# Server-Aided Revocable Bilateral Attribute-Based Encryption

Shengmin Xu[1], Jianting Ning[2,3], Guowen Xu[4], Xinyi Huang[2], Pawel Szalachowski[1], Jianying Zhou[1], and Robert H. Deng[3]

[1] Singapore University of Technology and Design, Singapore 487372, Singapore
{shengmin_xu,pawel,jianying_zhou}@sutd.edu.sg
[2] Fujian Normal University, Fuzhou 350007, P.R. China
{jtning88,xyhuang81}@gmail.com
[3] Singapore Management University, Singapore 178902, Singapore
robertdeng@smu.edu.sg
[4] University of Electronic Science and Technology of China, Sichuan 611731, P.R. China
guowen.xu@foxmail.com

**Abstract.** Attribute-based encryption has been diffusely deployed in a variety of outsourced computing scenarios to offer secure fine-grained access control. However, how to support efficient user revocation and fetch desirable data without expensive data decryption for resource-constrained end-devices remains a challenging problem. In this paper, we introduce a novel cryptographic primitive called the server-aided revocable bilateral attribute-based encryption (SRB-ABE). Our solution offers a secure and lightweight bilateral access control, including (1) fine-grained data user and data owner access control simultaneously; (2) outsourced data source identification; (3) server-aided user revocation with updatable ciphertexts; and (4) lightweight data decryption mechanism with one exponentiation computation. We give the formal definition and security models of SRB-ABE as well as a concrete construction with security proofs. The extensive comparison and experimental analysis demonstrate that our construction has supervisor functionality and comparable performance than the most relevant solutions.

**Keywords:** Attribute-based access control · server-aided · revocation.

## 1 Introduction

Attribute-based encryption as a novel paradigm has been widely deployed in the various outsourced computing environment (e.g., cloud computing and fog computing) to provide data sharing in terms of usability, scalability, and confidentiality. According to the forecast from Statista [28], North America, as the largest regional fog computing market, has estimated the market value of 1.2 billion U.S. dollars in 2019, and the expectation of North American market to see rapid growth in the coming years with increasing by almost threefold in the next five years. Although ABE has been shown to be an access control mechanism for encrypted data, some limiting factors affect it to practical. For instance, user revocation among a substantial amount of data users,

desirable data identification from a vast amount of ciphertexts without costly data decryption, and lightweight computation for devices constrained with memory, physical size, and battery. Although there are several possible approaches to address the above challenges, these solutions still remain some issues.

One potential approach for offering efficient user revocation is the certification revocation list [12], where a trusted party maintains a blacklist and anyone in that list cannot access any ciphertext. However, this approach requires the cloud server to keep the blacklist up-to-date and also be fully trusted. Besides, such an approach also requires the cloud server to authenticate every data user when they require the remote data and this additional authentication incurs large overheads. Another approach is revocable ABE (R-ABE) [7, 6]. The KGC helps every non-revoked data user to update their secret key periodically and the data users who cannot update their secret keys are implicitly revoked. However, [7] suffers from large costs to the secret key distribution since it requires the KGC maintaining the secure channel to each valid data user to update the secret key periodically, and [6] cannot prevent the revoked user to decrypt the ciphertext generated before being revoked and also suffers from the decryption key exposure attack [31]. Although Cui et al. [10] introduced a solution called server-aided R-ABE (SR-ABE) to achieve efficient user revocation and decryption key exposure resistance simultaneously, the revoked user still can decryption the ciphertext generated before being revoked and the data user cannot identify the desirable data from a vast amount of ciphertexts without expensive data decryption.

For supporting efficient data identification mechanism at a fine-grained level without expensive data decryption, one possible solution is attribute-based encryption with keyword search (ABKS) [37]. However, many literature [8, 23, 36] pointed out that ABKS cannot preserve the data privacy and suffers from a variety of attacks (e.g., leakage attack and file-injection attack). Another solution is matchmaking encryption recently proposed by Ateniese et al. [2] that offers bilateral access control. It enables the data owner to specify the data user to reveal the messages (data confidentiality, data user access control), and also allows the data user to determine whether ciphertexts are from the approved data owners (data source identification, data owner access control). However, no concrete construction of matchmaking encryption at a fine-grained level has been introduced.

For lightweight computation, we want to possess lightweight data decryption, server-aided revocation and outsourced data source identification. Lightweight data decryption was introduced by Green et al. [14], where the major workload of data decryption is processed by a third party and the data user only needs to perform the computation cost as the same as the regular ElGamal encryption scheme. However, [14] does not consider efficient user revocation. To address this problem, Cui et al. [10] proposed an SR-ABE, where a third party helps each non-revoked data user to derive the decryption key periodically and takes the heavy burden of data decryption enabling the data user to enjoy lightweight decryption and revocation simultaneously. Unfortunately, both schemes [14, 10] do not have data source identification and current matchmaking encryption [2] cannot support outsourced data source identification. Hence, it is still challenging to realize lightweight computation in the current fine-grained access control mechanisms.

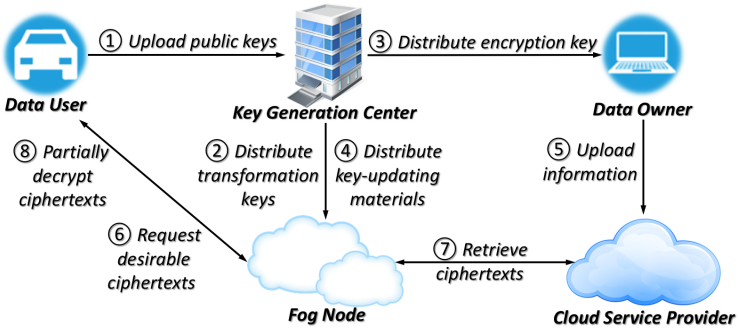


Fig. 1: System Architecture of the SRB-ABE

### 1.1 Our Contributions

To address the above challenges simultaneously, we first introduce a notion called server-aided revocable bilateral attribute-based encryption (SRB-ABE) to accomplish efficient and secure user revocation, data source identification and lightweight computation.

Our SRB-ABE is suitable for most of the outsourced computing scenarios since the remote cloud server need not be fully trusted. Specifically, we demonstrate fog computing [29] as one of the potential applications as shown in Fig. 1 since it extends the usability and functionality of the traditional cloud computing. By empowering fog nodes to carry out a large amount of storage, communication, and computation locally over the network, fog computing brings many advantages that cannot be supported by traditional cloud computing, including geographic distribution, mobility support, location-awareness, and low latency. The typical entities are a KGC, a cloud server provider (CSP), fog nodes, data users and data owners, and the workflow can be categorized into four phases: *system setup*, *user revocation*, *data uploading* and *data retrieving*. Note that due to the CSP and fog nodes are allowed to collude, we can merge the fog node and the CSP into one entity for simulating the traditional cloud computing.

*System setup*: The KGC setups the system via generating the system parameters. When a new data user joins the system, s/he generates a public and private key pair and sends the public key to the KGC (see ①). Based on this public key and the attribute set of this new data user, the KGC generates an attribute-based key, denoted as a transformation key, and sends it to an untrusted fog node (see ②). For supporting data source identification, the KGC also generates and sends the encryption key to each data owner based on their attribute set (see ③).

*User revocation*: The KGC generates key updating materials based on the latest revocation list and sends them to the fog nodes (see ④). Each fog node updates the transformation key based on the latest key updating materials. Notice that some revocable mechanisms [3, 6] operate the key updating process in the data users, and our solution enjoys lightweight computation since the workload of key updating is in fog nodes rather than data users themselves.

*Data uploading*: The data owner uploads the ciphertext to the CSP (see ⑤) by appending the attribute set of the data owner from the encryption key (from ③) and specifying an access structure of data user to offer bilateral access control at a fine-grained level.

*Data retrieving*: The data user requests data by specifying the access structure of data owners to a fog node (see ⑥). The fog node then requests ciphertexts from the CSP (see ⑦) and verifies the coming ciphertexts to offer outsourced data source identification. For each ciphertext passing the verification, the fog node generates the partially decrypted ciphertexts to reduce the workload of data decryption and returns those ciphertexts to the data user (see ⑧).

We give more details about our system architecture in Section 3.2. In a nutshell, our contributions in this paper can be summarized as follows.

- We first propose a notion called server-aided revocable bilateral attribute-based encryption (SRB-ABE), in which almost all data users' workloads including update keys, data decryption, and data source identification are delegated to an untrusted server (e.g. the fog node). Each data user only needs to keep a private key in constant size for decryption and only requires to perform one exponentiation computation to decrypt a ciphertext.
- We define a security model for SRB-ABE which considers all possible adversarial behaviors that could be executed by an adversary in the real world.
- Motivated by R-ABE [26], we apply revocable storage for SRB-ABE which allows ciphertexts publicly updatable periodically to prevent revoked data user decrypting ciphertexts before being revoked as a research gap in previous solutions [7, 5, 3, 6].

### 1.2 Related Work

*ABE*. The rudiment of ABE was introduced by Sahai and Waters [27], which only offers threshold access structure and provable secure in the selective model. After that, many ABE was proposed in two primary flavors: key-policy ABE (KP-ABE) [13, 24, 17, 4, 25, 1] and ciphertext-policy ABE (CP-ABE) [5, 9, 18, 16, 25, 34, 19, 20, 1, 35, 22, 15]. In KP-ABE, the user's private key is associated with an access structure, and ciphertexts are encrypted over an attribute set. The CP-ABE is processed in a dual way, an access structure associates ciphertexts, and the receiver's secret key is specified by an attribute set.

*S-ABE*. S-ABE [14] extends the usability of ABE by outsourcing major workloads of data decryption to a third party. Specifically, each data user generates a secret-public key pair and sends the public key to the KGC for obtaining a transformation key, where the transformation key is outsourced to a third party to process partially data decryption. The partially decrypted ciphertexts still remain secure since the secret keys of the data users are unknown to any party except the users themselves.

*R-ABE*. The approach of R-ABE can be categorized into two types: direct revocation [7, 5, 3] and indirect revocation [26, 32]. There are two strategies in direct revocation. One is the KGC broadcasting an updated secret key for each non-revoked users in which the secret key is representing as the attribute $att$ appending a timestamp $att\|t$ and $t$ is updated in each revocation epoch via secret channels, which is impractical since the

secret channel is expensive. The other one is embedding all the identities of the revoked user in the ciphertexts, which is also impractical since the data owners must keep the revocation list update to date. To address the above problems, indirect revocation was proposed, which divides a decryption key into a secret key and a public key-updating material. In each revocation epoch, the KGC broadcasts the key updating material via the public channel, and the non-revoked users update the corresponding secret key to be the decryption key in the new revocation epoch. To improve usability, Cui et al. [10] presented SR-ABE by combing the S-ABE and indirect R-ABE together, which allows the third party to generate the decryption periodically and partial ciphertext decryption simultaneously.

*ABKS.* ABKS was first introduced by Zheng et al. [37] for fetching ciphertext with specified keywords without expensive data decryption. However, many top-tie works [8, 23, 36] pointed out that many ABKS schemes suffer from passive (e.g., leakage attack) and active attacks (e.g., file-injection attack), where the privacy of searchable pattern is easy to compromise.

*ACE.* Damgard et al. [11] proposed the first access control encryption (ACE), which focus on the *no read up* and *no write down* data regulation. Specifically, this data regulation requires a sanitizer who acts as a fully trusted party to ensure secure data flow. The receiver cannot read the message generated by a user with higher reading rights, and the senders cannot write a message as a user with lower writing rights. Hence, the policy in ACE is suitable to the system with a hierarchical access policy rather than the access control in the cloud-fog architecture.

*Matchmaking encryption.* Matchmaking encryption as a novel paradigm protects the data communications with bilateral access control user privacy proposed by Ateniese et al. [2]. The current concrete construction of matchmaking encryption is in an identity-based setting, which cannot provide fine-grained access control.

### 1.3 Organization

Section 2 presents preliminaries for our proposed scheme. Section 3 defines the SRB-ABE scheme and the system architecture of the cloud-fog system. The concrete construction of SRB-ABE, and its security and performance analysis are given in Section 4. Section 5 concludes this paper.

## 2 Preliminaries

### 2.1 Bilinear Map

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic multiplicative groups of prime order $p$ and $g$ be a generator of $\mathbb{G}$. The map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is said to be an admissible bilinear pairing if following properties hold.

1. Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneration: $e(g, g) \neq 1$.
3. Computability: it is efficient to compute $e(u, v)$ for any $u, v \in \mathbb{G}$.

We say that $(\mathbb{G}, \mathbb{G}_T)$ are bilinear map groups if there exists a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ as above.

### 2.2 Linear Secret Sharing Scheme

We recall the definition of linear secret sharing scheme (LSSS) [31]. A LSSS policy is of the type $(\mathbb{M}, \rho)$, where $\mathbb{M}$ is a $\ell \times n$ matrix over the base field $\mathbb{F}$ and $\rho$ is a mapping function from the set $[\ell]$ to an attribute universe, where $[\ell]$ denotes a set from 1 to $]\ell$. A policy $(\mathbb{M}, \rho)$ satisfies an attribute set $\psi$ if $(1, 0, ..., 0) \in \mathbb{F}^n$ is contained in $\mathsf{Span}_{\mathbb{F}}(\mathbb{M}_i : \rho(i) \in \psi)$, where $\mathbb{M}_i$ is the $i^{th}$ row of $\mathbb{M}$. For simplicity, we use $\mathbb{F} = \mathbb{Z}_p$ for the rest of the paper.

