5-2022

Sanitizable access control system for secure cloud storage against malicious data publishers

Willy SUSILO
University of Wollongong

Peng JIANG
Beijing Institute of Technology

Jianchang LAI
Fujian Normal University

Fuchun GUO
University of Wollongong

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

See next page for additional authors

Sanitizable Access Control System for Secure Cloud Storage Against Malicious Data Publishers

Willy Susilo, *Fellow, IEEE*, Peng Jiang, Jianchang Lai, Fuchun Guo, Guomin Yang, *Senior Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

Abstract—Cloud computing is considered as one of the most prominent paradigms in the information technology industry, since it can significantly reduce the costs of hardware and software resources in computing infrastructure. This convenience has enabled corporations to efficiently use the cloud storage as a mechanism to share data among their employees. At the first sight, by merely storing the shared data as plaintext in the cloud storage and protect them using an appropriate access control would be a nice solution. This is assuming that the cloud is fully trusted for not leaking any information, which is impractical as the cloud is owned by a third party. Therefore, encryption is mandatory, and the shared data will need to be stored as a ciphertext using an appropriate access control. However, in practice, some of these employees may be malicious and may want to deviate from the required sharing policy. The existing protection in the literature has been explored to allow only legitimate recipients to decrypt the contents stored in the cloud storage, but unfortunately, *no existing work* deals with issues raised due to the presence of malicious data publishers. Malicious data publishers construct data following the given policy, but the ciphertexts can actually be decrypted by unauthorized users without valid keys, or simply, anyone else who is unauthorized. The impact of the involvement of malicious data publishers is detrimental, as it may damage intellectual properties from the corporations. Therefore, it remains an elusive research problem on how to enable a sound approach to resolve the issue when malicious data publishers are involved in the system, which is a very practical question. In this work, we present *a new direction of research* that can cope with the presence of malicious data publishers. We resolve the aforementioned problem by proposing the notion of Sanitizable Access Control System (SACS), which is designed for a secure cloud storage that can also resist against malicious data publishers. We define the threat model and its formal security model, as well as its design and scheme which is based on q-Parallel Bilinear Diffie-Hellman Exponent Assumption. We provide the security proof of our construction as well as its performance analysis. We believe that this work has opened a new area of research which has never been explored before, even though it is very practical. Therefore, this work will enhance the adoption of secure cloud storage in practice.

Index Terms—Secure cloud storage, access control, sanitizable, malicious data publishers

1 Introduction

The emergence of cloud storage technology has greatly influenced enterprise operations and the adoption of cloud technology has been one of the biggest changes of the digital age. Cloud storage offers low-cost solutions, which would be perfect for business, such as Small and Medium-sized Enterprises (SMEs). The existence of cloud storage enables business corporations to conveniently share their data among their employees. These data are supposed to be used solely by the employees of those corporations, since those may be related to their intellectual properties. At the first sight, by merely storing the data as plaintext in the cloud storage and protecting them by using an appropriate access control would be a sufficient solution. This is based on the assumption that the cloud is fully trusted and will not leak those data, which is impractical, as the cloud is owned by a third party. Therefore, it is essential to employ an encryption mechnism and store the data as a ciphertext in the cloud to avoid the data leakage.

The existing body of work in the literature make use of the notion of Attribute-based Encryption (ABE) [1], [2], [3] to enable such an unauthorized prevention by protecting the data with an appropriate access policy. Anyone who is equipped with a valid decryption key satisfying the access policy will be able to decrypt the data correctly. It implies that the data will be stored as a ciphertext instead of a plaintext in the cloud. This kind of protection *only* considers data privacy when data publishers are honest and follow the encryption algorithm.

Unfortunately, in practice, some of these employees may be malicious and they intentionally attempt to leak the contents of those data to unauthorized recipients, such as competing business corporations. These malicious employees may even would like to publish some sensitive contents and store them in the cloud, but also enable other unauthorized users to retrieve it - hence constituting a *malicious data*

- *Willy Susilo, Fuchun Guo, and Guomin Yang are with the Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: {wsusilo, fuchun, gyang}@uow.edu.au.*
- *Peng Jiang is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: pennyjiang0301@gmail.com.*
- *Jianchang Lai is with the School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350007, China. E-mail: jl967@uowmail.edu.au.*
- *Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 188065. E-mail: robertdeng@smu.edu.sg.*

Manuscript received 21 Feb. 2020; revised 22 Jan. 2021; accepted 2 Feb. 2021. Date of publication 9 Feb. 2021; date of current version 13 May 2022. (Corresponding authors: Willy Susilo and Peng Jiang.) Digital Object Identifier no. 10.1109/TDSC.2021.3058132

publisher. It is unfortunate that the approach based on ABE is not sound due to *malicious data publishers*, who encrypt data in a pernicious way. Here, the malicious data publisher constructs encrypted data following the given policy, but the ciphertexts can actually be decrypted by unauthorized users without valid keys. In practice, malicious data publishers refer to staff or virus-infected computers of a company that intend to leak internal sensitive information. As an example, a malicious data publisher may want to leak new product designs or commercial secrets that are only supposed to be accessed by certain people. The impact of the involvement of malicious data publishers is detrimental. In the above setting, malicious data publishers will construct ciphertexts containing copyrighted materials that seemingly adhere the required security access policy set by the organization. Nevertheless, the supposedly encrypted file can actually be decrypted by anyone illegally without any valid decryption key. Therefore, our main goal is to *achieve data privacy when data publishers are malicious and they do not follow the encryption algorithm accordingly.*

We aim to propose a very practical notion, called *Sanitizable Access Control System*, or simply SACS, which is designed for the cloud storage to resist against malicious data publishers. SACS enables a flexible access control for *both* data publishers and data receivers. As in ABE, SACS allows any valid receivers who are equipped with private keys satisfying the access policy to decrypt the ciphertext. However, SACS is equipped with sanitizing capability, which prevents malicious data publishers from generating ciphertexts that will be decryptable without any valid private keys. Although the malicious data publishers can maliciously generate ciphertexts which can be decrypted by anyone, the sanitizer will transform these ciphertexts into new ciphertexts which will only be decryptable by valid private key holders. We present our architecture as well as our scheme to achieve the above concept to build SACS. Furthermore, we also present an implementation of SACS.

Organization. The rest of this paper is organized as follows. In Section 2, we review the existing work that is related to SACS. In Section 3, we present SACS overview, which includes its system architecture, design goals, threat model and a brief workflow. The description of Sanitized ABE is provided in Section 4. Section 5 instantiates an SACS design, which is built from Sanitized ABE. Its security analysis and performance evaluation are provided in Sections 6 and 7. Finally, Section 8 concludes this work.

2 Related Work

In this section, we review some closely related work in the literature.

Access Control. Access control is able to guarantee data security in cloud storage systems. This has attracted much attention from academia and industry. IBM developed the capability-based model and systematic approaches to improve access control in the cloud services [4], [5]. Cryptographic primitives have been proposed for enabling access control on encrypted storage, such as broadcast encryption [6], proxy re-encryption [7], role-based encryption [8] and attribute-based encryption [1]. For the reason of security, scalability and flexibility, ABE has been regarded as one of the most suitable technologies for enabling access control [9]. Users whose attributes satisfying the access policy are able to access the plain data. ABE is mainly classified into two complementary forms, key-policy ABE [10] and ciphertext-policy ABE [2], [3]. In CP-ABE, attributes are used to describe the user's attributes and access policies over these attributes are attached to the encrypted data. Due to its flexibility and expressiveness, CP-ABE has more applications in cloud storage access control [11], [12], [13]. In this paper, we borrow CP-ABE as a component into our SACS design.

