On the Security of Data Access Control for Multiauthority Cloud Storage Systems

Xianglong Wu, Rui Jiang, and Bharat Bhargava, Fellow, IEEE

Abstract—Data access control has become a challenging issue in cloud storage systems. Some techniques have been proposed to achieve the secure data access control in a semitrusted cloud storage system. Recently, K. Yang et al. proposed a basic data access control scheme for multiauthority cloud storage system (DAC-MACS) and an extensive data access control scheme (EDAC-MACS). They claimed that the DAC-MACS could achieve efficient decryption and immediate revocation and the EDAC-MACS could also achieve these goals even though nonrevoked users reveal their Key Update Keys to the revoked user. However, through our cryptanalysis, the revocation security of both schemes cannot be guaranteed. In this paper, we first give two attacks on the two schemes. By the first attack, the revoked user can eavesdrop to obtain other users’ Key Update Keys to update its Secret Key, and then it can obtain proper Token to decrypt any secret information as a nonrevoked user. In addition, by the second attack, the revoked user can intercept Ciphertext Update Key to retrieve its ability to decrypt any secret information as a nonrevoked user. Secondly, we propose a new extensive DAC-MACS scheme (NEDAC-MACS) to withstand the above two attacks so as to guarantee more secure attribute revocation. Then, formal cryptanalysis of NEDAC-MACS is presented to prove the security goals of the scheme. Finally, the performance comparison among NEDAC-MACS and related schemes is given to demonstrate that the performance of NEDAC-MACS is superior to that of DAC-MACS, and relatively same as that of DAC-MACS.

Index Terms—Access control, attribute revocation, revocation security, CP-ABE, multiauthority cloud

1 INTRODUCTION

CLOUD computing extends the existing capabilities of Information Technology (IT) since cloud adaptively provides storage and processing services such as SaaS, IaaS, and PaaS that dynamically increase the capacity and add capabilities without investing in new infrastructure or licensing new software [1].

However, the data access control (DAC) issue of cloud computing systems has been escalated by the surge in attacks such as collusion, wiretapping and distort, so that DAC must be designed with sufficient resistance. DAC issues are mainly related to the security policies provided to the users accessing the uploaded data, and the techniques of DAC must specify their own defined security access policies and the further support of policy updates, based on which each valid user can have access to some particular sets of data whereas invalid users are unauthorized to access the data. One approach to alleviate attacks is to store the outsourcing data in encrypted form. However, due to the normally semitrusted cloud and its arrangement issues of administration rights, cloud-based access control approaches with traditional encryption are no longer applicable to cloud storage systems [2].

Sahai and Waters [4] laid a theoretical foundation for solving above encryption problem by introducing the new concept of attribute-based encryption (ABE) whose prototype is the identity-based encryption (IBE). The ABE notion has been the promising cryptographic approach on which more intensive research is based. V. Goyal et al. first proposed the key-policy attribute based encryption for fine-grained access control (KP-ABE) [5]. In KP-ABE, the data was encrypted by attribute set, and decryption was possible only when the user’s policy tree matched the attribute set in the ciphertext. Shortly after KP-ABE, J. Bethencourt introduced the mechanism of ciphertext policy attribute-based encryption (CP-ABE) [6], in which the user received attributes and secret keys from the attribute authority and was able to decrypt ciphertext only if it held sufficient attributes that satisfied the access policy embedded in the ciphertext.

Furthermore, the constructed CP-ABE scheme is deemed as one of the most appropriate techniques for data access control in cloud storage systems, since it can be configured to some DAC schemes which do not require the data owners to distribute keys and furnish the data owners with more efficient and attribute-level control on defined access policies offline. A myriad of data access control techniques based on CP-ABE (e.g. [2, 3], [7]-[19]) are proposed to construct the efficient, secure, fine-grained and attribute-level-revocable access schemes in a semi-trusted cloud storage system. However, based on the Dolev-Yao model [30], security goals such as active attack resistance, data confidentiality, anti-collusion, and attribute-revocation security of most solution designs cannot be all perfectly guaranteed since the capable Dolev-Yao adversaries can overhear, intercept, replay, and synthesis arbitrary information in the open communication channels. For example, in context of attribute revocation in the scenario of K.Yang et
proposed DAC-MACS and EDAC-MACS [2], due to the open and non-secure communication channel, the revoked users, as the Dolev-Yao adversaries, can still breach the backward revocation when they eavesdrop to obtain more than two valid users’ Key Update Keys to update their own Secret Keys, or when they intercept the Ciphertext Update Key delivered from attribute authority to cloud. In both scenarios, each revoked user can retrieve its ability to decrypt any secret information as a non-revoked user.

1.1 Our Contributions

In this paper, two attacks are first given on the DAC-MACS’s and EDAC-MACS’s revocation security which cannot be guaranteed through our cryptanalysis. Subsequently, a new extensive DAC-MACS scheme (NEDAC-MACS) is proposed to withstand above two attacks so as to support more secure attribute revocation. The main contributions of this paper are summarized as follows:

1. In this paper, two attacks are firstly constructed on the vulnerabilities of revocation security in DAC-MACS and EDAC-MACS. By the first attack, the revoked user can eavesdrop to obtain other users’ Key Update Keys to update its Secret Keys, and then it can obtain proper Token to decrypt any secret information as a non-revoked user as before. In addition, by the second attack, the revoked user can intercept the Ciphertext Update Key to retrieve its ability to decrypt any secret information as a non-revoked user as before.

