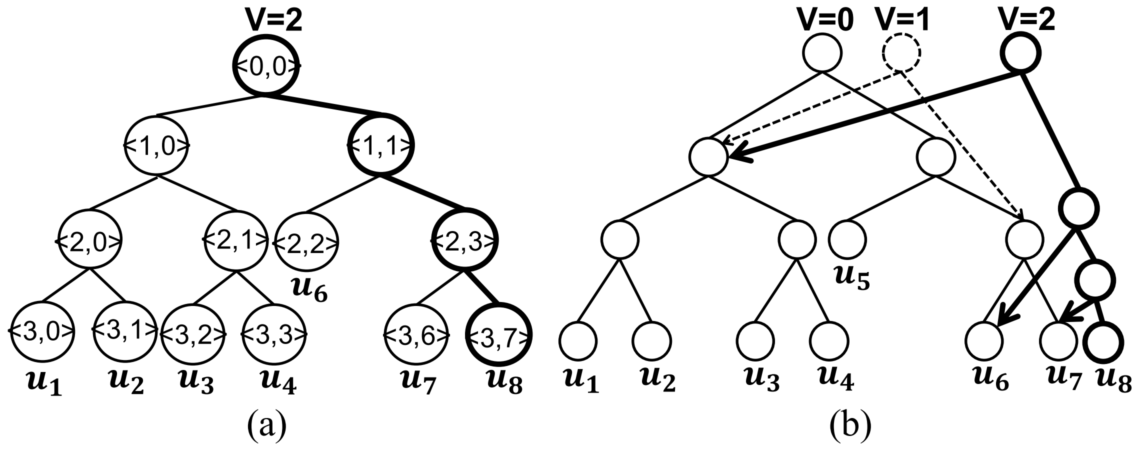

V=2
<0,0>
<1,0>
<1,1>
<2,0>
<2,1>
<2,2>
<2,3>
<3,0>
<3,1>
<3,2>
<3,3>
<3,6>
<3,7>
u_1 u_2 u_3 u_4 u_6 u_7 u_8
(a)
V=0 V=1 V=2
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8
(b)

Figure 4. Group admin and cloud's view of the TGDH and e-TGDH tree respectively after adding user u_8 in the group

Example: Figure 4a is the result of adding u_8 to the tree in Figure 3a by the admin. Note that u_8 has been added to node $< 2, 3 >$ that used to be the leaf node of u_7. The admin sends *_UID* = u_7, *UID* = u_8, and *list_BK* = $\{BK_{<0,0>}, BK_{<1,1>}, BK_{<2,3>}, BK_{<3,7>}\}$ to the cloud. To add u_8 to the e-TGDH tree, the cloud calls "ADD-User". The variable *path* becomes '11' since $(1, < 2, 2 >)$ has the greatest *version* in the e-TGDH index table for u_7 at that time. Then, the counterpart of Figure 4a is created in the e-TGDH tree by adding a new node for each *BK* in *list_BK*. Also, $(2, < 3, 6 >)$ and $(2, < 3, 7 >)$ are added in Table 2. $CurrVer_j$ becomes 2 after the function returns. Figure 4b represents the e-TGDH tree after adding u_8. Thick lines have been used to mark the changes caused by this event.

5.3. UNLIMITED USER REVOCATION AND JOINING

In order to revoke or add a user, the admin deletes the corresponding leaf node or adds a new node in the TGDH tree, respectively. The admin also updates the blinded keys along the path and sends to the cloud so that cloud can update the e-TGDH tree. Note that the secret key re-distribution to the existing users is not required as the existing users can compute the new group secret key ($K_{<0,0>}$) with the updated public blinded key available in the cloud. Since leaves can be deleted or new leaves can be added as many time as required, the number of revocation or new user joining in our scheme have no limit as well. On the contrary, the schemes such as [9, 10, 11, 8, 13, 5, 14] uses a static key tree because adding or removing leaves changes the secret membership keys of the existing users. Hence, the admin would require sending new secret membership keys to those users. It is avoided by creating a large static tree that limits the number of total users in the group.

6. PROPOSED REVO-ABE SCHEME

We construct our ReVO-ABE scheme by performing the following transformations to CP-ABE$_0$. First, the user is given a membership key (MbK) associated with the leaf of e-TGDH tree in addition to the attribute secret key (SK$_S$). Cloud1 and cloud2 are given the cloud1 master key (C1MK) and the update key (UK), respectively. For secure outsource decryption, cloud1 is given the transformation key (TK$_S$), created by blinding SK$_S$ with a random exponent t. The data owner creates ciphertext (CT) by encrypting with a policy and the group public key (GPK). The GPK is created from all the non-revoked users' MbKs so that revoked users cannot decrypt. We cryptographically bind SK$_S$, MbK, UK, and C1MK in a novel way so that collusion attacks can be prevented. Revocation or joining is done by removing or adding a node in the e-TGDH tree and updating the GPK from all nonrevoked users' MbKs. Since SK$_S$, MbK, UK, and C1MK are cryptographically binded together, a user's access is immediately revoked after removing him/her from the e-TGDH tree as long as one of the clouds acts honestly. Our detailed ReVO-ABE construction is as follows:

- `Setup`$(1^{\mathcal{K}})$: This algorithm is run by AA. The algorithm takes as input the security parameter $\mathcal{K}$ and generates a group $\mathbb{G}$ of prime order p with a generator g and chooses hash functions $\mathcal{H}$, $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$. In addition, it chooses *randomness extractor* F and two random exponents $\alpha, \beta \in \mathbb{Z}_p$. It randomly chooses $t_c \in \mathbb{Z}_p$ and sets cloud1 master secret key as C1MK $= t_c$, and computes the group master secret key as GMK=$g^{t_c/\beta}$.

The master secret key MK= (β, g^{α}, t_c) is kept secret, and the public key is published as PK= $(\mathbb{G}, g, \mathbb{G}_T, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, F, e(g,g)^{\alpha}, h = g^{\beta})$. The C1MK and GMK is sent to cloud1, and all group admins (GAs), respectively.

• `Group-Setup`(PK, GMK, N, j): This algorithm takes as input PK, GMK, the total number of initial users in the group (N), and the group id (j). Each user is identified with a user id (u_i), where $1 \leq i \leq N$. It initializes the TGDH key tree with N leaf nodes and associates each u_i with a leaf node that has a membership secret key $a_i \in \mathbb{Z}_p$. The blinded and secret key for all the intermediate nodes are calculated. Let K_0^j and $g^{K_0^j}$ be the initial root secret key ($K_{<0,0>}$) and blinded key ($BK_{<0,0>}$) of group j, respectively. Then, GA randomly chooses $b_0 \in \mathbb{Z}_p$ and publishes the initial group public key $\text{GPK}_0^j = (g^{K_0^j}, g^{b_0 K_0^j}, g^{\beta K_0^j})$, and keeps $\text{GSK}_0^i = K_0^j$ secret. GA also publishes the TGDH key tree (without the secret key of any node) so that cloud1 and the cloud2 can create the corresponding e-TGDH tree.

• `KeyGen`(PK, MK, $S, G, \{\text{GPK}_l^j\}_{\forall j \in G}$): The key generation algorithm chooses r and $\{r_1, r_2, \cdots, r_{|s|}\}$ randomly from $\mathbb{Z}_p$. Then, it computes $D_{1,G} = \{D_{1,j} = g^{(\alpha + r + b_l K_l^j) t_c / \beta}\}_{\forall j \in G}$ and $\text{SK}'_S = (\{D'_{2,i} = g^{r/\beta}.\mathcal{H}(i)^{r_i}, D'_{3,i} = g^{\beta r_i}\}_{\forall i \in S})$ where S and G are user's authorized attribute set and groups, respectively. Next, it chooses a random exponent $t \in \mathbb{Z}_p$ and computes the transformation key $\text{TK}_S = (\{D_{2,i} = g^{r/\beta t}.\mathcal{H}(i)^{r_i/t}, D_{3,i} = g^{\beta r_i / t}\}_{\forall i \in S})$. Finally, it sends $D_{1,G}$ to cloud2, TK_S to cloud1, and the attribute secret key $\text{SK}_S = (\text{TK}_S, t)$ to the user.

• `Group-KeyGen`(PK, GMK, j, GPK_l^j, u_i): If u_i is an existing user of group j, then this algorithm outputs the membership secret key a_i of the user's associated leaf node in the TGDH key tree. Otherwise, it replaces the shallowest node $< l', k >$ with a parent node having two children labeled as $< l' + 1, 2k >$ and $< l' + 1, 2k + 1 >$. The old $< l', k >$ node becomes $< l' + 1, 2k >$, and u_i is associated with $< l' + 1, 2k + 1 >$. Next, it randomly selects $a_i \in \mathbb{Z}_p$, sets $K_{<l'+1, 2k+1>} = a_i$ and updates the blinded and secret keys of all the nodes along the path. GA increments the version to $l + 1$, and updates the group secret key as $\text{GSK}_{l+1}^j = K_{l+1}^j = K_{<0,0>}$ and the group public key as $\text{GPK}_{l+1}^j = (g^{K_{l+1}^j}, g^{b_{l+1} K_{l+1}^j}, g^{\beta K_{l+1}^j})$.

GA also computes a proxy update key $\mathrm{UPr}^j_{l\to l+1} = g^{(b_{l+1}K^j_{l+1} - b_l K^j_l)t_c/\beta}$. It securely sends a_i and $\mathrm{UPr}^j_{l\to l+1}$ to the user and cloud2, respectively. Both cloud1 and cloud2 updates their copy of e-TGDH tree using Alg. 2. Cloud2 updates $D_{1,j}$ for all nonrevoked users as in $D'_{1,j} = D_{1,j}\mathrm{UPr}^j_{l\to l+1} = g^{(\alpha+r+b_{l+1}K^j_{l+1})t_c/\beta}$.

• `Encrypt`(PK, M, $\mathbb{A}$, $\{j, G\}$, $\{\mathrm{GPK}^{j'}_l\}_{\forall j' \in G\cup j}$): This algorithm selects a random polynomial P_y of degree d_y for each node y of Υ corresponding to $\mathbb{A}$ with the condition $d_y = t_y - 1$ in the following manner. First, a random $s \in \mathbb{Z}_p$ is chosen to set $P_R(0) = s$, where P_R represents the polynomial associated with the root R. Then, d_R number of other points on P_R are chosen randomly to define it completely. For any other node y, the algorithm completely defines the corresponding polynomial P_y by setting $P_y(0) = P_{parent(y)}(index(y))$ and choosing d_y random values on P_y. This process is carried out in a top-to-bottom fashion starting from the root R. Let $\mathcal{V}$ be the set of leaf nodes in Υ. Afterwards, it computes $C'_{0,j} = e(g,g)^{\alpha s + b_l K^j_l s}$, $\mathrm{CPr} = \{\mathrm{CPr}_{j\to j'} = e(g,g)^{b_{l'}K^{j'}_{l'}s - b_l K^j_l s}\}_{\forall j' \in G}$, $C_{1,G} = \{C_{1,j'} = g^{\beta K^{j'}_l s}\}_{\forall j' \in G\cup j}$, $\{C_{2,v} = g^{P_v(0)\beta}, C_{3,v} = \mathcal{H}(attr(v))^{P_v(0)}\}_{\forall v \in \mathcal{V}}$. Next, it chooses a random seed $K_R \in \mathbb{G}_T$ and computes $h_1 = \mathcal{H}_1(K_R)$, the symmetric encryption key $K_{SE} = F(K_R)$, symmetric ciphertext $\mathrm{CT}_{SE} = Enc_{K_{SE}}(\mathrm{M})$, and the verification key $\mathrm{VK} = \mathcal{H}_2(h_1 || \mathrm{CT}_{SE})$. Finally, the algorithm outputs the complete ciphertext as $\mathrm{CT} = (C_{0,j} = K_R C'_{0,j} = K_R e(g,g)^{\alpha s + b_l K^j_l s}, \{C_{2,v}, C_{3,v}\}_{\forall v \in \mathcal{V}}, \mathrm{CT}_{SE}, \mathrm{CPr}, C_{1,G}, \mathrm{VK})$.

• `Transform`(TK_S, CT, $D_{1,j}$, j, u_i): This algorithm partially decrypts CT by using a recursive algorithm called `DecryptNode` (CT, TK_S, v) that works as follows. It returns F_v when called on a node v in the access tree Υ corresponding to $\mathbb{A}$. The base case of the algorithm includes the fact when v is a leaf node. For an attribute $i = attr(v)$, if $i \notin$ S, then $F_v = \perp$. Otherwise,

$$F_v = \frac{e(C_{2,v}, D_{2,i})}{e(C_{3,v}, D_{3,i})} = \frac{e(g^{P_v(0)\beta}, (g^{r/\beta}.\mathcal{H}(i)^{r_i})^{1/t})}{e(\mathcal{H}(attr(v))^{P_v(0)}, g^{\beta r_i/t})}$$
$$= e(g,g)^{rP_v(0)/t}.$$

The recursive case of the algorithm is when v is a non-leaf node. Let us assume that $\mathcal{S}_v$ is the set of successfully decrypted child nodes of v. If $|\mathcal{S}_v| < t_v$, then $F_v = \perp$. Otherwise, the following value is returned:

$$\begin{aligned}
F_v &= \prod_{y \in \mathcal{S}_v} F_y^{\mathcal{L}_{j,\mathcal{S}'_v}(0)} \text{;where } {}^{j=index(y)}_{\mathcal{S}'_v=\{index(y):y\in\mathcal{S}_v\}} \\
&= \prod_{y \in \mathcal{S}_v} \left(e\left(g,g\right)^{r.P_{parent(y)}\left(index(y)\right)/t} \right)^{\mathcal{L}_{j,\mathcal{S}'_v}(0)} \\
&= \prod_{y \in \mathcal{S}_v} e\left(g,g\right)^{rP_v(j).\mathcal{L}_{j,\mathcal{S}'_v}(0)/t} = e(g,g)^{r.P_v(0)/t}.
\end{aligned}$$

Eventually, `DecryptNode` returns $F_R = e(g,g)^{rs/t}$ if root R of Υ is successfully decrypted. A failure symbol $\perp$ is returned otherwise.

If the user (u_i) is a valid member of group j, cloud1 chooses a random $b \in \mathbb{Z}_p$ and sends $(u_i, j, C'_{1,j} = (C_{1,j})^b)$ to cloud2. Then, cloud2 computes $T = e(C'_{1,j}, D_{1,j})$ and sends it back to cloud1. After that, cloud1 computes $T_1 = T^{1/t_c b} = e(g,g)^{(\alpha+r+b_l K_l^j)K_l^j s}$. Then, it either sets $T_2 = C_{0,j}$, or computes $T_2 = C_{0,j'}\text{CPr}_{j'\to j} = K_R e(g,g)^{\alpha s + b_l K_l^j s}$ (if $j \neq j'$and $j \in G$). Finally, cloud1 sends $\text{CT}_{part} = (T_1, T_2, F_R, \text{CT}_{SE}, \text{VK})$ to the user.

• `Decrypt`(PK, SK$_\text{S}$, CT$_{part}$, GSK$_l^j$): The decryption algorithm first computes $K'_R = T_2(F_R)^t / T_1^{1/K_l^j}$. Next, it verifies the outsourced delegation by

$$verify = \begin{cases} 1 & \text{if VK} = \mathcal{H}_2(\mathcal{H}_1(K'_R)||\text{CT}_{SE}) \\ 0 & \text{otherwise.} \end{cases}$$

If verification fails, $\perp$ is returned. Otherwise, it computes $K_{SE} = F(K'_R)$ and returns $\text{M} = Dec_{K_{SE}}(\text{CT}_{SE})$.

• `Revoke`(u_i, GMK, j, l): This algorithm removes the user (u_i) from the group j. Then, GA removes the leaf node associated with u_i from the TGDH tree, merges its sibling node with its parent, and updates all the blinded and secret keys along the path from the leaf to the root. GA increments the version to $l + 1$, and updates the group secret key as $\text{GSK}^j_{l+1} = K^j_{l+1} = K_{<0,0>}$ and the group public key as $\text{GPK}^j_{l+1} = (g^{K^j_{l+1}}, g^{b_{l+1}K^j_{l+1}}, g^{\beta K^j_{l+1}})$. GA also computes the proxy update key $\text{UPr}^j_{l\to l+1} = g^{(b_{l+1}K^j_{l+1} - b_l K^j_l)/\beta}$. It secretly sends $\text{UPr}^j_{l\to l+1}$ to cloud2. The e-TGDH tree is also modified by cloud1 and cloud2 using Alg. 1 to reflect the changes. Cloud2 updates $D_{1,j}$ for all nonrevoked users as in $D'_{1,j} = D_{1,j}\text{UPr}^j_{l\to l+1} = g^{(\alpha+r+b_{l+1}K^j_{l+1})t_c/\beta}$.

7. PROPOSED DMG-SDS SCHEME

In this section, we discuss how we use ReVO-ABE to construct our dynamic multi-group secure data sharing (DMG-SDS) scheme. The DMG-SDS scheme calls the ReVO-ABE algorithms in the backend. There are multiple groups in the system and each group is associated with an e-TGDH tree and a TGDH tree. They are maintained by the cloud (cloud1 and cloud2) and the corresponding group admin, respectively. DMG-SDS scheme supports group merge and group split. The data owner determines if re-encryption is required during file update by comparing the associated version l with $CurrVer_j$ of the TGDH. The details are as follows:

7.1. SYSTEM INITIALIZATION

The attribute authority calls `Setup` algorithm to generate PK and MK. It publishes PK and keeps MK secret. Let us assume that initially, there are n different groups $\{1, 2, \cdots, n\}$ in the system and each group j has m_j users (i.e., $j = \{u_{j,1}, u_{j,2}, \cdots, u_{j,m_j}\}$). For each group j, the group admin GA_j calls `Group-Setup` and initializes the TGDH key tree T_{G_j}. Then, each GA_j sends to the cloud the respective T_{G_j} with only the blinded keys and the cloud builds the initial e-TGDH tree from it. Both the cloud and GA_j sets $CurrVer_j$ equals to zero.

7.2. MEMBER JOINING

A user joins a set of groups (G) by the following two steps:

- Each GA_j first assigns a user id $u_{j,i}$ to the new user. Then, calls `Group-KeyGen` with arguments (PK, GMK, j, GPK_l^j, u_i) and returns membership secret key a_i to $u_{j,i}$.
- The user requests attribute secret keys for the authorized attribute set S. Attribute authority calls `KeyGen`(PK, MK, S, G, $\{\text{GPK}_l^j\}_{\forall j \in G}$). It sends $D_{1,G}$ to cloud2, TK_S to cloud1, and the attribute secret key $\text{SK}_S = (\text{TK}_S, t)$ to the user.

7.3. FILE OUTSOURCING

A data owner has to encrypt a file before uploading it to the cloud to protect its privacy from potential adversaries. To outsource a file M for a set of groups $\{G \cup j\}$, the owner follows the steps below:

- Gets the latest GPK_l^j for $\forall j' \in \{G \cup j\}$ from the cloud.
- Calls `Encrypt`(PK, M, $\mathbb{A}$, $\{j, G\}$, $\{\text{GPK}_l^{j'}\}_{\forall j' \in G \cup j}$) and uploads (ID, CT) to cloud1 where ID represents the file id.

7.4. FILE RETRIEVING

User $u_{j,i}$ requests a file from the cloud by sending Req=($u_{j,i}$, j, ID). Then, the cloud and the user act as follows:

Cloud:

- The cloud1 retrieves ciphertext (ID, CT) from its storage and the blinded key list $\mathbb{B}$ of the user's co-path from the e-TGDH tree.
- Cloud1 and cloud2 jointly run `Transform` algorithm, and either CT_{part} (if the user $u_{i,j}$ is a valid member of group j and the attributes in the secret key satisfy the policy) or $\perp$ is returned.
- Cloud1 sends (ID, $\text{CT}_{part}/\perp, \mathbb{B}$) to the user.

User:

- Let l be the associated version of CT_{part}. Then, $u_{j,i}$ computes GSK_l^j from a_i and $\mathbb{B}$ if $u_{j,i}$ has not already computed it previously.
- Finally, calls `Decrypt` that outputs either M or $\perp$.