Sanitizable Signatures. Sanitizable signatures (SS's) are proposed by Ateniese *et al.* [14] to allow controlling modifications of signed messages without invalidating the signature. SS is a variant of digital signatures where a designated party (the sanitizer) can update admissible parts of a signed message. Brzuska *et al.* [15] introduced most of security notions in SS's. Fehr and Fischlin [16] proposed sanitizable signcryption to hide the message-signature pair from the sanitizer. Many SS schemes [17], [18], [19], [20] have been proposed to satisfy different properties. SS provides the foundation to the concept of sanitization in encryption.

Access Control Encryption. Access Control Encryption (ACE) [21] was introduced to provide fine-grained access control. ACE gives different rights to different users not only in terms of which messages they are allowed to receive, but also which messages they are allowed to send. Here, the important property of *Sanitization* is included. ACE can prevent corrupted senders from sending information to corrupted receivers. In ACE, the sanitizer uses its sanitizer key from the authority to execute a specific randomized algorithm on the incoming ciphertext and thereafter passes the result to a database server or the receivers. By sanitizing, ACE ensures that no matter what the corrupted sender sends, what the receiver receives looks like a random encryption of a random message. In our SACS, the sanitizing operation does not need a sanitizer key from the authority. Only the valid receiver, who is assigned a valid private key by the authority, can recover the message.

3 SACS Overview

SACS aims to provide a flexible access control in terms of both data receivers and data publishers. It enables only valid data receivers, who hold private keys from the authority, to access the plain data. With ciphertext sanitizing, SACS prevents malicious data publishers from issuing information that can retrieve decryption key without valid private key generated from the trusted center, e.g., encryption key, such that any invalid receivers cannot access plain data even if they have encryption key. Our SACS does not use the notions of the data owner and the user. We call the entity who sends the cipher data as the data publisher and the entity who recovers the plain data as the receiver. We still consider cloud as the data storage platform.

3.1 System Architecture

We show SACS architecture in Fig. 1, where five kinds of independent entities are involved. They are the authority, the data publisher, the sanitizer, the receiver and the cloud server.

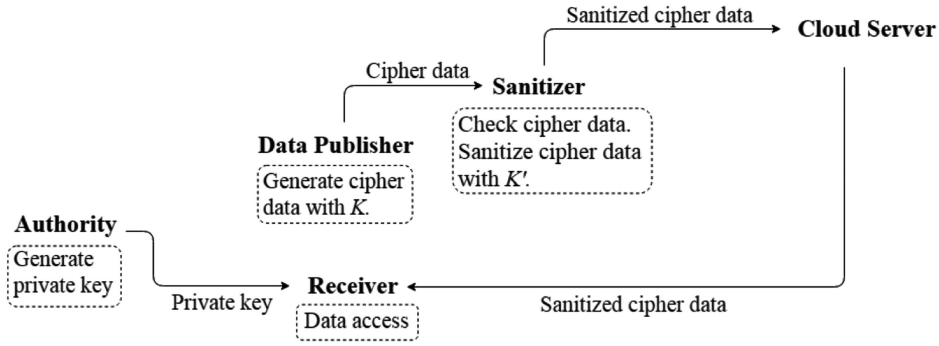


Fig. 1. SACS architecture.

- *The authority* manages and maintains the whole system. In SACS, we regard the authority as a trusted entity who holds the master secret key. The authority issues a unique private key to each receiver who registers into this system. Without loss of generality, we assume that the authority neither colludes with any other entities nor is compromised.
- *The data publisher* owns the plain data. He/she encrypts its plain data with an encryption key (e.g., K) and sets an access policy to deal with the encryption key. Then the data publisher sends the encrypted data (or *cipher data*) to the sanitizer. Actually, the publisher relies on this access policy to conduct data access control. Publishers are either honest or malicious. Both honest and malicious publishers execute the encryption operation on the plain data, but a malicious publisher might have extra behaviors, such as distributing the encryption key to some non-registered receivers. This incurs a failure of access control since some receivers can access the data without valid private key.
- *The sanitizer* is introduced to transform the original cipher data into the sanitized cipher data. Once getting cipher data from the data publisher, the sanitizer is instructed to do some specific processing on these cipher data. The processing includes two parts. One is to check whether the cipher data is under the claimed access policy and the other is to sanitize the cipher data with its encryption key K'. Then the sanitizer sends the sanitized cipher data to the cloud server for storage. Such a sanitization operation on the cipher data is to prevent malicious publishers and invalid data access. The sanitizer is an honest party, which means it just executes the sanitization following the sanitizing algorithm but no malicious operations, such as replacing/modifying the cipher data. The sanitizer learns nothing about the plain data.
- *The receiver* wants to access the plain data. He/she can freely download the cipher data that he/she is interested in from the cloud server. Prior to accessing the data, the receiver must register into the system and ask for a private key from the authority. When the registered receiver owns conditions satisfying the access policy, it is valid. Only valid receivers can access the plain data from the data publisher. Receivers will share neither their private keys nor the decrypted plain data with other entities. Here, we note that each receiver is unique.
- *The cloud server* provides a platform for cipher data storage. The cipher data stored in the cloud server can be acquired by any receivers. The cloud server just receives cipher data from the sanitizer and sends the cipher data to the receiver, while executes no computation operation. The cloud server will behave maliciously, e.g., delete the cipher data. Whether the cloud server is curious or not gives no effect on the security of SACS.

3.2 Design Goals

SACS aims to achieve the following design goals.

- *Sanitization.* SACS allows to sanitize the cipher data. The sanitization is to prevent any data publisher from malicious behaviors, which incurs invalid access to plain data. SACS makes an enhancement to data privacy in terms of publishers and provides secure data storage against malicious data publishers.
- *Integrity.* SACS allows the sanitizer to check whether the cipher data from the data publisher is indeed under the claimed access policy. This integrity check ensures the validity of the received cipher data. We requires such an operation to be prior to sanitizing.
- *Stronger Access Control.* SACS ensures stronger access control over encrypted data. It guarantees that only valid receivers can access the plain data while any receiver, even if he/she has an encryption key from the malicious data publisher, cannot decrypt the sanitized cipher data correctly.

3.3 Threat Model

In SACS, we consider a necessary sanitizer who controls the communication between the data publisher and the cloud server. All outgoing communication of data publishers must pass through this sanitizer. We consider a malicious data publisher who can illegally distribute the encryption key to invalid (even non-registered) receivers. We also consider that all of receivers in this system attempt to access the plain data.

The goal of an adversary in this system is to decrypt the sanitized cipher data and obtain the plain data. We give some illustrations about the adversary capability. First, it can have full access to sanitized cipher data in the cloud but have no right to obtain the original cipher data generated by

the data publisher. Second, it is allowed to register into this system like a receiver but cannot be assigned a valid private key corresponding to the specified access policy. Third, it can submit a requirement to a malicious data publisher and ask for his/her encryption key. Further, it is able to know the decryption algorithm and execute kinds of attacks to obtain the plain data from the given sanitized cipher data.

We also need to make some assumptions. One is that the public information (such as the system public parameters, the attribute space, the global identity space) are freely available in this system, while the secret information (such as master secret key, private keys and random numbers chosen by entities) are not available. One is that the authority is non-compromised. Another is that all entities honestly execute its embedded algorithm and no collusion exists between any two entities.

We do not consider the case where the cipher data generated from the data publisher is wrong, that is, all the receivers cannot decrypt the cipher data. This assumption is reasonable as it loses the encryption meaning if all the receivers cannot decrypt the cipher data.

3.4 Protocol Workflow

A SACS protocol mainly consists of five phases: System Initialization, Receiver Registration, Cipher Data Publishing, Cipher Data Sanitizing and Data Access. The protocol workflow is described as follows and its detailed construction will be presented in Section 5.

System Initialization. This phase is executed by the authority. The authority establishes the system by generating all system parameters. The public parameters are publicly available to all of entities while the master secret key is only kept by the authority.