### 2.3 Tree-Based Revocation Approach

The tree-based data structure has been widely used to reduce the computational cost of generating and transmitting key updating materials from linear to logarithmic. To revoke a user, the subset-cover algorithm $\mathsf{KUNode}(st, rl, t)$ [21] is proposed, where $st$ is the state representing the tree-based data structure, $rl$ is the revocation list recording identities of revoked users, and $t$ is the timestamp representing the revocation epoch. When a user joins the system, who will be assigned a random identifier $id$ and an undefined leaf node in $st$ will be labeled this $id$. The revocation method only requires the user $id$ to store the keys in $\mathsf{Path}(id)$, where $\mathsf{Path}(id)$ denotes nodes in the path from the root node to the leaf node $id$. Although this method sacrifices the length of the secret key from constant to logarithmic, the costs still significantly reduced since the secret key only distributes once, and key updating materials are broadcasted in each revocation epoch. More importantly, by applying our proposed SRB-ABE, the key is allowed to distribute via the public channel. We suggest readers to the literature [21] for the details of the subset-cover algorithm.

### 2.4 Updatable Time Structure

Xu et al. [33] proposed a time encoding algorithm $\mathsf{TEncode}(t, \mathcal{T})$ to shorten the ciphertext update, where $t$ is the timestamp when data encryption and $\mathcal{T}$ is the system bounded lifetime. In their scheme, the encryption algorithm takes the set $\mathcal{V}$ recording the all 0-bit of the timestamp instead of the user identity, and the ciphertext update algorithm is to updates the timestamp $t$ to the timestamp $t' \geq t$. Specifically, for updating the timestamp $t$ to $t'$, the ciphertext update algorithm allows the 0-bit in timestamp $t$ to be 1-bit in timestamp $t'$, while 1-bit in $t$ cannot change to be 0-bit in $t'$. We suggest readers to the literature [33] for the details of the time encoding algorithm.

## 3 Formal Definition and System Model

### 3.1 Formal Definition

**Definition 1 (SRB-ABE).** *An SRB-ABE scheme consists of five types of entities: a KGC, untrusted servers (e.g., fog nodes), a cloud service provider (e.g., cloud), data senders and data receivers, and has following algorithms:*

$\mathsf{Setup}(1^\lambda, \mathcal{N}, \mathcal{T}) \rightarrow (mpk, msk, st, rl)$*: The probabilistic setup algorithm is run by the KGC. This algorithm takes a security parameter* $\lambda \in \mathbb{N}$*, a number of system users* $\mathcal{N}$ *and*

*a bounded system lifetime $\mathcal{T}$ as input, and outputs a master public key $mpk$, a master secret key $msk$, a state $st$ and a revocation list $rl$. We implicitly assume that all other algorithms take $mpk$ as input.*

$\mathsf{KeyGen}(id) \rightarrow (pk_{id}, sk_{id})$*: The probabilistic key generation algorithm is run by each data receiver. This algorithm takes an identity $id$ as input, and outputs a public key $pk_{id}$ and a secret key $sk_{id}$.*

$\mathsf{TKGen}(msk, st, pk_{id}, \mathcal{R}) \rightarrow (tk_{id}, st)$*: The probabilistic transformation key generation algorithm is run by the KGC. This algorithm takes a master secret key $msk$, a state $st$, a public key $pk_{id}$ and an attribute set of receivers $\mathcal{R}$ as input, and outputs a transformation key $tk_{id}$ and an updated state $st$.*

$\mathsf{KUGen}(st, rl, t) \rightarrow ku_t$*: The probabilistic key updating material generation algorithm is run by the KGC. This algorithm takes a state $st$, a revocation list $rl$ and a timestamp $t$ as input, and outputs a key updating material $ku_t$.*

$\mathsf{TKUpdate}(tk_{id}, ku_t) \rightarrow utk_{id,t}/\perp$*: The probabilistic transformation key updating algorithm is run by untrusted servers. This algorithm takes a transformation key $tk_{id}$ and a key updating material $ku_t$ as input, and outputs an updated transformation key $utk_{id,t}$ or a failure symbol $\perp$ if the data receiver is revoked.*

$\mathsf{EKGen}(msk, \mathcal{S}) \rightarrow ek_{\mathcal{S}}$*: The probabilistic encryption key generation algorithm is run by the KGC. The algorithm takes a master secret key $msk$ and an attribute set of a sender $\mathcal{S}$ as input, and outputs an encryption key $ek_{\mathcal{S}}$.*

$\mathsf{Enc}(ek_{\mathcal{S}}, \hat{\mathcal{S}}, t, \mathbb{R}, m) \rightarrow c$*: The probabilistic encryption algorithm is run by each data sender. This algorithm takes an encryption key $ek_{\mathcal{S}}$, an attribute set of a sender $\hat{\mathcal{S}}$, a timestamp $t$, an access structure of a receiver $\mathbb{R}$ and a message $m$ as input, and outputs a ciphertext $c$.*

$\mathsf{CTUpdate}(c, t) \rightarrow c_t$*: The deterministic ciphertext update algorithm is run by the cloud service provider. This algorithm takes a ciphertext $c$ and a timestamp $t$ as input, and outputs an updated ciphertext $c_t$.*

$\mathsf{Verify}(\mathbb{S}, c_t) \rightarrow \{0, 1\}$*: The deterministic verification algorithm is run by untrusted servers. This algorithm takes an access structure of a sender $\mathbb{S}$ and a timestamp $t$ as input, and outputs $0$ if and only if $\mathcal{S} \models \mathbb{S}$; otherwise it outputs $0$.*

$\mathsf{Transfer}(utk_{id,t}, c_t) \rightarrow \widehat{c}$*: The deterministic transformation algorithm is run by untrusted servers. This algorithm takes an updated transformation key $utk_{id,t}$,, a ciphertext $c$ and a timestamp $t$ as input, and outputs a transformed ciphertext $\widehat{c}$.*

$\mathsf{Dec}(sk_{id}, \widehat{c}) \rightarrow m$*: The deterministic decryption algorithm is run by each data receiver. This algorithm takes a secret key $sk_{id}$ and a transformed ciphertext $\widehat{c}$ as input, and outputs a message $m$.*

$\mathsf{Rev}(rl, id, t) \rightarrow rl$*: The deterministic revocation algorithm is run by the KGC. This algorithm takes a revocation list $rl$, a identity $id$ and a timestamp $t$ as input, and outputs an updated revocation list $rl$.*

### 3.2 System Architecture

Recall the definition of SRB-ABE in Section 3.1 and reference to the system model in Fig. 1, the typical entities are a key generation center (KGC), a cloud server provider

(CSP), fog nodes, data users and data owners, and the workflow can be categorized into four phases: *system setup*, *user revocation*, *data uploading* and *data retrieving*.

*System setup*: The KGC runs the setup algorithm $\mathsf{Setup}(1^\lambda, \mathcal{N}, \mathcal{T})$ to generate the system parameters, including a master public key $mpk$, a master secret key $msk$, a state $st$ and a revocation list $rl$, and then initialize data users and data owners.

- Data user initialization: After receiving the master public key $mpk$, the data user runs the key generation algorithm $\mathsf{KeyGen}(id)$ to generate a key pair $(pk_{id}, sk_{id})$ and then outsources $pk_{id}$ to the KGC via a public channel (see ①). The KGC runs the transformation key generation algorithm $\mathsf{TKGen}(msk, st, pk_{id}, \mathcal{R})$ to generate a transformation key $tk_{id}$ and an updated state $st$. The KGC then keeps $st$ secret and sends $tk_{id}$ to fog nodes (see ②).
- Data owner initialization: The KGC runs the encryption key generation algorithm $\mathsf{EKGen}(msk, \mathcal{S})$ to derive the encryption key $ek_{\mathcal{S}}$ and sends $ek_{\mathcal{S}}$ to the data owner (see ③).

*User revocation*: The KGC updates the revocation list $rl$ by running the revocation algorithm $\mathsf{Rev}(rl, id, t)$ and runs the key updating material generation algorithm $\mathsf{KUGen}(st, rl, t)$ to generate the key updating material $ku_t$. The KGC then sends $ku_t$ to fog nodes (see ④). After receiving $ku_t$, each fog node runs the transformation key updating algorithm $\mathsf{TKUpdate}(tk_{id}, ku_t)$ to generate the updated transformation key $utk_{id,t}$ for each data user.

*Data uploading*: To share a message $m$ via the CSP, the data owner runs the encryption algorithm $\mathsf{Enc}(ek_{\mathcal{S}}, \hat{\mathcal{S}}, t, \mathbb{R}, m)$ to generate the ciphertext $c$ and outsources $c$ to the CSP (see ⑤), where $\hat{\mathcal{S}}$ is an attribute set, such that $\hat{\mathcal{S}} \subseteq \mathcal{S}$, indicating the attributes of data owners and $\mathbb{R}$ is an access structure specifying the data users.

*Data retrieving*: The data user sends the ciphertext requests by specifying the access structure of data owners $\mathbb{S}$ to a fog node (see ⑥). The fog node then requests ciphertexts from the CSP. The CSP runs the ciphertext update algorithm $\mathsf{CTUpdate}(c, t)$ to update the ciphertext under current timestamp $c_t$ and returns $c_t$ to the fog node (see ⑦). After receiving $c_t$, the fog node runs the verification algorithm $\mathsf{Verify}(\mathbb{S}, c_t)$ to find the ciphertexts such that the verification algorithm returns 1. For each ciphertext passing the verification, the fog node runs the transformation algorithm $\mathsf{Transfer}(utk_{id,t}, c_t)$ to generate the partially decrypted ciphertext $\widehat{c}$ and returns $\widehat{c}$ to the data user (see ⑧).

### 3.3 Threat Model & Security Model

We assume the KGC is a fully trusted entity. The CSP and fog nodes are semi-trusted, who faithfully carries out system operations but may launch any passive attacks. Data users and data owners are untrustworthy, who can launch any attacks. Data users may try to decrypt any unauthorized ciphertexts. Data owners may try to pretend any unauthorized person from generating messages to others. The malicious data users and data owners can get together to launch collusion attacks. For example, they may combine multiple decryption keys to decrypt unauthorized ciphertexts, and exchange encryption keys to generate the ciphertext without authorized attributes.

To simplicity, we present two security models to capture all the possible attacks in the threat model. There are selectively indistinguishably under chosen plaintext

attacks (sIND-CPA security) and existential unforgeability under chosen message attacks (EU-CMA security) for SRB-ABE between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

**Definition 2** (sIND-CPA **in SRB-ABE).** *The security definition of* sIND-CPA *for SRB-ABE is based on the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.*

***Initialization***. *$\mathcal{A}$ sends an access policy of a receiver $\mathbb{R}^*$ to $\mathcal{C}$.*

***Setup***. *$\mathcal{C}$ runs the setup algorithm to generate a master public key $mpk$, a master secret key $msk$, a state $st$ and a revocation list $rl$. $\mathcal{C}$ sends $mpk$ to $\mathcal{A}$, and keeps $(msk, st, rl)$.*

***Phase 1***. *$\mathcal{C}$ offers a sequence of oracles to allow $\mathcal{A}$ to query adaptively.*

- *Key generation oracle: $\mathcal{A}$ issues key generation queries on an identity $id$ and an attribute set $\mathcal{R}$. To each query, $\mathcal{C}$ returns a public key $pk_{id}$ by running the key generation algorithm* $\mathsf{KeyGen}(id)$*, and records $(id, pk_{id}, sk_{id})$ to a list so that the same $(pk_{id}, sk_{id})$ is used for all queries on $id$.*
- *Corrupt oracle: $\mathcal{A}$ issues corrupt queries on an identity $id$ and an attribute set $\mathcal{R}$. To each query, $\mathcal{C}$ returns a secret key $sk_{id}$ by running the key generation algorithm* $\mathsf{KeyGen}(id)$*, and records $(id, pk_{id}, sk_{id})$ to the list as in the key generation oracle.*
- *Transformation key generation oracle: $\mathcal{A}$ issues transformation key generation queries on an identity $id$ and an attribute set $\mathcal{R}$. To each query, $\mathcal{C}$ returns a transformation key $tk_{id}$ by running the transformation key generation algorithm* $\mathsf{TKGen}(msk, st, pk_{id}, \mathcal{R})$*. If $pk_{id}$ is not available, $\mathcal{C}$ runs the key generation oracle on an identity $id$ and an attribute set $\mathcal{R}$ first.*
- *Key updating materials generation oracle: $\mathcal{A}$ issues key updating materials generation queries on a timestamp $t$. To each query, $\mathcal{C}$ returns a transformation key $ku_t$ by running the key updating material generation algorithm* $\mathsf{KUGen}(st, rl, t)$*. Note that $ku_t$ is unique in each revocation epoch.*
- *Transformation key updating oracle: $\mathcal{A}$ issues transformation key updating queries on an identity $id$, an attribute set $\mathcal{R}$ and a timestamp $t$. To each query, $\mathcal{C}$ returns an updated transformation key $utk_{id,t}$ by running the transformation key updating algorithm* $\mathsf{TKUpdate}(tk_{id}, ku_t)$*. If $tk_{id}$ is not available, $\mathcal{C}$ runs the transformation key generation oracle on an identity $id$ and an attribute set $\mathcal{R}$ first. If $ku_t$ is not available, $\mathcal{C}$ runs the key updating materials generation oracle on a timestamp $t$.*
- *Revocation oracle: $\mathcal{A}$ issues revocation queries on an identity $id$ and a timestamp $t$. To each query, $\mathcal{C}$ updates the revocation list $rl$ by running the revocation algorithm* $\mathsf{Rev}(rl, id, t)$*.*
- *Encryption key generation oracle: $\mathcal{A}$ issues encryption key generation queries on an attribute set $\mathcal{S}$. To each query, $\mathcal{C}$ returns an encryption key $ek_{\mathcal{S}}$ by running the encryption key generation algorithm* $\mathsf{EKGen}(msk, \mathcal{S})$*.*