7.5. MEMBER REVOCATION

The steps to revoke $u_{j,i}$ from the group j are as follows:

- The GA_j calls `Revoke`($u_{j,i}$, GMK, j, l) that results in an incremented version of $l + 1$, proxy update key $\text{UPr}^j_{l \to l+1}$ and updated (GSK^j_{l+1}, GPK^j_{l+1}).
- Cloud2 updates $D_{1,j}$ of all non-revoked users by proxy re-encrypting the old $D_{1,j}$ with $\text{UPr}^j_{l \to l+1}$.

From this point, owners have to use the new GPK^j_{l+1} for encrypting or updating any file. Note that by repeating this process, any number of user revocation is possible.

7.6. FILE UPDATE

For file update, the data owner checks if $l \neq CurrVer_j$. In that case re-encryption is necessary, and the data owner creates anonymous updates for timestamp ts' as in $U_{ts \to ts'}$=(ts', $U1_{ts \to ts'} = e(g, g)^{b_{l'} K^j_{l'} s - b_l K^j_l s}$, $U2_{ts \to ts'} = g^{\beta K^j_{l'} s}$), where l' and l are the version numbers at timestamps ts' and ts ($ts' > ts$), respectively. The owner sends $U_{ts \to ts'}$ to cloud1, and cloud1 updates CT by replacing $C_{0,j}$ and $C_{1,j}$ with $C_{0,j}U1_{ts \to ts'}$ and $U2_{ts \to ts'}$, respectively. For this method to work properly, the owner needs to securely store s during creating the ciphertext (CT) so that it can be used later during creating updates (e.g., $U_{ts \to ts'}$).

7.7. GROUP MERGE

In our group merge operation, multiple groups are merged into a single group without requiring any user to change his/her membership secret key. Even this is done without the exchange of any secret information between group admins. To merge group$j$ with k, group admin GA_j and GA_k run between themselves a *Diffie-Hellman* key exchange protocol with authentication as follows. Let ($\mathrm{SK}_{G_j} = \mathrm{GSK}_l^j = a, \mathrm{PK}_{G_j} = g^a$) and ($\mathrm{SK}_{G_k} = \mathrm{GSK}_{l'}^k = b, \mathrm{PK}_{G_k} = g^b$) be the TGDH root secret and blinded key pair of group j and k, respectively. (KU_j, KR_j), (KU_k, KR_k) are the public and private digital signature key pairs of GA_j and GA_k, respectively. We assume that GA_j and GA_k know each other's public signature key beforehand from the attribute authority. One of the group admins (say GA_j) starts the protocol by sending PK_{G_j} to GA_k. Then, GA_k computes $\mathrm{SK}_{G_{jk}} = \mathcal{H}_3\left(g^{ab}\right)$, $h_1 = H_2(KU_k || \mathrm{PK}_{G_j} || \mathrm{PK}_{G_k} || \mathrm{SK}_{G_{jk}})$, $\sigma_1 = Sign(KR_k, h_1)$ and sends $(\mathrm{PK}_{G_k}, h_1, \sigma_1)$ to GA_j. Group admin GA_j verifies signature σ_1 as in $Verify(KU_k, \sigma_1) \stackrel{?}{=} h_1$. Then, GA_j computes $\mathrm{SK}_{G_{jk}}$ and verifies hash h_1 as in $H_2(KU_k || \mathrm{PK}_{G_j} || \mathrm{PK}_{G_k} || \mathrm{SK}_{G_{jk}}) \stackrel{?}{=} h_1$. At this point, GA_j knows he is talking to GA_k. Now GA_j sends $(h_2 = H_2(KU_j || \mathrm{SK}_{G_{jk}}), \sigma_2 = Sign(KR_j, h_2))$ to GA_k. Then, GA_k makes sure he is talking to GA_j by checking $Verify(KU_j, \sigma_2) \stackrel{?}{=} h_2$ and $H_2(KU_j || \mathrm{SK}_{G_{jk}}) \stackrel{?}{=} h_2$. Finally, admins compute corresponding root blinded key $\mathrm{PK}_{G_{jk}} = g^{\mathrm{SK}_{G_{jk}}}$ and send it to both cloud1 and cloud2. Both clouds simply merge the corresponding e-TGDH trees by making them left and right subtrees of a newly created root node with $\mathrm{PK}_{G_{jk}}$ as its blinded key. The merged group's secret and public keys are initialized as $\mathrm{GSK}_0^{jk} = K_0^{jk} = \mathrm{SK}_{G_{jk}}$ and $\mathrm{GPK}_0^{jk} = (g^{K_0^{jk}}, g^{b_0 K_0^{jk}}, g^{\beta K_0^{j}})$.

Table 3. Parameter of different pairing groups

Curves	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	p	k	*Security*
SS512	512	512	1024	160	2	80
MNT224	224	672	1344	224	6	100

7.8. GROUP SPLIT

For splitting a group into multiple subgroups, the group admin needs to split the TGDH tree and create a new TGDH tree for each sub-group. To reduce the number of blinded and secret key computation, the admin forms new TGDH trees from the sub-trees having users from the same group. Let L be the list of a list of leaf nodes corresponding to the same sub-group. The algorithm has the following steps: 1) For a leaf list in L, color all the nodes in the path of all other nodes in TGDH tree. 2) Find all the uncolored subtrees and form a new TGDH tree using their roots. 3) Reset colors of all colored nodes. 4) Repeat from step 1 for all lists in L. 5) Return roots of all the new TGDH trees. The admin then sends to the cloud all the updated blinded key and the cloud carries out the update accordingly. Note that users do not need to change their membership secret keys. For example, say that the admin wants to split the group of seven users in Figure 1 into two sub-groups with members $\{u_1, u_2, u_6, u_7\}$ and $\{u_3, u_4, u_5\}$. So, L = $\{\{u_1, u_2, u_6, u_7\}, \{u_3, u_4, u_5\}\}$. To create TGDH tree for the first group, in step 2 of the algorithm, a new root is created and its left and right subtrees are set as subtrees rooted at $< 2, 0 >$ and $< 2, 3 >$, respectively. Note that for this tree, the admin only needs to create secret and blinded key of the new root as in $K'_{<0,0>} = H_3(g^{K_{<2,0>}K_{<2,3>}})$ and $BK'_{<0,0>} = g^{K'_{<0,0>}}$ and reuses all existing keys in subtrees rooted at $< 2, 0 >$ and $< 2, 3 >$. Likewise, a TGDH tree for the second group is generated by creating a new root and making node $< 2, 1 >$ and $< 2, 3 >$ its left and right child, respectively.

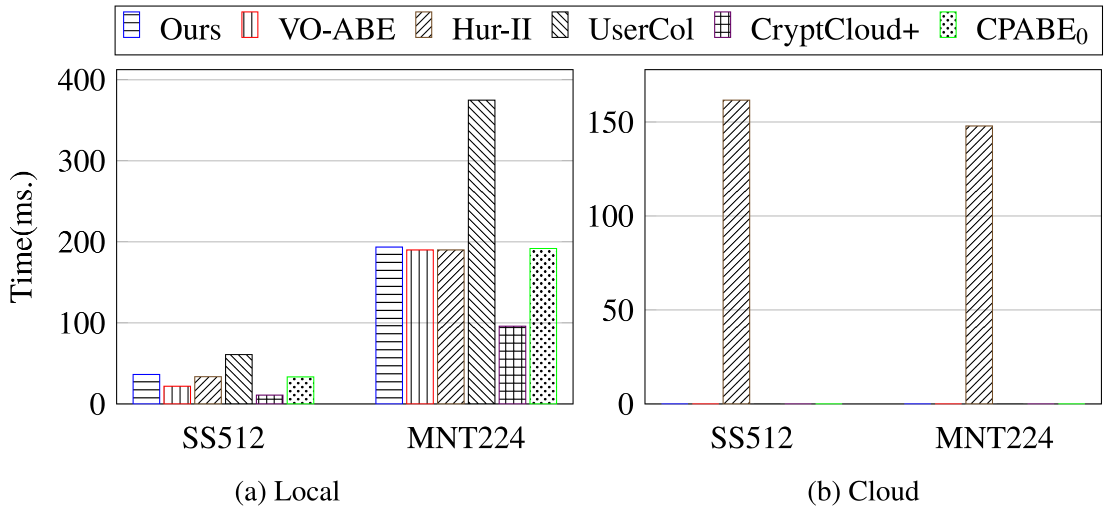


Figure 5. Encryption

8. IMPLEMENTATION AND EVALUATION

Implementation: We have implemented our scheme in `Charm` [33]. It is a Python-based framework developed for rapid prototyping of advanced cryptographic protocols. `Charm` uses PBC library [34] (written in C) for low-level system calls, including the most expensive group exponentiation and pairing operations. Hash functions $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2$, and $\mathcal{H}_3$ were implemented using SHA224, and the *randomness extractor F* was implemented using SHA256.

A detailed parameter description for our experimental setup is given in Table 3. SS512 is a super singular elliptic curve (with symmetric Type 1 pairing), and MNT224 is the Miyaji, Nakabayashi, Takano curve (with asymmetric Type 3 pairing). In Table 3, p = bit length of prime order p, k = embedding degree, *Security* is the security level in bits with respect to the discrete log problem, and the numbers associated with the curve name represent the base field size in bits (i.e., SS512 has a base field size of 512 bits). Though our ReVO-ABE construction is based on a symmetric pairing group ($\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$), we have tested our implementation in both symmetric and asymmetric group settings. `Charm` treats groups as asymmetric, though the actual setting depends on the type of underlying chosen

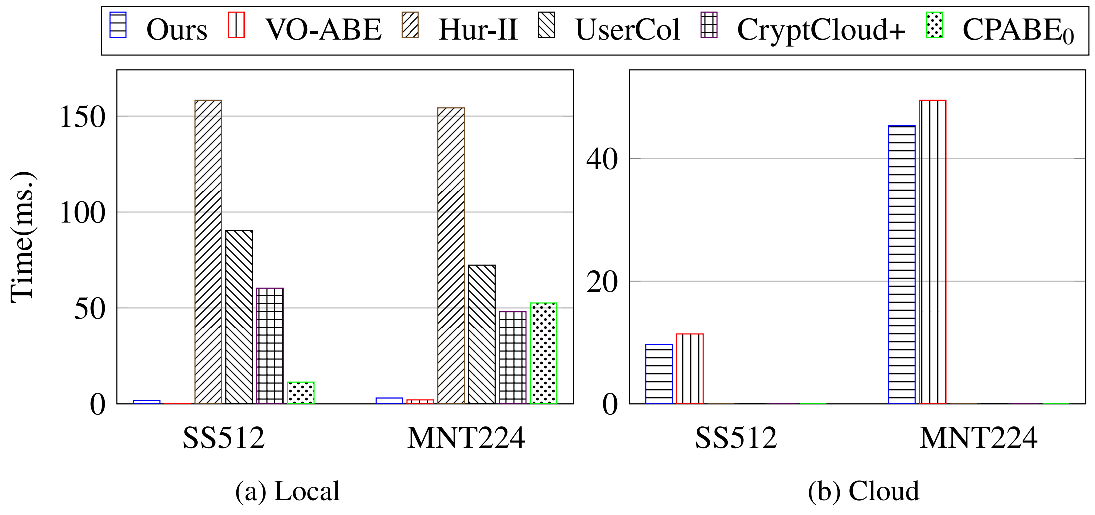
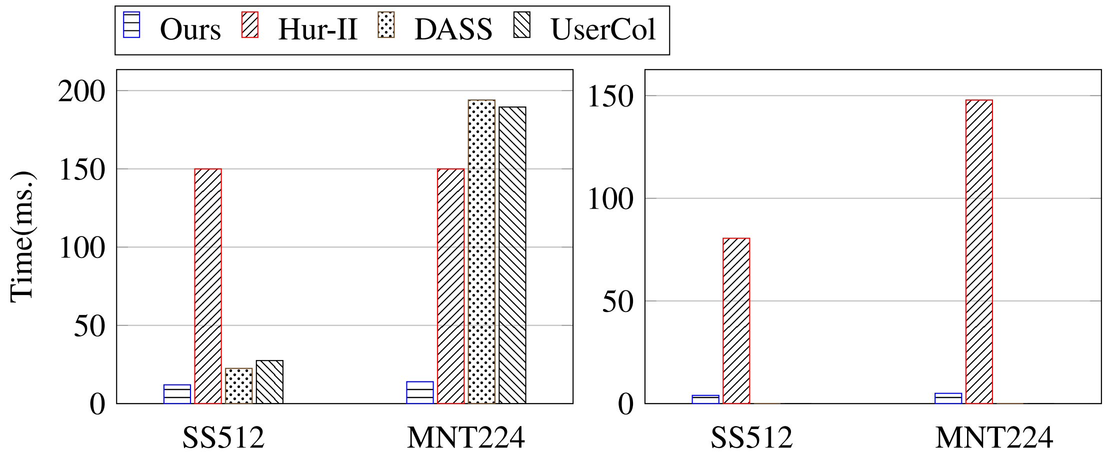


Figure 6. Decryption

curve. More specifically, there are three different groups ($\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$), and pairing is defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. We keep most of the terms in $\mathbb{G}_1$ while implementing our scheme in the asymmetric setting since operations in $\mathbb{G}_1$ are generally much faster than those in $\mathbb{G}_2$.

Testbed setup: We have simulated our user, attribute authority, and group admin in a desktop with Intel® Core i5-2400@3.1 GHz $\times$ 4 processor and 8 GB RAM running Ubuntu 16.04. For the cloud, we used a virtual instance in Amazon EC2 with Intel® Xeon(R) ES-1620v2@3.7 GHz $\times$ 8 processor and 16 GB RAM running Ubuntu server 16.04.3. Both machines run Python 3.5.2 and PBC-0.5.14 library.

Results: First, we compare the runtime of our encryption, decryption, and re-encryption (caused by revocation or joining) with the related schemes in Figure 5, 6, and 7, respectively. For each scheme, we first set up the system with 100 users. Next, we encrypt a message with the same access structure having eight distinct attributes. Then, we decrypt the ciphertext with a secret key that has five attributes, four of which being common attributes with the access structure. Finally, we re-encrypt the same ciphertext while we



(a) Local (b) Cloud

Figure 7. Re-encryption

keep access formula the same. We carried out the same experiment for SS512 and MNT224 curves. The time shown in Figure 5, 6, and 7 is the average of 50 individual trials. The local and cloud runtime is shown in separate sub-figures. For encryption and decryption, we compare our scheme with others who have similar features/objectives such as VO-ABE [17], Hur-II [7], UserCol [14], CryptCloud+ [5], and CPABE$_0$ [20]. We use CPABE$_0$ for the baseline comparison. For encryption, UserCol has the highest local runtime (Figure 5a), while the rest of the schemes have almost the same local runtime because UserCol needs more group exponentiation operations (in $\mathbb{G}_1$) locally. Only Hur-II requires cloud-side computation because the cloud needs to transform the ciphertext to enforce revocation (Figure 5b). On the other hand, only our scheme and VO-ABE supports outsourced decryption. So, local decryption cost is very small for both the schemes (Figure 6a). However, the cloud-side computation of our scheme is less than that of VO-ABE (Figure 6b) since runtime in the cloud for our scheme increases logarithmically with the number of attributes in the access structure, while for VO-ABE, the time increases linearly with the number of total attributes. However, the difference here is not that prominent because we have used a small access structure. The local cost for Hur-II is much higher than any

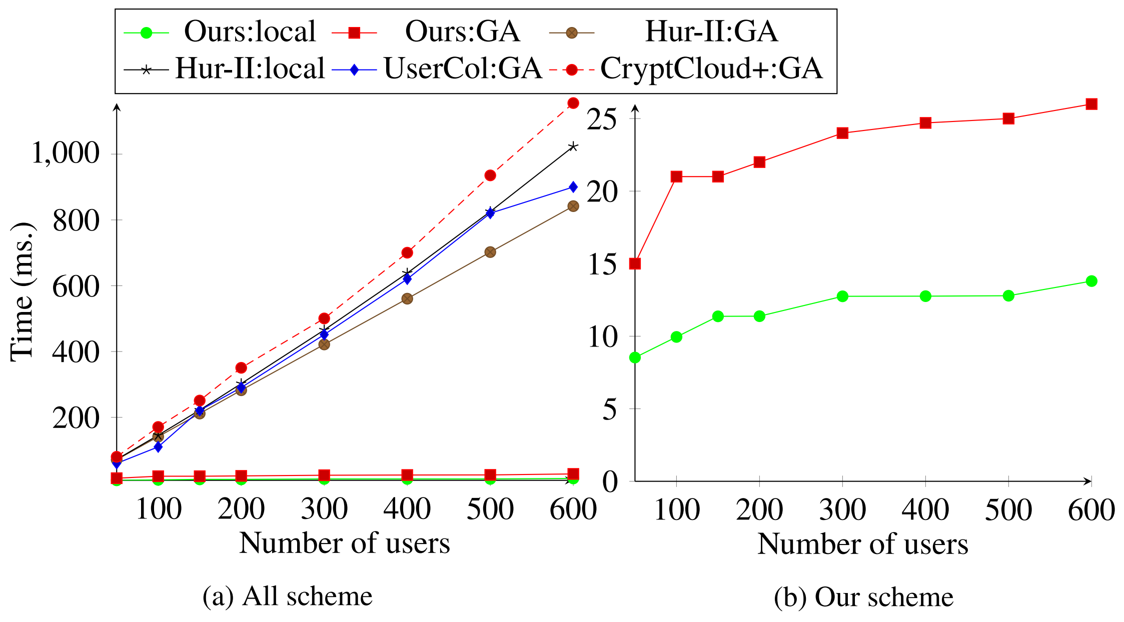


Figure 8. Re-keying time

other scheme because it requires decrypting a header that involves group exponentiation operations growing linearly with the number of total users. The performance for re-encryption of Hur-II, DASS, UserCol, and our scheme are shown in Figure 7. Note that our local computation cost for re-encryption is smaller than that of DASS (Figure 7a). Hur-II only needs the computation in the cloud-side as it does not involve the data owner and trusts the cloud for re-encryption. On the other hand, re-encryption is done by the group admin in UserCol (shown as local cost in Figure 7a), which can be a huge performance bottleneck considering the revocation and joining happens frequently. A very small amount of computation cost is associated with the cloud for our scheme because cloud only does multiplication operation to update the ciphertext (Figure 7b).

In Figure 8, we show the re-keying cost of Hur-II, UserCol, CryptCloud+, and our scheme caused by user revocation or new user joining. The re-keying runtime grows linearly with the number of total users in the group for all schemes except ours (Figure 8a). The runtime for both the user (local) and the group admin of our scheme appear to be constant in Figure 8a because runtime of other schemes is much higher which shrinks the graph along

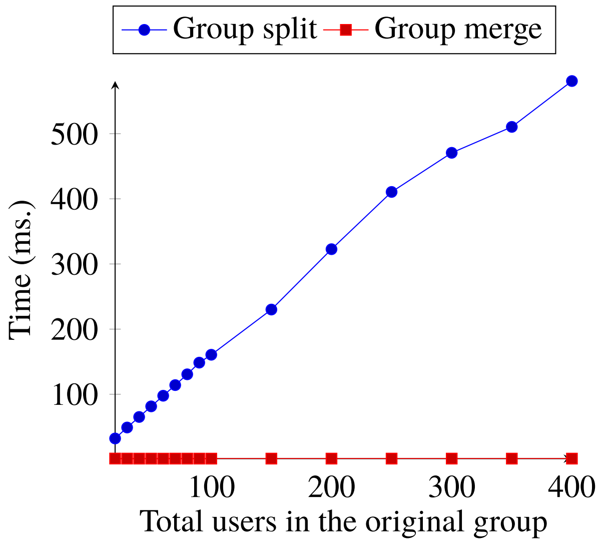


Figure 9. Group spit and merge cost

y axis. However, when drawn separately as shown in Figure 8b, we see that our runtime grows logarithmically with the total number of users. The results shown here are for the SS512 curve only. For other curves, a similar trend also follows.