Receiver Registration. This phase is mainly executed by the authority. In SACS system, a receiver sends a registration request to the authority and thereafter the authority returns a private key, which is used to decrypt the sanitized cipher data. Non-registered receivers impossibly finish a correct decryption while registered receivers have a certain possibility to retrieve the data with some access conditions. Our SACS design adopts CP-ABE, where a private key is associated with a set of attributes.

Cipher Data Publishing. This phase is mainly executed by the data publisher. For plain data confidentiality, the data publisher first leverages the encryption algorithm over the plain data and generates the cipher data. The data publisher combines both symmetric encryption and CP-ABE in the encryption. The plain data is directly encrypted with some symmetric encryption by using a symmetric key (i.e., the encryption key) and the symmetric key is encrypted with CP-ABE under a specified access policy. Such cipher data with a built-in access policy guarantees only registered receivers whose attribute set satisfies the specified access policy can get the symmetric key and thereby retrieve the plain data. After that, the cipher data is delivered to the sanitizer.

Cipher Data Sanitizing. This phase is mainly executed by the sanitizer. The sanitizer carries out a sanitization for the cipher data from the data publisher, which prevents malicious data publishers from distributing the encryption key and invalid receivers from accessing the plain data. This phase is split into two sub-phases, checking and sanitizing. Checking is to ensure the cipher data is indeed under the claimed access policy and sanitizing is to transfer the cipher data to the sanitized cipher data. The cipher data sanitizing further guarantees the access right of valid receivers, which means only valid receiver can access the plain data. After that, the sanitized cipher data is uploaded to the cloud server for storing and sharing.

Data Access. This phase is mainly executed by the receiver. The receiver downloads the sanitized cipher data from the cloud server and then decrypts with the decryption algorithm. Only holding a valid private key, the receiver can decrypt to retrieve the plain data. That is, if his/her attribute set in the private key satisfies the access policy in the sanitized cipher data, the encryption key can be obtained and then the plain data can be decrypted with the symmetric decryption.

4 Sanitized Attribute-Based Encryption

The SACS is based on a notion of *Sanitized Attribute-based Encryption (SABE)*. SABE allows to sanitize the ciphertext and prevents malicious encryptors, such that only valid private keys can be used to obtain the message. This section gives formal algorithm definitions and security model.

4.1 Algorithm Definitions

Definition 1. *A sanitized (ciphertext-policy) ABE scheme consists of the following algorithms.*

Setup(λ, U). The setup algorithm takes as input a security parameter λ and the number of universal attributes U. It returns system parameters Params and a master secret key msk.

KeyGen(S, msk, Params). The key generation takes as input an attribute set S, the master secret key msk and the system parameters Params. It returns the private key sk_S of S.

Encrypt($P, \mathcal{M}, \mathsf{Params}$). The encryption algorithm takes as input an access policy P, a message $\mathcal{M}$ and the system parameter Params. It returns a ciphertext $CT = \mathsf{Enc}[P, \mathcal{M}, \mathsf{Params}]$.

Sanitize(CT, Params). The sanitization algorithm takes as input the system parameter Params and a ciphertext CT. It returns a sanitized ciphertext $CT' = \mathsf{San}[\mathsf{Params}, CT]$.

Decrypt($CT', sk_S, \mathsf{Params}$). The decryption algorithm takes as input a sanitized ciphertext CT' for $P, \mathcal{M}$ and a private key sk_S for S. If the attribute set S satisfies the access policy P, the decryption algorithm returns $M = \mathsf{Dec}[CT', sk_S, \mathsf{Params}]$. Otherwise, it returns $\perp$.

Correctness. We give the correctness of Sanitized CP-ABE as follows. For all $\mathsf{Params}, msk, S, P$ such that the attribute set S satisfies the access policy P, if $sk_S \leftarrow$ **KeyGen** $(S, msk, \mathsf{Params})$, $CT \leftarrow$ **Encrypt**$(P, \mathcal{M}, \mathsf{Params})$ and $CT' \leftarrow$ **Sanitize**(CT, Params) , we have $M =$ **Decrypt**$(CT', sk_S, \mathsf{Params})$.

4.2 Security Model

For SABE, we consider the indistinguishability-based security. The adversary outputs an access structure and two

distinct ciphertexts for challenge, and tells apart which ciphertext is used to sanitize randomly chosen by the challenger. During this interaction, the adversary is allowed to query the private key of any attribute set as needed under the restriction that the queried attribute set does not satisfy the access structure. The security model is defined by a security game played between an adversary and a challenger below.

Setup. The challenger runs the system setup algorithm to generate the system public parameters Params and sends Params to the adversary.

Phase 1. In this phase, the adversary can issue private key query on any attribute set S_i as needed. The challenger runs the key generation algorithm to generate the corresponding private key and returns the result to the adversary.

Challenge. Once the adversary decides phase 1 is over, it outputs an access structure (M^*, ρ^*), two distinct ciphertexts CT_0, CT_1 for challenge. We require that none of the attribute sets S_i from phase 1 satisfies the access structure (M^*, ρ^*). Then the challenger randomly picks a bit $\mu \in \{0, 1\}$ and runs the ciphertext sanitizing algorithm to sanitizes the ciphertext CT_μ under (M^*, ρ^*) after checking the validity of CT_μ, and generates a challenge ciphertext CT^* which is given to the adversary.

Phase 2. In this phase, the adversary can issue more private key queries on attribute sets with the restriction established in the challenge phase. The challenger responds the same as phase 1.

Guess. Finally, the adversary outputs its guess μ' of μ.

We define the advantage of an adversary $\mathcal{A}$ in this game as

$$\mathsf{Adv}_{\mathcal{A}} = \left| \Pr[\mu' = \mu] - \frac{1}{2} \right|,$$

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

The security of our proposed scheme is conducted in a weaken security model, namely selective security. In the selective security model, the adversary must commit the access structure (M^*, ρ^*) before seeing the system public parameters, and the adversary is not allowed to make private key query on any attribute set which satisfies the access structure (M^*, ρ^*).

5 SACS Design

This section presents the SACS design in details. The SACS is built on top of SABE, where LSSS-based CP-ABE [3] is adopted. The data publisher encrypts the plain data with a random encryption key, which is encrypted with CP-ABE. To guarantee the integrity, the cipher data needs to be verified to be under the claimed access policy before sanitizing. A main challenge is that the traditional ABE ciphertext form (with only α-associated elements) cannot support the sanitization. To address it, a trick in construction is to add another element ($C_2 = e(g, g)^{\beta s}$ in the cipher data) corresponding to the α-associated element. This α-associated element is used to encrypt the encryption key K ($C_1 = K \cdot e(g, g)^{\alpha s}$). Then the sanitizer can chooses s', K' to sanitize the cipher data and a valid receiver can decrypt the sanitized cipher data by reconstructing K, K'. The used notations for SACS are summarized in Table 1 and the construction details are presented as follows.

TABLE 1
Notations Used in SACS

Notation	Description
Params	System public parameters.
msk	Master secret key, kept by the authority.
S	An attribute set.
sk_S	Private key for S.
$\mathcal{M}$	Plain data.
K	Encryption key chosen by the data publisher.
(M, ρ)	LSSS access structure.
CT	Cipher data
psk	Partial private key only used in **Checking** phase.
K'	Encryption key chosen by sanitizer in **Sanitizing**.
CT'	Sanitized cipher data.

5.1 System Initialization

The system is established by an authority. The authority first chooses a pairing group $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, e, p)$ [22] and the number of universal attributes U, and randomly picks group elements $g, h_1, h_2, \ldots, h_U$ from $\mathbb{G}$. It also chooses random exponents $a, \alpha, \beta \in \mathbb{Z}_p$, computes $g^a, e(g, g)^\alpha, e(g, g)^\beta$ and sets the system master secret key as $msk = (g^\alpha, g^\beta)$. It then chooses a pseudorandom generator PRG and publishes the system public parameters Params as

$$\mathsf{Params} = (\mathbb{PG}, g, h_1, h_2, \ldots, h_U, g^a, e(g, g)^\alpha, e(g, g)^\beta, PRG).$$

Here, we adopt the pseudorandom generator that is usually used to encrypt large-size message in the *Cipher Data Publishing* phase. PRG is constructed from the cryptographic hash functions in the form of $PRG(K) = H(K, 1)||H(K, 2)||\cdots||H(K, n)$, where H is a hash function. With such a form, we can see PRG as the random oracle in the Section 6.