***Challenge***. *$\mathcal{A}$ outputs two messages $(m_0, m_1)$, two attribute sets $(\mathcal{S}_0, \mathcal{S}_1)$ and a timestamp $t^*$. $\mathcal{C}$ continues the security game if the following restrictions are satisfied.*

1. *There is no overlap between two attribute sets $(\mathcal{S}_0, \mathcal{S}_1)$ such that $\mathcal{S}_0 \cap \mathcal{S}_1 = \emptyset$ to prevent trivial attack where $\mathcal{A}$ finds the sender's attribute set directly.*
2. *The key updating materials generation oracle, the transformation key updating oracle, and the revocation oracle can be queried at the time $t$, which is greater than or equal to that of all previous queries.*

3. *The revocation oracle cannot be queried at the timestamp $t$ if key updating materials generation oracle or transformation key updating oracle was queried at $t$.*
4. *If the corrupt oracle was queried on an identity $id$ with the attribute $\mathcal{R}$ set such that $\mathcal{R} \models \mathbb{R}^*$, then the revocation oracle must be queried on this identity $id$ at the timestamp $t \geq t^*$.*
5. *The transformation key updating oracle cannot be queried on any identity $id$ with the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}^*$ at the challenge timestamp $t^*$ or any identity $id$ has been revoked.*

*$\mathcal{C}$ randomly picks a bit $b \in \{0, 1\}$ and returns the challenge ciphertext $c^*$ to $\mathcal{A}$ by running the encryption algorithm* $\mathsf{Enc}(ek_{\mathcal{S}_b}, \hat{\mathcal{S}}, t^*, \mathbb{R}^*, m_b)$*, where $\mathcal{S}_0 \cap \mathcal{S}_1 = \hat{\mathcal{S}}$ and $ek_{\mathcal{S}_b}$ is from* $\mathsf{EKGen}(msk, \mathcal{S}_b)$.

***Phase 2**. $\mathcal{A}$ continues issuing queries to $\mathcal{C}$ as in Phase 1, following the constraints defined in the Challenge Phase.*

***Guess**. $\mathcal{A}$ outputs a bit $b'$, and it wins the security game if $b = b'$.*

*An SRB-ABE scheme is said to be* sIND-CPA *secure if for any probabilistic polynomial time adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{sIND\text{-}CPA}}_{\mathsf{SRB\text{-}ABE},\mathcal{A}}(1^\lambda, \mathcal{N}, \mathcal{T}) = \Pr[\mathcal{A}\ wins]$ is negligible, where $\lambda$ is a security parameter, $\mathcal{N}$ represents the number of system users and $\mathcal{T}$ denotes the system bounded lifetime.*

**Definition 3 (**EU-CMA **in SRB-ABE).** *The security definition of* EU-CMA *for SRB-ABE is based on the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.*

***Initialization**. $\mathcal{A}$ sends an access policy of a sender $\mathbb{S}^*$ as a challenge information to $\mathcal{C}$.*

***Setup**. The Setup phase is the same as the Setup phase in the* sIND-CPA *game.*

***Query**. $\mathcal{C}$ offers a sequence of oracles to allow $\mathcal{A}$ to query adaptively. The oracles include all the oracles in the* sIND-CPA *game and an additional oracle called encryption oracle, where the procedure of all oracles except encryption oracle are identical to the oracles in the* sIND-CPA *game. The definition of encryption oracle is given below.*

– *Encryption oracle: $\mathcal{A}$ issues encryption queries on an attribute set $\hat{\mathcal{S}}$, a timestamp $t$, an access structure $\mathbb{R}$ and a message $m$. To each query, $\mathcal{C}$ returns a ciphertext $c_t$ by running the encryption algorithm* $\mathsf{Enc}(ek_{\mathcal{S}}, \hat{\mathcal{S}}, t, \mathbb{R}, m)$.

***Forgery**. $\mathcal{A}$ outputs a ciphertext $c^*$ under the attribute set $\mathcal{S}$ such that $\mathcal{S} \models \mathbb{S}$. $\mathcal{A}$ wins the security game if*

1. *Any querying attribute set $\mathcal{S}$ to encryption key generation oracle is unsatisfied the challenge access policy $\mathbb{S}^*$ such that $\mathcal{S} \not\models \mathbb{S}^*$.*
2. *Any returning ciphertext $c_t$ from encryption oracle has no equivalent to the ciphertext $c^*$ such that $c^* \not\equiv c_t$, where equivalent meaning two ciphertext has different values but in same distribution domain (e.g., the same ciphertext with different randomness).*

*An SRB-ABE scheme is said to be* EU-CMA *secure if for any probabilistic polynomial time adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathsf{SRB\text{-}ABE},\mathcal{A}}(1^\lambda, \mathcal{N}, \mathcal{T}) = \Pr[\mathcal{A}\ wins]$ is negligible, where $\lambda$ is a security parameter, $\mathcal{N}$ represents the number of system users and $\mathcal{T}$ denotes the system bounded lifetime.*

# 4 Server-Aided Revocable Bilateral Attribute-Based Encryption

## 4.1 Concrete Construction

$\mathsf{Setup}(1^\lambda, \mathcal{N}, \mathcal{T}) \rightarrow (mpk, msk, st, rl)$: The setup algorithm generates a description of bilinear group $(e, \mathbb{G}, \mathbb{G}_T, g, p)$ according to a bilinear group parameter generator $\mathcal{G}(1^\lambda)$. Let $\ell$ be the bit length of the system bounded lifetime $\mathcal{T}$ such that $\ell = \log_2 \mathcal{T}$. The algorithm randomly chooses terms $\alpha, \beta \in \mathbb{Z}_p$ and $w, v, u, h, u_0, u_1, ..., u_\ell \in \mathbb{G}$. It then picks a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}$ and a binary tree $\mathsf{BT}$ with at least $\mathcal{N}$ leaf nodes. The algorithm returns a master public key $mpk = (g, w, v, u, h, u_0, u_1, ..., u_\ell, e(g,g)^\alpha, e(g,g)^\beta, \mathcal{H})$, a master secret key $msk = (\alpha, \beta)$, a state $st \leftarrow \mathsf{BT}$ and a revocation list $rl \leftarrow \emptyset$.

$\mathsf{KeyGen}(id) \rightarrow (pk_{id}, sk_{id})$: The key generation algorithm picks a random term $\gamma_{id} \in \mathbb{Z}_p$ and returns a key pair $(pk_{id}, sk_{id})$ as $pk_{id} = g^{\gamma_{id}}$ and $sk_{id} = \gamma_{id}$.

$\mathsf{TKGen}(msk, st, pk_{id}, \mathcal{R}) \rightarrow (tk_{id}, st)$: Parse the attribute set of a receiver $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_k)$ and the state $st = \mathsf{BT}$. The transformation key generation algorithm chooses an undefined leaf node from $\mathsf{BT}$, and stores $id$ in this node. For each node $\theta \in \mathsf{Path}(id)$, it retrieves $g_\theta$ if it is available in the node $\theta$. If $\theta$ has not been defined, it randomly picks $g_\theta \in \mathbb{G}$ and updates the node $\theta$ by updating the state $st \leftarrow st \cup (\theta, g_\theta)$. It then randomly chooses $r, r_1, r_2, ..., r_k \in \mathbb{Z}_p$ and computes $g'_\theta = pk_{id}^\alpha / g_\theta$ to derive the transformation key $tk_\theta = (tk_1 = g'_\theta w^r, tk_2 = g^r, \{tk_{3,\tau} = g^{r_\tau}, tk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau} v^{-r}\}_{\tau \in [k]})$. It returns $tk_{id} = ((id, \mathcal{R}), \{tk_\theta\}_{\theta \in \mathsf{Path}(id)})$ and a state $st$.

$\mathsf{KUGen}(st, rl, t) \rightarrow ku_t$: Parse the time $t$ is a bit string with $\ell$ bits such that $t \in \{0,1\}^\ell$. Let $t[i]$ be the $i^{th}$-bit of the time $t$. For each node $\theta \in \mathsf{KUNodes}(st, rl, t)$, the key updating material generation algorithm randomly picks a term $\bar{r} \in \mathbb{Z}_p$ and defines a set $\mathcal{V} \in [\ell]$ recording all indices $i$ for $t[i] = 0$ to compute a key updating material $ku_\theta = (ku_1 = g_\theta \cdot (u_0 \prod_{i \in \mathcal{V}} u_i)^{\bar{r}}, ku_2 = g^{\bar{r}})$. It returns a set of key updating materials $ku_t = (t, \{ku_\theta\}_{\theta \in \mathsf{KUNodes}(st, rl, t)})$.

$\mathsf{TKUpdate}(tk_{id}, ku_t) \rightarrow utk_{id,t} / \perp$: The transformation key updating algorithm returns a failure symbol $\perp$ if $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \emptyset$; otherwise, it has $\theta \in \mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t)$. Let $\mathcal{V} \in [\ell]$ be the set of all indices $i$ for $t[i] = 0$. The algorithm returns an updated transformation key $utk_{id,t} = ((id, \mathcal{R}, t), utk_1 = tk_1 \cdot ku_1 = pk_{id}^\alpha w^r \cdot (u_0 \prod_{i \in \mathcal{V}} u_i)^{\bar{r}}, utk_2 = tk_2 = g^r, \{utk_{3,\tau} = tk_{3,\tau} = g^{r_\tau}, utk_{4,\tau} = tk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau} v^{-r}\}_{\tau \in [k]}, utk_5 = ku_2 = g^{\bar{r}})$.

$\mathsf{EKGen}(msk, \mathcal{S}) \rightarrow ek_{\mathcal{S}}$: Parse the attribute set of a sender $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$. The encryption key generation algorithm randomly picks terms $s, s_1, s_2, ..., s_k \in \mathbb{Z}_p$ and returns an encryption key $ek_{\mathcal{S}} = (\mathcal{S}, ek_1 = g^\beta w^s, ek_2 = g^s, \{ek_{3,\tau} = g^{s_\tau}, ek_{4,\tau} = (u^{\mathcal{S}_\tau} h)^{s_\tau} v^{-s}\}_{\tau \in [k]})$.

$\mathsf{Enc}(ek_{\mathcal{S}}, \hat{\mathcal{S}}, t, \mathbb{R}, m) \rightarrow c$: Parse the access structure of a receiver $\mathbb{R} = (\mathbb{M}, \rho)$, where $\mathbb{M} \in \mathbb{Z}_p^{l \times n}$ is a matrix and $\rho : [l] \rightarrow \mathbb{Z}_p$ is a mapping function. The encryption algorithm randomly chooses a vector $\boldsymbol{x} = (\phi, x_2, ..., x_n)^\top \in \mathbb{Z}_p^{n \times 1}$ and computes $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_l)^\top = \mathbb{M}\boldsymbol{x}$. It randomly chooses terms $\phi_1, \phi_2, ..., \phi_l \in \mathbb{Z}_p$ and computes $c_0 = m \cdot e(g,g)^{\alpha\phi}, c_1 = g^\phi, c_{2,\tau} = w^{\lambda_\tau} v^{\phi_\tau}, c_{3,\tau} = (u^{\rho(\tau)} h)^{-\phi_\tau}$ and $c_{4,\tau} = g^{\phi_\tau}$. Let

$\tilde{\mathcal{V}} \in [\ell]$ be the set of all indices $i$ for $\tilde{t}[i] = 0$. It encodes the time $t$ to $\tilde{t}$ via running the time encoding algorithm $\mathsf{TEncode}(t, \mathcal{T})$ and computes $\tilde{c}_1 = u_0^{\phi}$ and $\tilde{c}_{2,i} = u_i^{\phi}$. Parse the attribute of a sender $\hat{\mathcal{S}} = (\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, ..., \hat{\mathcal{S}}_m)$ and $\hat{\mathcal{S}} \subseteq \mathcal{S}$. It randomly chooses $\hat{s}, \hat{s}_1, \hat{s}_2, ..., \hat{s}_m, \kappa \in \mathbb{Z}_p$ and computes $\hat{c}_1 = ek_2 \cdot g^{\hat{s}} = g^{s+\hat{s}}, \hat{c}_{2,\hat{\tau}} = ek_{3,\tau} \cdot g^{s_{\hat{\tau}}} = g^{s_\tau + s_{\hat{\tau}}}, \hat{c}_{3,\hat{\tau}} = ek_{4,\tau} \cdot (u^{\mathcal{S}_\tau} h)^{s_{\hat{\tau}}} v^{-\hat{s}} = (u^{\mathcal{S}_\tau} h)^{s_\tau + s_{\hat{\tau}}} v^{-(s+\hat{s})}$ and $\hat{c}_4 = g^{\kappa}$. Let $\ddot{c}$ be $\ddot{c} = c_0 \| c_1 \| c_{2,1} \| ... \| c_{2,l} \| c_{3,1} \| ... \| c_{3,l} \| c_{4,1} \| ... \| c_{4,l} \| \hat{c}_1 \| \hat{c}_{2,1} \| \ ... \| \hat{c}_{2,m} \| \hat{c}_{3,1} \| ... \| \ \hat{c}_{3,m} \| \hat{c}_4$. It computes $\hat{c}_0 = ek_1 \cdot w^{\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa} = g^{\beta} w^{s+\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa}$. The algorithm returns a ciphertext $c = (c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau \in [l]}, \tilde{c}_1, \{\tilde{c}_{2,i}\}_{i \in \tilde{\mathcal{V}}}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}\}_{\tau \in [m]}, \hat{c}_4)$.