2. Secondly, we propose a new extensive DAC-MACS scheme, denoted as the NEDAC-MACS, to withstand above two attacks and support more secure attribute revocation. We modify some DAC-MACS’s algorithms, and perform the vital ciphertext update communication between cloud server and AAs with some more secure algorithms. Our NEDAC-MACS scheme mainly includes two improvements on the DAC-MACS at Secret Key Generation phase and Attribute Revocation phase, and it can run correctly according to the correctness proof of NEDAC-MACS.

3. Then, formal cryptanalysis of the NEDAC-MACS is described to prove that the proposed NEDAC-MACS can guarantee collusion resistance, secure attribute revocation, data confidentiality, and provable security against static corruption of authorities based on the random oracle model.

4. Finally, performance analysis of our NEDAC-MACS are conducted by making an efficiency comparison among related CP-ABE schemes to testify that the NEDAC-MACS is security-enhanced without reducing more efficiency. The major overhead of decryption is also securely outsourced to the cloud servers, and the overall overheads of storage, communication and computation of the NEDAC-MACS are superior to that of DACC and relatively same as that of DAC-MACS.

1.2 Organizations

We first introduce related work in section 2. The system model and framework of DAC-MACS and EDAC-MACS are briefly reviewed in section 3. Then, two detailed attacks on the attribute revocation security of the two schemes are elaborated in section 4. Subsequently, a new extensive DAC-MACS scheme with enhanced revocation security is proposed in section 5. Section 6 and 7 present the formal cryptanalysis and performance simulation of our NEDAC-MACS scheme, respectively. Finally, the conclusion is given in Section 8.

2 Related Work

Data Access Control: A plurality of data access control systems (e.g. [2, 3, 7]-[19]) based on the promising CP-ABE technique are proposed to construct the efficient, secure, fine grained and revocable access schemes. S.Ruj et al. (2011) proposed a distributed access control scheme in clouds (DACC) [9] that supported attribute revocation. In DACC, one or more key distribution centers (KDCs) distributed keys to data owners and users. Technically, it requires not only forward security but more indispensable backward security in context of the attribute revocation. However, DACC supported attribute revocation with vulnerable forward security [2].

J.Hur et al. (2011) proposed an attribute-based DAC scheme [12] with efficient revocation in cloud storage systems, whereas it was designed only for the cloud systems with single trusted authority. In addition, the above two schemes both require data owners to reencrypt the outsourced ciphertext after revocation.

Liu et al. (2013) presented a secure multi-owner data sharing scheme called Mona [20]. It is claimed that the scheme can achieve fine-grained access control and secure revocation. However, the scheme will easily suffer from collusion attack by the revoked user and the cloud [21].

Recently, K.Yang et al. proposed a data access control scheme for multiauthority cloud storage system (DAC-MACS) [2] and [3] which both supported more efficient decryption and secure attribute revocation without reencryption by the data owners. In reference [2], due to a strong security assumption in DAC-MACS that the non-revoked users will not reveal their key update keys to the revoked user, the authors further removed the assumption and proposed the extensive data access control scheme (EDAC-MACS). In context of secure attribute revocation, DAC-MACS and EDAC-MACS could both achieve forward revocation security irrespective of active attacks. However, the backward revocation security both in DAC-MACS and EDAC-MACS still cannot be guaranteed when the revoked user eavesdrops to obtain more than two users’ Key Update Keys to update its Secret Key, or when the revoked user intercepts the Ciphertext Update Key. In both scenarios, the revoked user can retrieve its ability to decrypt any secret information as a non-revoked user just as before.

Efficiency of Outsourcing Decryption: Green et al. [22] (2011) introduced the notion of outsourcing ABE decryption, and presented two concrete ABE schemes with outsourced decryption, which outsourced the main computation of the decryption and only incurred a small overhead of plaintext recovery for the user by using a token-based
3.2 System Model of DAC-MACS

As shown in Fig. 1, a cloud storage system with multiple attribute authorities (DAC-MACS) has five types of entities involved: global certificate authority (CA), users, cloud servers, data owners, and attribute authority (AA). Table II presents the roles and behaviors of all involved parties in DAC-MACS.

In DAC-MACS, the global certificate authority (CA) accepts both users’ and attribute authorities’ registrations to initialize the system by two steps CAsetup and AAssetup, and hence assign a global unique identity uid to each valid user and a global unique aid to each AA.

After registration, each AA_k ∈ S_k runs Secret Key generation algorithm to compute valid user’s secret keys (SK) according to the user’s role or hierarchy in a defined access policy to some sensitive data.

Then, for each data m, data owners first define an access structure [24], [25] A = (M, ρ), encrypt the data under this access structure and then outsource the encrypted data CT to the proxy cloud server.

Thereafter, the user U_i ∈ S_i can upload A-related secret keys (SK) and its global public key GPK to cloud for a decryption token TK computed by cloud servers, then the user can decrypt the data m with the TK and its global secret key. The CA, AAs, and cloud servers cannot decrypt the data m without user’s global secret key.

For attribute revocation, the corresponding AA, which supervises the revoked attribute, first assigns a version key to each attribute and then generates Ciphertext Update Key for cloud to update CT and Key Update Key for users to update SK. Only those CTs, SKs related to the revoked attribute need to be updated to implicitly contain the latest version key of the revoked attribute. After attribute revocation, all algorithms in system stay unaltered.

3.3 Framework of DAC-MACS

The framework of DAC-MACS mainly consists of five phases: System initialization, Secret Key generation by AAs, Data encryption by data owners, Data decryption by users with the help of cloud, and Attribute revocation.