5.2 Receiver Registration

In this phase, receivers can register to the system and receives the correspond private keys from the authority. Suppose a receiver has an attribute set $S \subseteq \{1, 2, \ldots, U\}$.

The authority first checks whether the receiver has these attributes indeed by artificial methods, such as certificate verification. Otherwise, taking as input $(S, msk, \mathsf{Params})$, the authority chooses two random exponents $t, t' \in \mathbb{Z}_p$ and creates the private key for S as

$$sk_S = (g^\alpha g^{at}, g^\beta g^{at'}, g^t, g^{t'}, h_x^t, h_x^{t'} : \forall x \in S).$$

5.3 Cipher Data Publishing

For a given data $\mathcal{M}$, the data publisher specifies an easy expressed monotone boolean formula as the access policy and turns it into an LSSS access structure (M, ρ) by following the terminology in [23]. M is an $m \times n$ matrix, where m is the scale of a specific attribute set and n is variable

depending on the monotone boolean formula definition and the LSSS turning method. The function ρ maps each row of M to a specific attribute, marked as $\rho(i) \in \{Att_1, \ldots, Att_U\}$. To encapsulate the encryption key, the data publisher chooses a vector $\vec{v} = (s, y_2, y_3, \ldots, y_n) \in \mathbb{Z}_p^n$, where $y_2, y_3, \ldots, y_n$ are randomly chosen. Each share θ_i can be calculated with $\theta_i = \vec{v} \cdot M_i \in \mathbb{Z}_p$, where M_i denotes the ith row of M.

The data publisher chooses an encryption key $K \in \mathbb{G}_T$ and $z_1, z_2, \ldots, z_m \in \mathbb{Z}_p$, and creates the corresponding cipher data $CT = \left(C_0, C_1, C_2, D_0, (D_{1i}, D_{2i})_{i=1}^m\right)$ as

$$C_0 = PRG(K) \oplus \mathcal{M},\ C_1 = K \cdot e(g,g)^{\alpha s},\ C_2 = e(g,g)^{\beta s},$$

$$D_0 = g^s, \left(D_{1i} = g^{a\theta_i} h_{\rho(i)}^{-z_i},\ D_{2i} = g^{z_i}\right), i = 1, 2, \ldots, m.$$

After generating the cipher data CT, the data publisher sends the cipher data CT to the sanitizer.

5.4 Cipher Data Sanitizing

After receiving the cipher data, the sanitizer performs the following two steps to sanitize the cipher data.

Step 1: Checking.

The goal of this step is to check whether the cipher data is generated under the access structure (M, ρ) as it claimed in the received cipher data. To do the verification, the sanitizer chooses a random exponent γ as the dummy master secret key and computes γ-relevant partial private key psk_S whose attribute set S satisfies the access structure as below:

$$psk_S = (g^\gamma g^{a\tilde{t}},\ h_x^{\tilde{t}},\ \ g^{\tilde{t}} : \forall x \in S),$$

where $\tilde{t}$ is a random from $\mathcal{Z}_p$ chosen by the sanitizer. Then it uses this private key to compute

$$\begin{aligned}
&\prod_{i\in I}\left(e(D_{1i}, g^{\tilde{t}}) e\left(D_{2i}, h_{\rho(i)}^{\tilde{t}}\right)\right)^{w_i} \\
&= \prod_{i\in I}\left(e\left(g^{a\theta_i} h_{\rho(i)}^{-z_i}, g^{\tilde{t}}\right) e\left(g^{z_i}, h_{\rho(i)}^{\tilde{t}}\right)\right)^{w_i} \\
&= \prod_{i\in I} e(g,g)^{a\tilde{t}\sigma_i w_i} \\
&= e(g,g)^{\sum_{i\in I} a\tilde{t}\sigma_i w_i} \\
&= e(g,g)^{a\tilde{t}s},
\end{aligned}$$

$$\frac{e(D_0, g^\gamma g^{a\tilde{t}})}{e(g,g)^{a\tilde{t}s}} = \frac{e(g,g)^{\gamma s} e(g,g)^{a\tilde{t}s}}{e(g,g)^{a\tilde{t}s}} = e(g,g)^{\gamma s}.$$

Finally, it checks whether the following equation holds:

$$e(D_0, g^\gamma) = e(g,g)^{\gamma s},$$

and performs *Step 2* cipher data sanitizing below if the answer is yes. Otherwise, the sanitizer rejects the cipher data. By this, the sanitizer can believe that the cipher data is valid generated under the claimed access policy.

Step 2: Sanitizing.

The sanitizer chooses a vector $\vec{v}' = (s', y_2', y_3', \ldots, y_n') \in \mathbb{Z}_p^n$, where $s', y_2', y_3', \ldots, y_n'$ are randomly chosen. Each share θ_i' can be calculated with $\theta_i' = \vec{v}' \cdot M_i \in \mathbb{Z}_p$, where M_i denotes the ith row of M. It chooses another encryption key $K' \in \mathbb{G}_T$, and randomly picks $z_1', z_2', \ldots, z_m' \in \mathbb{Z}_p$. It then computes

$$C_0' = PRG(K'),\ C_1' = e(g,g)^{\alpha s'},\ C_2' = K' \cdot e(g,g)^{\beta s'},$$

$$D_0' = g^{s'}, \left(D_{1i}' = g^{a\theta_i'} h_{\rho(i)}^{-z_i'},\ D_{2i}' = g^{z_i'}\right), i = 1, 2, \ldots, m.$$

Then it sanitizes CT and generates the sanitized cipher data $CT' = \left(V_0, V_1, V_2, V_3, (A_i, B_i)_{i=1}^m\right)$ as

$$\begin{aligned}
V_0 &= C_0 \oplus C_0' = PRG(K') \oplus PRG(K) \oplus \mathcal{M}, \\
V_1 &= C_1 \cdot C_1' = K \cdot e(g,g)^{\alpha(s+s')}, \\
V_2 &= C_2 \cdot C_2' = K' \cdot e(g,g)^{\beta(s+s')}, \\
V_3 &= D_0 \cdot D_0' = g^{s+s'},
\end{aligned}$$

together with for $i = 1, 2, \ldots, m$

$$A_i = D_{1i} \cdot D_{1i}' = g^{a\theta_i + a\theta_i'} h_{\rho(i)}^{-z_i - z_i'}.$$

$$B_i = D_{2i} \cdot D_{2i}' = g^{z_i + z_i'}.$$

The sanitized cipher data CT' will be sent to and stored in the cloud.

5.5 Data Access

Given a sanitized cipher data $CT' = (V_0, V_1, V_2, V_3, (A_i, B_i)_{i=1}^m)$ The receiver executes the decryption algorithm to access the plain data $\mathcal{M}$. The decryption mainly recovers K and K'. The former can be computed with $e(g,g)^{\alpha(s+s')}$, which is obtained by using private key $g^\alpha g^{at}$, and the latter can be computed with $e(g,g)^{\beta(s+s')}$, which is obtained by using private key $g^\beta g^{at}$. The concrete decryption processes are described as follows.