We show the group split and merge cost associated with the admin in Figure 9. We split each group into two equal sized groups and record the time by taking the average of 50 trials. In our experiment, the subgroup assignment of each user was done randomly, and hence the time appears to grow linearly with the number of users in the original group. If members from the same sub-group are part of a larger subtree in the original tree, then the time becomes less as all keys in the subtree are re-used during forming new TGDH tree. Group merging cost, on the other hand, is constant (about 1.3 ms.) because it requires only constant number of group exponentiation operation in $\mathbb{G}_1$.

To summarize, our experimental results show that the proposed scheme is able to reduce the user-side computation of ABE significantly compared to other ABE schemes. Our scheme also keeps the re-keying and re-encryption cost at the user end relatively small.

Table 4. Symbols used in performance analysis

Symbol	Meaning
u	Total number of attributes in the system
a	Number of attributes in access structure $\mathbb{A}$
b	Number of attributes in user secret key
s	Number of satisfied attributes by user secret key
$\lvert\mathbb{G}_i\rvert$	Size of a single element in group $\mathbb{G}_i$
$\lvert\mathbb{K}\rvert$	Size of symmetric key
$\lvert\mathbb{C}\rvert$	Ciphertext size of actual file
$\lvert p\rvert$	Size of a single element in $\mathbb{Z}_p$
l_2	Output size of the hash function $\mathcal{H}_2$
C_i	Single exponentiation time in group $\mathbb{G}_i$
P	Computation time of a pairing operation
t	Maximum number of revocations allowed
m	Total number of users in the group
r	Subset cover of all revoked users in the binary key tree

9. THEORETICAL PERFORMANCE ANALYSIS

In this section, we first compare our scheme with other CP-ABE-based schemes from a theoretical perspective in terms of storage, communication, and computational efficiency. Finally, we analyze the space complexity of the TGDH and e-TGDH trees maintained by the group admin and clouds, respectively. Table 4 illustrates different symbols that have been used for this purpose.

9.1. STORAGE AND COMMUNICATION EFFICIENCY

The space efficiency comparison in terms of ciphertext and secret and public key size has been summarized in Table 5. The ciphertext size, secret key size, and public key size represent the storage cost required by the cloud, each user, and the attribute authority to store them, respectively. Additionally, they represent the communication cost when these are sent from one party to another. Our ciphertext has fewer group elements compared to [5, 12, 14]. Compared to other schemes, our scheme and VO-ABE [17] have additional l_2 bytes in the ciphertex to support verifiability.

Table 5. Comparison of storage and communication efficiency with other schemes

Scheme	**Ciphertext size**	**Secret key size**	**Public key size**
DASS [8]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert$	$(b + 1)\lvert\mathbb{G}_1\rvert + (\log m)\lvert\mathbb{K}\rvert$	$(u + 2)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
Hur-I [13]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert$	$(2b + 1)\lvert\mathbb{G}_1\rvert + (\log m)\lvert\mathbb{K}\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
Hur-II [7]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert$	$2(b+1)\lvert\mathbb{G}_1$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
VO-ABE [17]	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert + l_2$	$(b+3)\lvert\mathbb{G}_0\rvert + \lvert p\rvert$	$(u + 2)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$
PIRATTE [6]	$(a + 1)\lvert\mathbb{G}_1\rvert + a\lvert\mathbb{G}_2\rvert+\lvert\mathbb{G}_T\rvert+\lvert\mathbb{C}\rvert$	$2b\lvert\mathbb{G}_1\rvert + (b + 1)\lvert\mathbb{G}_2\rvert + 2\lvert p\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert + \lvert\mathbb{G}_T\rvert$
CryptCloud+ [5]	$(2a + 5)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert$	$(b + 4 + 2\log m)\lvert\mathbb{G}_1\rvert$	$(u+6)\lvert\mathbb{G}_1\rvert+3\lvert p\rvert$
Flexible [12]	$(2a + 6)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + 2\lvert p\rvert + \lvert\mathbb{C}\rvert$	$(b+4)\lvert\mathbb{G}_1\rvert+2\lvert p\rvert$	$3\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_T\rvert + \lvert p\rvert$
UserCol [14]	$(4a + ra + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert$	$4b\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$2(u + 3)\lvert\mathbb{G}_1\rvert + 2\lvert\mathbb{G}_T\rvert + (2m - 1)\lvert p\rvert$
Ours	$(2a + 1)\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert + \lvert\mathbb{C}\rvert + l_2$	$2b\lvert\mathbb{G}_1\rvert + 2\lvert p\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$

The user's secret key consists of the attribute secret key (SK_S) and the membership key (MbK). Because of using the e-TGDH tree, our MbK consists of a single $\mathbb{Z}_p$ element (of size $|p|$). On the other hand, [8, 13, 5] use a static binary tree, and the MbK consists of all the keys along the path from the leaf to the root resulting in a MbK of size $\log m|\mathbb{K}|$ for [8] and [13], and MbK of size $2\log m|\mathbb{G}_1|$ for [5]. The scheme [14] reduces the MbK size to $2b|\mathbb{G}_1|$ (assuming $\log m < 2b$). To create MbK, [14] generates two $\mathbb{G}_1$ elements from the leaf node for each attribute in SK_S (resulting in a total of $2b$ elements). Due to the reduced MbK size, the total secret key size (SK_S and MbK) in our scheme is smaller than that of [13, 5, 14]. The scheme [17] have smaller secret key size than ours because [17] does not support revocation, hence does not have any MbK.

The public key size has a direct correlation with the underlying CP-ABE scheme used. The scheme [6, 13, 7, 12] and our scheme are based on CP-ABE$_0$ [20]. Hence, our scheme is equally good or slightly better than those schemes. On the other hand, the public key size of [17, 8, 5, 14] is very large as they are based on [35], whose public key size increases with the total number of attributes in the system. In addition to the space cost shown in Table 5, all schemes except [17] need to store some additional information in the cloud to support revocation. For [13, 8, 5, 14], and our scheme, this cost is O(2^m) since these schemes store all the user information in a binary tree.

9.2. COMPUTATION COST ANALYSIS

We show the computation cost of our scheme and compare it with other schemes in Table 6. The computation cost has been expressed in terms of group exponentiation and pairing operation in a similar manner to [17, 7, 13, 6], etc. This is a reasonable consideration since these two operations dominate other lightweight hash, multiplication, division, and addition operations. Our encryption cost is less than all other schemes except that of [5]. For the local decryption cost, our scheme outperforms others except [17] because we need one more C_1 and P locally for decryption (compared to [17]) to support dynamic changes in the group. Nevertheless, our decryption cost in the cloud-side is a lot less than that of [17, 12] if the number of attributes (a) in the access structure grows bigger. Though [6] does not support outsourced decryption, the cloud needs to do aC_2 operations for each decryption.

9.3. OVERHEAD DUE TO CHANGE IN GROUP DYNAMICS

In [8, 13, 7] and [12], the user has to update all the common attributes between his and the revoked user's attribute secret keys. Each attribute update requires one group exponentiation operation (in $\mathbb{G}_1$). Consequently, the user has to do O($\log m$) group expo-

Table 6. Comparison with other schemes in terms of computation cost.

Scheme	**Encryption**	**Decryption**	
		local	cloud
DASS [8]	$(3a+1)C_1+C_T$	$(s+1)P + s(C_1+C_T)$	N/A
Hur-I [13]	$(3a+1)C_1 + C_T$	$(2s+1)P + C_1 + C_T\log a$	N/A
Hur-II [7]	$(3a+2m+3)C_1 + C_T$	$(3s+1)P + C_T\log a + (m+1)sC_1$	N/A
VO-ABE [17]	$(3a+1)C_1+C_T$	C_T	$(2s+1)P + aC_T$
PIRATTE [6]	$(a+1)C_1 + C_T + aC_2$	$(s+\log a)C_T + (3s+1)P$	aC_2
CryptCloud+ [5]	$(a+5)C_1 + C_T$	$2C_1 + sC_T + (2s+5)P$	N/A
Flexible [12]	$2(a+3)C_1 + 2C_T$	$4C_T$	$(2s+4)P + C_T\log a$
UserCol [14]	$(3a+ra+1)C_1 + C_T$	$(2s-1)C_T + (3s+1)P$	N/A
Ours	$2(a+1)C_1 + C_T + P$	$2C_T$	$(2s+1)P + C_1 + C_T\log a$

nentiation operations. In our scheme, the user needs to compute all the secret keys along the e-TGDH tree path to ultimately compute the new GSK which requires $\log m$ exponentiation operations in $\mathbb{G}_1$ as shown in Table 7. The group admin (GA) on the other hand needs to compute new blinded and secret keys along the path, creates one proxy update key (UPr), and updates GPK that requires $2\log m$, one, and two group exponentiation operations in $\mathbb{G}_1$, respectively, incurring a total of $(2\log m + 3)C_1$ cost. Our key update operation cost for the GA is less than that of [7, 6, 12, 5, 14] because the admin needs to create update keys for m nonrevoked users that requires $O(m)$ group exponentiation operations. There is no cost associated with the user in [6, 5, 14], and the cloud or the GA in [13, 8] because there is no expensive group exponentiation or pairing operations involved.

For re-encryption (Table 7), all a attributes in the ciphertext policy need to be updated in [8, 13, 7]. Each attribute update requires three group exponentiation operations (in $\mathbb{G}_1$). Two additional group exponentiation operations (one in $\mathbb{G}_1$ and one in $\mathbb{G}_T$) are required to update two other ciphertext components. So the total computation cost becomes $(3a + 1)C_1 + C_T$. This entire cost is associated with the data owner in [8], and with the cloud in [13, 7]. In [6], the data owner alone does the re-encryption task. It requires two group exponentiation operations (one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$) for updating each attribute and two additional group exponentiation operations (one in $\mathbb{G}_1$ and one in $\mathbb{G}_T$) for updating two other ciphertext components, incurring a total cost of $(a + 1)C_1 + C_T + aC_2$. CryptCloud+ does not consider re-encrypting the ciphertext after revocation. The group admin in [12] generates a ciphertext component update that requires only a pairing and a group exponentiation operation (a total cost of $C_1 + P$) and the cloud multiplies this component with the ciphertext for re-encryption. Although it is very efficient, the re-encryption process cannot prevent a cloud-revoked user collusion attack. To prevent the collusion attack the group admin in [14] re-encrypts the ciphertext that requires three group exponentiation operations (in $\mathbb{G}_1$) to update each attribute, and three additional group exponentiation operations (two in $\mathbb{G}_1$ and one in $\mathbb{G}_T$) to compute three updated ciphertext components (a total of $(3a + 2)C_1 + C_T$ cost). In our scheme, the owner needs to compute $U1_{ts \to ts'} = e(g, g)^{b_{l'} K^j_{l'} s - b_l K^j_l s}, U2_{ts \to ts'} = g^{\beta K^j_{l'} s}$, which require two pairing and two group exponentiation operations, incurring a total cost of $C_1 + C_T + 2P$.

9.4. SPACE COMPLEXITY ANALYSIS OF KEY TREES

In this section, we discuss the space complexity of key trees. Each admin has to maintain a TGDH tree, and clouds have to maintain a corresponding e-TGDH. At any point, if the total number of users in the group is m, then the space complexity of the TGDH tree for the admin would be $2.2^{\log m+1} - 1 = 4.2^{\log m} - 1 = 4.m - 1 \approx \mathrm{O}(m)$.

Table 7. Cost of group dynamic change

Scheme	**Key update**		**Re-encryption**	
	user	Cloud/GA	Owner	Cloud/GA
DASS [8]	bC_1	0	$(3a+1)C_1 + C_T$	0
Hur-I [13]	bC_1	0	0	$(3a+1)C_1 + C_T$
Hur-II [7]	bmC_1	$2(m+1)C_1$	0	$(3a+1)C_1 + C_T$
PIRATTE [6]	0	amC_2	$(a+1)C_1 + C_T + aC_2$	0
CryptCloud+ [5]	0	$3mC_1$	N/A	N/A
Flexible[12]	$(b+1)C_1$	$2mC_1 + P$	0	$C_1 + P$
UserCol [14]	0	$(2m-1)C_1$	0	$(3a+2)C_1 + C_T$
Ours	$C_1\log m$	$(2\log m+3)C_1$	$C_1 + C_T + 2P$	0

The space complexity of the e-TGDH tree for the cloud is more difficult to determine. On average, each revocation or joining operation adds $\log m$ nodes to the tree. Lets assume that there are initially $X_m = 2m-1$ nodes in the e-TGDH tree and d number of random revocation or joining has been performed. The space complexity can be calculated as follows:

$$\begin{aligned} & X_m + \log m + \log(m+1) + \cdots\cdots + \log(m+d) \\ & = X_m + \log(m(m+1)(m+2)\cdots(m+d)) \\ & = X_m + \log\left(\frac{m!}{d!}\right) \approx \mathrm{O}(m + \log m!). \end{aligned}$$

Efficiency of the e-TGDH tree: The cloud needs to create only $\log m$ new nodes along the path from the affected leaf to the root on average to create a new version of the tree if it uses our proposed e-TGDH tree structure, as opposed to $2m-1$ new nodes if using the original TGDH tree. For each revocation or joining, the cloud's space saving would be $2m - \log m - 1$, and d such operations would save $d(2m - \log m - 1)$ space.

10. SECURITY ANALYSIS

10.1. ANALYSIS OF COLLUSION ATTACKS

In this section prove that our scheme is secure against the three type of collusion attacks discussed in the threat model, and at the same time achieves immediate revocation against those attacks. Note that immediate revocation is possible only if the user does not locally store the symmetric encryption key (K_{SE}). Otherwise K_{SE} can be directly used to decrypt CT_{SE}. Suppose two users (of group j), $u_{j,k}$ and $u_{j,k'}$ are authorized for the attribute set S and S', respectively. The transformation key of $u_{j,k}$ and $u_{j,k'}$ are denoted as $\mathrm{TK}_S = \{D_{2,i} = g^{r/\beta t}.\mathcal{H}(i)^{r_i/t}, D_{3,i} = g^{\beta r_i/t}\}_{\forall i \in S}$ and $\mathrm{TK}'_{S'} = \{D_{2,i} = g^{r'/\beta t'}.\mathcal{H}(i)^{r'_i/t'}, D_{3,i} = g^{\beta r'_i/t'}\}_{\forall i \in S'}$, respectively, and their update keys for group j are denoted as $\mathrm{UK} = D_{1,j} = g^{(\alpha+r+b_l K_l^j)t_c/\beta}$ and $\mathrm{UK}' = D'_{1,j} = g^{(\alpha+r'+b_l K_l^j)t_c/\beta}$, respectively. The user $u_{j,k}$ and $u_{j,k'}$ have the attribute secret key $\mathrm{SK}_S = (\mathrm{TK}_S, t)$ and $\mathrm{SK}'_{S'} = (\mathrm{TK}'_{S'}, t')$, respectively, and the membership key $\mathrm{MbK} = a_k$ and $\mathrm{MbK}' = a_{k'}$, respectively. Now, cloud1 has TK_S and TK'_S, and its master key C1MK=t_c, while cloud2 has UK and UK′. For the sake of simplicity, we denote a ciphertext encrypted for group j as $\mathrm{CT} = (C_{0,j} = K_R e(g,g)^{\alpha s + b_l K_l^j s}, C_{1,j} = g^{\beta K_l^j s}, \{C_{2,v} = g^{P_v(0)\beta}, C_{3,v} = \mathcal{H}(attr(v))^{P_v(0)}\}_{\forall v \in \mathcal{V}})$. The assumption is that attribute set S satisfies the policy of CT while S' does not. Now in the following we prove that CT cannot be decrypted by launching type–1, type–2, or type–3 collusion attacks.

10.1.1. Security Against Type–1 Collusion Attack. Let us assume that $u_{j,k}$ gets revoked from group j and colludes with cloud1 to decrypt CT. However, cloud2 remains honest and removes $u_{j,k}$ from its e-TGDH tree and increases its version number ($CurrVer_j$) to $l+1$ as per the protocol. Note that cloud1, with SK_S and CT, can compute $F_R = e(g,g)^{rs/t}$. Now, cloud1 requests cloud2 to compute T by sending $(u_{j,k}, j, C'_{1,j} = (C_{1,j})^b)$ to cloud2. However, cloud2 finds out that $u_{j,k}$ has been removed from the latest ($CurrVer_j = l+1$) e-TGDH tree, and declines the request. Hence, cloud1 cannot compute T_1. Without T_1 it is computationally impossible for $u_{j,k}$ to compute $e(g,g)^{\alpha s + b_l K_l^j s}$, and retrieve K_R

from $T_2 = K_R e(g,g)^{\alpha s + b_l K_l^j s}$. For the same reason $u_{j,k}$ cannot retrieve K_R after the data owner updates the CT components for the version $l+1$ (e.g., updates $C_{0,j}$ and $C_{1,j}$ to $K_R e(g,g)^{\alpha s + b_{l+1} K_{l+1}^j s}$ and $g^{\beta K_{l+1}^j s}$, respectively). As a result, type–1 collusion attack fails. This also proves the security of immediate revocation against type–1 collusion attack since the revocation becomes effective even before the data owner updates CT to version $l+1$.

10.1.2. Security Against Type–2 Collusion Attack. Let us assume that $u_{j,k}$ gets revoked from the group j and colludes with cloud2 to decrypt CT. However, cloud1 remains honest and removes $u_{j,k}$ from its e-TGDH tree, and increases its version number ($CurrVer_j$) to $l+1$ as per the protocol. To help $u_{j,k}$ decrypt CT, cloud2 gives to $u_{j,k}$ the update key UK $= D_{1,j} = g^{(\alpha + r + b_l K_l^j)t_c/\beta}$. Then, $u_{j,k}$ can compute $F_R = e(g,g)^{rs/t}$ (since S satisfies the policy), and $T' = e(D_{1,j}, (C_{1,j})^{1/K_l^j}) = e(g^{(\alpha + r + b_l K_l^j)t_c/\beta}, g^{\beta s}) = e(g,g)^{(\alpha + r + b_l K_l^j)st_c}$. However, the revoked user has to compute $e(g,g)^{(\alpha + r + b_l K_l^j)s}$ from T' which is computationally hard (because of the CDH assumption) without the cloud1 master key (C1MK). Consequently, $u_{j,k}$ cannot retrieve K_R from $T_2 = K_R e(g,g)^{\alpha s + b_l K_l^j s}$. For the same reason $u_{j,k}$ cannot retrieve K_R after the data owner updates the CT components for the version $l+1$ (e.g., updates $C_{0,j}$ and $C_{1,j}$ to $K_R e(g,g)^{\alpha s + b_{l+1} K_{l+1}^j s}$ and $g^{\beta K_{l+1}^j s}$, respectively). As a result, type–2 collusion attack fails. This also proves the security of immediate revocation against type–2 collusion attack since the revocation becomes effective even before the data owner updates CT to version $l+1$.

10.1.3. Security Against Type–3 Collusion Attack. Let us assume that $u_{j,k}$ gets revoked from group j, and colludes with $u_{j,k'}$ and one of the clouds to decrypt CT. We have already proved in the previous sections that $u_{j,k}$ cannot successfully decrypt CT by colluding with one of the clouds. However, in this collusion attack $u_{j,k}$ has some advantage since $u_{j,k'}$ is also colluding. Following two cases are possible in this collusion attack:

Cloud1 colludes: In this case, cloud1 can request cloud2 to compute T by sending it $(u_{j,k'}, j, C'_{1,j} = (C_{1,j})^b)$. Since $u_{j,k'}$ is a nonrevoked user, cloud2 computes $T = e(C'_{1,j}, UK') = e(g,g)^{(\alpha + r' + b_l K_l^j)bt_c K_l^j s}$, and sends to cloud1. Now, cloud1 computes

$T_1 = (T)^{1/bt_c} = e(g,g)^{(\alpha+r'+b_l K_l^j)K_l^j s}$, and sends CT_{part} to $u_{j,k}$. Then, $u_{j,k}$ computes $K'_R = T_2(F_R)^t / T_1^{1/K_l^j} = K_R e(g,g)^{(r-r')s}$. This means $u_{j,k}$ cannot retrieve K_R, and hence unable to decrypt CT. Note that when the data owner updates CT for the current version (e.g., $l+1$), T_1 and T_2 becomes $e(g,g)^{(\alpha+r'+b_{l+1}K_{l+1}^j)K_{l+1}^j s}$ and $K_R e(g,g)^{\alpha s + b_{l+1}K_{l+1}^j s}$, respectively. In this case, $u_{j,k'}$ may compute K_{l+1}^j from MbK′ and the blinded key list $\mathbb{B}$ of his or her co-path, and give to $u_{j,k}$. With K_{l+1}^j, $u_{j,k}$ computes $K'_R = T_2(F_R)^t / T_1^{1/K_{l+1}^j} = K_R e(g,g)^{(r-r')s}$, meaning that K_R cannot be retrieved.