3.3.1 System Initialization

The whole system can be set up with following steps:
1. CA setup: The certificate authority initializes the system with the CA setup algorithm:

\[ \text{CASetup}(1^k) \rightarrow (\text{MSK, SP, (sk}_{\text{CA}}, v_k\text{CA})). \]

It takes a security parameter \( \lambda \) as inputs and it outputs the system’s master key MSK and the public parameters SP and a pair of signature and verification key \( (sk}_{\text{CA}}, v_k\text{CA}) \).

2. User Registration: The users send their identity information to CA, then CA conducts UserReg algorithm:

\[ \text{UserReg(Sp, sk}_{\text{CA}}, \text{info}_u) \rightarrow (uid, GPK}_{\text{uid}}, GS{\text{K}}}_{\text{uid}}\text{.cert}(uid)) \]

to compute and return each user’s unique identity \( uid \), global public key \( GPK}_{\text{uid}} = g^{uid} \), a global secret key \( GS{\text{K}}}_{\text{uid}} = z_{uid} \) and a user certification \( cert(uid) = \text{Sign}_{sk_{\text{CA}}}(uid, u_{uid}, g^{1/uid}) \).

3. AA Registration: Similar to the user registration, each AA sends their identity information to CA for its unique identity \( aid \).

4. AA Setup: Each \( AA_{\text{aid}}, aid \in S_A \) initializes itself with the AA setup algorithm:

\[ \text{AASetup(Sp, aid)} \rightarrow (SK}_{\text{aid}}, PK}_{\text{aid}}, \{ VK{x}_{\text{aid}}, PK{x}_{\text{aid}}\}). \]

The outputs \( SK}_{\text{aid}} = (\alpha_x, \beta_x, \gamma_x) \), \( PK_x = (e(g, g)^{a_x}, g^{1/\beta_x}, t) \), and \( \{ VK_x, PK_x \} = (g^\gamma x H(x_k^\gamma)) \) are the secret versions of public keys and public key of each attribute \( x_k \) supervised by \( AA_x \).

3.3.2 Secret Key Generation by AAs

Each attribute authority \( AA_x(k \in S_A) \) assigns each valid user \( U_j (j \in S_U) \) a set of attributes \( j, k \) to each user’s secret attribute key \( SK}_{j,k} \):

\[ \text{SKGen(SK}_{\text{aid}}, \text{SP, (PK}_{x_{\text{aid}}}), S_{\text{aid}}, \text{aid}.cert(uid)) \rightarrow SK_{\text{aid},\text{aid}} \text{ to generate the user’s secret attribute key SK}_{j,k} \]  

For \( \forall j \in S_U \) and \( \forall k \in S_A \):

\[ \text{SK}_{j,k} = (K_{j,k}, L_{j,k}, R_{j,k} x_k \in S_j : K_{j,k} = x_k) \]

\[ = (g^\frac{a_x}{\gamma_x^*} g^{a_x k} \gamma_x, g^\frac{1}{\gamma_x^*} L_{j,k} = g^\frac{a_x}{\gamma_x^*} \frac{1}{\gamma_x^*} R_{j,k} = g^\gamma \frac{\gamma_x}{\gamma_x^*}, \quad \forall x_k \in S_j : k_{x,k} = g^\gamma \frac{\gamma_x}{\gamma_x^*} (PK_x)^{\gamma_t k}) \]

where the value \( \gamma_t \) is randomly chosen in \( Z_p \).

3.3.3 Data Encryption by Owners

For each data \( m \), according to the data’s logic attribute granularities, data owners define a monotone access structure \( A \) which can be efficiently realized by a linear secret sharing schemes (LSSS [24]), then an efficient monotone span program (MSP) \( (M, \rho) \) can be constructed due to the proved equivalence between LSSS and MSP [24, 25]. Under \( A \), data owners perform the Encrypt algorithm:

\[ \text{Encrypt(Sp, (PK_x)}^k_{\text{aid}}\{ SK_{x_{\text{aid}},i}, m, A \}) \rightarrow CT \]

to compute CT for the data \( m \):

\[ CT = (\text{Enc}(m), C, C', C'', \forall i = 1 \text{ to } l : C_i, D_{i,1}, D_{i,2}) \]

\[ = (\text{Enc}(m), C = \gamma^k \cdot (\prod_{x_{\text{aid}}} e(g, g)^{a_x})^s, C' = g^s, C'' = g^s_{\gamma^k}, \quad \forall i = 1 \text{ to } l : C_i = g^{a_x t}, (PK_x)^{\gamma_t k}), D_{i,1} = g^{\frac{a_x}{\gamma_x^*}} R_{\gamma_x}, D_{i,2} = g^{\frac{1}{\gamma_x^*}} \]

where values \( k \in I_A, r_i, s \), and vector \( \vec{v} = (s, y_2, ..., y_s) \) are randomly chosen, \( s \) is the secret value in LSSS, \( \lambda_i = (M \cdot \vec{v})_i \) is a share of secret \( s \) and belongs to \( \rho(i) \), \( M \) is a \( l \times n \) matrix in monotone span program, and \( \rho \) is a function from \{1, 2, ..., \} \rightarrow \{ x_k \in S_{\lambda_k}, k \in I_A \}.