The receiver runs the decryption algorithm, which taking as input a sanitized cipher data CT' for access structure (M, σ) and a private key for a set S. Suppose that S satisfies the access structure and let $I \subset \{1, 2, \ldots, m\}$ be defined as $I = \{i : \rho(i) \in S\}$. Let $\{w_i \in \mathbb{Z}_p\}_{i\in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of any secret s according to M, then $\sum_{i\in I} w_i \theta_i = s$. Then the receiver computes

$$\begin{aligned}
&\prod_{i\in I}\left(e(A_i, g^t) e\left(B_i, h_{\rho(i)}^t\right)\right)^{w_i} \\
&= \prod_{i\in I}\left(e\left(g^{a\theta_i + a\theta_i'} h_{\rho(i)}^{-z_i - z_i'}, g^t\right) e\left(g^{z_i + z_i'}, h_{\rho(i)}^t\right)\right)^{w_i} \\
&= \prod_{i\in I} e(g,g)^{at(\theta_i + \theta_i')w_i} \\
&= e(g,g)^{\sum_{i\in I} at(\theta_i + \theta_i')w_i} \\
&= e(g,g)^{at(s+s')}, \\
&\prod_{i\in I}\left(e(A_i, g^{t'}) e\left(B_i, h_{\rho(i)}^{t'}\right)\right)^{w_i} \\
&= \prod_{i\in I}\left(e\left(g^{a\theta_i + a\theta_i'} h_{\rho(i)}^{-z_i - z_i'}, g^{t'}\right) e\left(g^{z_i + z_i'}, h_{\rho(i)}^{t'}\right)\right)^{w_i} \\
&= \prod_{i\in I} e(g,g)^{at(\theta_i + \theta_i')w_i} \\
&= e(g,g)^{\sum_{i\in I} at(\theta_i + \theta_i')w_i} \\
&= e(g,g)^{at'(s+s')},
\end{aligned}$$

$$\frac{e(V_3, g^{\alpha} g^{at})}{e(g,g)^{at(s+s')}} = \frac{e(g,g)^{\alpha(s+s')} e(g,g)^{at(s+s')}}{e(g,g)^{at(s+s')}} = e(g,g)^{\alpha(s+s')},$$

$$\frac{e(V_3, g^{\beta} g^{at'})}{e(g,g)^{at'(s+s')}} = \frac{e(g,g)^{\beta(s+s')} e(g,g)^{at'(s+s')}}{e(g,g)^{at'(s+s')}} = e(g,g)^{\beta(s+s')}.$$

After obtaining $e(g,g)^{\alpha(s+s')}$, $e(g,g)^{\beta(s+s')}$, the receiver can retrieve the encryption keys K and K' by computing

$$K = \frac{V_1}{e(g,g)^{\alpha(s+s')}},$$

$$K' = \frac{V_2}{e(g,g)^{\beta(s+s')}},$$

and get the message as

$$\mathcal{M} = V_0 \oplus PRG(K) \oplus PRG(K').$$

From our setting, after sanitizing, the cipher data actually is encrypted by two encryption keys K and K′, where K′ is chosen by the sanitizer secretly. As the sanitizer is full trusted, K′ can only be retrieved via decryption by providing a valid private key. Therefore, even the malicious data publisher reveals the decryption key K in the cipher data, the non-authorized receivers cannot retrieve the plain data. Our approach can efficiently prevent the malicious data publisher from leaking the plain data.

6 Security Analysis

In this section, we give a formal security proof of our proposed scheme. Before starting to analyze the security, we show the complexity assumption which the security of our scheme based on.

6.1 Complexity Assumption

The security of the proposed scheme is derived from the below complexity assumption.

Decisional q-Parallel Bilinear Diffie-Hellman Exponent Assumption [3]. Let $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, e, p)$ be a pairing group, $a, s, b_1, b_2, \ldots, b_q \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}$. If an adversary is given $\vec{Y} =$

$$g, g^s, g^a, \ldots, g^{(a^q)}, g^{(a^{q+2}),\ldots,g^{(a^{2q})}}$$
$$\forall_{1\le j\le q} \quad g^{s\cdot b_j}, g^{a/b_j}, \ldots, g^{(a^q/b_j)}, g^{(a^{q+2}/b_j),\ldots,g^{(a^{2q}/b_j)}}$$
$$\forall_{1\le j,k\le q, k\ne j} \quad g^{a\cdot s\cdot b_k/b_j}, \ldots, g^{(a^q\cdot s\cdot b_k/b_j)},$$

it must remain hard to distinguish $e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element Z in $\mathbb{G}_T$.

An algorithm $\mathcal{D}$ that outputs $\mu \in \{0,1\}$ has advantage ϵ in solving decisional q-parallel BDHE in $\mathbb{G}$ if

$$|\Pr[\mathcal{B}(\vec{Y}, Z = e(g,g)^{a^{q+1}s}) = 1] - \Pr[\mathcal{B}(\vec{Y}, Z = R) = 1]| \ge \epsilon.$$

Definition 2. *We say that the decisional q-parallel BDHE assumption holds if no polynomial time algorithm has non-negligible advantage in solving the decisional q-parallel BDHE problem.*

6.2 Security Proof

The security of the proposed scheme is fulfilled by the following theorem.

Theorem 1. *Suppose the decisional q-parallel BDHE assumption holds and the pseudorandom generator PRG is a random oracle, then no polynomial time adversary can selectively break our proposed scheme with a challenge matrix of size $m^* \times n^*$, where $m^*, n^* \le q$.*

Proof. Suppose there is an adversary $\mathcal{A}$ with non-negligible advantage ϵ in the selective security game against our construction. Then we can construct a simulator $\mathcal{B}$ to solve the decisional q-parallel BDHE problem. Suppose $\mathcal{B}$ is given a decisional q-parallel BDHE problem instance as input.

Init. The simulator takes in a q-parallel BDHE challenge $\vec{Y}, Z$. The adversary gives the algorithm the challenge access structure (M^*, ρ^*), where M^* is a $m^* \times n^*$ matrix with $m^*, n^* \le q$.

Setup. The simulator randomly chooses $\alpha', \beta', \eta \in \mathbb{Z}_p$ and implicitly $\alpha = \alpha' + a^{q+1}$, $\beta = \beta' + \eta a^{q+1}$ and computes $e(g,g)^{\alpha} = e(g^a, g^{a^q}) e(g,g)^{\alpha'}, e(g,g)^{\beta} = e(g^{a^{\theta}}, g^{a^q})$ $e(g,g)^{\beta'}$ and $g^{\alpha} = g^{a^{q+1}} g^{\alpha'}$.

Next, we show how to simulate the group elements $h_1, h_2, \ldots, h_U$. For each $x \in [1, U]$, we choose a random value l_x. Let X denote the set of indices i such that $\rho^*(i) = x$. The simulator computes

$$h_x = g^{l_x} \prod_{i\in X} g^{aM^*_{i,1}/b_i} \cdot g^{a^2 M^*_{i,2}/b_i} \cdots g^{a^{n^*} M^*_{i,n^*}/b_i}.$$

We note that if $X = \emptyset$, we have $h_x = g^{l_x}$. As l_x is chosen at random, h_x is distributed randomly.

PRG-Query. Upon receiving a PRG query for K_i, the simulator $\mathcal{B}$ responds as follow. $\mathcal{B}$ maintains a list $\mathcal{L}$ of a tuple (K_i, F_i). This list is initially empty. If K_i already appears in the list in a tuple (K_i, F_i), it returns the corresponding F_i. Otherwise, $\mathcal{B}$ picks a random $F_i \in \mathbb{Z}_p$ as the value of $PRG(K_i)$, and adds the tuple (K_i, F_i) to $\mathcal{L}$ and then sends F_i to $\mathcal{A}$.