Cloud2 colludes: In this case, cloud2 gives UK $= g^{\alpha+r+b_l K_l^j}$ (if CT is not updated by the owner) or UK $=$ UK.UPr$_{l\to l+1}^j = g^{(\alpha+r+b_l K_l^j)t_c/\beta} g^{(b_{l+1}K_{l+1}^j - b_l K_l^j)t_c/\beta} = g^{(\alpha+r+b_{l+1}K_{l+1}^j)t_c/\beta}$ (if CT is updated by the owner). Then, $u_{j,k'}$ can compute K_{l+1}^j and give to $u_{j,k}$. Similar to type–2 collusion attack (Section 10.1.2), $u_{j,k}$ may compute T'. However, $u_{j,k}$ cannot retrieve K_R since T' is blinded with C1MK.

We can see that type–3 collusion attack also fails as long as one of the clouds follows the protocol. This also proves the security of immediate revocation against type–3 collusion attack since the revocation becomes effective even before updating CT to version $l+1$.

10.2. SEMANTIC SECURITY ANALYSIS

In this section, we define the formal security model of our ReVO-ABE scheme by a CPA (chosen plaintext attack) game. In this security game, the adversary launches a collusion attack where it tries to correctly guess a challenged ciphertext by combining previously queried revoked user's key with a nonrevoked user's key. To keep things simple, without loosing generality, we will assume a single group setting during our security game formulation.

Init: The adversary commits to an access structure $\mathbb{A}^*$ by giving it to the challenger.

Setup: The challenger runs `Setup`$(1^{\mathcal{K}})$ and `Group-Setup` algorithms. The `Setup` algorithm generates the master secret key MK, public key PK, cloud1 master secret key C1MK, and group master secret key GMK. On the other hand, the `Group-Setup` algorithm initializes the TGDH key tree with N leaves and generates the initial group public and secret key GPK_0 and GSK_0, respectively. The challenger then gives PK and GPK to $\mathcal{A}$.

Phase 1: The challenger initializes an empty table T, an empty set D, and an integer $j = 0$. The adversary $\mathcal{A}$ adaptively makes the following queries polynomial number of times:

- `Create`(S, u_j): Challenger sets $j = j + 1$ and runs `KeyGen` and `Group-KeyGen` algorithms to generate $(\text{SK}_S, \text{TK}_S, D_1)$, and a_j, respectively. Next, it gives TK_S to the adversary and stores tuple $(j, u_j, S, \text{SK}_S, \text{TK}_S, a_j, D_1)$ in table T.

- `Corrupt`(i): If there exists i^{th} tuple in table T, then the challenger retrieves tuple $(i, u_i, S, \text{SK}_S, \text{TK}_S, a_i, D_1)$. Finally, updates D := D $\cap\{(S, u_i)\}$ and returns (a_i, SK_S) to the adversary if either of the following is true:

  - S does not satisfy $\mathbb{A}^*$.
  - The user u_i is a revoked user.

  However, if S satisfies $\mathbb{A}^*$ and u_i is a nonrevoked user or no such i^{th} tuple exists in T, then it returns $\perp$.

Challenge: The adversary submits two equal length messages M_0 and M_1 along with an access structure $\mathbb{A}^*$ such that for $\forall(S, u_j) \in \text{D}$, either S does not satisfy $\mathbb{A}^*$ or u_j is a revoked user. The challenger then randomly picks a bit $b \in \{0, 1\}$. Finally, it runs `Encrypt` and returns the output CT to the adversary.

Phase 2: The adversary repeats Phase1 with the following constraints:

- The adversary cannot submit a `Corrupt` query for a nonrevoked user u_i and an attribute set S such that S satisfies $\mathbb{A}^*$ and S is added to D.

Guess: The adversary outputs a guess $b' \in \{0, 1\}$ and wins if $b' = b$.

Definition 5 *The security game defined above is called IND-CPA game. Our proposed ReVO-ABE scheme is CPA (chosen plaintext attack) secure if a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ has a negligible advantage in wining this IND-CPA game.*

Theorem 4 *If the CDH assumption described in Section 3.3 holds, then our ReVO-ABE scheme is secure in the selective model. In other words, if there exists a polynomial time adversary $\mathcal{A}$ that can win the IND-CPA game with a non-negligible advantage after a total of q^* queries, then we can find a polynomial time algorithm $\mathcal{B}$ that breaks the CDH assumption with a non-negligible advantage.*

Proof: The challenger produces the bilinear group parameters $\{p, G, G_T, g, e\}$, where $\mathbb{G}$ and $\mathbb{G}_T$ represent two multiplicative cyclic groups of prime order p, g is a generator of $\mathbb{G}$, and e is a bilinear map. The challenger then randomly selects $x, y \in \mathbb{Z}_p$, and computes $X = g^x, Y = g^y$. The CDH challenge (X, Y) is then given to the algorithm $\mathcal{B}$. Now, the goal of $\mathcal{B}$ is to compute $Z = g^{xy}$ by utilizing $\mathcal{A}$. To achieve this goal, $\mathcal{B}$ interacts with $\mathcal{A}$ by programming the parameters of the original IND-CPA game in the following manner:
Init: The adversary $\mathcal{A}$ commits to an access structure $\mathbb{A}^*$ by giving it to the challenger.
Setup: The system parameters are programmed by $\mathcal{B}$ as follows:

- $\mathcal{B}$ randomly chooses $\alpha, \beta \in \mathbb{Z}_p$. Then, it randomly chooses $t_c \in \mathbb{Z}_p$ and sets cloud1 master secret key as C1MK $= t_c$, and computes the group master secret key as GMK=$g^{t_c/\beta}$. The master secret key MK= (β, g^α, t_c) is kept secret, and the public key is published as PK= $(\mathbb{G}, g, \mathbb{G}_T, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, F, e(g, g)^\alpha, h = g^\beta)$. Here, $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, F$ are replaced by the following oracles:

- Oracle $\mathcal{H}$: It maintains a dictionary to answer to the attribute hash queries. For a query on attribute $attr_i$, if the attribute does not exist in the dictionary, it randomly chooses $n_i \in \mathbb{G}$, stores tuple $(attr_i, n_i, \mathcal{H}(attr_i) = g^{n_i})$ in the dictionary and returns g^{n_i} as a query response. However, if $attr_i$ already exists in the dictionary, it returns the preexisting g^{n_i}.
- Oracle $\mathcal{H}_1$: For a query on a value $K_i \in \mathbb{G}_T$, if it does not exist in the dictionary, the oracle randomly chooses $h_{1_i} \in \{0,1\}^{l_1}$, stores tuple $(K_i, \mathcal{H}_1(K_i) = h_{1_i})$ in the dictionary and returns h_{1_i} as a query response. However, if K_i already exists in the dictionary, it returns the preexisting h_{1_i}.
- Oracle $\mathcal{H}_2$: For a query on a value $v_i \in \{0,1\}^*$, if it does not exist in the dictionary, the oracle randomly chooses $\mathrm{VK}_i \in \{0,1\}^{l_2}$, stores tuple $(v_i, \mathcal{H}_2(V_i) = \mathrm{VK}_i)$ in the dictionary and returns VK_i as a query response. However, if v_i already exists in the dictionary, it returns the preexisting VK_i.
- Oracle $\mathcal{H}_3$: For a query on a value $g_i \in \mathbb{G}$, if it does not exist in the dictionary, the oracle randomly chooses $m_i \in \mathbb{Z}_p$, stores tuple $(g_i, \mathcal{H}_2(g_i) = m_i)$ in the dictionary and returns m_i as a query response. However, if g_i already exists in the dictionary, it returns the preexisting m_i directly.
- Oracle F: For a query on a value $K_i \in \mathbb{G}_T$, if it does not exist in the dictionary, the oracle randomly chooses $K_{SE_i} \in \mathbb{K}$, stores tuple $(K_i, F(K_i) = K_{SE_i})$ in the dictionary and returns K_{SE_i} as a query response. However, if K_i already exists in the dictionary, it returns the preexisting K_{SE_i}.

• $\mathcal{B}$ programs the parameters of `Group-Setup` algorithm by initializing the TGDH key tree with N leaf nodes and associates each u_i with a leaf node that has a membership secret key $a_i \in \mathbb{Z}_p$. The blinded and secret keys for all the intermediate nodes are calculated. Let K_0 and g^{K_0} be the initial root secret key ($K_{<0,0>}$) and blinded key

$(BK_{<0,0>})$. Then, $\mathcal{B}$ randomly chooses $b_0 \in \mathbb{Z}_p$ and publishes the initial group public key $\text{GPK}_0 = (g^{K_0}, X^{b_0 K_0} = g^{x b_0 K_0}, g^{\beta K_0})$, and keeps $\text{GSK}_0 = K_0$ secret. $\mathcal{B}$ also publishes the TGDH key tree.

Phase 1: $\mathcal{B}$ initializes an empty table T, an empty set D, and an integer $j = 0$. Then, $\mathcal{A}$ adaptively makes the following queries polynomial number of times:

- `Create`(S, u_j): Algorithm $\mathcal{B}$ sets $j = j + 1$ and responds to adversary $\mathcal{A}$'s queries as follows:
  - If S does not satisfy $\mathbb{A}^*$ but u_j is a nonrevoked user, then $\mathcal{B}$ randomly chooses r and $\{r_1, r_2, \cdots, r_{|s|}\}$ from $\mathbb{Z}_p$. Then, it computes $D_1 = g^{(\alpha + r + x b_l K_l) t_c / \beta}$ and $\text{SK}'_S = (\{D'_{2,i} = g^{r/\beta}.\mathcal{H}(i)^{r_i}, D'_{3,i} = g^{\beta r_i}\}_{\forall i \in S})$. Next, it chooses a random exponent $t \in \mathbb{Z}_p$, computes the transformation key $\text{TK}_S = (\{D_{2,i} = g^{r/\beta t}.\mathcal{H}(i)^{r_i/t}, D_{3,i} = g^{\beta r_i/t}\}_{\forall i \in S})$, and sets $\text{SK}_S = (\text{TK}_S, t)$. For the sake of simplicity, lets assume that user u_j already exists in the TGDH tree and a_j is the associated leaf node of u_j. Finally, $\mathcal{B}$ gives TK_S to the adversary and stores tuple $(j, u_j, S, \text{SK}_S, \text{TK}_S, a_j, D_1)$ in table T.
  - If S satisfies $\mathbb{A}^*$ but u_j is a revoked user, then $\mathcal{B}$ randomly chooses r and $\{r_1, r_2, \cdots, r_{|s|}\}$ from $\mathbb{Z}_p$. For the revoked user u_j, instead of using $g^{x b_l K_l}$ for the computation of D_1, $\mathcal{B}$ computes $X^* = X^{b_l^* K_l^*}$ for unknown $y^* \in \mathbb{Z}_p$ such that $b_l^* K_l^* = y^* b_l K_l$. Then, $\mathcal{B}$ computes $D_1^* = g^{(\alpha + r + x b_l^* K_l^*) t_c / \beta}$ and $\text{SK}'_S = (\{D'_{2,i} = g^{r/\beta}.\mathcal{H}(i)^{r_i}, D'_{3,i} = g^{\beta r_i}\}_{\forall i \in S})$. Next, it chooses a random exponent $t \in \mathbb{Z}_p$, computes the transformation key $\text{TK}_S^* = (\{D_{2,i} = g^{r/\beta t}.\mathcal{H}(i)^{r_i/t}, D_{3,i} = g^{\beta r_i/t}\}_{\forall i \in S})$, and sets $\text{SK}_S^* = (\text{TK}_S^*, t)$. To capture the security regarding the revoked user u_j, instead of associating u_j with any leaf of the TGDH key, $\mathcal{B}$ randomly picks $a_j^* \in \mathbb{Z}_p$. Finally, $\mathcal{B}$ gives TK_S^* to the adversary and stores tuple $(j, u_j, S, \text{SK}_S^*, \text{TK}_S^*, a_j^*, D_1^*)$ in table T.

  - If S satisfies $\mathbb{A}^*$ and u_j is a nonrevoked user, then $\mathcal{B}$ returns a 'fake' transformation key to $\mathcal{A}$.

- `Corrupt`(i): If there exists i^{th} tuple in table T, then the challenger retrieves tuple $(i, u_i, S, \mathrm{SK}_S, \mathrm{TK}_S, a_i, D_1)$. Finally, updates D := D $\cap\{S\}$ and returns (a_i, SK_S) to the adversary if either of the following is true:

  - S does not satisfy $\mathbb{A}^*$.
  - The user u_i is a revoked user.

  However, if S satisfies $\mathbb{A}^*$ and u_i is a nonrevoked user or no such i^{th} tuple exists in T, then it returns $\perp$.

  Challenge: The adversary $\mathcal{A}$ submits two equal length messages M_0 and M_1 along with an access structure $\mathbb{A}^*$ such that for $\forall(S, u_j) \in$ D, either S does not satisfy $\mathbb{A}^*$ or u_j is a revoked user. The challenger then randomly picks a bit $b^* \in \{0, 1\}$ and encrypts M_{b^*} as follows:

  - $\mathcal{B}$ selects a random polynomial P_w of degree d_w for each node w of access tree Υ corresponding to $\mathbb{A}^*$ with the condition $d_w = t_w - 1$ in the following manner. First, a random $s \in \mathbb{Z}_p$ is chosen to set $P_R(0) = s$, where P_R represents the polynomial associated with the root R. Then, d_R number of other points on P_R are chosen randomly to define it completely. For any other node w, the algorithm completely defines the corresponding polynomial P_w by setting $P_w(0) = P_{parent(w)}(index(w))$ and choosing d_w random values on P_w. This process is carried out in a top-to-bottom fashion starting from the root R. Let $\mathcal{V}$ be the set of leaf nodes in Υ. Afterwards, it computes $C_0' = e(g, g)^{\alpha s} e(X^{b_l K_l}, Y^s) = e(g, g)^{\alpha s + xy b_l K_l s}, C_1 = g^{\beta K_l s}, \{C_{2,v} = g^{P_v(0)\beta}, C_{3,v} = \mathcal{H}(attr(v))^{P_v(0)}\}_{\forall v \in \mathcal{V}}$.

- Next, $\mathcal{B}$ chooses a random seed $K_R \in \mathbb{G}_T$ and computes $h_1 = \mathcal{H}_1(K_R)$, the symmetric encryption key $K_{SE} = F(K_R)$, symmetric ciphertext $\mathrm{CT}^*_{SE} = Enc_{K_{SE}}(\mathrm{M}_{b^*})$, and the verification key $\mathrm{VK} = \mathcal{H}_2(h_1||\mathrm{CT}_{SE})$.

Finally, the $\mathcal{B}$ sends out to the adversary $\mathcal{A}$ the complete ciphertext as $\mathrm{CT}^* = (C_0^* = K_R C_0' = K_R e(g,g)^{\alpha s + xyb_l K_l s}, C_1, \{C_{2,v}, C_{3,v}\}_{\forall v \in \mathcal{V}}, \mathrm{CT}^*_{SE}, \mathrm{VK})$. Note that in $\mathcal{B}$'s simulation of this `Encrypt` algorithm, we have omitted the ciphertext components CPr and $C_{1,G}$ since we are only considering a single group scenario for simplicity.

Phase 2: The adversary repeats Phase1 with the following constraints:

- The adversary cannot submit a `Corrupt` query for a nonrevoked user u_i and an attribute set S such that S satisfies $\mathbb{A}^*$ and S is added to D.

Guess: The adversary $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$.

At this point, we can see that by successfully programming all the parameters of the original IND-CPA game, $\mathcal{B}$ can interact with $\mathcal{A}$ as if from $\mathcal{A}$'s point of view, its interaction with algorithm $\mathcal{B}$ is the same as the interaction with the challenger of the original game. If $\mathcal{A}$ wins in the security game, then it must have done corrupt queries for a revoked user and a nonrevoked user to get (a_j^*, SK_S^*) and (a_j, SK_S), respectively such that by combining both keys, $\mathcal{A}$ has successfully decrypted the challenge ciphertext CT^*.

To find the corrupt keys that $\mathcal{A}$ has used to decrypt the challenge ciphertext, $\mathcal{B}$ ignores $\mathcal{A}$'s output guess and randomly chooses two tuples $(j, u_j, S, \mathrm{SK}_S, \mathrm{TK}_S, a_j, D_1)$ and $(j^*, u_j^*, S^*, \mathrm{SK}_S^*, \mathrm{TK}_S^*, a_j^*, D_1^*)$ from table T. Let us assume that tuple $(j, u_j, S, \mathrm{SK}_S, \mathrm{TK}_S, a_j, D_1)$ represents the corrupt query result for a nonrevoked user u_j and a set S that does not satisfy $\mathbb{A}^*$ whereas tuple $(j^*, u_j^*, S^*, \mathrm{SK}_S^*, \mathrm{TK}_S^*, a_j^*, D_1^*)$ represents the corrupt query result for a revoked user u_j^* and a set S^* that satisfies $\mathbb{A}^*$. Then, $\mathcal{B}$ carries out the decryption as follows:

The algorithm $\mathcal{B}$ calls the recursive algorithm `DecryptNode` ($\mathrm{CT}^*, \mathrm{TK}_S^*, v$) that works as follows. It returns F_v when called on a node v in the access tree Υ corresponding to $\mathbb{A}^*$. The base case of the algorithm is when v is a leaf node. For an attribute $i = attr(v)$, if $i \notin \mathrm{S}$, then $F_v = \perp$. Otherwise,

$$\begin{aligned} F_v &= \frac{e(C_{2,v}, D_{2,i})}{e(C_{3,v}, D_{3,i})} = \frac{e(g^{P_v(0)\beta}, (g^{r/\beta}.\mathcal{H}(i)^{r_i})^{1/t})}{e(\mathcal{H}(attr(v))^{P_v(0)}, g^{\beta r_i/t})} \\ &= e(g,g)^{rP_v(0)/t}. \end{aligned}$$

The recursive case of the algorithm is when v is a non-leaf node. Let us assume that $\mathcal{S}_v$ is the set of successfully decrypted child nodes of v. If $|\mathcal{S}_v| < t_v$, then $F_v = \perp$. Otherwise, the following value is returned:

$$\begin{aligned} F_v &= \prod_{w \in \mathcal{S}_v} F_w^{\mathcal{L}_{j,\mathcal{S}'_v}(0)} \text{ ;where } {}^{\;\;j=index(w)}_{\mathcal{S}'_v=\{index(w):w\in\mathcal{S}_v\}} \\ &= \prod_{w \in \mathcal{S}_v} \left(e\,(g,g)^{r.P_{parent(w)}(index(w))/t}\right)^{\mathcal{L}_{j,\mathcal{S}'_v}(0)} \\ &= \prod_{w \in \mathcal{S}_v} e\,(g,g)^{rP_v(j).\mathcal{L}_{j,\mathcal{S}'_v}(0)/t} = e(g,g)^{r.P_v(0)/t}. \end{aligned}$$

Eventually, `DecryptNode` returns $F_R = e(g,g)^{rs/t}$ if root R of Υ is successfully decrypted. A failure symbol $\perp$ is returned otherwise. Then, it chooses a random $b \in \mathbb{Z}_p$ and computes $C_1' = (C_1)^b), T = e(C_1', D_1^*)$. After that using C1MK, it computes $T_1 = T^{1/t_c.b} = e(g,g)^{(\alpha+r+xb_l^*K_l^*)K_ls}$. Then, sets $T_2 = C_0^* = K_R e(g,g)^{\alpha s + xyb_lK_ls}$. Eventually, $\mathcal{B}$ computes TGDH root secret key K_l from a_j, and retrieves K_R' as in

$$\begin{aligned} K_R' &= T_2(F_R)^t / T_1^{1/K_l} \\ &= \frac{K_R e(g,g)^{\alpha s + xyb_lK_ls} e(g,g)^{rs}}{e(g,g)^{(\alpha+r+xb_l^*K_l^*)s}} = \frac{K_R e(g,g)^{xyb_lK_ls}}{e(g,g)^{xb_l^*K_l^*s}} \\ &= \frac{K_R e(g^{xy},g)^{b_lK_ls}}{e(g^{xy^*},g)^{b_lK_ls}} = \frac{K_R e(Z,g)^{b_lK_ls}}{e(g^{xy^*},g)^{b_lK_ls}}. \end{aligned}$$

From the above equation, if $K'_R = K_R$, then Z must be equal to g^{xy^*} or $y = y*$. This means that $\mathcal{B}$ must have computed $Z = g^{xy}$ during computing D_1^*. Now, if the adversary $\mathcal{A}$ makes a total of q* corrupt queries that results in q* tuples in table T, then $\mathcal{B}$'s probability of randomly choosing tuple $(j^*, u_j^*, S^*, \mathrm{SK}_S^*, \mathrm{TK}_S^*, a_j^*, D_1^*)$ is $1/\mathrm{q}^*$. Given that $\mathcal{A}$ wins the IND-CPA game with a non-negligible advantage, $\mathcal{B}$'s probability of successfully computing Z is $1/\mathrm{q}^*$. Since q* is only polynomially large, $1/\mathrm{q}^*$ is non-negligible. This proves our theorem 4. Since the CDH assumption is believed to be hard, by contradiction, we can conclude that there exists no such polynomial time adversary $\mathcal{A}$ that can win the IND-CPA game with a non-negligible advantage.

11. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a directly revocable ABE scheme called ReVO-ABE using our proposed data structure called e-TGDH tree. It utilizes our federated cloud architecture (using two clouds) and a novel key binding technique to prevent collusion attacks and achieve revocation under the assumption that at least one of the two clouds acts honestly. Unlike existing schemes, ReVO-ABE does not put any cap on the number of user revocation or joining. The local decryption cost has been reduced by securely outsourcing most computationally expensive tasks to the cloud. Finally, we have built a multi-group secure data sharing scheme called DMG-SDS to demonstrate that our ABE scheme supports a muti-group setting. We have implemented our scheme, and the performance results show the effectiveness of the solution in comparison to others. In the future, we plan to develop an Android mobile client application based on our scheme and conduct more performance tuning for mobile platforms. In addition, we have only considered static access policy in this work. Thus, it will be interesting to see how it affects our system if dynamic access policy change is allowed.

REFERENCES

[1] Y. Yang, J. Liu, Z. Wei, and X. Huang, "Towards revocable fine-grained encryption of cloud data: Reducing trust upon cloud," in *Australasian Conference on Information Security and Privacy*, pp. 127–144, Springer, 2017.

[2] M. Lyu, X. Li, and H. Li, "Efficient, verifiable and privacy preserving decentralized attribute-based encryption for mobile cloud computing," in *Data Science in Cyberspace (DSC), 2017 IEEE Second International Conference on*, pp. 195–204, IEEE, 2017.

[3] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *Advances in Cryptology–CRYPTO 2012*, pp. 199–217, Springer, 2012.

[4] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355–370, 2014.

[5] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K.-K. R. Choo, "Cryptcloud+: secure and expressive data access control for cloud storage," *IEEE Transactions on Services Computing*, 2018.

[6] S. Jahid and N. Borisov, "Piratte: Proxy-based immediate revocation of attribute-based encryption," *arXiv:1208.4877*, 2012.

[7] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2013.

[8] C. Lyu, S.-F. Sun, Y. Zhang, A. Pande, H. Lu, and D. Gu, "Privacy-preserving data sharing scheme over cloud for social applications," *Journal of Network and Computer Applications*, vol. 74, 2016.

[9] H. Wang, Z. Zheng, L. Wu, and P. Li, "New directly revocable attribute-based encryption scheme and its application in cloud storage environment," *Cluster Computing*, vol. 20, no. 3, pp. 2385–2392, 2017.

[10] G. Yu, X. Ma, Z. Cao, W. Zhu, and G. Zeng, "Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation," in *International Conference on Information and Communications Security*, pp. 172–179, Springer, 2017.

[11] R. Zhang, L. Hui, S. Yiu, X. Yu, Z. Liu, and Z. L. Jiang, "A traceable outsourcing cp-abe scheme with attribute revocation," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pp. 363–370, IEEE, 2017.

[12] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2018.

[13] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, 2011.

[14] J. Li, W. Yao, J. Han, Y. Zhang, and J. Shen, "User collusion avoidance cp-abe with efficient attribute revocation for cloud storage," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1767–1777, 2018.

[15] R. R. Al-Dahhan, Q. Shi, G. M. Lee, and K. Kifayat, "Survey on revocation in ciphertext-policy attribute-based encryption," *Sensors*, vol. 19, no. 7, p. 1695, 2019.

[16] M. Green, S. Hohenberger, B. Waters, *et al.*, "Outsourcing the decryption of abe ciphertexts.," in *USENIX Security Symposium*, vol. 2011, 2011.

[17] B. Qin, R. H. Deng, S. Liu, and S. Ma, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1384–1393, 2015.

[18] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, 2005.

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

[20] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007.

[21] J. Li, Y. Shi, and Y. Zhang, "Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage," *International Journal of Communication Systems*, vol. 30, no. 1, p. e2942, 2017.

[22] G. Zeng, "Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation," in *Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6-8, 2017, Proceedings*, vol. 10631, p. 172, Springer, 2018.

[23] B. Qin, Q. Zhao, D. Zheng, and H. Cui, "(dual) server-aided revocable attribute-based encryption with decryption key exposure resistance," *Information Sciences*, vol. 490, pp. 74–92, 2019.

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

[25] H. Cui, T. Hon Yuen, R. H. Deng, and G. Wang, "Server-aided revocable attribute-based encryption for cloud computing services," *Concurrency and Computation: Practice and Experience*, p. e5680, 2020.

[26] X. Wang, Y. Chi, and Y. Zhang, "Traceable ciphertext policy attribute-based encryption scheme with user revocation for cloud storage," in *2020 International Conference on Computer Engineering and Application (ICCEA)*, pp. 91–95, IEEE, 2020.

[27] D. Han, N. Pan, and K.-C. Li, "A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[28] J. K. Liu, M. H. Au, X. Huang, R. Lu, and J. Li, "Fine-grained two-factor access control for web-based cloud computing services," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 484–497, 2016.

[29] J. Li, X. Lin, Y. Zhang, and J. Han, "Ksf-oabe: outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, 2017.

[30] J. Li, Y. Wang, Y. Zhang, and J. Han, "Full verifiability for outsourced decryption in attribute based encryption," *IEEE Transactions on Services Computing*, 2017.

[31] J. Li, F. Sha, Y. Zhang, X. Huang, and J. Shen, "Verifiable outsourced decryption of attribute-based encryption with constant ciphertext length," *Security and Communication Networks*, vol. 2017, 2017.

[32] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 60–96, 2004.

[33] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[34] B. Lynn, "The stanford pairing based crypto library," *Privacy Preservation Scheme for Multicast Communications in Smart Buildings of the Smart Grid*, vol. 324, 2013.

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

IV. A PERMISSIONED BLOCKCHAIN-BASED ACCESS CONTROL SYSTEM FOR IOT

MD Azharul Islam and Sanjay Madria

Department of Computer Science

Missouri University of Science and Technology

Rolla, Missouri 65401

Email: mdazharul.islam@mst.edu and madrias@mst.edu

ABSTRACT

IoT devices produce a lot of valuable and sensitive data that is often shared with external parties to provide different kinds of useful services. Traditional IoT access control systems are centralized and do not include all the stakeholders in the access control decision-making process. To fill this gap, we propose a permissioned blockchain-based access control system for IoT, where a different phase of access control like creating access policy and making the access control decision happens based on the consensus of all the stakeholders. To be more specific, we design and implement attribute-based access control (ABAC) in a permissioned blockchain called Hyperledger Fabric and leverage its smartcontract and distributed consensus to enable a distributed access control for IoT. The effectiveness of our proposed system is demonstrated by the performance evaluation result in an IoT testbed.

Keywords: IoT, Blockchain, Access control

1. INTRODUCTION

As IoT devices are becoming more popular, security and privacy of the heterogeneous data produced by these devices have become more important than ever before. This is because the data produced by IoT devices can contain extremely private information like audio and video clips from smart surveillance systems, medical information from fitness devices, location and activity pattern or even daily schedule of individuals in the house hold. Often times, IoT devices are not utilized to their full potential unless this data is shared with different service providers. For example, data from fitness devices may need to be shared with the physician and the hospital, temperature sensor data may need to be shared with the emergency department and service providers like Amazon and Google can collect user data through smart home devices like Echo, Google home etc. to ensure better quality of service. While sharing the IoT device data with other parties, there are two fundamental questions that need to be asked: 1) **Who** is accessing the shared data? and 2) **What** data is accessed? Answer to the first question determines whether the data falls into the hands of the wrong parties. On the other hand, the second question is to find out whether IoT data requester is collecting anything without the data owner's consent.

Currently, how data requester collects user data from IoT devices lacks transparency and even doubtful in some cases. This is because the owner has no role in the access control of how the data will be shared with the data consumer. Although in some cases, requesters provide the owner with some kind of agreement policy that the owner has to agree on to enjoy the intended service. This leaves the data owner with no other choice but to trust the data consumer blindly. These agreement policies are often very high level and obscure. Moreover, there is no way for the data owner to verify whether the requester is complying with the agreement and not collecting anything more than what was agreed upon. On top of that, it is hard to tell if different service providers implement their security mechanisms properly. This gives the malicious parties an opportunity to get access to the user's confidential and sensitive IoT device data by exploiting any security backdoor that may exist.

The above mentioned problems of IoT data security mainly stem from the fact that different parties involved in the IoT ecosystem are under different administrative entity and there may be a lack of trust between them. Using traditional approaches like [1], it is impossible to ensure the active participation of all mutually untrusted parties in different aspects of the IoT access control mechanism, like fixing the access policy and making the access control decision. Blockchain offers a great platform to build distributed applications for mutually untrusted parties by eliminating the need of a trusted central authority. At a high level, blockchain is a distributed immutable ledger maintained by a network of peers where all the peers in the network at any given point of time agrees on a single identical version of the ledger through some consensus protocol.

The blockchain empowering the cryptocurrencies like bitcoin [2] and Ethereum [3] are called public or permissionless blockchain as no permission is needed for a peer to participate in the blockchain. While public blockchain is well-suited for cryptocurrency, it has a scalability issue that limits the number of transactions the network can process referred to as blockchain bloat [4]. For example, bitcoin can process only a maximum of seven transactions per minute. This is due to the fact that the block creation frequency (1 block per 10 minutes) and size (1MB) is limited [2]. The security of the public blockchain relies on the proof of work (PoW) where all the peers in the network validate all the transactions and try to solve a computationally intensive cryptographic puzzle. The hardness of the puzzle is set so that a new block is created every 10 minutes. Due to the network latency, there exist multiple forks of the blockchain and it can take up to six hours to eventually reach a consensus. That is why transaction wait time is very high in pubic blockchain (sometimes up to six hours). Though, consensus protocols like proof of stake (PoS) are there, the transaction wait time is still high in public blockchain [5]. However, in the private or permissioned blockchain, the transactions are much faster. This is because it does not rely on PoW or PoS. Rather, it incorporates much faster consensus protocols like Byzantine Fault Tolerance (BFT) and yet provides a way to secure the transactions among a group of participants with verified identities who have a common goal but do not fully trust each other [5]. As a result, it is a better fit for IoT access control. This inspires us to propose an access control system for IoT which is based on permissioned blockchain.

By utilizing the smartcontract, we implement attribute based access control (ABAC) in the blockchain which is a great fit for IoT than other access control mechanisms such as role based access control or identity based access control. This is because IoT ecosystem consists of a huge number of IoT devices that varies in functionalities, characteristics and capabilities. Only ABAC can offer expressive fine-grained access control in such a diverse environment. We implement our access control system in Hyperledger Fabric [5] which is an open source implementation of a permissioned blockchain, and evaluate its effectiveness in an IoT testbed. Our results also demonstrates the practicality of our proposed system. The most closely related work to ours is [6] which is a physical access control management system based on permissioned blockchain. However, it is not specifically intended for IoT and it can not therefore enforce fine-grained access control like ours as role based access control was used. IoT access control systems proposed in [7, 8, 9, 10] are also related to ours. Since they are based on public blockchain, it is needless to say that all of them inherit the the existing limitations of the public blockchain. Besides, access control mechanisms proposed in these works are not as sophisticated as ABAC.

Our contributions can be summerized as follows:

- To the best of our knowledge, we are the first to propose a private blockchain based IoT access control system based on attribute based access control. Since our scheme is based on private blockchain, the access requests are resolved much faster than that of a public blockchain.
- We report a full implementation of our proposed system in a permissioned blockchain platform called Hyperledger Fabric and prove its practicality by evaluating its performance in an IoT testbed environment.

2. RELATED WORK

Besides cryptocurrency, the adoption of blockchain is also noticeable in the IoT spectrum. For example, an access control mechanism based on bitcoin was proposed in [4, 7]. Since, bitcoin does not support smartcontract, the proposed access control mechanism is very basic and does not offer fine-grained access control for heterogeneous IoT devices. [8] proposed an architecture for scalable access management of IoT devices based on Ethereum smartcontract where blockchain is run in the IoT devices. It was implemented in a small scale local Ethereum test network. However, it is not clear how it is going to work in the original pubic Ethereum network. Dorri et al. proposed a solution in [11, 9, 10] where an overlay network is formed by the IoT nodes. Multiple overlay nodes form a cluster with an elected cluster head (CH) for each cluster. The CHs maintains a newly proposed pubic blockchain. It has a new transaction format and does not have smartcontract which makes implementation of sophisticated access control policy really hard. An access control management system based on private blockchain has been proposed in [6] although it was not directly intended for IoT. Besides, role-based access control was considered in this work which is not a good fit for IoT access control involving a large scale of heterogeneous devices with lack of standardization.

3. BACKGROUND

In this section, we discuss the necessary background of for our proposed system. First, we discuss how resource is managed inside an IoT network. Then, we discuss different components of Hyperledger Fabric.

3.1. RESOURCE MANAGEMENT IN AN IOT NETWORK

Many standards in line with the IEEE 802.15.4 radio [12] were defined by the IETF for IoT networks. IoT-Auth [1] adopted many of these well defined protocols to manage resource within an IoT network. For this purpose, the *gateway* maintains three data structures: *Routing Table, Resource Table*, and *Data Table*. There are following steps involved:

3.1.1. Device Discovery. The first step is to do the device discovery. It is done according to the Routing Protocol for Low Power and Lossy Networks (RPL) protocol. More specifically, a network coordinator or *sink node* initiates this process by periodically sending in the IoT network the Destination Oriented Directed Acyclic Graph (DODAG) Information Object (DIO) message. A newly joining device replies with the Destination Advertisement Object (DAO) message. The *sink node* forwards this message to the *gateway*. The *gateway* stores the identifier of all the active IoT end nodes along with their communication path in the *Routing Table*.

3.1.2. Resource Discovery. The *gateway* starts the process by sending a GET message to the well-known URI of the IoT device called *./well-known/core*. The IoT device sends a response message in the Constrained RESTful Environments (CoRE) Link Format. The response message includes information such as resource type *(rt)*, interface description *(if)*, and maximum size estimate *(sz)*. These information are processed and stored in the *Resource Directory*.

3.1.3. Data Collection. The *gateway* collects data from the intended IoT device as new data is available. Data is stored in the *Data Table* along with its identity information.

3.2. HYPERLEDGER FABRIC

Hyperledger Fabric is an opensource implementation of a permissioned blockchain [5]. It offers a modular architecture allowing different kinds of pluggable functionalities by leveraging well known and proven technologies. One of the most powerful aspect of Hyperledger Fabric is that it gives a platform to run smartcontracts on the blockchain. Smartcontracts are special kinds of programs that can be written using traditional programming language such as GO to perform different kinds of operations on the underlying blockchain. We discuss the core building blocks of Hyperledger Fabric as follows:

3.2.1. Peer, Organization, and Client. In Hyperledger Fabric, peers are the nodes that host the blockchain and runs the smartcontract. There could be two basic types of peers based on the role it takes up: *validating peer* and *non-validating peer*. A *validating peer* runs the consensus protocol, executes and validates transactions, and maintains the blockchain. A *non-validating peer* acts as a proxy to connect the external applications/clients to the *validating peers*. Some peers can take up a special role of endorsing transactions referred to as *endorsing peers*. Peers are part of a conceptual entity called the organization. Each peer is part of some organization and multiple organizations collectively maintain the blockchain. On the other hand, clients are the entities that submit transaction requests to the blockchain. They are normally third-party applications written by the provided SDK.

3.2.2. Membership Service Provider (MSP). The permission to participate in the blockchain is handled by the MSP. For example, every peer needs to collect *enrollment certificate* and *transaction certificate* from the designated certificate authority (CA) of the MSP to connect to the network and submit transactions, respectively. Each organization can have a separate MSP that independently operates its own membership service.

3.2.3. Transaction Endorsement. Hyperledger fabric does not rely on *proof-of-work* or *proof-of-stake* to maintain the immutability of the blockchain or to prevent double spending. Rather, it relies on the *endorsement policy* that states which peers need to endorse a transaction to be considered as a valid one. An *endorsement policy* is written using *endorsement policy syntax*. For example, endorsement policy *OR('Org1.peer1', AND('Org2.peer2', 'Org3.peer3'))* states that transaction needs to be endorsed by either peer1 of Org1 or by both peer2 of Org2 and peer3 of Org3.

3.2.4. Ordering Service. The ordering service accepts endorsed transactions from the client, orders them according to the plugged-in consensus protocol, and delivers them to the designated peers to be written in the blockchain. It guarantees the proper ordering of the transactions that ensures the consensus.

3.2.5. Chaincode. Chaincode is similar to smartcontract in the context of Hyperledger Fabric. These are piece of programs written in traditional programming language such as Go, java, and node.js and can manipulate the blockchan. In this paper, we will use the term smart contract and chaincode interchangeably.

3.2.6. Blockchain Data Structures. Blockchain in Hyperledger Fabric incorporating two different kinds of data structures: *state* and *ledger*. *State* stores the latest state of the blockchain by modeling it as a key-value storage (KVS). It is maintained and hosted by the peers and can be manipulated from the chaincode, triggered by transactions. On the other hand, *ledger* stores the verifiable history of all the unsuccessful attempts and successful change made in the *state* as a totally ordered *hashchain* of blocks of transactions.

4. PROPOSED SYSTEM ARCHITECTURE

We discuss our system architecture in this section. Our proposed architecture is depicted in Figure 1. In the following subsections, we first discuss the main actors of our system. Then, we discuss the main components followed by how the constrained IoT resource is accessed by a requester who is external to the IoT network.

4.1. ACTORS

There are mainly two types of actors in our system as discussed below:

4.1.1. Resource Provider/Owner. This actor is basically the owner of the IoT equipped smart home, smart office, school etc. where variety of IoT devices produce different kinds of data. The data could range from environment sensing data (such as temperature, pressure, humidity, luminosity, etc.) to healthcare data generated by wearable devices or even image, audio and video data generated by surveillance systems.

4.1.2. Requester. Any party that accesses data generated by IoT devices is a requester. Normally, a party who relies on the IoT data to provide different kinds of service is considered as data requester. For instance, google, amazon provides services like music streaming, voice search result etc. based on the data provided by google home, amazon echo. Emergency service providers such as hospitals, fire service etc. are also requesters since emergency services may rely on the IoT device such as elderly monitoring device, or different healthcare devices for data. Different research organizations may also rely on user IoT data to conduct scientific research and survey. Finally, regulatory organizations are also considered as requester since they may need to access IoT data for auditing purpose. These requesters are generally external to the IoT local network and access IoT data by through access control mechanism imposed by blockchain.