$\mathsf{CTUpdate}(c, t) \rightarrow c_t$: Let $\mathcal{V} \in [\ell]$ be the set of all indices $i$ for $t[i] = 0$. The ciphertext update algorithm computes $\tilde{c} = \tilde{c}_1 \prod_{i \in \mathcal{V}} \tilde{c}_{2,i} = (u_0 \prod_{i \in \mathcal{V}} u_i)^{\phi}$. The algorithm returns the ciphertext $c_t = (c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau \in [l]}, \tilde{c}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}\}_{\tau \in [m]}, \hat{c}_4)$.

$\mathsf{Verify}(\mathbb{S}, c_t) \rightarrow \{0, 1\}$: Parse the access structure of a sender $\mathbb{S} = (\mathbb{N}, \pi)$, where $\mathbb{N} \in \mathbb{Z}_p^{l \times n}$ is a matrix and $\pi : [\ell] \rightarrow \mathbb{Z}_p$ is a mapping function. The verification algorithm randomly picks a vector $\boldsymbol{y} = (1, y_2, ..., y_n)^{\top} \in \mathbb{Z}_p^{n \times 1}$ and computes $\boldsymbol{\mu} = (\mu_1, \mu_2, ..., \mu_l)^{\top} = \mathbb{N}\boldsymbol{y}$. Let $\mathcal{I}$ be $\mathcal{I} = \{i : \pi(i) \in \mathcal{S}\}$ for $\{\omega_i \in \mathbb{Z}_p\}_{i \in \mathcal{I}}$ such that $\sum_{i \in \mathcal{I}} \omega_i \mathbb{N}_i = (1, 0, ..., 0)$. The verification checks

$$\frac{e(\hat{c}_0, g)}{\prod_{i \in \mathcal{I}} (e(\hat{c}_1, w^{\mu_i} v) \cdot e(\hat{c}_{2,\tau}, (u^{\pi(i)} h)^{-1}) \cdot e(\hat{c}_{3,\tau}, g))^{\omega_i} \cdot e(\hat{c}_4, \mathcal{H}(\ddot{c}))} \stackrel{?}{=} e(g, g)^{\beta}.$$

The algorithm outputs 1 if the above formula is valid; otherwise, it outputs 0.

$\mathsf{Transfer}(utk_{id,t}, c_t) \rightarrow \widehat{c}$: Parse the attribute set of a receiver $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_k)$ and the access structure of a receiver $\mathbb{R} = (\mathbb{M}, \rho)$, where $\mathbb{M} \in \mathbb{Z}_p^{l \times n}$ is a matrix and $\rho : [l] \rightarrow \mathbb{Z}_p$ is a mapping function. Let $\mathcal{J}$ be $\mathcal{J} = \{j : \rho(j) \in \mathcal{R}\}$ for $\{\delta_j \in \mathbb{Z}_p\}_{j \in \mathcal{J}}$ such that $\sum_{j \in \mathcal{J}} \delta_j \mathbb{M}_j = (1, 0, ..., 0)$. The transformation algorithm computes

$$\widehat{c}_0 = \frac{e(c_1, utk_1)}{\prod_{j \in \mathcal{J}} (e(c_{2,j}, utk_2) \cdot e(c_{3,j}, utk_{3,\tau}) \cdot e(c_{4,j}, utk_{4,\tau}))^{\delta_j} \cdot e(\tilde{c}, utk_5)} = e(pk_{id}, g)^{\alpha\phi}.$$

The algorithm returns a transformed ciphertext $\widehat{c} = (c_0, \widehat{c}_0)$.

$\mathsf{Dec}(sk_{id}, \widehat{c}) \rightarrow m$: The decryption algorithm returns a message $m = c_0 / \widehat{c}_0^{1/sk_{id}}$.

$\mathsf{Rev}(rl, id, t) \rightarrow rl$: The revocation algorithm returns an updated revocation list $rl$ as $rl \leftarrow rl \cup (id, t)$.

### 4.2 Security Analysis

**Theorem 1.** *If Waters' IBE [30] is* IND-CCA *secure and Rouselakis-Waters' ABE [25] is* sIND-CPA *secure, then all probabilistic polynomial-time adversaries have a negligible advantage in breaking* sIND-CPA *security for our scheme*[5].

**Theorem 2.** *If the computational $q$-1 assumption holds, then all probabilistic polynomial-time adversaries have a negligible advantage in breaking* EU-CMA *security for our scheme*[6].

[5] The security proof is in Appendix B.

[6] Appendix A shows the computational $q$-1 assumption. The security proof is in Appendix C.

### 4.3 Performance Analysis

In this subsection, we concrete the performance analysis in terms of functionality, theoretical costs and experimental simulations.

For the functionality, we give a detailed comparison in Table 1. Our functionality analysis focuses on the following properties:

- *Data privacy* is the fundamental requirement of cryptographic tools, which requires the encrypted data cannot be revealed by any party without a valid decryption key.
- *Data source identification* is used to discard messages from undesirable data owners without costly data decryption, where it not only ensures the quantity of the data source but also prevents a variety of attacks, such that the impersonate attack and the denial-of-service attack.
- *Revocation* enables the trusted party (e.g., the KGC) to revoke users whom secret keys are compromised or credentials has been changed (e.g., retire and promotion). Note that revocation is very useful in the subscribe system with sporadic memberships (e.g., commercial cloud services, pay-TV, etc.).
- *Outsourced computation* relieves major workloads of the end-devices, which improve the performance and usability of the IoT devices constrained with memory, physical size, and battery. In this paper, we focus on the expensive workloads at the end-devices, including data decryption, user revocation, and data source identification, where user revocation refers to the cost of generating the decryption key in each revocation epoch.
- We consider four strategies about *access policy*: one-to-one data sharing, one-way LSSS, bilateral LSSS, and "no read up, no write down" regulation. One-to-one data sharing represents the coarse-level access control, such as public key infrastructure (PKI) and identity-based encryption (IBE). One-way LSSS standards for the fine-grained access control in most of ABE schemes. Bilateral LSSS is for the schemes with sender access control and receiver access control simultaneously, such as generic construction of matchmaking encryption [2] and our proposed SRB-ABE. The regulation "no read up, no write down" is a novel access control approach proposed in access control encryption (ACE) [11].

From Table 1, our proposed SRB-ABE has supervision performance in functionality since it has data privacy, data source identification, and revocation simultaneously. In addition, SRB-ABE supports outsourced computation in terms of data decryption, user revocation, and data source identification. Moreover, bilateral LSSS is more powerful than the traditional one-way LSSS since it prevents various attacks and filters undesirable messages. Furthermore, the LSSS access control is more suitable than the "no read up, no write down" regulation in cloud-fog computing.

For the theoretical costs, we concentrate on the most relevant works with outsourced computation since the relevant works without outsourced computation allocate the expensive workload to the end-device, which is not suitable to the IoT ecosystem and the performance of them is much inefficient than the works with outsourced computation.

In Table 2, we present the theoretical analysis among the most relevant solutions with outsourced computation [14, 10], and ours. Our proposed SRB-ABE applies the

Table 1: Functionality Comparison among the Most Relevant Concrete Solutions

| *Mode* | *Data Privacy* | *Data Identification* | *Revocation* | *Outsourced Computation* | *Access Policy* |
|---|---|---|---|---|---|
| ***ABE*** [27] | ✓ | ✗ | ✗ | *N/A* | *One-way LSSS* |
| ***S-ABE*** [14] | ✓ | ✗ | ✗ | *Decryption* | *One-way LSSS* |
| ***R-ABE*** [26] | ✓ | ✗ | ✓ | *N/A* | *One-way LSSS* |
| ***SR-ABE*** [10] | ✓ | ✗ | ✓ | *Decryption and revocation* | *One-way LSSS* |
| ***ABKS*** [37] | ✓ | ✗ | ✗ | *N/A* | *One-way LSSS* |
| ***ACE*** [11] | ✓ | ✓ | ✗ | *N/A* | *No read up, no write down* |
| ***MIBE*** [2] | ✓ | ✓ | ✗ | *N/A* | *One-to-one data sharing* |
| ***SRB-ABE*** | ✓ | ✓ | ✓ | *Decryption, revocation and data source identification* | *Bilateral LSSS* |

Table 2: Theoretical Analysis among Relevant Solutions with Outsourced Computation

| | *Mode* | *Setup* | *KeyGen* | *TKGen* | *KUGen* | *EKGen* | *Enc* | *Verify* | *Transfer* | *Dec* |
|---|---|---|---|---|---|---|---|---|---|---|
| **CHW11** | ***S-ABE*** | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(\mathcal{R})$ | *N/A* | *N/A* | $\mathcal{O}(\mathbb{R})$ | *N/A* | $\mathcal{O}(\mathbb{R})$ | $\mathcal{O}(1)$ |
| **CDLQ16** | ***SR-ABE*** | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(\log \mathcal{N} \cdot \mathcal{R})$ | $\mathcal{O}(\log \mathcal{N})$ | *N/A* | $\mathcal{O}(\mathbb{R})$ | *N/A* | $\mathcal{O}(\mathbb{R})$ | $\mathcal{O}(1)$ |
| **Ours** | ***SRB-ABE*** | $\mathcal{O}(\log \mathcal{T})$ | $\mathcal{O}(1)$ | $\mathcal{O}(\log \mathcal{N} \cdot \mathcal{R})$ | $\mathcal{O}(\log \mathcal{N})$ | $\mathcal{O}(\mathcal{S})$ | $\mathcal{O}(\mathcal{S} + \mathbb{R})$ | $\mathcal{O}(\mathbb{S})$ | $\mathcal{O}(\mathbb{R})$ | $\mathcal{O}(1)$ |

large universe ABE with the constant-size public parameter [25] to setup the system with the order of $\mathcal{O}(1)$. By applying the ElGamal-like key pair as in S-ABE [14], each data user only requests to generate a secret-public key pair, which takes $\mathcal{O}(1)$. The transformation key generation process is in the fog node, which depends on the attribute set of data user $\mathcal{O}(\log \mathcal{N} \cdot \mathcal{R})$, where $\mathcal{N}$ is the number of system users. Note that sacrificing the secret key is to reduce the cost of key updating materials, which broadcast in each revocation epoch rather than the secret key only distributes once. By applying tree-based revocation list as in Section 2.3 and updatable time structure as in 2.4, the key updating materials requires $\mathcal{O}(\log \mathcal{N})$. The cost of encryption key generation depends on the attribute set of the sender $\mathcal{O}(\mathcal{S})$. Our SRB-ABE takes extra cost $\mathcal{O}(\mathcal{S} + \mathbb{R}$ to derive ciphertext since it requires the data owner to embed the corresponding attribute set $\mathcal{S}$. Similarly, the cost of verification process depends on the access policy of the sender $\mathcal{O}(\mathbb{S})$. For the ciphertext transformation and the data decryption processes, all the schemes with outsourced computation require transformation depending on the access structure of the receiver $\mathcal{O}(\mathbb{R})$ and decryption in the constant $\mathcal{O}(1)$. Hence, our proposed SRB-ABE has comparable performance among the most relevant solutions with outsourced computation [14, 10] in the term of theoretical complexity.

For experimental simulation, we focus on evaluating S-ABE [14] denoted GHW11, SR-ABE [10] denoted CDLQ16 and ours. Our mainly experiential simulation was performed on a MacBook Pro running in macOS Mojave Version 10.14.6 with 2.2 GHz Intel Core i7 and 16 GB 2400 MHz DDR4 and data decryption process was simulated on MI 5s running in Android 10 with Quad-core Max 2.15 GHz Snapdragon 821 and 4 GB 2400 MHz LPDDR4. In particular, our implementation is in Eclipse with Java Version 13.0.2 via JPBE library with Type A elliptic curve from "a.properties" provided by JPBE library. Hence, in our experimental performances, $p$ is a 160-bit prime number, and elements in $\mathbb{G}$ and $\mathbb{G}_T$ have 512 bits and 1024 bits, respectively. The experimental performances are presented in Fig. 2. To simplicity, we set the number of system users to $2^{10} = 1024$ and the bounded system lifetime to $2^{10}$ if no specification.