Phase 1. In this phase, the adversary can issue the private key queries. For a private key query of a set S where S does not satisfy the access structure (M^*, ρ^*), the simulator first randomly chooses $r \in \mathbb{Z}_p$. It then finds a vector $\vec{w} = (w_1, \ldots, w_{n^*}) \in \mathbb{Z}_p^{n^*}$ such that $w_1 = -1$ and for all i where $\rho^*(i) \in S$ we have that $\vec{w} \cdot M_i^* = 0$. From the definition of LSSS, such a vector must exist. The simulator then implicitly sets

$$t = r + w_1 a^q + w_2 a^{q-1} + \cdots + w_{n^*} a^{q-n^*+1}.$$

Similarly, the simulator chooses a random $r' \in \mathbb{Z}_p$ and finds a vector $\vec{w'} = (w'_1, \ldots, w'_{n^*}) \in \mathbb{Z}_p^{n^*}$ such that $w'_1 = -\eta$ and for all i where $\rho^*(i) \in S$ we have that $\vec{w'} \cdot M_i^* = 0$. Then it implicitly sets

$$t' = r' + w'_1 a^q + w'_2 a^{q-1} + \cdots + w'_{n^*} a^{q-n^*+1},$$

and computes

$$g^t = g^r \prod_{i=1,\ldots,n^*} (g^{a^{q+1-i}})^{w_i},$$

$$g^{t'} = g^{r'} \prod_{i=1,\ldots,n^*} (g^{a^{q+1-i}})^{w'_i},$$

$$g^{\alpha} g^{at} = g^{\alpha'} g^{ar} \prod_{i=2,\ldots,n^*} (g^{a^{q+2-i}})^{w_i},$$

$$g^{\beta} g^{at'} = g^{\beta'} g^{ar'} \prod_{i=2,\ldots,n^*} (g^{a^{q+2-i}})^{w'_i}.$$

All these values can be computed from the hard problem instance. For the computation of $h_x^t, h_x^{t'}\ \forall x \in S$, we first consider $x \in S$ for which there is no i such that $\rho^*(i) = x$. In this case, we can simply let $h_x^t = (g^t)^{l_x}$ and $h_x^{t'} = (g^{t'})^{l_x}$.

For another case, $x \in S$, where x is used in the access structure. Let X be the set of all i such that $\rho(i)^* = x$. The simulator computes h_x^t and $h_x^{t'}$ respectively as follows:

$$(g^t)^{l_x} \prod_{i \in X} \prod_{j=1,\ldots,n^*} \left(g^{(a^j/b_i)r} \prod_{k=1,\ldots,n^*, k \neq j} (g^{a^{q+1+j-k}/b_i})^{w_k} \right)^{M^*_{i,j}},$$

$$(g^t)^{l_x} \prod_{i \in X} \prod_{j=1,\ldots,n^*} \left(g^{(a^j/b_i)r} \prod_{k=1,\ldots,n^*, k \neq j} (g^{a^{q+1+j-k}/b_i})^{w'_k} \right)^{M^*_{i,j}}.$$

One might observe that in this case, there are terms of the form g^{a^{q+1}/b_i} that we cannot simulate. However, all these terms will be canceled as $M_i^* \cdot \vec{w} = 0$ and $M_i^* \cdot \vec{w'} = 0$. Therefore, the simulation of private keys is correct.

Challenge. Once the adversary decides that phase 1 is over, it outputs two distinct ciphertexts CT_0, CT_1. The simulator $\mathcal{B}$ randomly picks a bit $\mu \in \{0,1\}$, an encryption key $K^* \in \mathbb{G}_T$ with $K^* \notin \mathcal{L}$ and computes the challenge ciphertext as follows.

The simulator first performs the ciphertext checking process on CT_μ to check whether CT_μ was generated under the the access structure (M^*, ρ^*), and aborts if no. Otherwise, the simulator believes that CT_μ has the correct ciphertext structure and performs as follows.

Let $CT_\mu = \left(C_0, C_1, C_2, D_0, (D_{1i}, D_{2i})_{i=1}^m\right)$. $\mathcal{B}$ first obtains $PRG(K^*)$, computes $V_3^* = D_0 \cdot g^s$ and

$$V_0^* = C_0 \oplus PRG(K^*),$$

$$V_1^* = C_1 \cdot Z \cdot e(g^s, g^{\alpha'}),$$

$$V_2^* = C_2 \cdot K^* \cdot Z^{\eta} \cdot e\left(g^s, g^{\beta'}\right).$$

For the simulation of $U_i^* = (A_i^*, B_i^*)$, the simulator randomly chooses $y'_2, \ldots, y'_{n^*}$ and implicitly sets the shared secret using the vector

$$\vec{v} = \left(s, sa + y'_2, sa^2 + y'_3, \ldots, sa^{n^*-1} + y'_{n^*}\right) \in \mathbb{Z}_p^{n^*}.$$

It also chooses random values $z'_1, z'_2, \ldots, z'_{m^*}$ from $\mathbb{Z}_p$. For $i = 1, \ldots, m^*$, we define R_i as the set of all $k \neq i$ such that $\rho^*(i) = \rho^*(k)$. i.e., the set of all other row indices that have the same attribute as row i. Finally, the simulator implicitly sets $z_i = -z'_i - sb_i$ and computes the components A_i, B_i as

$$A_i^* = D_{1i} \cdot h_{\rho^*(i)}^{z'_i} \left(\prod_{j=2,\ldots,n^*} (g^a)^{M^*_{i,j} y'_j} \right) (g^{b_i \cdot s})^{-l_{\rho^*(i)}} \cdot \left(\prod_{k \in R_i} \prod_{j=1,\ldots,n^*} (g^{a^j \cdot s \cdot (b_i/b_k)})^{M^*_{k,j}} \right),$$

$$B_i^* = D_{2i} \cdot g^{z_i} = D_{2i} \cdot g^{-z'_i} g^{-sb_i}.$$

The output challenge ciphertext is

$$CT^* = \left(V_0^*, V_1^*, V_2^*, \left(A_i^*, B_i^*\right)_{i=1}^{m^*}\right).$$

If $Z = e(g,g)^{a^{q+1}s}$, we have

$$\begin{aligned} V_1^* &= C_1 \cdot Z \cdot e(g^s, g^{\alpha'}) \\ &= C_1 \cdot e(g,g)^{a^{q+1}s} \cdot e\left(g^s, g^{\alpha'}\right) \\ &= C_1 \cdot e(g,g)^{\left(\alpha' + a^{q+1}\right)s} \\ &= C_1 \cdot e(g,g)^{\alpha s}, \\ V_2^* &= C_2 \cdot K^* \cdot Z^{\eta} \cdot e(g^s, g^{\beta'}) \\ &= C_2 \cdot K^* \cdot e(g,g)^{\eta a^{q+1}s} \cdot e(g^s, g^{\beta'}) \\ &= C_2 \cdot K^* \cdot e(g,g)^{\left(\beta' + \eta a^{q+1}\right)s} \\ &= C_2 \cdot K^* \cdot e(g,g)^{\beta s}. \end{aligned}$$

Therefore, the challenge ciphertext is valid ciphertext and indistinguishable from the real scheme when $Z = e(g,g)^{a^{q+1}s}$.

Phase 2. In this phase, the adversary can issue more private key queries. The simulator responds the same as phase 1.

Guess. Finally, the adversary outputs its guess μ' of μ. Then the simulator outputs 1 if $\mu = \mu'$ to indicate that it believes $Z = e(g,g)^{a^{q+1}s}$. Otherwise, it outputs 0 to indicate that it believe Z is a random element in group $\mathbb{G}_T$.