3.3.4 Data Decryption by Users with the Help of Cloud Servers

1. Token Generation by Cloud

The user \( U_j (j \in S_U) \) from the user set \( S_U \) queries for a decryption Token \( TK \) and \( CT \) by sending its secret keys \( \{ SK_{j,k} \}_{k \in A} \) and \( GPK_j \). Then \( TK \) is computed by TKGen algorithm:

\[ \text{TKGen}(CT, GPK}_{\text{uid}}\{ SK}_{\text{uid},k\} \rightarrow TK, \text{ and the output is} \]

\[ \text{TK} = \prod_{k \in A} \prod_{i \in S_k} [\text{e}(C_i, GPK_j) \cdot e(D_{i,1}, K_{j,k}) \cdot e(D_{2,2}, L_{j,k})]^{w_i k} \]

where \( N_k = |I_A|, l_k = \{ i : \rho(i) \in S_{\lambda_k} \}, l = \{ l_k \}_{k \in A} \), and \( \{ w_i \}_{i \in S_k} \) are the chosen constants which can reconstruct the secret \( s \) if \( \{ l_k \}_{k \in A} \) are valid shares of \( s \).

2. Data Decryption by Users

After receiving TK and CT, the user \( U_j \) can decrypt the ciphertext with its \( GS{\text{K}}_j \) by the Decrypt algorithm:

\[ \text{Decryp}(CT, TK, GS{\text{K}}_j) \rightarrow m. \]

The user \( U_j \) first computes the content key:

\[ K = C/TK^x, \text{ where } GS{\text{K}}_j = z_k, \]

then it can decrypt the ciphertext:

\[ m = D_{e_k}(\text{Enc}(m)). \]

3.3.5 Attribute Revocation

Suppose \( x_k \) of user \( U_{\mu} \) is revoked from \( AA_x \).

1. Update Key Generation by AAs

The \( x_k \)-corresponding authority \( AA_x \) first generates a new attribute version key \( VK'_{x_k} \), and then performs the UKeyGen algorithm:

\[ \text{UKeyGen(SK}_{\text{aid}}\{ u_{\text{aid}}, VK_{x_{\text{aid}}}' \}) \rightarrow CUK_{x_{\text{aid}}}, \text{CUK}_{x_{\text{aid}}}', VK_{x_{\text{aid}}}'. \]

to calculate the Attribute Update Key \( AUK_{x} \), the Key Update Key \( KUK_{x} \), and the Ciphertext Update Key \( CUK_{x} \):

\[ \text{AUK}_{x} = \gamma_{x} (VK_{x} - VK_{x_{\text{aid}}}'), \quad KUK_{x} = \gamma_{x} AUK_{x_{\text{aid}}}, \quad \text{CUK}_{x} = \beta_{x} AUK_{x_{\text{aid}}}. \]

Then, \( AA_x \) sends \( KUK_{x}, CUK_{x} \) to nonrevoked user \( U_j \) and cloud server respectively. Meanwhile, the public key of the revoked attribute \( x_k \) is changed to the latest version:

\[ PK'_{x} = PK_{x} \cdot e^{AUK_{x_{\text{aid}}}}. \]

2. Secret Key Update by Nonreproved Users:

Upon receiving \( KUK_{x} \) from the user \( U_j (j \neq \mu) \) can run the SKUpdate algorithm:

\[ \text{SKUpdate(SK}_{\text{aid}}, CUK_{x_{\text{aid}}}) \rightarrow SK_{\text{aid}} \]

so as to update its \( SK_{j,k} \) to the latest version:

\[ SK'_{j,k} = (K'_{j,k} = K_{j,k} \cdot L_{j,k} = L_{j,k} \cdot R_{j,k} = R_{j,k}), \quad K'_{j,x} = K_{j,x} \cdot CUK_{j,x} \cdot x_k \in S_j : x_k \neq \tilde{x}_k : K'_{j,x} = K_{j,x}, \]

3. Ciphertext Update by Cloud

Receiving \( CUK_{x_{\text{aid}}} \) from \( AA_x \), cloud servers can run the CTUpdate algorithm:

\[ \text{CTUpdate}(CT, CUK_{x_{\text{aid}}}) \rightarrow CT' \]

to update its current ciphertext.
CT = (En_k(m), C, C', C''), ∀i = 1 to l: C_i, D_{1,i}, D_{2,i}) into the latest version:
CT' = (En_k(m), C, C', C''), ∀i = 1 to l: C_i, D_{1,i}, D_{2,i}),
therein ∀i = 1 to l: if ρ(i) = x_k: C_i' = C_i, D_{2,i}' = g^a_i 
(PK_s^{k})^{-r_i}, else C_i' = C_i.

For the previous ciphertext CT’ which is updated after Attribute Revocation phase, it is called updated previous ciphertext in this paper. Meanwhile, the newly outsourced data can also be denoted by CT’ since they are both corresponding to the current version PK_s^{k}.

3.4 EDAC-MACS Description
In DAC-MACS [2], K.Yang et al. first gave DAC-MACS a strong security assumption that all the nonrevoked users will not send their received Key Update Keys to the revoked user, since they found the revoked user can technically update its secret key to the latest vision via using other user’s Key Update Key.

Then they removed this assumption and propose the extensive data access control scheme (EDAC-MACS). Compared to DAC-MACS, three algorithms’ outputs are modified: SKeyGen, TKGen and UKeyGen. With these fraction modifications, they claimed that the revoked user has no chance to update its Secret Key even if it can corrupt some AAs and collude with some nonrevoked users. However, this conclusion cannot be guaranteed according to the following section 4.