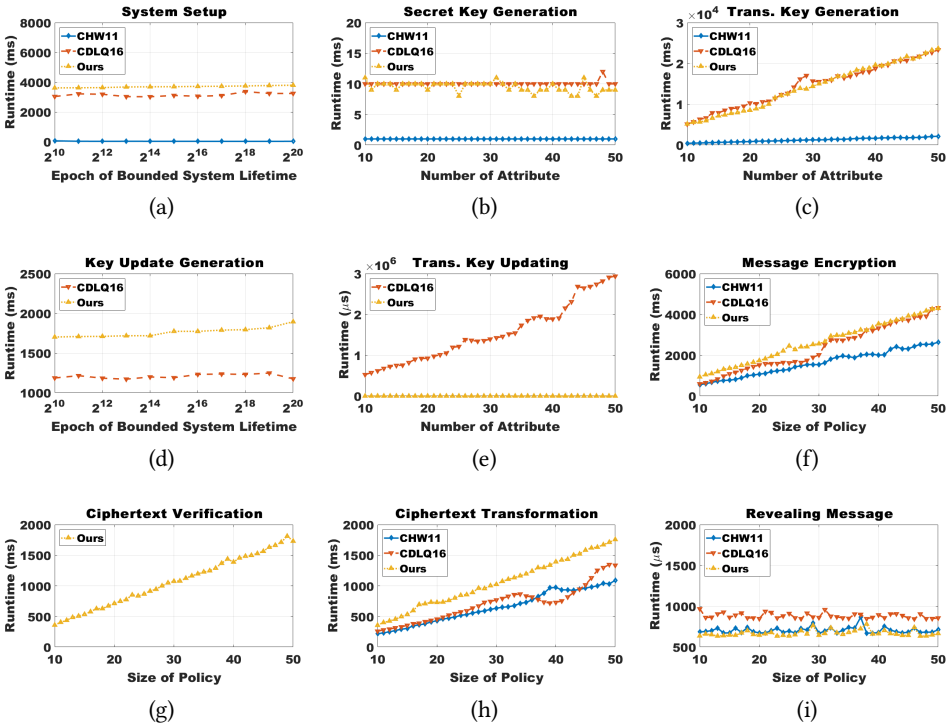


Fig. 2: Experimental Simulations

Fig. 2a presents the running time for system setup as a function of the epoch of bounded system lifetime since the time consumption is irrelevant to the size of the attribute universe. Our scheme takes much more time than other schemes since the time component in our scheme is provably secure in the standard model, where CLDQ16 is provably secure in the random oracle model. CHW11 has the least time since it cannot support revocation.

Fig. 2b illustrates the time for secret key generation as a function of the number of attributes. CDLQ16 and ours have a comparable performance and take more time than CHW11 since both schemes derive a secret-public key pair and outsource the public key to derive transformation key. CHW11 only requires to derive a secret key and the generation of the transformation key needs the secret key. Hence, CHW11 requires a secure channel to derive transformation key but CDLQ16 and ours are not.

Fig. 2c displays the time for transformation key generation versus the number of attributes. CDLQ16 and ours have a comparable performance and take more time than CHW11 since each user in both schemes has 11 secret keys ($\log_2 \mathcal{N} + 1 = 11$ and the number of system user $\mathcal{N} = 2^{10}$) for the efficient tree-based revocation setting and CHW11 requires 1 secret key only.

Fig. 2d presents the time for key update generation versus the epoch of bounded system lifetime. Ours takes much more time than CDLQ16 since our solution can achieve strong security requirement. CDLQ16 applies the random oracle to generate the key update and our key update is in the standard model.

Fig. 2e illustrates the time for transformation key updating versus the number of attributes. CDLQ16 takes much more time since they apply key re-randomization technology to derive the transformation key. However, the security of our scheme is based on the user secret key. In other words, the leakage of the transformation key does not affect the security of the security of the other revocation epochs. Hence, our scheme has less time since no key re-randomization.

Fig. 2f displays the time for message encryption versus the size of policies. Our scheme takes much more time than others since our scheme considers efficient revocation in the strong security model (e.g., the standard model). CDLQ16 has less time than ours since it support revocation in the weak security model (e.g., the random oracle model). CHW11 has the better performance than others since it cannot provide revocation.

Fig. 2g presents the time for ciphertext verification as a function of the size of policies. We demonstrate the performance of our scheme only since it is an additional property provided by ours only. The time for ciphertext verification is linear to the size of policies. Our scheme has smaller decryption key size than others since the simple fine-grained access control policy is used to control the receivers.

Fig. 2h illustrates the time for ciphertext transformation as a function of the size of policies. The performance of ciphertext transformation is similar to the performance of message encryption as in Fig. 2f. The additional overheads are incurred by the time component for efficient revocation. CHW11 has the best performance of all since it has no revocation.

Fig. 2i displays the time for revealing message as a function of the size of policies. All three schemes take a very small cost for revealing message (under 1000 $\mu$s = 1 ms). Our scheme takes less time than CDLQ16 and has comparable performance to CHW11.

Hence, our proposed SRB-ABE has comparable performance among the most relevant solutions with outsourced computation [14, 10] in the term of the performance in the real-time simulation.

## 5 Conclusion

In this work, we investigated the problems of data sharing in cloud-fog computing and produced a secure cloud-fog data sharing system adopting to IoT ecosystem with following properties: (1) a fine-grained receiver and sender access control simultaneously; (2) server-aided user revocation with ciphertext update; (3) lightweight data decryption via fog nodes; and (4) outsourced data source identification. We believe that our proposed matchmaking attribute-based encryption finds many applications for providing data privacy and ciphertext identification simultaneously.

## References


1. Shashank Agrawal and Melissa Chase. FAME: fast attribute-based message encryption. In *CCS*, pages 665–682, 2017.
2. Giuseppe Ateniese, Danilo Francati, David Nuñez, and Daniele Venturi. Match me if you can: Matchmaking encryption and its applications. In *CRYPTO*, pages 701–731, 2019.
3. Nuttapong Attrapadung and Hideki Imai. Attribute-based encryption supporting direct/indirect revocation modes. In *IMA*, pages 278–300, 2009.
4. Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In *PKC*, volume 6571, pages 90–108, 2011.
5. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P*, pages 321–334, 2007.
6. Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In *CCS*, pages 417–426, 2008.
7. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.
8. David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. *IACR Cryptology ePrint Archive*, 2016:718, 2016.
9. Ling Cheung and Calvin C. Newport. Provably secure ciphertext policy ABE. In *CCS*, pages 456–465, 2007.
10. Hui Cui, Robert H. Deng, Yingjiu Li, and Baodong Qin. Server-aided revocable attribute-based encryption. In *ESORICS*, pages 570–587, 2016.
11. Ivan Damgård, Helene Haagh, and Claudio Orlandi. Access control encryption: Enforcing information flow with cryptography. In *TCC*, pages 547–576, 2016.
12. Dirk Fox. Certificate revocation list (CRL). *Datenschutz und Datensicherheit*, 25(8), 2001.
13. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, pages 89–98, 2006.
14. Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *USENIX*, 2011.
15. Jinguang Han, Liqun Chen, Willy Susilo, Xinyi Huang, Aniello Castiglione, and Kaitai Liang. Fine-grained information flow control using attributes. *Inf. Sci.*, 484:167–182, 2019.
16. Javier Herranz, Fabien Laguillaumie, and Carla Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In *PKC*, pages 19–34, 2010.
17. Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.
18. Xiaohui Liang, Zhenfu Cao, Huang Lin, and Dongsheng Xing. Provably secure and efficient bounded ciphertext policy attribute based encryption. In *ASIACCS*, pages 343–352, 2009.
19. Joseph K. Liu, Man Ho Au, Xinyi Huang, Rongxing Lu, and Jin Li. Fine-grained two-factor access control for web-based cloud computing services. *IEEE Trans. Information Forensics and Security*, 11(3):484–497, 2016.
20. Qutaibah M. Malluhi, Abdullatif Shikfa, and Viet Cuong Trinh. A ciphertext-policy attribute-based encryption scheme with optimized ciphertext size and fast decryption. In *ASIACCS*, pages 230–240, 2017.
21. Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, pages 41–62, 2001.
22. Jianting Ning, Zhenfu Cao, Xiaolei Dong, Kaitai Liang, Hui Ma, and Lifei Wei. Auditable $\sigma$-time outsourced attribute-based encryption for access control in cloud computing. *IEEE Trans. Information Forensics and Security*, 13(1):94–105, 2018.

23. Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. Passive attacks against searchable encryption. *IEEE Trans. Information Forensics and Security*, 14(3):789–802, 2019.
24. Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *CCS*, pages 195–203, 2007.
25. Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *CCS*, pages 463–474, 2013.
26. Amit Sahai, Hakan Seyalioglu, and Brent Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In *CRYPTO*, pages 199–217, 2012.
27. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, volume 3494, pages 457–473, 2005.
28. Statista. Edge computing market size forecast worldwide from 2017 to 2024.
29. Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *FedCSIS*, pages 1–8, 2014.
30. Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.
31. Shengmin Xu, Guomin Yang, and Yi Mu. Revocable attribute-based encryption with decryption key exposure resistance and ciphertext delegation. *Information Sciences*, 479:116–134, 2018.
32. Shengmin Xu, Guomin Yang, Yi Mu, and Robert H. Deng. Secure fine-grained access control and data sharing for dynamic groups in the cloud. *IEEE Trans. Information Forensics and Security*, 13(8):2101–2113, 2018.
33. Shengmin Xu, Guomin Yang, Yi Mu, and Ximeng Liu. A secure iot cloud storage system with fine-grained access control and decryption key exposure resistance. *Future Generation Comp. Syst.*, 97:284–294, 2019.
34. Yinghui Zhang, Dong Zheng, Xiaofeng Chen, Jin Li, and Hui Li. Computationally efficient ciphertext-policy attribute-based encryption with constant-size ciphertexts. In *ProvSec*, pages 259–273, 2014.
35. Yinghui Zhang, Dong Zheng, and Robert H. Deng. Security and privacy in smart health: Efficient policy-hiding attribute-based access control. *IEEE Internet of Things Journal*, 5(3):2130–2145, 2018.
36. Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX*, pages 707–720, 2016.
37. Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *INFOCOM*, pages 522–530, 2014.

## A Hard Assumption

**Definition 4 (Computational $q$-1 Assumption).** *Let $a, s, b_1, b_2, ..., b_q \in \mathbb{Z}_p$ be random terms and $g \in \mathbb{G}$ be a group generator of bilinear group $\mathbb{G}$ with prime order $p$. The computational $q-1$ assumption is that no probabilistic polynomial-time algorithm cannot output the term $e(g,g)^{sa^{q+1}}$ with more than a negligible advantage by giving $g, g^s$; $\forall(i,j) \in [q,q] : g^{a^i}, g^{b_j}, g^{sb_j}, g^{a^i b_j}, g^{a^i/b_j^2}$ ; $\forall(i,j,j') \in [2q,q,q]$ and $j \neq j' : g^{a^i b/b_{j'}^2}$; $\forall(i,j) \in [2q,q]$ and $i \neq q+1 : g^{a^i/b_j}$; $\forall(i,j,j') \in [q,q,q]$ and $j \neq j' : g^{sa^i b_j/b_{j'}}, g^{sa^i b_j/b_{j'}^2}$.*

## B Security Proof of Theorem 1

*Proof.* Suppose there exists a polynomial-time adversary $\mathcal{A}$ who can break sIND-CPA security of our proposed scheme with non-negligible advantage. We can build a probabilistic polynomial-time simulator $\mathcal{B}$ to break Waters' IBE [30] denoted $\mathcal{C}_{\mathsf{IBE}}$ and Rouselakis-Waters' ABE [25] as $\mathcal{C}_{\mathsf{ABE}}$.

**Initialization**: $\mathcal{B}$ receives a challenge access policy $\mathbb{R}^* = (\mathbb{M}^*, \rho^*)$ from $\mathcal{A}$ and randomly chooses a bit $rev \in \{0, 1\}$. $\mathcal{B}$ then processes the simulation based on $rev$.

- If $rev = 0$, $\mathcal{B}$ assumes $\mathcal{A}$ is a non-revoked user. $\mathcal{B}$ aborts if any identity $id$ associated with the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}^*$ is queried.
- If $rev = 1$, $\mathcal{B}$ assumes $\mathcal{A}$ is a revoked user. $\mathcal{B}$ aborts if any identity $id$ associated with the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}^*$ is not revoked at the timestamp $t$, where $t \leq t^*$ and $t^*$ is a challenge timestamp to derive a challenge ciphertext $c^*$.

$\mathcal{B}$ forwards $\mathbb{R}^*$ to $\mathcal{C}_{\mathsf{ABE}}$ if $rev = 0$. Note that Waters' IBE [30] is IND-CCA secure, and hence the challenge timestamp $t^*$ does not declare at the beginning of the game.

**Setup**: $\mathcal{B}$ randomly chooses a term $\beta \in \mathbb{Z}_p$, a collusion-resistant hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{G}$ and initializes an empty revocation list $rl$ and a binary tree $\mathsf{BT}$ with at least $\mathcal{N}$ leaf nodes as the state $st$, where $\mathcal{N}$ is the number of system users. $\mathcal{B}$ then simulates the master public key as follows:

- If $rev = 0$, $\mathcal{B}$ receives the master public key $mpk_{\mathsf{ABE}}$ from $\mathcal{C}_{\mathsf{ABE}}$, and randomly chooses $\bar{u}_0, \bar{u}_1, ..., \bar{u}_\ell \in \mathbb{Z}_p$ to compute $u_i = g^{\bar{u}_i}$ for $i \in [\ell]$, where $\ell$ is the bit length of the system bounded lifetime $\mathcal{T}$ such that $\ell = \log_2 \mathcal{T}$. $\mathcal{B}$ returns the master public key $mpk = (mpk_{\mathsf{ABE}}, u_0, u_1, ..., u_\ell, e(g, g)^\beta, \mathcal{H})$ to $\mathcal{A}$.
- If $rev = 1$, $\mathcal{B}$ receives the master public key $mpk_{\mathsf{IBE}}$ from $\mathcal{C}_{\mathsf{IBE}}$, and randomly chooses $\bar{w}, \bar{v}, \bar{u}, \bar{h} \in \mathbb{Z}_p$ to compute $w = g^{\bar{w}}, v = g^{\bar{v}}, u = g^{\bar{u}}, h = g^{\bar{h}}$. $\mathcal{B}$ returns the master public key $mpk = (mpk_{\mathsf{IBE}}, w, v, u, h, e(g, g)^\beta, \mathcal{H})$ to $\mathcal{A}$.