4.2. COMPONENTS

The main components of our system are the local IoT networks and the blockchain. A brief discussion of these components are given below:

4.2.1. Local IoT Network. Each local IoT network is composed of one or more IoT devices, a sink node and a gateway. The sink node works like a network coordinator for all the IoT devices and is connected to the gateway. The gateway acts as an interface to the external world to access any resource within the local IoT network. A gateway can

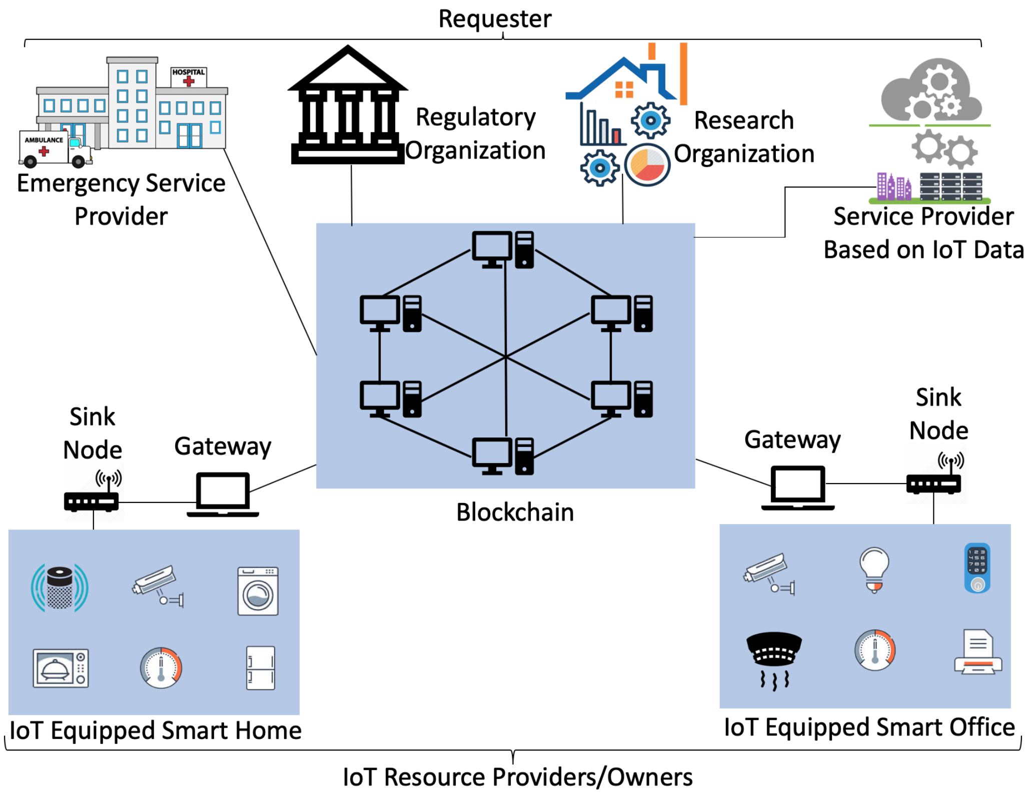


Figure 1. System Architecture

have its own public IP address or may be connected to the cloud that provides it with a public interface so that resource requesters can access IoT data from outside the local IoT network. It manages all the information available within the IoT network. For this purpose, it maintains three data structures named as *Routing Table, Resource Dictionary,* and *Data Table*. According to our architecture, there can be many local IoT networks as such, each representative of an IoT equipped smart home, office or school etc.

4.2.2. Blockchain. The blockchain in our architecture is a permissioned one, implemented in Hyperledger Fabric. All the attributes and ABAC policies upon validation are stored in the blockchain. It also works as a policy enforcement point for any access request to a particular IoT resource. In brief, the blockchain provides an unified ABAC platform for the entire IoT ecosystem.

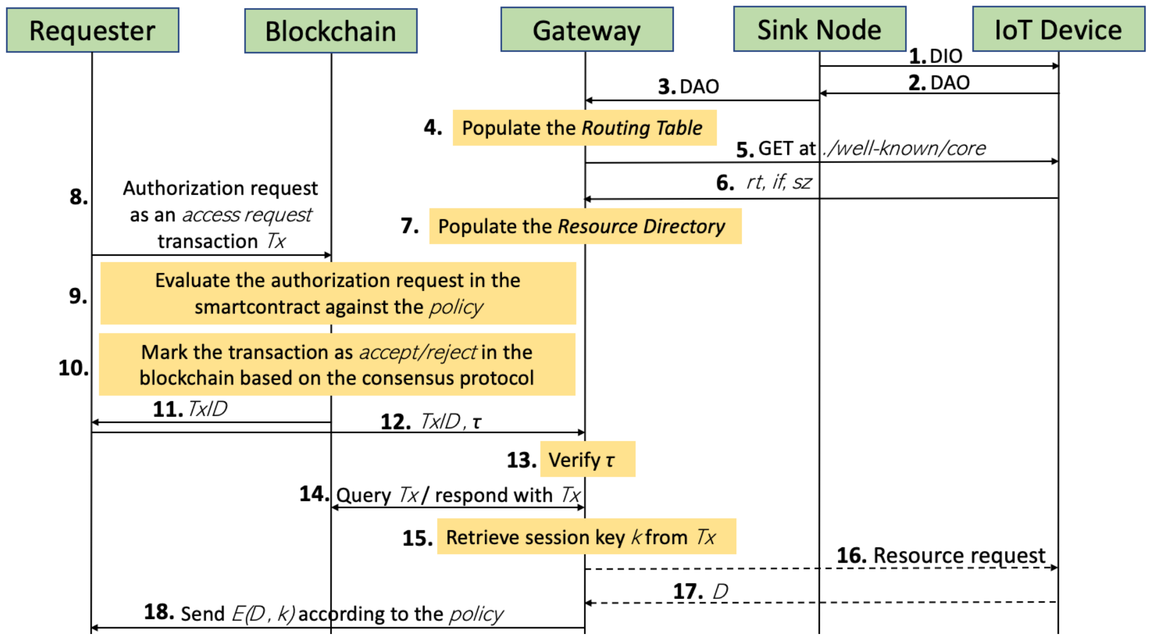

Requester
Blockchain
Gateway
Sink Node
IoT Device
1. DIO
2. DAO
3. DAO
4. Populate the *Routing Table*
5. GET at ./well-known/core
6. rt, if, sz
7. Populate the *Resource Directory*
8. Authorization request as an *access request* transaction *Tx*
9. Evaluate the authorization request in the smartcontract against the *policy*
10. Mark the transaction as *accept/reject* in the blockchain based on the consensus protocol
11. TxID
12. TxID, τ
13. Verify τ
14. Query Tx / respond with Tx
15. Retrieve session key k from Tx
16. Resource request
17. D
18. Send E(D, k) according to the *policy*

Figure 2. Resource Access

4.3. RESOURCE ACCESS PROCESS BY THE REQUESTER

The sequence diagram presented in Figure 2 shows how the IoT resource is accessed by the requester. Steps 1, 2, 3, and 4 is for the *Device Discovery* and populating the *Routing Table*. Steps 5, 6, and 7 are to complete the *Resource Discovery* process and populate the *Resource Table* as described in Section 3.1. At this point, a requester who is external to the IoT local network, can request for IoT resource. First, the requester sends the authorization request as an *access request* transaction *Tx* (step 8). Let PK_{Ow} and DK_{Ow} be the public and private key of the IoT resource provider/owner respectively; *Enc* and *Dec* be the public key encryption and decryption algorithm respectively; SK_{Re} and VK_{Re} be the requester's signing and verification key respectively; *Sig* and *Ver* be the DSA (Digital Signature Algorithm) signing and verification algorithm respectively, and *H(.)* be a secure hash function. In the authorization request, the requester sends the following information:

$$Req = (Resource\ Location,\ policyID,\ C),\ \sigma \tag{1}$$

Resource Location is Equation 1 is the IP address or public interface of the *gateway* of the intended IoT network, *PolicyID* is the ID of the ABAC *policy* that determines the access control rule of the resource. $C = Enc(k, PK_{Ow})$, where k is a session key. Signature σ is computed as $\sigma = Sig(H(Req), SK_{Re})$.

In step 9, each *endorsing peer* in the blockchain first checks the validity of the request by checking the following equality:

$$H(Req) == Ver(\sigma, VK_{Re}) \tag{2}$$

Then, the *policy* corresponding to the *PolicyID* is retrieved from the blockchain. Each *endorsing peer* evaluates the request in the smartcontract against the *policy*. In step 10, the transaction is marked as *accept / reject* based on the consensus protocol and committed in the blockchain. The following information is saved in the transaction:

$$TxID, Tx = (Req, \sigma, Decision) \tag{3}$$

Here, *TxID* is a unique ID for *Tx*, and *Decision* can have as a value either *accept* or *reject*. Then, the requester gets the *TxID* from the blockchain (step 11). Later, the requester sends *TxID*, $\tau = Sig(H(TxID), SK_{Re})$ to the *gateway* indicating its intention to access the IoT resource (step 12). In step 13, the *gateway* checks the authenticity of the request by verifying the requester's signature as in $H(TxID) == Ver(\tau, VK_{Re})$. Later in step 14, the *gateway* queries the blockchain for transaction *Tx* with *TxID* and gets the *Tx* as response. Then (in step 15), upon confirming if the transaction was marked as *accept*, the *gateway* retrieves the session key k by decrypting C with its decryption key as $k = Dec(C, DK_{Ow})$. The *gateway* checks if the resource is sent to the proper requester by verifying σ using Equation 2 with the same verification key used to verify τ. If the requested resource according to the *policy* is available in the *Data Table*, the *gateway* sends to the requester the data *D* right away by symmetrically encrypting it with *k* as in *E(D,k)* (step 18). Otherwise, the *gateway* polls the resource from the designated IoT device(s). That is why steps 16 and 17 are shown in dotted arrows.

5. DETAILS OF OUR ABAC MODEL

In this section, we discuss our attribute based access control (ABAC) model in detail. This includes how we model attributes and policies and how those policies are evaluated against attributes.

5.1. MODELING ATTRIBUTES

Attribute is a core component of our ABAC system. We model attribute as a multi-component data structure. Each attribute is composed of three components: name, value, and type. Name is simply a string. On the other hand, value can be any data type such as string, numeric, date etc. The general representation of an attribute is as follows:

$$att_i^t = (name,\ t,\ val) \tag{4}$$

In Equation 4 *name, t,* and *val* stands for the name, type, and value of the attribute, respectively. We use the dot (.) operator to access an assigned member of an attribute, i.e. $att_i^t.val$ is used to access the assigned values of att_i^t. The set of all possible values of an attribute att_i^t is denoted as $Val(att_i^t) = \{val_{i,j} : 1 \leq j \leq M_i\}$. M_i is the total number of possible values of att_i. In our system, we consider four types of attributes: subject attribute, resource attribute, environment attribute, and action attribute. Description of these four types of attributes is as follows:

5.1.1. Subject Attribute. This type of attribute is used to completely describe the requester or the resource owner. Examples of subject attribute can be the organization, title, rank etc. Let att_i^{Sub} and N_{Sub} be a subject attribute and the total number of subject attributes in the system, respectively. Then, the set of all subject attributes are expressed as $Attr(Sub) = \{att_i^{Sub} : 1 \leq i \leq N_{Sub}\}$.

5.1.2. Resource Attribute. Anything that can describe the IoT resource properly are considered as resource attributes and is denoted as att_i^{Res}. For example, the IoT resource name, type, identifier etc. are resource attributes. If there are N_{Res} resource attributes in the system, then all resource attributes are represented as $Attr(Res) = \{att_i^{Res} : 1 \leq i \leq N_{Res}\}$.

5.1.3. Environment Attribute. Time, location etc. related attributes are considered as environment attributes and are represented as att_i^{Env}. We assume that there are N_{Env} environment attribute in total and express them as $Attr(Env) = \{att_i^{Env} : 1 \leq i \leq N_{Env}\}$.

5.1.4. Action Attribute. Any type of action the requester or resource owner is allowed to perform falls into this category. Example includes attributes such as read, write, delete, update etc. Action attributes are represented as att_i^{Act}. Let $Attr(Act)$ be the set of all N_{Act} possible action attributes in the system which is expressed as $Attr(Act) = \{att_i^{Act} : 1 \leq i \leq N_{Act}\}$.

5.2. MODELING POLICY

A policy is expressed as a boolean expression that defines the access rule to a resource in terms of attributes and their corresponding values. In our ABAC model, policies are very expressive as both attribute values and attributes themselves are boolean expressions. A complete policy consists of *attribute value expression* and *attribute expression* as discussed below:

5.2.1. Attribute Value Expression. Allowed values of an attribute att_i^t in a policy are expressed as the following boolean expression:

$$E_{att_i^t} := Exp\ (\mathcal{E}[, \mathcal{E}]) \tag{5}$$

In Equation 5, *Exp* is either *AND* or *OR* joining two boolean expressions and $\mathcal{E}$ is either an attribute value $val_{i,j} \in Val(att_i^t)$ or a recursive call to *Exp*.

5.2.2. Attribute Expression. For each attribute type *t*, we use a separate policy $\mathbb{P}_t$. Each such policy is a boolean expression composed of *t* type attributes as in:

$$\mathbb{P}_t := Exp\ (\mathbb{E}[, \mathbb{E}]) \tag{6}$$

In the above equation, *E* is either an attribute value expression $E_{att_i^t}$ or a recursive call to *Exp*. Finally, the combined policy is written as a conjunction of all four types of policies as in Equation 7:

$$\mathbb{P} = \mathbb{P}_{Sub}\ AND\ \mathbb{P}_{Res}\ AND\ \mathbb{P}_{Env}\ AND\ \mathbb{P}_{Act} \tag{7}$$

5.3. POLICY EVALUATION

Granting an access request for a resource is determined by evaluating a policy against a set of subject, object, environment and action attributes. A complete policy $\mathbb{P}$ contains four *attribute expressions* and each attribute in an *attribute expression* contains an *attribute value expression*. So, evaluation of $\mathbb{P}$ can be broken down into following two parts:

5.3.1. Evaluation of Attribute Value Expression. If $\mathcal{S} \subset Val(att_i^t)$ is a set of assigned values to a particular attribute att_i^t, and $\mathcal{I}$ is the minimum number of values required to satisfy the boolean expression in $E_{att_i^t}$, then we say that att_i^t satisfies $E_{att_i^t}$ if

$\mathcal{I} \subset \mathcal{S}$. It is expressed by the following notation:

$$E_{att_i^t} \vdash att_i^t = \begin{cases} true, & \text{if } \mathcal{I} \subset \mathcal{S}, \\ false, & \text{otherwise.} \end{cases}$$

5.3.2. Evaluation of Attribute Expression. Let $\mathbb{I}_t$ be the minimum number of required attributes to satisfy the boolean expression in $\mathbb{P}_t$, and $E_{att_i^t} \vdash att_i^t = true$ for $\forall att_i^t \in \mathbb{I}_t$ where $E_{att_i^t} \in \mathbb{P}_t$. Then we say that attribute set $\mathbb{A}_t \subset Att(t)$ satisfies $\mathbb{P}_t$ if $\mathbb{I}_t \subset \mathbb{A}_t$, and it is represented by the following notation:

$$\mathbb{P}_t \vdash \mathbb{A}_t = \begin{cases} true, & \text{if } \mathbb{I}_t \subset \mathbb{A}_t, \\ false, & \text{otherwise.} \end{cases}$$

Finally, the satisfaction of a complete policy $\mathbb{P}$ by an attribute set $\mathbb{A} = \{\mathbb{A}_{Sub}, \mathbb{A}_{Res}, \mathbb{A}_{Env}, \mathbb{A}_{Act}\}$ is represented by the following notation:

$$\mathbb{P} \vdash \mathbb{A} = \begin{cases} false, & \text{if } \exists \mathbb{A}_t \in \mathbb{A} : \mathbb{P}_t \vdash \mathbb{A}_t = false, \\ true, & \text{otherwise.} \end{cases}$$

6. ABAC IMPLEMENTATION IN HYPERLEDGER FABRIC

In this section, we discuss how our ABAC model discussed in the previous section is implemented in Hyperledger Fabric. The first step towards our ABAC implementation is to form the blockchain network. After that, attribute creation and assignment, policy creation and resource access request are done by sending transactions to the blockchain by the requester. These steps are discussed in the following subsections.

6.1. FORMING THE BLOCKCHAIN NETWORK

The permission to join the blockchain is managed by the MSP (Managed Service Provider) of some high-level organizations. For our ABAC implementation in the Hyperledger Fabric, we assume that there exist multiple high level organizations. For example, an organization may represent all the regulatory institutions, companies like google, amazon who provide IoT based services can have their own authoritative organizations in the blockchain, and research institutions can have an authoritative organization as well. Finally, the data owners must be part of some organization also. For example, a smart city can play the role of an organization for all the smart home owners of that city. Each organization may have one or more running peers. It is worth noting that the data owners need to have their own running peers to be able to directly take part in the access control decision making process of their data. There will be some dedicated nodes that will perform the ordering service. Peers from different organizations along with the ordering service collectively form and run the blockchain.

6.2. SETTING UP THE ENDORSEMENT POLICY

The consensus in hyperledger fabric largely depends on the endorsement policy. This is because it dictates who need to endorse a particular transaction to be considered by the validating peers as a valid one. A data owner fixes the endorsement policy by creating a configuration transaction in the blockchain. The endorsement policy along with the identity of all the endorsing peers are embedded in this transaction. An endorsement policy creates a logical channel between the endorsing peers and the ordering service. In order to be committed in the blockchain, transactions submitted to a channel need to be endorsed by the channel's endorsing peers according to the endorsement policy. Different data owners will have different endorsement policies. Hence, many channels as such will exist in the Hyperledger Fabric.

6.3. ATTRIBUTE MANAGEMENT

For an ABAC system to function properly, attributes should be created with name, type and a set of allowable values. After creating attributes, they need to be assigned to different entities. Subject attributes are created and assigned to the specific subject by an attribute authority through an administration point. There are multiple such authorities, each with authority over different set of subject attributes. In our blockchain based ABAC model, the MSP of each organization plays the role of this attribute authority. Resource attributes on the other hand are created and assigned to the specific resource by the owner of that resource. Environment and action attributes are system wide common, and must be created by the regulatory organizations. We use general terms attribute creator and issuer to refer to the entity responsible for attribute management. In practice, they are the same entity. Attribute creation and assignment is handled by a smart contract function named *AttributeMgr* and details are discussed below.

6.3.1. Attribute Creation. No attribute can be used in our system without registering it in the blockchain. To register an attribute, the creator first sends a transaction request in the bockchain. Within the transaction, the complete attribute structure as in Equation 4 is embedded. *AttributeMgr* parses the attribute from the transaction, checks the semantics, and converts it into a json object. Besides name, type and value, some additional fields such as creatorID, organization name (orgName) are also added in the json object. Upon endorsement, ordering and verification phase, the json object is written in the key-value storage of Hyperledger Fabric called the *state*. One critical issue of ABAC system is the conflict resolution of attributes. Conflict during the attribute creation occurs when two different attributes with the same name are created by two different creators. For example, *manager* attribute of org A is different from the *manager* attribute of org B. The system should allow both org A and org B to create *manager* attribute. At the same time, the system should know the difference between them. *AttributeMgr* resolves this issue prior to writing the attribute in the *state* by creating unique IDs for each attribute as follows:

$$ID_{att_i^t} = H\left(orgName \mid\mid \text{creatorID} \mid\mid att_i^t.name \mid\mid att_i^t.t\right)$$

$ID_{att_i^t}$ is used as the key for the attribute when stored in the *state*. Two attributes with the same key cannot be stored in the *state*. This allows the creation of attributes with same name by two different creators while restricting same creator from creating duplicate attributes.