4 Vulnerability Analysis of DAC-MACS and EDAC-MACS
In this section, attack model and two attacks on the attribute revocation security of DAC-MACS and EDAC-MACS are described in detail. In 4.1, we present the adopted attack model. Then, the first attack is elaborated in section 4.2 on the EDAC-MACS’s vulnerability that the revoked user (attacker) can update its Secret Key with other users’ Key Update Keys, and hence decrypt any secret information as a nonrevoked user before.

4.1 Attack Model
In this paper, we make the cryptanalysis and propose our new extensive scheme based on the Dolev-Yao model [30], in which the adversary can overhear, intercept, insert arbitrary information into, synthesis, and replay any message delivered in the communication channels. Under the Dolev-Yao model, the only way to protect the transmitted information from passive or active attacks by eavesdroppers or malicious adversaries is to design the effective security protocols. This means there is no “secure communication channels” assumption between all the involved communication entities. Therefore, it is reasonable that Dolev-Yao model can be more appropriate and practical to describe the attackers and demonstrate the communication protocols in reality.

4.2 Attack I
The attack 1 includes two phases: attack preparation and attack implementation. At the preparation phase, the revoked user (attacker) eavesdrops to obtain any two nonrevoked users’ Key Update Keys at Attribute Revocation phase of EDAC-MACS. Then at the implementation phase, the revoked user can update its own Secret Key SK and then successfully decrypt corresponding CT’ as a nonrevoked user.

4.2.1 Attack Preparation Phase
At the Attribute Revocation phase of EDAC-MACS, when x_k of user U_j is revoked from AA_i, AA_k sends computed Key Update Keys to each nonrevoked user by implementing UKeyGen algorithm. In principle, the revoked user U_µ cannot decrypt any x_k-corresponding ciphertext. However, as an attacker in EDAC-MACS, the revoked user U_µ can eavesdrop to obtain any two nonrevoked users’ Key Update Keys: KUK_p,x_k of U_p and KUK_q,x_k of U_q (p, q ≠ µ):

KUK_p,x_k = g^{(u_p x_k y_a)}AUk_s k, KUK_q,x_k = g^{(u_q x_k y_a)}AUk_s k,
where AUk_s k = γ_µ(v_s k − v_s k).

The revoked user (attacker U_µ) can also obtain the u_p, u_q of two users from the cert(uid) with the CA’s verification key v_C A:

cert(uid) = Sign_{v_C A}(uid, u_mid, g^{1/z uid}), uid = p, q.

Then U_µ can compute its Key Update Key KUK_p,x_k and successfully decrypts CT’ at the following phase.

4.2.2 Attack Implementation Phase
Having obtained u_p, u_q, KUK_p,x_k and KUK_q,x_k, the attacker U_µ starts generating its own KUK_p,x_k as follows.

Attacker U_µ first computes an interim parameter:

Δ = KUK_p,x_k / KUK_q,x_k = g^{(u_p x_k y_a b) v_s k (v_s k − v_s k)}.

Afterwards, it can compute its own Key Update Key:

KUK_p,x_k = Δ^{u_p u_q} · KUK_p,x_k^{u_p u_q}.

Then, attacker U_µ can update its current SK_p,k = (K_p,k, L_p,k, R_p,k, ∀x_k ∈ S_p,k; K_p,x_k) to the latest version with following algorithm:

SKUpdate(SK_p,k, KUK_p,x_k) → SK_p,k.

It outputs:

SK_p,k = [K_p,k, L_p,k, R_p,k, ∀x_k, x_k ≠ x_k, K_p,x_k = x_k, x_k].

Then U_µ can upload the latest version SK_p,k to freely query the cloud for proper Token TK and the objective CT’:

TK = e(C', K' p,k) · e(B' p,k, C')^{-1} = e(g, g)^{x_a u_N A} · e(D_1,i, K' p,i) · e(D_2,i, L' p,i) w_N A
= e(g, g)^{x_a u_N A} · e(g, g) w_N A 
· e(g, g) w_N A 
· e(g, g) w_N A

Afterwards, the attacker U_µ can successfully calculate the symmetric encryption key k:

k = c/TK_p,k, where GSK_µ = z_µ.

Finally U_µ can successfully finish the attack for decrypt-
ing the CT’, whether the CT’ is updated previous one or newly outsourced one, as follow:
\[ m = D_{\text{new}}(E_{\text{new}}(m)). \]

### 4.3 Attack II

The attack 2 also includes two phases: attack Preparation and Attack Implementation. At the preparation phase, the revoked user (attacker U_u) intercepts the previous CUK_{\hat{x}_k} at the Attribute Revocation phase in DAC-MACS or EDAC-MACS. Then at the implementation phase, the revoked user can use the previous CUK_{\hat{x}_k} to decrypt any secret information as a nonrevoked user. Furthermore, the revoked user U_u can properly complete all related operations on its own since it can learn the algorithms CTUpdate, TKGen and all the corresponding inputs.

#### 4.3.1 Attack Preparation Phase

At Attribute Revocation phase of DAC-MACS or EDAC-MACS, when the AA_k sends Ciphertext Update Key CUK_{\hat{x}_k} to cloud server after implementing the UKKeyGen algorithm, the revoked user U_u, as an attacker, can eavesdrop to obtain the transmitted CUK_{\hat{x}_k} = \beta_k\text{AUK}_{\hat{x}_k}/y_k.

Then it can successfully decrypt CT’ at the following implementation phase.