**Phase 1 & 2**: $\mathcal{A}$ is allowed to query the following oracles.

Key generation oracle: $\mathcal{A}$ queries on an identity $id$ and an attribute set $\mathcal{R}$. $\mathcal{B}$ returns the same $pk_{id}$ if $id$ has been queried before; otherwise, $\mathcal{B}$ randomly chooses $\gamma_{id} \in \mathbb{Z}_p$ to set $sk_{id} = \gamma_{id}$ and compute $pk_{id} = g^{sk_{id}}$ and records $(id, pk_{id}, sk_{id})$. $\mathcal{B}$ returns $pk_{id} = g^{sk_{id}}$ to $\mathcal{A}$.

Corrupt oracle: $\mathcal{A}$ queries on an identity $id$ and an attribute set $\mathcal{R}$. $\mathcal{B}$ returns the same $sk_{id}$ if $id$ has been queried before; otherwise, $\mathcal{B}$ randomly chooses $\gamma_{id} \in \mathbb{Z}_p$ to set $sk_{id} = \gamma_{id}$ and compute $pk_{id} = g^{sk_{id}}$ and records $(id, pk_{id}, sk_{id})$. $\mathcal{B}$ returns $sk_{id} = g^{sk_{id}}$ to $\mathcal{A}$.

Transformation key generation oracle: $\mathcal{A}$ queries on an identity $id$ and an attribute set $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_k)$. $\mathcal{B}$ answers the query as follows:

Case 1: $rev = 0$ and $\mathcal{R} \models \mathbb{R}^*$, $\mathcal{B}$ aborts since $\mathcal{A}$ is a non-revoked user and the secret key for the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}^*$ cannot be queried.

Case 2: $rev = 0$ and $\mathcal{R} \not\models \mathbb{R}^*$, $\mathcal{B}$ sends the attribute set $\mathcal{R}$ to $\mathcal{C}_{\mathsf{ABE}}$. $\mathcal{B}$ receives the secret key $sk_{\mathcal{R}} = (sk_1, sk_2, \{sk_{3,\tau}, sk_{4,\tau}\}_{\tau \in [k]})$. $\mathcal{B}$ retrieves $sk_{id}$ if it is available;

otherwise, $\mathcal{B}$ randomly chooses $\gamma_{id} \in \mathbb{Z}_p$ to set $sk_{id} = \gamma_{id}$ and compute $pk_{id} = g^{sk_{id}}$ and records $(id, pk_{id}, sk_{id})$. $\mathcal{B}$ assigns $id$ to an undefined leaf node in BT. For all nodes $\theta$ in $\mathsf{Path}(id)$, $\mathcal{B}$ retrieves $g_\theta$ if it is available; otherwise, $\mathcal{B}$ randomly picks $g_\theta \in \mathbb{G}$. According to Rouselakis-Waters' ABE [25], $\mathcal{B}$ randomly pick $r_\theta, r_{\theta,1}, r_{\theta,2}, ..., r_{\theta,k} \in \mathbb{Z}_p$ and computes $tk_\theta = (tk_1, tk_2, \{tk_{3,\tau}, tk_{4,\tau}\}_{\tau\in[k]})$ as:

$$
\begin{aligned}
&tk_1 = sk_1^{\gamma_{id}} \cdot w^{r_\theta}/g_\theta = (g^{\alpha\gamma_{id}}/g_\theta) \cdot w^{r\gamma_{id}+r_\theta} = (pk_{id}^{\alpha}/g_\theta) \cdot w^{r\gamma_{id}+r_\theta},\\
&tk_2 = sk_2^{\gamma_{id}} \cdot g^{r_\theta} = g^{r\gamma_{id}+r_\theta}, tk_{3,\tau} = sk_{3,\tau}^{\gamma_{id}} \cdot g^{r_{\theta,\tau}} = g^{r_\tau\gamma_{id}+r_{\theta,\tau}},\\
&tk_{4,\tau} = sk_{4,\tau}^{\gamma_{id}} \cdot (u^{\mathcal{R}_\tau}h)^{r_{\theta,\tau}} v^{-r_\theta} = (u^{\mathcal{R}_\tau}h)^{r_\tau\gamma_{id}+r_{\theta,\tau}} v^{-(r\gamma_{id}+r_\theta)}.
\end{aligned}
$$

In the view of $\mathcal{A}$, $tk_\theta$ is uniform embedding the master secret key $\alpha$ and randomnesses $r\gamma_{id} + r_\theta, r_1\gamma_{id} + r_{\theta,1}, r_2\gamma_{id} + r_{\theta,2}, ..., r_k\gamma_{id} + r_{\theta,k}$ in the randomness domain $\mathbb{Z}_p$. $\mathcal{C}$ returns the transformation key $tk_{id} = ((id, \mathcal{R}), \{tk_\theta\}_{\theta\in\mathsf{Path}(id)})$ to $\mathcal{A}$.

Case 3: $rev = 1$. Let $id^*$ denote the users associated with the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}$. $\mathcal{B}$ assigns $id$ to an undefined leaf node in BT. For all nodes $\theta$ in $\mathsf{Path}(id)$, $\mathcal{B}$ retrieves $g_\theta$ if it is available; otherwise, $\mathcal{B}$ randomly picks $g_\theta \in \mathbb{G}$. $\mathcal{B}$ then randomly picks $r, r_1, r_2, ..., r_k \in \mathbb{Z}_p$ and computes the transformation key in the node $\theta$ $tk_\theta = (tk_1, tk_2, \{tk_{3,\tau}, tk_{4,\tau}\}_{\tau\in[k]})$ as $tk_1 = g_\theta w^r, tk_2 = g^r, tk_{3,\tau} = g^{r_\tau}, tk_{4,\tau} = (u^{\mathcal{R}_\tau}h)^{r_\tau}v^{-r}$. In the view of $\mathcal{A}$, $tk_\theta$ is uniform. Specifically, $\mathcal{A}$ is assumed to be a revoked user who will break the security game via breaking the security of $\mathcal{C}_{\mathsf{IBE}}$. Hence, the simulation of $tk_\theta$ is identical to the real scheme except the key components $\alpha$ and $\gamma_{id}$ is allocated in the key updating materials $ku_t$ rather than the transformation key $tk_{id}$. $\mathcal{C}$ returns $tk_{id} = ((id, \mathcal{R}), \{tk_\theta\}_{\theta\in\mathsf{Path}(id)})$ to $\mathcal{A}$.

Key updating materials generation oracle: $\mathcal{A}$ queries on a timestamp $t$. $\mathcal{B}$ defines a set $\mathcal{V} \in [\ell]$ recording all indices $i$ for $t[i] = 0$ and answers the query as follows:

Case 1: $rev = 0$. For all nodes $\theta \in \mathsf{KUNodes}(st, rl, t)$, $\mathcal{B}$ retrieves $g_\theta$ and randomly picks $\bar{r} \in \mathbb{Z}_p$ and computes $ku_\theta = (ku_1, ku_2)$ as: $ku_1 = g_\theta \cdot (u_0 \prod_{i\in\mathcal{V}} u_i)^{\bar{r}}, ku_2 = g^{\bar{r}}$. $\mathcal{B}$ returns $ku_t = (t, \{ku_\theta\}_{\theta\in\mathsf{KUNodes}(st,rl,t)})$ to $\mathcal{A}$.

Case 2: $rev = 1$. $\mathcal{C}$ forwards $t$ to $\mathcal{C}_{\mathsf{IBE}}$ and receives $sk_t = (sk_1, sk_2)$ according to Waters' IBE [30]. $\mathcal{B}$ retrieves $\gamma_{id}$ and $g_\theta$, randomly picks $\bar{r}' \in \mathbb{Z}_p$, and computes $ku_\theta = (ku_1, ku_2)$ as $ku_1 = sk_1^{\gamma_{id}}/g_\theta \cdot (u_0 \prod_{i\in\mathcal{V}} u_i)^{\bar{r}'} = (g^{\alpha\gamma_{id}}/g_\theta) \cdot (u_0 \prod_{i\in\mathcal{V}} u_i)^{\bar{r}\gamma_{id}+\bar{r}'}$, $ku_2 = sk_2^{\gamma_{id}} \cdot g^{\bar{r}'} = g^{\bar{r}\gamma_{id}+\bar{r}'}$. $\mathcal{B}$ returns $ku_t = (t, \{ku_\theta\}_{\theta\in\mathsf{KUNodes}(st,rl,t)})$ to $\mathcal{A}$. In the view of $\mathcal{A}$, $ku_t$ is uniform and the valid updated transformation key $utk_{id,t}$ from $tk_{id}$ and $ku_t$. Please refer to the transformation key updating oracle (the next oracle) for the correctness of $utk_{id,t}$.

Transformation key updating oracle: $\mathcal{A}$ queries on an identity $id$, an attribute set $\mathcal{R}$ and a timestamp $t$. $\mathcal{B}$ answers the query as follows:

Case 1: $rev = 0$ and $\mathcal{R} \models \mathbb{R}^*$, $\mathcal{B}$ aborts since $\mathcal{A}$ is a non-revoked user and the secret key for the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}^*$ cannot be queried.

Case 2: $rev = 0$ and $\mathcal{R} \not\models \mathbb{R}^*$. $\mathcal{B}$ retrieves the transformation key $tk_{id}$ and the key updating material $ku_t$. $\mathcal{B}$ runs the transformation key generation oracle on the identity $id$ and the corresponding attribute set $\mathcal{R}$ if $tk_{id}$ is not available, and key updating materials generation oracle on the timestamp $t$ if $ku_t$ is not available. $\mathcal{B}$ returns a failure

symbol $\perp$ if $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \emptyset$; otherwise, there is exists a node $\theta$ such that $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \theta$. Parse $tk_\theta = (tk_1, tk_2, \{tk_{3,\tau}, tk_{4,\tau}\}_{\tau \in [k]})$ and $ku_\theta = (ku_1, ku_2)$. $\mathcal{B}$ computes

$$
\begin{aligned}
&utk_1 = tk_1 \cdot ku_1 = pk_{id}^{\alpha} w^{r\gamma_{id}+r_\theta} \cdot (u_0 \textstyle\prod_{i \in \mathcal{V}} u_i)^{\bar{r}},\\
&utk_2 = tk_2 = g^{r\gamma_{id}+r_\theta}, utk_{3,\tau} = tk_{3,\tau} = g^{r_\tau \gamma_{id}+r_{\theta,\tau}},\\
&utk_{4,\tau} = tk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau \gamma_{id}+r_{\theta,\tau}} v^{-(r\gamma_{id}+r_\theta)}, utk_{5,\tau} = ku_2 = g^{\bar{r}}.
\end{aligned}
$$

$\mathcal{B}$ returns $utk_{id,t} = ((id, \mathcal{R}, t), utk_1, utk_2, \{utk_{3,\tau}, utk_{4,\tau}\}_{\tau \in [k]}, utk_5)$ to $\mathcal{A}$. In the view of $\mathcal{A}$, $utk_{id,t}$ is uniform with the secret key $\alpha$ and consistent randomnesses.

Case 3: $rev = 1$. $\mathcal{B}$ retrieves the transformation key $tk_{id}$ and the key updating material $ku_t$. $\mathcal{B}$ runs the transformation key generation oracle on the identity $id$ and the corresponding attribute set $\mathcal{R}$ if $tk_{id}$ is not available, and key updating materials generation oracle on the timestamp $t$ if $ku_t$ is not available. $\mathcal{B}$ returns a failure symbol $\perp$ if $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \emptyset$; otherwise, there is exists a node $\theta$ such that $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \theta$. Parse $tk_\theta = (tk_1, tk_2, \{tk_{3,\tau}, tk_{4,\tau}\}_{\tau \in [k]})$ and $ku_\theta = (ku_1, ku_2)$. $\mathcal{B}$ computes

$$
\begin{aligned}
&utk_1 = tk_1 \cdot ku_1 = pk_{id}^{\alpha} w^{r} \cdot (u_0 \textstyle\prod_{i \in \mathcal{V}} u_i)^{\bar{r}\gamma_{id}+\bar{r}'},\\
&utk_2 = tk_2 = g^{r}, utk_{3,\tau} = tk_{3,\tau} = g^{r_\tau},\\
&utk_{4,\tau} = tk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau} v^{-r}, utk_{5,\tau} = ku_2 = g^{\bar{r}\gamma_{id}+\bar{r}'}.
\end{aligned}
$$

$\mathcal{B}$ returns $utk_{id,t} = ((id, \mathcal{R}, t), utk_1, utk_2, \{utk_{3,\tau}, utk_{4,\tau}\}_{\tau \in [k]}, utk_5)$ to $\mathcal{A}$. In the view of $\mathcal{A}$, $utk_{id,t}$ is uniform with the secret key $\alpha$ and consistent randomnesses.

Revocation oracle: $\mathcal{A}$ queries on an identity $id$ and a timestamp $t$. $\mathcal{B}$ updates the revocation list $rl$ such that $rl \leftarrow rl \cup (id, t)$.