This completes the description of the simulation and the solution of decisional q-parallel BDHE problem. It is not hard to verify that the simulation is indistinguishable from the real scheme. If $Z = e(g,g)^{a^{q+1}s}$, according to the assumption, we have

$$\Pr[\mathcal{B}(\vec{Y}, Z = e(g,g)^{a^{q+1}s}) = 1] = \frac{1}{2} + \epsilon.$$

When Z is a random group element in $\mathbb{G}_T$, as PRG is viewed as random oracle, we have $PRG(K^*)$ is random. Thus V_0^* is random. As $\alpha', \beta', \eta, K^*, z'_i, y'_i$, are random chosen, we have $V_1^*, V_2^*, B_i^*, A_i^*$ are random and independent of CT_μ. Therefore the ciphertext CT_μ is completely hidden from the adversary and the challenge ciphertext CT^* is a one-time pad. We have

$$\Pr[\mathcal{B}(\vec{Y}, Z = R) = 1] = \frac{1}{2}.$$

Therefore, we have the advantage of the simulator $\mathcal{B}$ in solving the decisional q-parallel BDHE problem is

TABLE 2
Communication Analysis

	Authority → Receiver	Data publisher → Sanitizer	Sanitizer → Cloud server	Cloud server → Receiver
Receiver Registration	$(4+\|S\|)\|\mathbb{G}\|$	-	-	-
Cipher Data Publishing	-	$(1+m)\|\mathbb{G}\|+2\|\mathbb{G}_T\|+\|P\|$	-	-
Cipher Data Sanitizing	-	-	$(1+m)\|\mathbb{G}\|+2\|\mathbb{G}_T\|+\|P\|$	-
Data Access	-	-	-	$(1+m)\|\mathbb{G}\|+2\|\mathbb{G}_T\|+\|P\|$

$$\begin{aligned}\mathsf{Adv}_{\mathcal{B}} &= |\Pr[\mathcal{B}(\vec{Y}, Z = e(g,g)^{a^{q+1}s}) = 1] \\ &\quad - \Pr[\mathcal{B}(\vec{Y}, Z = R) = 1]| \\ &= \frac{1}{2} + \epsilon - \frac{1}{2} \\ &= \epsilon.\end{aligned}$$

□

7 Performance Analysis

We present the performance analysis for our SACS protocol in terms of both communication complexity and computation time. Due to no comparable protocols in the literature, we only evaluate our protocol in the following analysis.

To present a fair communication analysis, consider four phases except *System Initialization*. We use $|\mathbb{Z}_p|$, $|\mathbb{G}|$ and $|\mathbb{G}_T|$ to denote the size of an element in groups $\mathbb{Z}_p$, $\mathbb{G}$ and $\mathbb{G}_T$, respectively. Also, the size of an attribute set S is denoted as $|S|$ and the output length of the pseudorandom generator is denoted as $|P|$. The communication cost results are summarized in Table 2. In the *Receiver Registration* phase, the communication cost is contributed by the private key assigned by the authority to the receiver. The length of the private key is $(4+|S|)|\mathbb{G}|$. In the *Cipher Data Publishing* phase, the communication cost is mainly from the cipher data, which is uploaded from the data publisher to the sanitizer. The size of the cipher data is $(1+m)|\mathbb{G}|+2|\mathbb{G}_T|+|P|$, where m is the scale of a specific attribute set in the access policy. In the *Cipher Data Sanitizing* phase, the communication cost is contributed by the sanitized cipher data sent from the sanitizer to the cloud server. The size of the sanitized cipher data is the same as that of the cipher data. In the *Data Access* phase, the receiver has no need to send anything to others but needs to download the sanitized cipher data stored on the cloud server. The communication cost is definitely the size of the sanitized cipher data. Here, we do not consider the communication process of the request launched by the receiver.

For the computation analysis, we conduct the simulation of our SACS protocol on a machine with 2.2 GHz Intel Core i7 CPU and 16GB 2133 MHz DDR3 memory. In our experiments, we utilize PBC library and the used language is Python. We choose the symmetric pairing which is constructed based on the curve $y^2 = x^3 + x$ over the finite field F_p. the order of the base field p, satsifying $p = 3 \bmod 4$, is 512-bit. The embedding degree is 2. $\mathbb{G}$ is the group of points on $E(F_p)$ and the group order is 160-bit. We implement the pseudorandom generator by invoking hashlib [24], where HMAC-SHA256 is used to implement the hash function H. The ABE simulation is based on cpabe toolkit [25], assisted with Python Cryptography toolkit [26]. We note that the simulation is just to test the computation time of our protocol but not implement a prototype. Hence we consider neither the deployment of entities nor the latency of data transmitting among different entities.

In our test, three variable factors are considered. They are the length of the plain data, the number of attributes (i.e., the size of the attribute set S) and the scale of a specific attribute set in the access policy (i.e., m from the matrix $M_{m\times n}$ in Section 5.3). We usually fix two factors and vary the rest to execute kinds of tests. The number of universal attributes is pre-set to be 10. In trials, we only collect results with successful decryption.

We first consider to run the SACS protocol and test the computational time. In this test, we vary the length of the plain data, where the number of attributes in the private key is fixed to be 5 and the scale of a specific attribute set in the access policy is fixed to be 10. Three used files (to-be-encrypted plain data) are pre-selected and their sizes are 10, 100 and 500 KB, respectively. The time is collected from random element selection to decryption completion. We plot the computational time of the SACS protocol in Fig. 2. We notice that the time to execute the SACS protocol increases quite slightly with the plain data length. From SACS protocol description, the computational cost is almost linearly dependent on the size of the attribute set, corresponding to the access structure, but not on the plain data. The file length theoretically has insignificant influence on the running time of the complete SACS protocol. The above experiment results are also consistent with their theoretical complexity.

Then we consider different phases and measure the computational time in each phase. We execute the *Receiver Registration* phase, the *Cipher Data Publishing* phase, the *Cipher Data Sanitizing* phase and the *Data Access* phase. The *System Initialization* phase is excluded because its computational time is independent of variable factors mentioned above. Figs. 3, 4, 5, and 6 illustrate their corresponding computational time, respectively. In the *Receiver Registration* phase, the computation is mainly from the generation of the private key, which needs to take the attribute set as input. So we vary the number of the attributes and run the algorithm of the *Receiver Registration* phase accordingly. In the *Cipher Data Publishing* phase, we focus on the computational time of the cipher data generation, where we select the plain

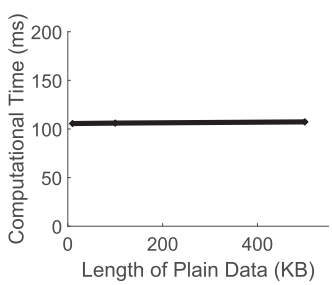


Fig. 2. Computational time of SACS protocol.

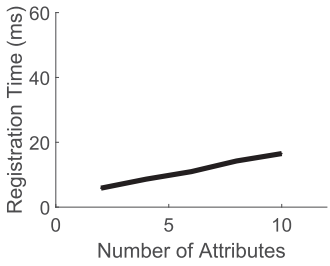


Fig. 3. Registration time.

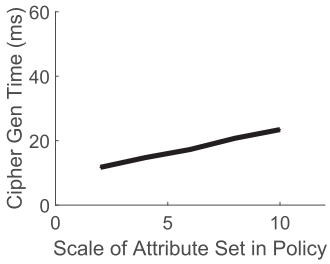


Fig. 4. Cipher gen time.

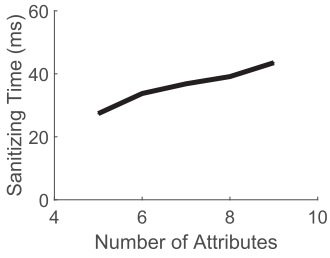


Fig. 5. Sanitizing time.

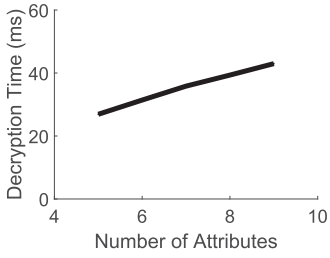


Fig. 6. Decryption time.

data with size of 10 KB. In terms of construction in Section 5, the computational time will be associated with only the scale of a specific attribute set in the access policy and hence we execute trials with respect varying m. In the *Cipher Data Sanitizing* phase, the computational time is associated with both the scale of a specific attribute set in the access policy and the used attribute set. Here we choose to fix the former to be 10 and vary the latter. To test the computational time of the *Data Access* phase, we fix the scale of a specific attribute set in the access policy in the sanitized cipher data to be 10 and vary the number of attributes for the private key generated in the *Receiver Registration* phase. Then we test the decryption time in the *Data Access* phase. Overall, these experiment results show that the computational time in each phase has a growth with increase of the specified variable factor although the amplitude variation is different.