6.3.2. Attribute Assignment. After attribute creation, the issuer has to assign the attribute along with the appropriate set of values to the proper entities. It is important to take enough security measures so that attributes cannot be altered or tampered with to maliciously satisfy an access policy. To accomplish this, we cryptographically bound attributes to the entities. During attribute assignment, the attribute issuer would add the attributes to the `attributes` field of an X.509 attribute certificate (AC) according to the IETF standard [13]. The issuer sends this certificate by embedding it in a blockchain transaction. *AttributeMgr* verifies the certificate and converts it in a json object to store it in the *state*. Finally, it is stored in the *state* after endorsement, ordering and verification phase.

6.4. ABAC POLICY MANAGEMENT

In our blockchain based access control system, access to the restricted IoT resource is controlled by the ABAC access policy. Both the IoT resource owner and requester first agrees on a policy which is expressed according to our policy model as discussed in Section 5.2. The policy is sent in a blockchain transaction. The transaction has to be endorsed by both the resource owner and the requester. It is then written in the *state* as a key-value pair with key being $policyID = H(\mathbb{P})$, and the value being the policy itself.

Meta Policy: Some policies may require meta policy. It determines things like who can modify or delete the actual policy $\mathbb{P}$, the validity period of $\mathbb{P}$ etc. Meta policy is denoted by $\mathbb{MP}$ and modeled in the similar fashion as the original policy except the resource attribute expression $\mathbb{P}_{Res}$ points to $\mathbb{P}$. Meta policy is included in the blockchain transaction when the policy is created. Default meta policy applies to the policies that do not have any meta policy. According to the default policy, only the resource owner can modify or delete the policy. For the security purpose, meta policy is assumed to be immutable.

A smartcontract function named *PolicyMgr* is responsible for policy management tasks such as checking the semantics while policy creation and modifying the policy according to the meta policy.

Algorithm 4

```
Procedure: ACDecMaker ( Req, $\sigma$ )
 1: get policy $\mathbb{P}$ from the state database for key Req.policyID
 2: get the relevant attribute set $\mathbb{A} = \{\mathbb{A}_{Sub}, \mathbb{A}_{Res}, \mathbb{A}_{Env}, \mathbb{A}_{Act}\}$ from the state database
 3: if $H(Req) \mathrel{!}= Ver(\sigma, VK_{Re})$ then
 4:     return ($TxID, Tx = (Req, \sigma, reject)$)
 5: end if
 6: for each $(\mathbb{A}_t, \mathbb{P}_t)$ in $(\mathbb{A}, \mathbb{P})$ do
 7:     if $\mathbb{P}_t \vdash \mathbb{A}_t == true$ then
 8:         continue
 9:     else
10:         return ($TxID, Tx = (Req, \sigma, reject)$)
11:     end if
12: end for
13: return ($TxID, Tx = (Req, \sigma, accept)$)
```

6.5. POLICY EVALUATION

The policy evaluation logic is implemented in a smart contract function called *ACDecMaker* (access control decision maker). The endorsing peers responsible for endorsing resource access request transaction *Tx* (Equation 3) invoke this smartcontract.

ACDecMaker takes as input *Req = (Resource Location, policyID, C)*, σ and checks if the access policy $\mathbb{P}$ corresponding to the *policyID* is satisfied by the relevant attribute set $\mathbb{A}$. The algorithm for *ACDecMaker* is shown in Alg. 4.

7. EXPERIMENT

We did a full implementation of our blockchain based access control system in order to demonstrate the practicality of our solution. In the following section, we first provide details of our implementation, i.e. blockchain and IoT network testbed implementation. Then, we evaluate the performance of our system by experiments.

7.1. THE IOT NETWORK

We have developed an IoT testbed in our lab using MEMSICâĂŹs TelosB Mote TPR2420CA devices [14]. TPR2420CA bundles many essential elements required to perform IoT based lab studies such as an integrated temperature, light and humidity sensor, an IEEE 802.15.4 radio with integrated antenna etc. In our testbed, we divide the sensors into three groups: group 1, 2, and 3. Each group has four TPR2420CA devices- three of them act as IoT end device and one serves as the purpose of a sink node. All three sink nodes are connected to a PC through a USB hub and directly communicate with the PC through USB port. For each group, the PC runs a separate gateway program. The gateway program is written in java and uses SQLite for storing *Routing Table, Resource Dictionary,* and *Data Table*. Each group along with the gateway forms an individual IoT network. The device discovery and the resource discovery within the IoT network is done according to the IEEE 802.15.4 standard as discussed in Section 3.1. Each group of our testbed represents an IoT equipped smart home, office or school etc. in real life.

Table 1. Endorsement policy for IoT resource access control of different IoT groups

ID	Endorsement Policy	Intended IoT group
1	Any peer from OrgA, OrgB or OrgC	Group 1
2	At least one peer from each Org	Group 2
3	All peers from OrgA, OrgB, and OrgC	Group 3

7.2. THE BLOCKCHAIN NETWORK

Our blockchain network has been implemented in Hyperledger Fabric v1.3 [15]. We have considered three different organizations: OrgA, OrgB, and OrgC for our Hyperledger Fabric setup. We assume that all IoT resource owners belong to OrgA and each of them has a running peer under OrgA (OrgA.peer1-3). OrgB is the representative organization for all resource requesters and there are five running peers (OrgB.peer1-5) under it. Finally, we assume that OrgC represents all regulatory organizations and there are three running peers (OrgC.peer1-3) under it. All peers of OrgA are hosted in a desktop with Intel® Core i5-2400@3.1 GHz $\times$ 4 processor and 8 GB RAM running Ubuntu 16.04. A desktop with Xeon(R) ES-1620v2@3.7 GHz $\times$ 8 processor and 16 GB RAM running Ubuntu server 16.04.3 was used to host all the OrgB peers. Peers of OrgC run in a desktop which has the similar configuration as that of OrgA. As the ordering service, we have two orderer nodes backed by a kafka-zookeeper cluster. The ordering service runs in a separate desktop having the same configuration as that of OrgA.

We have written a java web based client application using the fabric client SDK to interact with the blockchain. This client application provides a simple interface to create attributes, assign attributes to a particular entity and create ABAC policy targeting a specific resource.

During the bootstrap phase, the MSP of each organization provides the participants with necessary crypto materials, i.e. certificates, signature keys, and encryption keys. Then, three different endorsement policies are created for three different IoT groups as in Table 1.

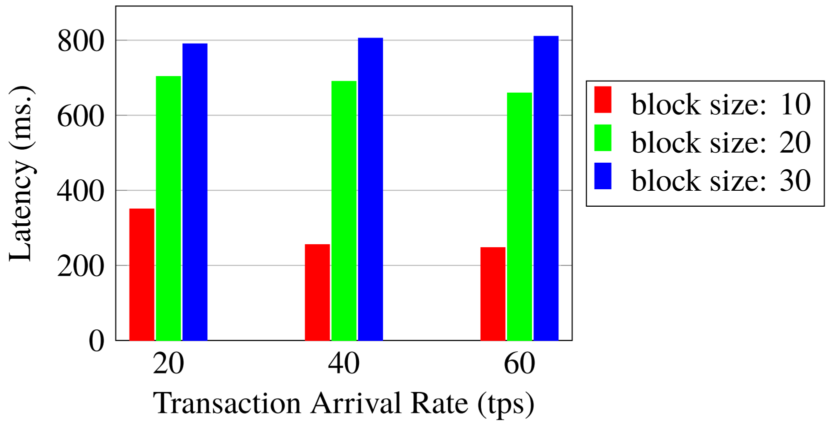


Figure 3. Attribute Creation

After setting up the endorsement policies, our smartcontract with three main functions, i.e. *AttributeMgr, PolicyMrg,* and *ACDecMaker* is installed in each peer. Then, the MSP of each organization creates subject attributes and registers in the blockchain. Environment and action attributes are created by the MSP of OrgC. Finally, the resource attributes for each IoT group is created by their respective owner using the client application. After attribute creation, attribute is assigned to different entities. In our experimental setup, we create 20 subject, 20 resource, 10 environment and 5 action attributes, each having 5 values. Five different ABAC policies were created with the number of attributes required to be satisfied ranging from 10 to 50.

7.3. WORKLOADS AND EXPERIMENTAL RESULTS

We show the performance of our scheme in terms of how fast different ABAC actions can be performed. All the results presented are averaged over five runs. Three of the most important fabric configurable parameters are the *stateDB*, endorsement policy, and block size. Between the two choices of GoLevelDB and CouchDB, we choose *stateDB* as it was shown in [16] that GoLevelDB has better throughput and faster read/write. A new block is created when either the number of pending transactions since the last written block reaches

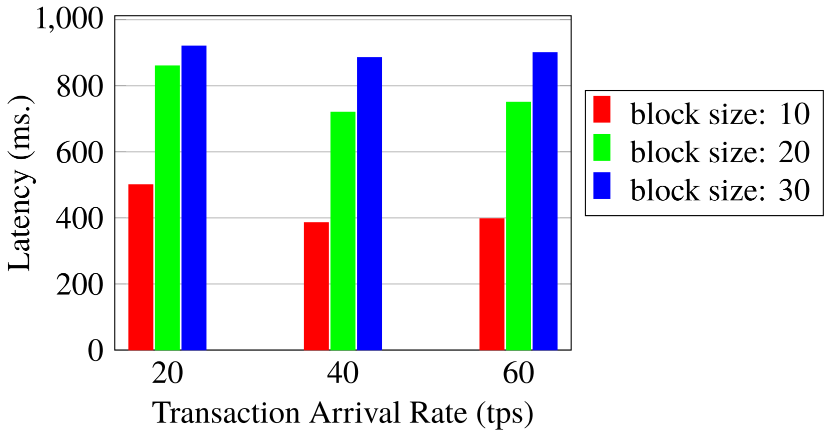


Figure 4. Attribute Assignment

the block size or timeout happens. We set the timeout to be 1 second. Then, we examine the latency for attribute creation, attribute assignment and policy creation operations for block size values 10, 20, and 30 with three different transaction arrival rates (20, 40 and 60 transactions per second). It is noticeable from Figure 3, 4, and 5 that the latency increases with the increase in block size. For example, when the transaction arrival rate for attribute creation (Figure 3) is at 40, an increase in the block size from 10 to 30 increases the latency by 3-fold from 255 ms. to 805 ms. This is because with larger block size, a pending transaction has to wait a little longer at the orderer queue causing delay in the transaction writing rate in the blockchain on average. Between attribute creation, attribute assignment, and policy creation, we observe that attribute assignment takes longer on average compared to the other two operations. The reason for this is that *AttributeMgr* in this case verifies the `signatureVale` in X.509 attribute certificate which is an expensive cryptographic operation. Across the board, we notice the lowest latency when the transaction arrival rate is at 40 tps and the block size is 10. Note that, we use the first endorsement policy in Table 1 for the three experiments discussed above.

For the next experiment (Figure 6), we set the block size to be 10 and transaction arrival rate at 40 tps which was found to be the optimum for our Hyperledger Fabric setup.

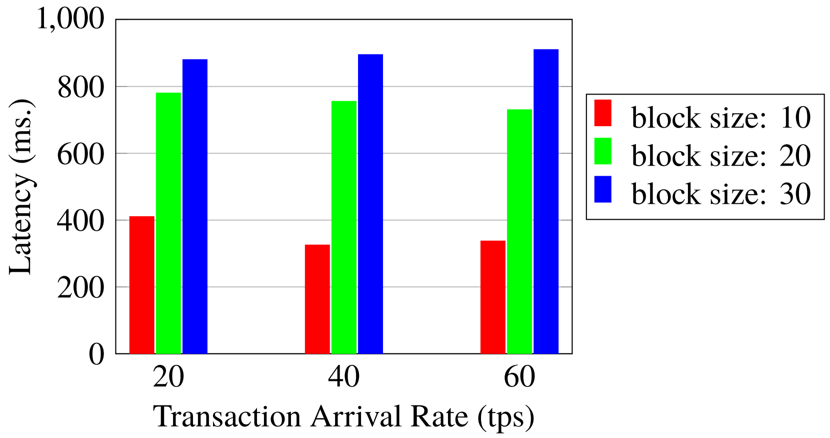


Figure 5. Policy Creation

Besides, hash function *H*, public key encryption (*Enc, Dec*), and digital signature (*Sig, Ver*) were implemented using SHA-256, RSA-1024, and DSA with 1024 bits key size, respectively. With these parameters fixed, we measure the latency of serving IoT resource access request for different types of ABAC policies in different IoT groups configured with different endorsement policies as stated in Table 1. As a baseline comparison, the latency of serving IoT resource is shown when no access control mechanism is in place. We observe that latency is the lowest (.4 sec.) when there is no access control mechanism in place. On the other hand, group 3 has much higher latency compared to group 1 and 2. The reason is that the endorsement policy of group 3 requires all 11 peers to endorse a transaction while group 1 and 2 requires only 1 and 3, respectively. The latency also increases with the increase in required number of attributes to satisfy ABAC policy. This is because ABAC policy evaluation algorithm is NP-complete and therefore, the complexity increases with the number of attributes in the policy.

Public blockchain based IoT access control schemes are still limited by the very high transaction latency. For example, it may take several hours before a bitcoin transaction is committed in the blockchain. So, it is not suitable for any IoT access control scenario requiring low transaction latency. For instance, medical emergency service requires quick

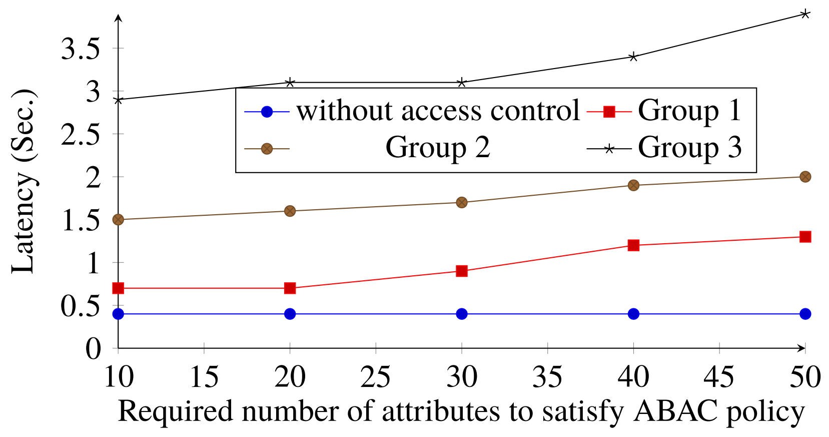


Figure 6. Latency of serving IoT resource access request

access to the wearable device data of elderly people. Our permissioned blockchain based scheme can serve IoT resource access request much faster (around 4 sec.) by utilizing the low transaction latency in Hyperledger Fabric.

8. CONCLUSION AND FUTURE WORK

Any public blockchain based IoT access control system inherits the shortcomings of a public blockchain. On the other hand, permissioned blockchain can overcome these limitations with a very small number of peers whose identities are verified but are not necessarily trusted to each other. This motivates us to design and implement a permissioned blockchain-based access control system for IoT in Hyperledger Fabric. By using attribute based access control (ABAC) as our access control model, we can provide fine-grained access control while IoT devices share resource (data) with external parties. By running experiments in an IoT testbed, we have fine-tuned our blockchain network for access control by finding the optimum parameter values for our network (block timeout = 1s, block size = 20 at 40 transactions per second). Using the optimum parameter values, we show that our access control system can serve access request of IoT resources much faster than the public blockchain. The ABAC policy evaluation algorithm we have used is NP-complete. Therefore, in our future work, we plan to reduce the latency even more by optimizing this algorithm.

REFERENCES

[1] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "Oauth-iot: An access control framework for the internet of things based on open standards," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 676–681, IEEE, 2017.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2019.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[4] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, pp. 5943–5964, 2016.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, D. De Caro, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, p. 30, ACM, 2018.

[6] S. Rouhani, V. Pourheidari, and R. Deters, "Physical access control management system based on permissioned blockchain," *arXiv preprint arXiv:1901.09873*, 2019.

[7] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 523–533, Springer, 2017.

[8] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[9] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 618–623, IEEE, 2017.

[10] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *The second international conference on Internet-of-Things design and implementation*, pp. 173–178, ACM, 2017.

[11] A. Dorri, S. S. Kanhere, and R. Jurdak, “Blockchain in internet of things: challenges and solutions,” *arXiv preprint arXiv:1608.05187*, 2016.

[12] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, “Standardized protocol stack for the internet of (important) things,” *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[13] S. Farrell, R. Housley, and S. Turner, “An internet attribute certificate profile for authorization,” tech. rep., 2010.

[14] “Telosb motes, 2017.” http://www.memsic.com/info/aceinna-landing.cfm?nu=/wireless-sensor-networks.

[15] “Hyperledger fabric v1.3.” https://hyperledger-fabric.readthedocs.io/en/release-1.3/whatsnew.html.

[16] P. Thakkar, S. Nathan, and B. Viswanathan, “Performance benchmarking and optimizing hyperledger fabric blockchain platform,” in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 264–276, IEEE, 2018.

SECTION

3. CONCLUSION AND FUTURE WORK

In this research we have proposed three novel attribute-based encryption schemes for secure data sharing in the cloud. All three schemes support efficient revocation while being resilient against both *type I* and *type II* collusion attacks. The scheme presented in Paper I does not require any trusted entity to achieve revocation, and the revocation does not affect the secret keys of the non-revoked user. As a result, there is no key update cost associated with the non-revoked users. However, this scheme only supports user-level revocation which is equivalent to revoking a particular user's all attributes at the same time. It is not possible to revoke some attribute(s) from a user, while keeping the others intact. In Paper II, we overcome this issue by proposing an attribute-based encryption scheme that supports revocation at the attribute-level. In order to achieve attribute-level revocation, some additional components are created in both secret key and ciphertext that not only increases the ciphertext and secret key size, but also adds additional computational overhead. So, this scheme should be used only if there is a necessity of revoking selective attribute(s) from the user. Otherwise, the scheme proposed in Paper I would perform better. The scheme proposed in Paper III is more appropriate for secure data sharing in a multi-group setting where a group can split into multiple groups or multiple groups can combine together to form a larger group. The group members can move between different groups and one group can collaborate with another group.

In order to evaluate the performance of our schemes, we have implemented them in a desktop environment. It will be interesting to see how our schemes perform in a mobile environment such as android or iOS. In our research we have only considered static access policy where it is not possible to change the policy without completely re-encrypting the data. So, enabling dynamic policy can be a good future extension to our work. Researchers believe that most of the public key cryptography schemes are vulnerable to adversaries equipped with the capability of a quantum computer. Hence, researchers are trying to develop post-quantum variants of different cryptographic schemes. We think developing quantum-safe variants of our schemes would be an interesting research direction.

For secure data sharing in IoT, we propose a permissioned blockchain-based access control system. The system provides a secure and convenient way for the data owner to share IoT data with anyone. The data owner uploads the data sharing policy in the blockchain. In order to access the data, one needs to send a request to the blockchain. The request is evaluated by the blockchain nodes against the policy uploaded by the data owner. The access permission is granted only if the blockchain nodes reach to a consensus about granting permission. It not only ensures proper access control but also ensures better transparency as all requests are recorded in the blockchain and can be verified later on. This helps in resolving any kind of dispute that may arise later on. In our work, we have only considered the access control and security aspect of data sharing. It will be interest to also consider the economic aspect of it. For example, the data owner can share data with third party in exchange for some financial incentive. The data owner can charge less for sharing approximate location data while charge more for sharing more accurate location data. Since our scheme supports fine-grained access control, the data owner can create two types of policies and upload in the blockchain. While our current system takes care of the access control part, further research is required to take care of the various financial aspect such as payment, refund, any kind of dispute resolution over payment.

REFERENCES

[1] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage.," in *NDSS*, vol. 3, pp. 131–145, 2003.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual International Cryptology Conference*, pp. 213–229, Springer, 2001.

[4] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 60–96, 2004.

[5] X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: secure multi-owner data sharing for dynamic groups in the cloud," *ieee transactions on parallel and distributed systems*, vol. 24, no. 6, pp. 1182–1191, 2013.

[6] J. Kim, W. Susilo, M. H. Au, and J. Seberry, "Adaptively secure identity-based broadcast encryption with a constant-sized ciphertext," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 3, pp. 679–693, 2015.

[7] K. Xue and P. Hong, "A dynamic secure group sharing framework in public cloud computing," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 459–470, 2014.