#### 4.3.2 Attack Implementation Phase

Having obtained CUK_{\hat{x}_k}, the revoked user (attacker U_u) can freely obtain the objective CT’ anywhere and anytime from cloud servers, whether the CT’ is updated previous one or newly outsourced one:
\[
\text{CT’} = \begin{cases} 
    \tilde{E}_n(m), & \text{if } \rho(i) = \tilde{x}_k; \mathcal{C}'_i = g^{\tilde{a}_i}.(\mathcal{PK}'_{\text{new}})^{-\tilde{r}_i} \text{, else } \mathcal{C}'_i = \mathcal{C}_i. 
\end{cases}
\]

Then, U_u starts invoking CTUpdate algorithm to reverse the received CT’ back to previous nonrevoked state for U_u:
\[
\text{CTUpdate}(\text{CT’}, -\text{CUK}_{\hat{x}_k}) \rightarrow \text{CT}.
\]

It outputs
\[
\text{CT} = \begin{cases} 
    \tilde{E}_n(m), & \text{if } \rho(i) = \tilde{x}_k; \mathcal{C}'_i = g^{\tilde{a}_i}.(\mathcal{PK}'_{\text{new}})^{-\tilde{r}_i} \text{, else } \mathcal{C}'_i = \mathcal{C}_i. 
\end{cases}
\]

**Correctness.**

If \( \rho(i) = \tilde{x}_k; \mathcal{C}'_i = g^{\tilde{a}_i}.(\mathcal{PK}'_{\text{new}})^{-\tilde{r}_i} \text{, else } \mathcal{C}'_i = \mathcal{C}_i. \)

Afterwards, the attacker U_u can successfully calculate TK by itself:
\[
\text{TK} = \prod_{k \in \Lambda} \prod_{i \in \mathbb{Z}_k} e(c_0, \mathcal{GPK}_0^k) \cdot e(D_{1,i}, \mathcal{C}^{\text{CUK}_k}g^{\text{\mathcal{C}''}_{i}})^{-\tilde{r}_i} \cdot e(D_{2,i}, \mathcal{C}_i^k)^{\tilde{r}_i} = e(g, g)^{\mathcal{Z}_k^\mu}.
\]

Hence the symmetric encryption key \( \kappa \) can be calculated with the TK:
\[
\kappa = \frac{C}{\text{TK}^{\mu}}, \text{ where } \text{GSK}_\mu = z_\mu.
\]

Finally, U_u can decrypt the CT’ as:
\[
m = D_{\text{new}}(E_{\text{new}}(m)).
\]

### 5 Our New Extensive DAC-MACS Scheme

In order to withstand above two attacks and to support more secure attribute revocation, a more robust extensive DAC-MACS scheme, denoted as the NEDAC-MACS, is proposed. We modify the vulnerable algorithms of DAC-MACS so that the vital ciphertext update communications between cloud and AAs are performed with security-enhanced algorithms. Our NEDAC-MACS scheme mainly includes two improvements on EDAC-MACS schemes at the Secret Key Generation phase and the Attribute Revocation phase.

### 5.1 Preliminaries

#### 5.1.1 Bilinear Pairing

**Definition 1.** Let \( G_1, G_2 \) and \( G_2 \) be three multiplicative cyclic groups of the same prime order \( p \). Let \( e: G_1 \times G_2 \rightarrow G_3 \) denote a bilinear map defined with the following three properties:

- **Bilinear:** \( \forall \mu \in G_1, \forall \nu \in G_2, a, b \in \mathbb{Z}_p, \text{ we have } e(a\mu, b\nu) = e(\mu, \nu)^{ab}. \)
- **Nondegenerate:** \( \exists P \in G_1, \forall Q \in G_2 \text{ such that } e(P, Q) \neq 1, \text{ where } I \text{ is the identity element of } G_1. \)
- **Computable:** There exists an efficient algorithm to compute \( e(P, Q) \), for \( \forall \mu \in G_1, \forall \nu \in G_2. \)

In this paper, we adopt the symmetric bilinear pairings on elliptic curves groups (let \( G_1 = G_2 \) denoted as \( G \)).

#### 5.1.2 Decisional q-Parallel Bilinear Diffie-Hellman Exponent Problem

**Definition 2 (q-parallel BDHE [9]).** Let \( g \) be a generator of group \( G \) with prime order \( p \) and \( a, s \in \mathbb{Z}_p \) be randomly chosen. Given a vector \( y \):
\[
(g, g^s, g^{s1}, g^{s2}, ..., g^{s\gamma}), g^a, g^b, ..., g^{aq}, g^{aq+1}, ..., g^{aq+q},
\]

\[ \forall 1 \leq j \leq q, g^{asb_j}, g^{a/b_j}, ..., g^{a\gamma/b_j}, g^{a\gamma/b_j}, g^{a\gamma/b_j}. \]

It must be hard to distinguish a valid tuple
\[
e(e(g, g)^{a\gamma/b_j}g^{a\gamma/b_j}).
\]

**Definition 3.** An algorithm \( \mathcal{A} \) that outputs \( z \in \{0,1\} \) has advantage \( \epsilon \) in solving decisional \( q \)-parallel BDHE problem in group \( G \) if
\[
\left| Pr_{\mathcal{A}(y, T = e(g, g)^{a\gamma/b_j})} = 0 \right| - Pr_{\mathcal{A}(y, T = R)} = 0 \right| \geq \epsilon.
\]

#### 5.1.3 Linear Secret Sharing Scheme (LSSS) [24]

A secret sharing scheme over a set of parties \( P \) is called linear over \( \mathbb{F}_p \):

- The shares for each party form a vector over \( \mathbb{F}_p \).
- There exists a share-generating matrix \( M \) with \( l \) rows and \( c \) columns, for all \( i = 1, ..., l \), we define the function \( \rho(i) \) labeled with the \( i \)-th row of \( M \).
- Let \( s \in \mathbb{Z}_p \) be the secret to be shared, and randomly choose \( r_2, ..., r_l \in \mathbb{Z}_p \) to construct the column vector \( \bar{y} = (s, r_2, ..., r_l) \), the party \( \rho(i) \) gets the share \( \lambda_i = \langle \bar{y}, m_i \rangle \) of the secret \( s \) from \( \bar{y} \).