8 Conclusion

We initiated the study of secure cloud storage in the presence of malicious data publishers, which is a very practical situation that unfortunately has never been studied in the literature previously. In this setting, malicious data publishers construct data following the given access control policy, but the ciphertexts can actually be decrypted by unauthorized users without the need of valid keys. We designed a system and its secure scheme to enable protection against this kind of attack. Our scheme is proven secure under q-Parallel Bilinear Diffie-Hellman Exponent Assumption. We also provided an implementation of our system for performance analysis. We believe this work will open future research work in cloud storage, since this notion is very practical. We note that this notion will further encourage the adoption of cloud storage in practice.

Acknowledgments

This work was supported in part by the Australian Research Council Discovery Project (Grant No. DP200100144). The work of Peng Jiang was supported by NSFC (Grant No. 61902025), Beijing Natural Science Foundation (Grant No. 4204111), Beijing Institute of Technology Research Fund Program for Young Scholar and Open Foundation of State key Laboratory of Networking and Switching Technology (BUPT) (SKLNST-2020-1-04). The work of Fuchun Guo was supported in part by NSFC (Grant No. 61972094).

References

[1] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.

[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, 2007, pp. 321–334.

[3] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. Int. Workshop Public Key Cryptogr.*, 2011, pp. 53–70.

[4] S. Berger *et al.*, "Security intelligence for cloud management infrastructures," *IBM J. Res. Develop.*, vol. 60, no. 4, pp. 11:1–11:13, 2016.

[5] Secure access control for cloud storage. Accessed: Feb. 13, 2021. [Online]. Available: https://www.research.ibm.com/haifa/projects/storage/cloudstorage/secure_access.shtml.

[6] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proc. Annu. Int. Cryptol. Conf.*, 2005, pp. 258–275.

[7] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 1–30, 2006.

[8] L. Zhou, V. Varadharajan, and M. Hitchens, "Achieving secure role-based access control on encrypted data in cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 1947–1960, Dec. 2013.

[9] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *IEEE Comput.*, vol. 48, no. 2, pp. 85–88, Feb. 2015.

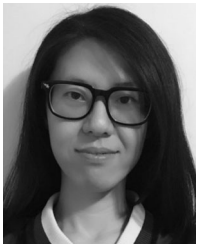
[10] N. Attrapadung, B. Libert, and E. de Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *Proc. Int. Workshop Public Key Cryptogr.*, 2011, pp. 90–108.

[11] Z. Wan, J. Liu, and R. H. Deng, "HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 743–754, Apr. 2012.

[12] Y. Wu, Z. Wei, and R. H. Deng, "Attribute-based access to scalable media in cloud-assisted content sharing networks," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 778–788, Jun. 2013.

[13] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2271–2282, Oct. 2013.
[14] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik, "Sanitizable signatures," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2005, pp. 159–177.
[15] C. Brzuska *et al.*, "Security of sanitizable signatures revisited," in *Proc. Int. Workshop Public Key Cryptogr.*, 2009, pp. 317–336.
[16] V. Fehr and M. Fischlin, "Sanitizable signcryption: Sanitization over encrypted data (full version)," *IACR Cryptol. ePrint Arch.*, vol. 2015, 2015, Art. no. 765.
[17] R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schröder, "Efficient sanitizable signatures without random Oracles," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2016, pp. 363–380.
[18] M. T. Beck *et al.*, "Practical strongly invisible and strongly accountable sanitizable signatures," in *Proc. Australas. Conf. Inf. Secur. Privacy*, 2017, pp. 437–452.
[19] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures," in *Proc. Int. Workshop Public Key Cryptogr.*, 2017, pp. 152–182.
[20] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin, "Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys," in *Proc. Int. Workshop Public Key Cryptogr.*, 2016, pp. 301–330.
[21] I. Damgård, H. Haagh, and C. Orlandi, "Access control encryption: Enforcing information flow with cryptography," in *Proc. Theory Cryptogr. Conf.*, 2016, pp. 547–576.
[22] D. Boneh, X. Boyen, and E. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2005, pp. 440–456.
[23] Z. Liu, Z. Cao, and D. S. Wong, "Efficient generation of linear secret sharing scheme matrices from threshold access trees," *IACR Cryptol. ePrint Arch.*, vol. 2010, 2010, Art. no. 374.
[24] Hashlib. Accessed: Feb. 13, 2021. [Online]. Available: https://docs.python.org/3/library/hashlib.html
[25] Cpabe toolkit. Accessed: Feb. 13, 2021. [Online]. Available: http://acsc.cs.utexas.edu/cpabe/
[26] Python cryptography toolkit. Accessed: Feb. 13, 2021. [Online]. Available: https://pypi.python.org/pypi/pycrypto

Willy Susilo (Fellow, IEEE) is a senior professor with the School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong (UOW), Australia. He is the director at the Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, UOW and the head of the School of Computing and Information Technology, UOW (2015 - now). Prior to this role, he was awarded the prestigious Australian Research Council Future Fellowship in 2009. In 2016, he was awarded the "Researcher of the Year at UOW, due to his research excellence and contributions. He is the editor-in-chief of the Elsevier's *Computer Stanrdards and Interfaces* and the MDPI's *Information Journal*. He is currently an associate editor of the *IEEE Transactions on Dependable and Secure Computing*, *ACM Computing Surveys*, and Elsevier's *Computers and Security*. He has also served as the program committee member of several international conferences.



Peng Jiang is currently an associate professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology. Previously, she was a postdoctoral fellow with the Hong Kong Polytechnic University, Hong Kong, and a visitor with the University of Wollongong, Australia. She has published more than 30 papers in the area of cybersecurity and has served as a program committee member in many international conferences. Her research interests include cryptography, information security, and blockchain.

Jianchang Lai received the MS degree from Xiamen University, Xiamen, China, in 2014, and the PhD degrees from the University of Wollongong, Wollongong, Australia, in 2018. He is currently a lecturer with the School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China. His current major research interests include public-key cryptography including encryption, security proof, data security, and user privacy.

Fuchun Guo received the PhD degree from the University of Wollongong, Wollongong, Australia, in 2013. He is currently a senior lecturer with the Institute of Cybersecurity and Cryptology, University of Wollongong. He received the prestigious Australian Research Council DECRA Fellowship award in 2017. His primary research interests include the public-key cryptography; in particular, protocols, encryption, and signature schemes, and security proof.

Guomin Yang (Senior Member, IEEE) received the PhD degree from the Computer Science Department, City University of Hong Kong, Hong Kong, in 2009. Formerly, he worked as a research scientist with the Temasek Laboratories, National University of Singapore. He is currently an associate professor with the School of Computing and Information Technology, University of Wollongong, Australia. He has been awarded a prestigious Australian Research Council DECRA Fellowship Award. His research interests include cryptography and network security.

Robert H. Deng (Fellow, IEEE) is currently the AXA chair professor of cybersecurity, the director of the Secure Mobile Centre, and the deputy dean for faculty and research with the School of Information Systems, Singapore Management University (SMU). He is also a fellow of the Academy of Engineering Singapore. His research interests include the areas of data security and privacy, cloud security, and Internet of Things security. He received the Outstanding University Researcher Award from the National University of Singapore, the Lee Kuan Yew Fellowship for Research Excellence from SMU, and the Asia Pacific Information Security Leadership Achievements Community Service Star from the International Information Systems Security Certification Consortium. He served as the Steering Committee chair for the ACM Asia Conference on Computer and Communications Security. He served on many editorial boards and conference committees, including the editorial boards of the *IEEE Security & Privacy Magazine*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, *International Journal of Information Security*, and *Journal of Computer Science and Technology*.