[8] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, 2005.

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

[10] N. Attrapadung, B. Libert, and E. De Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *International Workshop on Public Key Cryptography*, pp. 90–108, Springer, 2011.

[11] J. Han, W. Susilo, Y. Mu, and J. Yan, "Privacy-preserving decentralized key-policy attribute-based encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2150–2162, 2012.

[12] C.-J. Wang and J.-F. Luo, "A key-policy attribute-based encryption scheme with constant size ciphertext," in *2012 Eighth International Conference on Computational Intelligence and Security*, pp. 447–451, IEEE, 2012.

[13] J. Lai, R. H. Deng, Y. Li, and J. Weng, "Fully secure key-policy attribute-based encryption with constant-size ciphertexts and fast decryption," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 239–248, 2014.

[14] Q. Li, J. Ma, R. Li, J. Xiong, and X. Liu, "Large universe decentralized key-policy attribute-based encryption," *Security and communication Networks*, vol. 8, no. 3, pp. 501–509, 2015.

[15] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007.

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

[17] K. Emura, A. Miyaji, A. Nomura, K. Omote, and M. Soshi, "A ciphertext-policy attribute-based encryption scheme with constant ciphertext length," in *International Conference on Information Security Practice and Experience*, pp. 13–23, Springer, 2009.

[18] J. Li, Q. Huang, X. Chen, S. S. Chow, D. S. Wong, and D. Xie, "Multi-authority ciphertext-policy attribute-based encryption with accountability," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pp. 386–390, ACM, 2011.

[19] L. Ibraimi, Q. Tang, P. Hartel, and W. Jonker, "Efficient and provable secure ciphertext-policy attribute-based encryption schemes," in *International Conference on Information Security Practice and Experience*, pp. 1–12, Springer, 2009.

[20] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Infocom, 2010 proceedings IEEE*, pp. 1–9, Ieee, 2010.

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

[22] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355–370, 2014.

[23] S. Ruj, M. Stojmenovic, and A. Nayak, "Decentralized access control with anonymous authentication of data stored in clouds," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 2, pp. 384–394, 2014.

[24] C.-C. Lee, P.-S. Chung, and M.-S. Hwang, “A survey on attribute-based encryption schemes of access control in cloud environments.,” *IJ Network Security*, vol. 15, no. 4, pp. 231–240, 2013.

[25] K. Liang, M. H. Au, J. K. Liu, W. Susilo, D. S. Wong, G. Yang, Y. Yu, and A. Yang, “A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing,” *Future Generation Computer Systems*, vol. 52, pp. 95–108, 2015.

[26] A. Sahai, H. Seyalioglu, and B. Waters, “Dynamic credentials and ciphertext delegation for attribute-based encryption,” in *Advances in Cryptology–CRYPTO 2012*, pp. 199–217, Springer, 2012.

[27] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K.-K. R. Choo, “Cryptcloud+: secure and expressive data access control for cloud storage,” *IEEE Transactions on Services Computing*, 2018.

[28] S. Jahid and N. Borisov, “Piratte: Proxy-based immediate revocation of attribute-based encryption,” *arXiv:1208.4877*, 2012.

[29] J. Hur, “Improving security and efficiency in attribute-based data sharing,” *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2013.

[30] C. Lyu, S.-F. Sun, Y. Zhang, A. Pande, H. Lu, and D. Gu, “Privacy-preserving data sharing scheme over cloud for social applications,” *Journal of Network and Computer Applications*, vol. 74, 2016.

[31] H. Wang, Z. Zheng, L. Wu, and P. Li, “New directly revocable attribute-based encryption scheme and its application in cloud storage environment,” *Cluster Computing*, vol. 20, no. 3, pp. 2385–2392, 2017.

[32] G. Yu, X. Ma, Z. Cao, W. Zhu, and G. Zeng, “Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation,” in *International Conference on Information and Communications Security*, pp. 172–179, Springer, 2017.

[33] R. Zhang, L. Hui, S. Yiu, X. Yu, Z. Liu, and Z. L. Jiang, “A traceable outsourcing cp-abe scheme with attribute revocation,” in *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pp. 363–370, IEEE, 2017.

[34] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, “Flexible and fine-grained attribute-based data storage in cloud computing,” *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2018.

[35] J. Hur and D. K. Noh, “Attribute-based access control with efficient revocation in data outsourcing systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, 2011.

[36] J. Hur, “Improving security and efficiency in attribute-based data sharing,” *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2011.

[37] X. Wang and J. Fang, “A revocable outsourcing attribute-based encryption scheme,” in *Cloud Computing, Security, Privacy in New Computing Environments: 7th International Conference, CloudComp 2016, and First International Conference, SPNCE 2016, Guangzhou, China, November 25–26, and December 15–16, 2016, Proceedings*, vol. 197, p. 145, Springer, 2017.

[38] J. Li, W. Yao, J. Han, Y. Zhang, and J. Shen, “User collusion avoidance cp-abe with efficient attribute revocation for cloud storage,” *IEEE Systems Journal*, vol. 12, no. 2, pp. 1767–1777, 2018.

[39] H. Deng, Z. Qin, Q. Wu, Z. Guan, and Y. Zhou, “Flexible attribute-based proxy re-encryption for efficient data sharing,” *Information Sciences*, vol. 511, pp. 94–113, 2020.

[40] X. Tao, C. Lin, Q. Zhou, Y. Wang, K. Liang, and Y. Li, “Secure and efficient access of personal health record: a group-oriented ciphertext-policy attribute-based encryption,” *Journal of the Chinese Institute of Engineers*, vol. 42, no. 1, pp. 80–86, 2019.

[41] A. Michalas, “The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing,” in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 146–155, 2019.

[42] H. Xiong, H. Zhang, and J. Sun, “Attribute-based privacy-preserving data sharing for dynamic groups in cloud computing,” *IEEE Systems Journal*, vol. 13, no. 3, pp. 2739–2750, 2018.

[43] J. Shen, D. Liu, J. Shen, Q. Liu, and X. Sun, “A secure cloud-assisted urban data sharing framework for ubiquitous-cities,” *Pervasive and mobile Computing*, vol. 41, pp. 219–230, 2017.

[44] B. Familiar, “Iot and microservices,” in *Microservices, IoT, and Azure*, pp. 133–163, Springer, 2015.

[45] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, “Internet of things security: A survey,” *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, 2017.

[46] W. H. Hassan *et al.*, “Current research on internet of things (iot) security: A survey,” *Computer Networks*, vol. 148, pp. 283–294, 2019.

[47] M. A. Khan and K. Salah, “Iot security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.

[48] J. L. H. Ramos, J. B. Bernabé, and A. F. Skarmeta, “Towards privacy-preserving data sharing in smart environments,” in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 334–339, IEEE, 2014.

[49] I. Chiuchisan, I. Chiuchisan, and M. Dimian, “Internet of things for e-health: An approach to medical applications,” in *2015 International Workshop on Computational Intelligence for Multimedia Understanding (IWCIM)*, pp. 1–5, IEEE, 2015.

[50] S. Pérez, D. Rotondi, D. Pedone, L. Straniero, M. J. Núñez, and F. Gigante, “Towards the cp-abe application for privacy-preserving secure data sharing in iot contexts,” in *International conference on innovative mobile and internet services in ubiquitous computing*, pp. 917–926, Springer, 2017.

[51] H. Tao, M. Z. A. Bhuiyan, A. N. Abdalla, M. M. Hassan, J. M. Zain, and T. Hayajneh, “Secured data collection with hardware-based ciphers for iot-based healthcare,” *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 410–420, 2018.

[52] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2019.

[53] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[54] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, D. De Caro, C. Ferris, G. Laventman, Y. Manevich, *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*, p. 30, ACM, 2018.

[55] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, “Ripple: Overview and outlook,” in *International Conference on Trust and Trustworthy Computing*, pp. 163–180, Springer, 2015.

[56] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, “Fairaccess: a new blockchain-based access control framework for the internet of things,” *Security and Communication Networks*, vol. 9, pp. 5943–5964, 2016.

[57] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[58] D. Chen and H. Zhao, “Data security and privacy protection issues in cloud computing,” in *2012 International Conference on Computer Science and Electronics Engineering*, vol. 1, pp. 647–651, IEEE, 2012.

[59] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, “Plutus: Scalable secure file sharing on untrusted storage.,” in *Fast*, vol. 3, pp. 29–42, 2003.

[60] D. Naor, A. Shenhav, and A. Wool, "Toward securing untrusted storage without public-key operations," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, pp. 51–56, 2005.

[61] G. Zhao, C. Rong, J. Li, F. Zhang, and Y. Tang, "Trusted data sharing over untrusted cloud storage providers," in *2nd IEEE International Conference on Cloud Computing Technology and Science*, pp. 97–103, IEEE, 2010.

[62] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 127–144, Springer, 1998.

[63] Z. Qin, H. Xiong, S. Wu, and J. Batamuliza, "A survey of proxy re-encryption for secure data sharing in cloud computing," *IEEE Transactions on Services Computing*, 2016.

[64] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[65] M. Jakobsson, "On quorum controlled asymmetric proxy re-encryption," in *International Workshop on Public Key Cryptography*, pp. 112–121, Springer, 1999.

[66] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.

[67] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 185–194, 2007.

[68] R. H. Deng, J. Weng, S. Liu, and K. Chen, "Chosen-ciphertext secure proxy re-encryption without pairings," in *International Conference on Cryptology and Network Security*, pp. 1–17, Springer, 2008.

[69] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *International Conference on Applied Cryptography and Network Security*, pp. 288–306, Springer, 2007.

[70] H. Wang, Z. Cao, and L. Wang, "Multi-use and unidirectional identity-based proxy re-encryption schemes," *Information Sciences*, vol. 180, no. 20, pp. 4042–4059, 2010.

[71] K. Emura, A. Miyaji, and K. Omote, "An identity-based proxy re-encryption scheme with source hiding property, and its application to a mailing-list system," in *European Public Key Infrastructure Workshop*, pp. 77–92, Springer, 2010.

[72] J. Shao, G. Wei, Y. Ling, and M. Xie, "Identity-based conditional proxy re-encryption," in *2011 IEEE International Conference on Communications (ICC)*, pp. 1–5, IEEE, 2011.

[73] J. Shao, "Anonymous id-based proxy re-encryption," in *Australasian Conference on Information Security and Privacy*, pp. 364–375, Springer, 2012.

[74] J. Shao and Z. Cao, "Multi-use unidirectional identity-based proxy re-encryption from hierarchical identity-based encryption," *Information Sciences*, vol. 206, pp. 83–95, 2012.

[75] C. Sur, C. D. Jung, Y. Park, and K. H. Rhee, "Chosen-ciphertext secure certificateless proxy re-encryption," in *IFIP International Conference on Communications and Multimedia Security*, pp. 214–232, Springer, 2010.

[76] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *International conference on the theory and application of cryptology and information security*, pp. 452–473, Springer, 2003.

[77] L. Xu, X. Wu, and X. Zhang, "Cl-pre: a certificateless proxy re-encryption scheme for secure data sharing with public cloud," in *Proceedings of the 7th ACM symposium on information, computer and communications security*, pp. 87–88, 2012.

[78] H. Guo, Z. Zhang, J. Zhang, and C. Chen, "Towards a secure certificateless proxy re-encryption scheme," in *International Conference on Provable Security*, pp. 330–346, Springer, 2013.

[79] Y. Lu and J. Li, "A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds," *Future Generation Computer Systems*, vol. 62, pp. 140–147, 2016.

[80] M. Chase, "Multi-authority attribute based encryption," in *Theory of Cryptography Conference*, pp. 515–534, Springer, 2007.

[81] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 121–130, ACM, 2009.

[82] H. Lin, Z. Cao, X. Liang, and J. Shao, "Secure threshold multi authority attribute based encryption without a central authority," in *International Conference on Cryptology in India*, pp. 426–436, Springer, 2008.

[83] H. Lin, Z. Cao, X. Liang, and J. Shao, "Secure threshold multi authority attribute based encryption without a central authority," *Information Sciences*, vol. 180, no. 13, pp. 2618–2632, 2010.

[84] Y. Rouselakis and B. Waters, "Efficient statically-secure large-universe multi-authority attribute-based encryption," in *International Conference on Financial Cryptography and Data Security*, pp. 315–332, Springer, 2015.

[85] H. Cui, R. H. Deng, Y. Li, and B. Qin, "Server-aided revocable attribute-based encryption," in *European Symposium on Research in Computer Security*, pp. 570–587, Springer, 2016.

[86] B. Qin, Q. Zhao, D. Zheng, and H. Cui, "(dual) server-aided revocable attribute-based encryption with decryption key exposure resistance," *Information Sciences*, vol. 490, pp. 74–92, 2019.

[87] Y. Shi, Q. Zheng, J. Liu, and Z. Han, "Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation," *Information Sciences*, vol. 295, pp. 221–231, 2015.

[88] J. Li, Y. Shi, and Y. Zhang, "Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage," *International Journal of Communication Systems*, vol. 30, no. 1, p. e2942, 2017.

[89] M. Green, S. Hohenberger, B. Waters, *et al.*, "Outsourcing the decryption of abe ciphertexts.," in *USENIX Security Symposium*, vol. 2011, 2011.

[90] B. Qin, R. H. Deng, S. Liu, and S. Ma, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1384–1393, 2015.

[91] J. K. Liu, M. H. Au, X. Huang, R. Lu, and J. Li, "Fine-grained two-factor access control for web-based cloud computing services," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 484–497, 2016.

[92] J. Li, X. Lin, Y. Zhang, and J. Han, "Ksf-oabe: outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, 2017.

[93] J. Li, Y. Wang, Y. Zhang, and J. Han, "Full verifiability for outsourced decryption in attribute based encryption," *IEEE Transactions on Services Computing*, 2017.

[94] J. Li, F. Sha, Y. Zhang, X. Huang, and J. Shen, "Verifiable outsourced decryption of attribute-based encryption with constant ciphertext length," *Security and Communication Networks*, vol. 2017, 2017.

[95] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 21–30, 2002.

[96] W. Mostowski and P. Vullers, "Efficient u-prove implementation for anonymous credentials on smart cards," in *International Conference on Security and Privacy in Communication Systems*, pp. 243–260, Springer, 2011.

[97] Y. Yang, X. Zheng, and C. Tang, "Lightweight distributed secure data management system for health internet of things," *Journal of Network and Computer Applications*, vol. 89, pp. 26–37, 2017.

[98] H. Shafagh, A. Hithnawi, L. Burkhalter, P. Fischli, and S. Duquennoy, "Secure sharing of partially homomorphic encrypted iot data," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pp. 1–14, 2017.

[99] W. Wang, P. Xu, and L. T. Yang, "Secure data collection, storage and access in cloud-assisted iot," *IEEE cloud computing*, vol. 5, no. 4, pp. 77–88, 2018.

[100] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proceedings of the 2017 on Cloud Computing Security Workshop*, pp. 45–50, 2017.

[101] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, "Integrating blockchain for data sharing and collaboration in mobile healthcare applications," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–5, IEEE, 2017.

[102] A. Manzoor, M. Liyanage, A. Braeke, S. S. Kanhere, and M. Ylianttila, "Blockchain based proxy re-encryption scheme for secure iot data sharing," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 99–103, IEEE, 2019.

[103] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K.-C. Li, "A secure fabric blockchain-based data transmission technique for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3582–3592, 2019.

[104] Y. Jiang, W. Susilo, Y. Mu, and F. Guo, "Ciphertext-policy attribute-based encryption with key-delegation abuse resistance," in *Australasian Conference on Information Security and Privacy*, pp. 477–494, Springer, 2016.

[105] G. Zeng, "Server-aided directly revocable ciphertext-policy attribute-based encryption with verifiable delegation," in *Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6-8, 2017, Proceedings*, vol. 10631, p. 172, Springer, 2018.

[106] H. Cui, T. Hon Yuen, R. H. Deng, and G. Wang, "Server-aided revocable attribute-based encryption for cloud computing services," *Concurrency and Computation: Practice and Experience*, p. e5680, 2020.

[107] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[108] B. Lynn, "The stanford pairing based crypto library," *Privacy Preservation Scheme for Multicast Communications in Smart Buildings of the Smart Grid*, vol. 324, 2013.

[109] Y. Rouselakis and B. Waters, "Practical constructions and new proof methods for large universe attribute-based encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 463–474, ACM, 2013.

[110] M. A. Islam and S. Madria, "A collusion-resistant revocable attribute-based encryption scheme for secure data sharing in cloud," in *The 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May 11-14, 2020, Melbourne, Victoria, Australia*, pp. 21–30, IEEE/ACM, 2020.

[111] R. R. Al-Dahhan, Q. Shi, G. M. Lee, and K. Kifayat, "Survey on revocation in ciphertext-policy attribute-based encryption," *Sensors*, vol. 19, no. 7, p. 1695, 2019.

[112] Y. Zhang, R. H. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng, "Attribute-based encryption for cloud computing access control: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–41, 2020.

[113] X. Wang, Y. Chi, and Y. Zhang, "Traceable ciphertext policy attribute-based encryption scheme with user revocation for cloud storage," in *2020 International Conference on Computer Engineering and Application (ICCEA)*, pp. 91–95, IEEE, 2020.

[114] D. Han, N. Pan, and K.-C. Li, "A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[115] Y. Yang, J. Liu, Z. Wei, and X. Huang, "Towards revocable fine-grained encryption of cloud data: Reducing trust upon cloud," in *Australasian Conference on Information Security and Privacy*, pp. 127–144, Springer, 2017.

[116] M. Lyu, X. Li, and H. Li, "Efficient, verifiable and privacy preserving decentralized attribute-based encryption for mobile cloud computing," in *Data Science in Cyberspace (DSC), 2017 IEEE Second International Conference on*, pp. 195–204, IEEE, 2017.

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

[118] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "Oauth-iot: An access control framework for the internet of things based on open standards," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 676–681, IEEE, 2017.

[119] S. Rouhani, V. Pourheidari, and R. Deters, "Physical access control management system based on permissioned blockchain," *arXiv preprint arXiv:1901.09873*, 2019.

[120] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 523–533, Springer, 2017.

[121] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[122] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Blockchain for iot security and privacy: The case study of a smart home,” in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 618–623, IEEE, 2017.

[123] A. Dorri, S. S. Kanhere, and R. Jurdak, “Towards an optimized blockchain for iot,” in *The second international conference on Internet-of-Things design and implementation*, pp. 173–178, ACM, 2017.

[124] A. Dorri, S. S. Kanhere, and R. Jurdak, “Blockchain in internet of things: challenges and solutions,” *arXiv preprint arXiv:1608.05187*, 2016.

[125] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, “Standardized protocol stack for the internet of (important) things,” *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[126] S. Farrell, R. Housley, and S. Turner, “An internet attribute certificate profile for authorization,” tech. rep., 2010.

[127] “Telosb motes, 2017.” http://www.memsic.com/info/aceinna-landing.cfm?nu=/wireless-sensor-networks.

[128] “Hyperledger fabric v1.3.” https://hyperledger-fabric.readthedocs.io/en/release-1.3/whatsnew.html.

[129] P. Thakkar, S. Nathan, and B. Viswanathan, “Performance benchmarking and optimizing hyperledger fabric blockchain platform,” in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 264–276, IEEE, 2018.

VITA

MD Azharul Islam was born in Bangladesh, a beautiful small country in the South Asia. He received his bachelor's degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in February, 2012. After completing his bachelor's degree, he worked as a software engineer at REVE Systems and helped in the development of several VOIP software solutions.

He obtained his M.S. and Ph.D. in Computer Science in May, 2021 from Missouri University of Science and Technology under the supervision of Dr. Sanjay Madria. His core research focus was on cryptography, cybersecurity, privacy, and blockchain. More specifically, his research goal was to develop secure protocols to solve real-world security problems such as secure data sharing in the cloud and IoT. During his Ph.D., he published several papers in top-tier conference and journals. He also did a research internship at VERISIGN in the summer of 2018.

Azharul was very passionate about travelling because it is a great way to experience new culture, food, and getting to know new people. He was a big fan of cricket and football. His other hobbies included photography and cooking.