### 5.2 Security Model of NEDAC-MACS

Similar to DAC-MACS, the authorities can only be corrupted statically, whereas the adversary can query adaptively secret keys under condition that queried secret keys cannot be used in decrypting the challenge ciphertext. The security model of the NEDAC-MACS is presented by
defining a game between a challenger and an adversary as following steps.

Init: After performing the CSetup algorithm, a set of corrupted attribute authorities \( S'_{A} \) are selected by the adversary in the set of all authorities \( S_{A} \). The challenger generates the public keys and secret keys, then sends all public keys and secret keys to the querying adversary in authority set \( S'_{A} \), whereas sends only public keys in \( S_{A} \). The adversary can ensure the real generation by our NEDAC text update communications between cloud servers and Attribute Revocation phase, when in context. Due to the assumption holds, NEDAC MACS apply some components of random, such as \( h_{k} \) on the exponent of bilinear pairing, to each user’s secret attribute keys. Thus, when the discrete logarithm assumption holds, the malicious adversary or collusive users are blinded by the randomness, and it is hard for them to launch passive or active attacks such as adaptive chosen message attack or our attack 1 and 2 in section 4.

5.3 NEDAC-MACS Architecture

Similar to DAC-MACS, the NEDAC-MACS, new extensive data access control for multiple authorities cloud storage system, also has five types of entities involved: global certificate authority (CA), users, cloud servers, data owners, and attribute authorities (AA).

The security assumptions of each entity are the same as EDAC-MACS.

The framework of the NEDAC-MACS model also consists of five phases: System Initialization, Secret Key Generation by AAs, Data Encryption by Owners, Data Decryption by Users with the help of cloud, and Attribute Revocation.

At System Initialization phase of NEDAC-MACS, all corresponding algorithms remain the same as in DAC-MACS.

Then at the Secret Key Generation phase, compared to DAC-MACS, the output of the Secret Key generation algorithm are modified in NEDAC-MACS by adding a randomly chosen number \( h_{uid,aid} \) for AA to compute valid user \( u_{uid} \)’s secret keys \( SK \). Meanwhile, the component \( l_{uid,aid} \) in SK is correspondingly changed to \( l_{uid,aid} \) linked with attribute.

Then at the Data Encryption and Decryption phase, the encryption algorithm by data owner and the decryption algorithm by users is the same as in DAC-MACS.

Finally at the Attribute Revocation phase, when attribute \( \tilde{x}_{aid} \) of \( AA_{aid} \) is revoked from user \( u_{aid} \), the corresponding update key generation algorithm takes as four inputs users’ \( SK_{aid} \), current \( VK_{uid} \), plus the CT’s components \( D_{2j} (\rho (d) = \tilde{x}_{aid}) \) transmitted from cloud servers, and it outputs a new version key for \( \tilde{x}_{aid} \), the ciphertext update keys for cloud to update CT, and the key update keys for users to update SK. Only those CTs, SKs related to the revoked attribute \( \tilde{x}_{aid} \) need to be updated to implicitly contain the latest version key of \( \tilde{x}_{aid} \). The update key generation and secret key update algorithms’ outputs are correspondingly changed to contain the randomly chosen number \( h_{uid,aid} \) piece, and the ciphertext update algorithm is converted into taking as inputs the ciphertext CT, \( CUK_{sid} \), \( \tilde{x}_{aid} \), \( PK_{aid} \), and a new randomly picked value \( \tilde{f} \).

After attribute revocation, all the cryptography algorithms in the NEDAC-MACS also stay unaltered except the public key of the involved revoked attribute. Those modified or added fragments of DAC-MACS’s algorithms are detailed as the two improvements below.

5.3.2 Improvement at Secret Key Generation Phase

At the Secret Key Generation by AAs phase, we add a randomly chosen number \( h_{j,k} \) stored by the \( AA_{k} \) for future
attribute revocation from the user $U_j$.
Each $AA_k (k \in S_A)$ assigns each valid user $U_j (j \in S_U)$ a set of attributes $S_{ij,k}$ after verifying user’s cert$(j)$ by using verification key $vk_{CA}$, then $AA_k$ performs the $S$KeyGen algorithm:

$$S\text{KeyGen}\left(SK_{aid,i}, \{PK_{a,i}\}, S_{aid,a}, SP, \text{cert}(uid), h_{aid,a}\right) \to SK_{uid,a}$$

to generate user’s secret key $SK_{j,k}$, for $\forall j \in S_U, \forall k \in S_A$:

$$SK_{j,k} = (K_{j,k}, R_{j,k}, \forall x_k \in S_{j,k}: K_{j,k}, L_{j,k})$$

$$= \left\{ \begin{array}{ll}
K_{j,k} = g^{a_{j,k} \ast j}\cdot g\cdot a_{j,k}\cdot \beta_k, & R_{j,k} = g^{a_{j,k}} \vspace{1em} \\forall x_k \in S_{j,k}: & L_{j,k} = g^{\beta_{x_k} (h_{j,k} - 1)} \cdot \sum_i \gamma_k \beta_k^i \vspace{1em} \\K_{j,k} = g^{\beta_{x_k} (h_{j,k} - 1)} & + g^{a_{j,k} \ast j}\cdot g\cdot a_{j,k}\cdot \beta_k \end{array} \right.$$
Then the user $U_j$ can perform the decryption algorithm Decrypt to obtain plaintext $m$:

$$\kappa = C/TK^{2i}, \quad m = \text{Dec}_k(\text{Enc}_m(m)),$$

where $\text{GSK}_j = z_j$.