Encryption key generation oracle: Parse the attribute set of a sender $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$. $\mathcal{B}$ randomly picks terms $s, s_1, s_2, ..., s_k \in \mathbb{Z}_p$ and computes $ek_1 = g^{\beta} w^{s}, ek_2 = g^{s}, ek_{3,\tau} = g^{s_\tau}, ek_{4,\tau} = (u^{\mathcal{S}_\tau} h)^{s_\tau} v^{-s}$. $\mathcal{B}$ returns $ek_{\mathcal{S}} = (\mathcal{S}, ek_1, ek_2, \{ek_{3,\tau}, ek_{4,\tau}\}_{\tau \in [k]})$ to $\mathcal{A}$. Note that $\beta$ is initialized in the setup phase and is known by $\mathcal{B}$.

**Challenge**: $\mathcal{B}$ receives $(m_0, m_1, \mathcal{S}_0, \mathcal{S}_1, t^*)$ and picks a random bit $b \in \{0, 1\}$. Parse $\mathcal{S}_b = (\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k)$. $\mathcal{B}$ generates the challenge ciphertext as follows:

**Case 1**: $rev = 0$. $\mathcal{B}$ forwards $(m_0, m_1)$ to $\mathcal{C}_{\mathsf{ABE}}$ and $\mathcal{C}_{\mathsf{ABE}}$ returns the ciphertext $ct_{\mathbb{R}^*} = (c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau \in [l]})$, where $c_1 = g^{\phi}$. $\mathcal{B}$ then generates the missing components. Let $\tilde{\mathcal{V}} \in [\ell]$ be the set of all indices $i$ for $\tilde{t}[i] = 0$. $\mathcal{B}$ randomly picks $\phi \in \mathbb{Z}_p$ and computes $\tilde{c} = c_2^{\bar{u}_0} \prod_{i \in \mathcal{V}} c_2^{\bar{u}_i} = (u_0 \prod_{i \in \mathcal{V}} u_i)^{\phi}$. $\mathcal{B}$ randomly picks terms $s, s_1, s_2, ..., s_k \in \mathbb{Z}_p$ to compute the encryption key $ek_{\mathcal{S}} = (\mathcal{S}, ek_1, ek_2, \{ek_{3,\tau}, ek_{3,\tau}\}_{\tau \in [k]})$ as: $ek_1 = g^{\beta} w^{s}, ek_2 = g^{s}, ek_{3,\tau} = g^{s_\tau}, ek_{4,\tau} = (u^{\mathcal{S}_\tau} h)^{s_\tau} v^{-s}$. Let the attribute of the sender $\hat{\mathcal{S}} = (\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, ..., \hat{\mathcal{S}}_m)$ such that $\hat{\mathcal{S}} = \mathcal{S}_0 \cap \mathcal{S}_1$. $\mathcal{B}$ randomly chooses $\hat{s}, \hat{s}_1, \hat{s}_2, ..., \hat{s}_m, \kappa \in \mathbb{Z}_p$ and computes $\hat{c}_1 = ek_2 \cdot g^{\hat{s}} = g^{s+\hat{s}}, \hat{c}_{2,\hat{\tau}} = ek_{3,\tau} \cdot g^{s_{\hat{\tau}}} = g^{s_\tau + s_{\hat{\tau}}}, \hat{c}_{3,\hat{\tau}} = ek_{4,\tau} \cdot (u^{\mathcal{S}_\tau} h)^{s_{\hat{\tau}}} v^{-\hat{s}} = (u^{\mathcal{S}_\tau} h)^{s_\tau + s_{\hat{\tau}}} v^{-(s+\hat{s})}, \hat{c}_4 = g^{\kappa}$. Let $\ddot{c}$ be $\ddot{c} = c_0 \| c_1 \| c_{2,1} \| ... \| c_{2,l} \| c_{3,1} \| ... \| c_{3,l} \| c_{4,1} \| ... \| c_{4,l} \| \hat{c}_1 \| \hat{c}_{2,1} \| \ ... \| \hat{c}_{2,m} \| \hat{c}_{3,1} \| ... \| \hat{c}_{3,m} \| \hat{c}_4$. It computes $\hat{c}_0 = ek_1 \cdot w^{\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa} = g^{\beta} w^{s+\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa}$. The algorithm returns a ciphertext $c = (c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau \in [l]}, \tilde{c}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}\}_{\tau \in [m]}, \hat{c}_4)$.

**Case 2**: $rev = 1$ and for all $t \leq t^*$, it has $(id^*, t) \notin rl$ such that the attribute set of $id^*$ is $\mathcal{R} \models \mathbb{R}$, then $\mathcal{B}$ aborts since all users associated with the attribute set $\mathcal{R}$ such that $\mathcal{R} \models \mathbb{R}$ must be revoked before or at the timestamp $t^*$ when $\mathcal{A}$ plays revoked users.

**Case 3**: $rev = 1$ and for all $t \leq t^*$, it has $(id^*, t) \in rl$ such that the attribute set of $id^*$ is $\mathcal{R} \models \mathbb{R}$, then $\mathcal{B}$ forwards $(m_0, m_1, t^*)$ and to $\mathcal{C}_{\mathsf{IBE}}$. $\mathcal{C}_{\mathsf{IBE}}$ returns the ciphertext $c_t = (c_0, c_1, \tilde{c})$, where $c_1 = g^{\phi}$ Parse $\mathbb{R}^* = (\mathbb{M}^*, \rho^*)$, where $\mathbb{M}^* \in \mathbb{Z}_p^{l \times n}$ is a matrix and $\rho^* : [l] \to \mathbb{Z}_p$ is a mapping function. $\mathcal{B}$ randomly chooses a vector $\boldsymbol{x} = (\phi, x_2, ..., x_n)^\top \in \mathbb{Z}_p^{n \times 1}$ (note that $\phi$ is unknown to $\mathcal{B}$ but $g^{\phi}$ is known to $\mathcal{B}$) and computes $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_l)^\top = \mathbb{M}\boldsymbol{x}$. It randomly chooses terms $\phi_1, \phi_2, ..., \phi_l \in \mathbb{Z}_p$ and computes $c_{2,\tau} = w^{\lambda_\tau} v^{\phi_\tau}, c_{3,\tau} = (u^{\rho(\tau)} h)^{-\phi_\tau}, c_{4,\tau} = g^{\phi_\tau}$. $\mathcal{B}$ randomly picks terms $s, s_1, s_2, ..., s_k \in \mathbb{Z}_p$ to compute the encryption key $ek_{\mathcal{S}} = (\mathcal{S}, ek_1, ek_2, \{ek_{3,\tau}, ek_{3,\tau}\}_{\tau \in [k]})$ as $ek_1 = g^{\beta} w^s, ek_2 = g^s, ek_{3,\tau} = g^{s_\tau}, ek_{4,\tau} = (u^{\mathcal{S}_\tau} h)^{s_\tau} v^{-s}$. $\mathcal{B}$ then chooses Let $\ddot{c}$ be $\ddot{c} = c_0 \| c_1 \| c_{2,1} \| ... \| c_{2,l} \| c_{3,1} \| ... \| c_{3,l} \| c_{4,1} \| \ ... \| c_{4,l} \| \hat{c}_1 \| \hat{c}_{2,1} \| ... \| \hat{c}_{2,m} \| \hat{c}_{3,1} \| ... \| \hat{c}_{3,m} \| \hat{c}_{4,1} \| ... \| \hat{c}_{4,m}$. It computes $\hat{c}_0 = ek_1 \cdot w^{\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa} = g^{\beta} w^{s+\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa}$. The algorithm returns a ciphertext $c = (c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau \in [l]}, \tilde{c}_1, \{\tilde{c}_{2,i}\}_{i \in \tilde{\mathcal{V}}}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}, \hat{c}_{4,\tau}\}_{\tau \in [m]})$.

**Guess**: $\mathcal{B}$ receives a bit $b'$ and forwards $b'$ to $\mathcal{C}_{\mathsf{ABE}}$ if $rev = 0$; otherwise, to $\mathcal{C}_{\mathsf{IBE}}$.

If $rev = 0$, $\mathcal{B}$ simulates the game for non-revoked users. By interacting with $\mathcal{C}_{\mathsf{ABE}}$, $\mathcal{B}$ can break the underlying hard problem of $\mathcal{C}_{\mathsf{ABE}}$ (e.g., decisional $q$-1 assumption) based on the returning value $b'$ from $\mathcal{A}$. If $rev = 1$, $\mathcal{B}$ simulates the game for revoked users. By interacting with $\mathcal{C}_{\mathsf{IBE}}$, $\mathcal{B}$ can break the underlying hard problem of $\mathcal{C}_{\mathsf{IBE}}$ (e.g., decisional BDH assumption) based on the returning value $b'$ from $\mathcal{A}$. However, two aborts exist and are happened by $\mathcal{B}$ guessing $\mathcal{A}$ whether revoked or non-revoked incorrectly. Hence, we have $1/2$ security loss since $rev$ is a random bit from domain $\{0, 1\}$. Therefore, we have the advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathsf{sIND\text{-}CPA}}_{\mathsf{SRB\text{-}ABE},\mathcal{A}}(1^\lambda, \mathcal{N}, \mathcal{T}) = (\epsilon_{\mathsf{ABE}} + \epsilon_{\mathsf{IBE}})/2$, where $\epsilon_{\mathsf{ABE}}$ is the advantage of Rouselakis-Waters' ABE [25] to break decisional $q$-1 assumption and $\epsilon_{\mathsf{IBE}}$) is the advantage of Waters' IBE [30] to break decisional BDH assumption.

## C Security Proof of Theorem 2

*Proof.* Suppose there exists $\mathcal{A}$ who can break EU-CMA security of our scheme with non-negligible advantage. We can build $\mathcal{B}$ to break the computational $q$-1 assumption with non-negligible advantage.

**Initialization**: $\mathcal{B}$ receives the given terms from the computational $q$-1 assumption and a challenger policy $\mathbb{S}^* = (\mathbb{N}^*, \pi^*)$ from $\mathcal{A}$.

**Setup**: $\mathcal{B}$ randomly chooses a term $\beta \in \mathbb{Z}_p$, a collusion-resistant hash function $\mathcal{H} : \{0, 1\}^* \to \mathbb{G}$ and initializes an empty revocation list $rl$ and a binary tree $\mathsf{BT}$ with at least $\mathcal{N}$ leaf nodes as the state $st$. $\mathcal{B}$ then randomly chooses $\tilde{b} \in \mathbb{Z}_p$ and implicitly sets $\beta = a^{q+1} + \tilde{b}$, where $a$ and $q$ are defined in the computational $q$-1 assumption. $\beta$ has correct distribution since $a$ is information-theoretically hidden from $\mathcal{A}$. $\mathcal{B}$ then picks $\alpha, \tilde{v}, \tilde{u}, \tilde{h} \in \mathbb{Z}_p$ and derives the following terms based on the parameters from the computational $q$-1 assumption: $g = g, u = g^{\tilde{u}} \cdot \prod_{(j,k) \in [\ell,n]} (g^{a^k / b_j^2})^{\mathbb{M}^*_{j,k}}, h = g^{\tilde{h}} \cdot \prod_{(j,k) \in [\ell,n]} (g^{a^k / b_j^2})^{-\rho^*(j) \mathbb{M}^*_{j,k}}, w = g^a, v = g^{\tilde{v}} \cdot \prod_{(j,k) \in [\ell,n]} (g^{a^k / b_j})^{\mathbb{M}^*_j}, e(g, g)^{\beta} =$

$e(g^a, g^{a^q}) \cdot e(g,g)^{\tilde{b}}$. $\mathcal{B}$ chooses a hash function $\mathcal{H}: \{0,1\}^* \to \mathbb{G}$ and $\bar{u}_0, \bar{u}_1, ..., \bar{u}_\ell \in \mathbb{Z}_p$ to compute $u_i = g^{\bar{u}_i}$ for $i \in [\ell]$, where $\ell$ is the bit length of $\mathcal{T}$ such that $\ell = \log_2 \mathcal{T}$. $\mathcal{B}$ then returns $mpk = (g, w, v, u, h, u_0, u_1, ..., u_\ell, e(g,g)^\alpha, e(g,g)^\beta, \mathcal{H})$. to $\mathcal{A}$.

**Query**: $\mathcal{A}$ is allowed to query the following oracles adaptively.

Key generation oracle: $\mathcal{A}$ queries on $(id, \mathcal{R})$. $\mathcal{B}$ returns the same $pk_{id}$ if $id$ has been queried before; otherwise, $\mathcal{B}$ randomly chooses $\gamma_{id} \in \mathbb{Z}_p$ to set $sk_{id} = \gamma_{id}$, computes $pk_{id} = g^{sk_{id}}$ and records $(id, pk_{id}, sk_{id})$. $\mathcal{B}$ returns $pk_{id}$.

Corrupt oracle: $\mathcal{A}$ queries on $(id, \mathcal{R})$. $\mathcal{B}$ returns the same $sk_{id}$ if $id$ has been queried before; otherwise, $\mathcal{B}$ randomly chooses $\gamma_{id} \in \mathbb{Z}_p$ to set $sk_{id} = \gamma_{id}$, computes $pk_{id} = g^{sk_{id}}$ and records $(id, pk_{id}, sk_{id})$. $\mathcal{B}$ returns $sk_{id} = g^{sk_{id}}$.

Transformation key generation oracle: $\mathcal{A}$ queries on $(id, \mathcal{R})$. $\mathcal{B}$ chooses an undefined leaf node from $\mathsf{BT}$, and stores $id$ in this node. For each $\theta \in \mathsf{Path}(id)$, it runs as follows: $\mathcal{B}$ retrieves $g_\theta$ if it is available in $\theta$. If $\theta$ has not been defined, it randomly picks $g_\theta \in \mathbb{G}$ and updates the node $\theta$ by updating the state $st \leftarrow st \cup (\theta, g_\theta)$. $\mathcal{B}$ randomly chooses $r, r_1, r_2, ..., r_k \in \mathbb{Z}_p$ and computes $g'_\theta = pk_{id}^\alpha / g_\theta$ to derive $tk_\theta = (tk_1, tk_2, \{tk_{3,\tau}, tk_{4,\tau}\}_{\tau \in [k]})$ as $tk_1 = g'_\theta w^r, tk_2 = g^r, tk_{3,\tau} = g^{r_\tau}, tk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau} v^{-r}$. $\mathcal{B}$ returns $tk_{id} = ((id, \mathcal{R}), \{tk_\theta\}_{\theta \in \mathsf{Path}(id)})$.

Transformation key updating oracle: $\mathcal{A}$ queries on $(id, \mathcal{R}, t)$. $\mathcal{B}$ retrieves $tk_{id}$ and $ku_t$. $\mathcal{B}$ runs the transformation key generation oracle on $(id, \mathcal{R})$ if $tk_{id}$ is not available, and key updating materials generation oracle on $t$ if $ku_t$ is not available. $\mathcal{B}$ returns $\perp$ if $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \emptyset$; otherwise, there is exists a node $\theta$ such that $\mathsf{Path}(id) \cap \mathsf{KUNodes}(st, rl, t) = \theta$. Let $\mathcal{V} \in [\ell]$ be the set of all indices $i$ for $t[i] = 0$. $\mathcal{B}$ returns $utk_{id,t} = ((id, \mathcal{R}, t), utk_1, utk_2, \{utk_{3,\tau}, utk_{4,\tau}\}_{\tau \in [k]}, utk_5)$ as $utk_1 = pk_{id}^\alpha w^r \cdot (u_0 \prod_{i \in \mathcal{V}} u_i)^{\bar{r}}, utk_2 = g^r, utk_{3,\tau} = g^{r_\tau}, utk_{4,\tau} = (u^{\mathcal{R}_\tau} h)^{r_\tau} v^{-r}, utk_5 = g^{\bar{r}}$.

Revocation oracle: $\mathcal{A}$ queries on $(id, t)$. $\mathcal{B}$ updates the revocation list $rl \leftarrow rl \cup (id, t)$.

Encryption key generation oracle: Parse $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$. Because of $\mathcal{S} \not\models \mathbb{S}^*$, there exists $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)^\top \in \mathbb{Z}_p^n$ such that $\lambda_1 = -1$ and $\mathbb{N}_i^* \lambda_i = 0$ for all $i \in \mathcal{I}$ for $\mathcal{I} = \{i | i \in [\ell] \wedge \rho^*(i) \in \mathcal{S}\}$. $\mathcal{B}$ then picks $\tilde{s} \in \mathbb{Z}_p$ and implicit sets the random term $s$ as $s = \tilde{s} + \lambda_1 a^q + \lambda_2 a^{q-1} + \cdots + \lambda_n a^{q+1-n} = \tilde{s} + \sum_{i \in [n]} \lambda_i a^{q+1-i}$. $\mathcal{B}$ computes $tk_1$ and $tk_2$ as $ek_1 = g^\beta w^s = g^{a^{q+1}} g^{\tilde{b}} g^{a\tilde{s}} \prod_{i \in [n]} g^{\lambda_i a^{q+2-i}}, ek_2 = g^s = g^{a\tilde{s}} \prod_{i \in [n]} g^{\lambda_i a^{q+2-i}}$. For all $\tau \in [k]$, it generates the additional components $ek_{3,\tau} = g^{s_\tau}$ and $ek_{4,\tau} = (u^{\mathcal{S}_\tau} h)^{s_\tau} v^{-s}$, where $v^{-s} = v^{-\tilde{r}} \prod_{i \in [n]} (g^{a^{q+1-i}})^{-\tilde{v}\lambda_i} \prod_{(i,j,k) \in [n,\ell,n], i \neq k} (g^{a^{q+1+k-i}/b_j})^{-\lambda_i \mathbb{N}^*_{j,k}} \prod_{(i,j) \in [n,\ell]} g^{-\lambda_i \mathbb{N}^*_{j,i} a^{q+1}/b_j}$. $\mathcal{B}$ randomly picks $\tilde{s} \in \mathbb{Z}_p$ and compute $s_\tau = \tilde{s}_\tau + s \cdot \sum_{i' \in [\ell], \pi^*(i') \notin \mathcal{S}} b_{i'}/(\mathcal{S}_\tau - \pi^*(i')) + \sum_{(i,i) \in [n,\ell], \pi^*(i') \notin \mathcal{S}} \theta_i b_{i'} a^{q+1-i}/(\mathcal{S}_\tau - \pi^*(i'))$. For the component $(u^{\mathcal{S}_\tau} h)^{s_\tau}$, it computes $(u^{\mathcal{S}_\tau} h)^{s_\tau} = (u^{\mathcal{S}_\tau} h)^{\tilde{s}_\tau} \cdot (ek_{3,\tau}/g^{\tilde{s}})^{\tilde{u}\mathcal{S}_\tau + \tilde{h}} \cdot \prod_{(i',j,k) \in [\ell,\ell,n], \pi^*(i') \notin \mathcal{S}} g^{\mathcal{S}_\tau - \pi^*(j) \mathbb{N}^*_{j,k} b_{i'} a^k / (\mathcal{S}_\tau - \pi^*(i'))b_j^2} \cdot \prod_{(i,i',j,k) \in [n,\ell,\ell,n], \rho^*(i') \notin \mathcal{S}} g^{(\mathcal{S}_\tau - \pi^*(j))/(\mathcal{S}_\tau - \pi^*(i'))b_j^2} \cdot \prod_{(i,i',j,k) \in [n,\ell,\ell,n], \rho^*(i') \notin \mathcal{S}} g^{(\lambda_i \mathbb{N}^*_{j,k} b_{i'} a^{q+1+k-i}/(\mathcal{S}_\tau - \pi^*(i'))b_j^2}$. It returns $ek_{id} = (\mathcal{S}, ek_1, ek_2, \{ek_{3,\tau}, ek_{4,\tau}\}_{\tau \in [k]})$.

Encryption oracle: $\mathcal{A}$ issues encryption queries on $(\hat{\mathcal{S}}, t\mathbb{R}, m)$. $\mathcal{B}$ randomly chooses a vector $\boldsymbol{x} = (\phi, x_2, ..., x_n)^\top \in \mathbb{Z}_p^{n \times 1}$ and computes $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_l)^\top = \mathbb{M}\boldsymbol{x}$. $\mathcal{B}$ randomly chooses terms $\phi_1, \phi_2, ..., \phi_l \in \mathbb{Z}_p$ and computes $c_0 = m \cdot e(g,g)^{\alpha\phi}, c_1 = g^\phi, c_{2,\tau} = w^{\lambda_\tau} v^{\phi_\tau}, c_{3,\tau} = (u^{\rho(\tau)} h)^{-\phi_\tau}, c_{4,\tau} = g^{\phi_\tau}$. Let $\tilde{\mathcal{V}} \in [\ell]$ be the set of all indices $i$ for $\tilde{t}[i] = 0$. It encodes the time $t$ to $\tilde{t}$ via $\mathsf{TEncode}(t, \mathcal{T})$ and computes $\tilde{c} =$

$(u_0 \prod_{i\in\mathcal{V}} u_i)^{\phi}$. For the rest part of the ciphertext, the simulation depends on the relationship between $\mathcal{S}$ and $\mathbb{S}$. If $\mathcal{S} \not\models \mathbb{S}$. For $\mathcal{S} \models \mathbb{S}$, $\mathcal{B}$ randomly picks $\tilde{y}_2, \tilde{y}_3, ..., \tilde{y}_n \in \mathbb{Z}_p$ and implicit have $\boldsymbol{y} = (s, sa+\tilde{y}_2, sa^2+\tilde{y}_3, ..., sa^{n-1}+\tilde{y}_n)^{\top}$. $\mathcal{B}$ simulates $\boldsymbol{\lambda} = \mathbb{M}^*\boldsymbol{y}$ for each row $\tau \in [\ell]$ as $\lambda_\tau = \sum_{i\in[n]} \mathbb{M}^*_{\tau,i} sa^{i-1} + \sum_{i=2}^{n} \mathbb{M}^*_{\tau,i}\tilde{y}_i = \sum_{i\in[n]} \mathbb{M}^*_{\tau,i} sa^{i-1} + \tilde{\lambda}_\tau$, where $\tilde{\lambda}_\tau = \sum_{i=2}^{n} \mathbb{M}^*_{\tau,i}\tilde{y}_i$ are known to $\mathcal{B}$. $\mathcal{B}$ computes $\hat{c}_1 = g^s$, then implicitly sets $t_\tau = -sb_\tau$ and computes $\hat{c}_{2,\tau} = w^{\lambda_\tau} \cdot (g^{sb_\tau})^{-\tilde{v}} \cdot \prod_{(j,k)\in[\ell,n], j\neq\tau} (g^{sa^k b_\tau / b_j})^{\mathbb{M}^*_{j,k}}$, $\hat{c}_{3,\tau} = (g^{sb_\tau})^{-(\tilde{u}\rho^*(\tau))+\tilde{h}} \cdot \prod_{(j,k)\in[\ell,n], j\neq\tau} (g^{sa^k b_\tau / b_j^2})^{-(\rho^*(\tau)-\rho^*(j))\mathbb{M}^*_{j,k}}$. $\mathcal{B}$ randomly picks $\kappa \in \mathbb{Z}_p$ and computes $\hat{c}_4 = g^{\kappa}$. Let $\ddot{c}$ be $\ddot{c} = c_0 \| c_1 \| c_{2,1} \| ... \| c_{2,l} \| c_{3,1} \| ... \| c_{3,l} \| c_{4,1} \| ... \| c_{4,l} \| \hat{c}_1 \| \hat{c}_{2,1} \| ... \| \hat{c}_{2,m} \| \hat{c}_{3,1} \| ... \| \hat{c}_{3,m} \| \hat{c}_4$. $\mathcal{B}$ computes $\hat{c}_0 = ek_1 \cdot w^{\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa} = g^{\beta} w^{s+\hat{s}} \cdot \mathcal{H}(\ddot{c})^{\kappa}$. $\mathcal{B}$ returns $c = ((\mathcal{S}, \mathbb{R}), c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau\in[l]}, \tilde{c}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}, \hat{c}_{4,\tau}\}_{\tau\in[m]})$.

**Forge**: $\mathcal{A}$ outputs a ciphertext $c^* = ((\mathcal{S}, \mathbb{R}), c_0, c_1, \{c_{2,\tau}, c_{3,\tau}, c_{4,\tau}\}_{\tau\in[l]}, \tilde{c}, \hat{c}_0, \hat{c}_1, \{\hat{c}_{2,\tau}, \hat{c}_{3,\tau}\}_{\tau\in[m]}, \hat{c}_4)$ to $\mathcal{A}$. Parse $\mathbb{S}^* = (\mathbb{N}^*, \pi^*)$ with $\mathbb{N} \in \mathbb{Z}_p^{\ell\times n}$ and $\pi : [\ell] \to \mathbb{Z}_p$. $\mathcal{B}$ randomly picks $\tilde{y}_2, \tilde{y}_3, ..., \tilde{y}_n \in \mathbb{Z}_p$ and implicit have $\boldsymbol{y} = (s, sa+\tilde{y}_2, sa^2+\tilde{y}_3, ..., sa^{n-1}+\tilde{y}_n)^{\top}$. Let $\mathcal{I}$ be $\mathcal{I} = \{i : \pi(i) \in \mathcal{S}\}$ for $\{\omega_i \in \mathbb{Z}_p\}_{i\in\mathcal{I}}$ $s.t.$ $\sum_{i\in\mathcal{I}} \omega_i \mathbb{N}_i = (1, 0, ..., 0)$. $\mathcal{B}$ computes $A = e(\hat{c}_0, g^s)$, $B = \prod_{i\in\mathcal{I}} (e(\hat{c}_1, w^{\mu_i} v) \cdot e(\hat{c}_{2,\tau}, (u^{\pi(i)} h)^{-1}) \cdot e(\hat{c}_{3,\tau}, g))^{\omega_i} \cdot e(\hat{c}_4, \mathcal{H}(\tilde{c}))$ and $A/B = e(g,g)^{\beta s} = e(g^a, g^{a^q})^s \cdot e(g,g)^{\tilde{b}s}$. Because $\mathcal{B}$ knows $\tilde{b}$ and $g^s$, we have $e(g,g)^{a^{q+1}s}$ as the returning value. Hence, there is no abort during the simulation and $\mathcal{B}$ have a non-negligible advantage to break the computational $q$-1 assumption. Therefore, we have $\mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathsf{SRB\text{-}ABE},\mathcal{A}}(1^{\lambda}, \mathcal{N}, \mathcal{T}) = \epsilon$, where $\epsilon$ is the probability to break the computational $q$-1 assumption.