Therefore, $U_j$ can successfully decrypt arbitrary outsourced ciphertext corresponding to its attribute set. □

### 6 Security Analysis of NEDAC-MACS

In this section, the formal security analysis of NEDAC-MACS is given to prove that our NEDAC-MACS can guarantee collusion resistance, revocation security, data confidentiality and provable security against static corruption of authorities under security model 5.2.

#### 6.1 Collusion Resistance

Theorem 2 proves that our NEDAC-MACS can withstand the colluding attack between the legitimate users. For example, given that a valid user $U_1$ with attribute set $S_1$ and another user $U_2$ with $S_2$, according to Theorem 2, it is infeasible for $U_1$ and $U_2$ to collude together for decrypting the ciphertext CT encrypted with $W = S_1 \cup S_2$.

**Theorem 2.** NEDAC-MACS scheme is secure with users collusion resistance.

**Proof.** In NEDAC-MACS, Secret Keys issued by different AA$k$ to each user is associated with the user’s unique identity $u_i$, and meanwhile two random elements $t_{j,k}$, $h_{j,k}$ chosen by AA$k$. Those collusive users are blinded by the random numbers $t_{j,k}$, $h_{j,k}$, and it is hard for them to calculate one user’s secret key with other users’ secret keys. Therefore, those collusive users cannot decrypt those ciphertext which each individual of them cannot decrypt alone, even though the whole attribute set of them satisfies the access policy. Moreover, those collusive users also cannot selectively replace the components of Secret Key issued by AA$k$ with the components of secret key issued by AA$l$ ($k \neq l$). □

#### 6.2 Revocation Security

In this section, formal cryptanalysis on the security of attribute revocation in NEDAC-MACS is given. Theorem 3 proves that our NEDAC-MACS can ensure the revocation security, which means in context of attribute revocation in NEDAC-MACS, the revoked users, as Dolev-Yao attackers, cannot launch attack 1 in section 4 and update their Secret Keys to breach revocation security and retrieve the ability to decrypt any secret information as non-revoked users as before, even though they intercept any valid users’ Key Update Keys.

**Theorem 3.** In the NEDAC-MACS, the revoked user has no chance to update its Secret Key even if it can corrupt some AAs (not the AA corresponding to the revoked attribute) and collude with some nonrevoked users.

**Proof.** In NEDAC-MACS, when $\tilde{x}_k$ of user $U_\mu$ is revoked from AA$k$, each key update key $\text{KUK}_{i,j,k} = g^{h_{j,k}u_i\beta_k\text{AUK}_{x_k}}$ $j \neq \mu$ is associated with both the user’s unique identity $u_i$ and an item $h_{j,k}\beta_k$ defined by corresponding AA$k$. The item $h_{j,k}\beta_k$ in the secret key prevents users from updating their secret keys with the other users’ update keys, since it is only known by the noncorrupted AA$k$ and kept different and secret to all the users. □

We describe the formal definitions of the backward and forward revocation security as following definition 8 and 9 respectively, which are the basis of proofs in theorem 4 and 5.

**Definition 8.** NEDAC-MACS scheme supports backward security in context of attribute revocation if the $\tilde{x}_k$-revoked user has no chance to passively retrieve its ability to decrypt any $\tilde{x}_k$-corresponding ciphertext CT as a nonrevoked user, whether the CT is updated previous ciphertext or the newly outsourced ciphertext.

**Definition 9.** NEDAC-MACS scheme supports forward security in context of attribute revocation if the newly recruited user $U_\mu$ who has been assigned the attribute $\tilde{x}_k$ (suppose $\tilde{x}_k$ is revoked from other user $U_\mu$ $\mu \neq n$), is able to decrypt any authorized $\tilde{x}_k$-corresponding ciphertext CT, whether the CT is updated previous ciphertext or newly outsourced ciphertext.

**Theorem 4.** Theorem 4 gives the proof that our NEDAC-MACS can ensure the backward revocation security, which means in context of attribute revocation in NEDAC-MACS, the revoked users cannot launch attack 1 and 2 in section 4 and breach the backward revocation security even though they eavesdrop to intercept any Cipher-text Update Keys delivered from AAs to cloud servers on open and non-secure communication channel. For example, suppose that the AA$k$-monitoring attribute $\tilde{x}_k$ is revoked from user Alice $U_\mu$, the NEDAC-MACS is able to guarantee that Alice cannot decrypt any $\tilde{x}_k$-related ciphertext CT whether or not the CT is authorized to Alice before the $\tilde{x}_k$ revocation.

**Proof.** When $\tilde{x}_k$ of user $U_\mu$ is revoked from AA$k$:

1. For the previous ciphertext CT’ which is updated after the Attribute Revocation phase:

$$CT' = (\text{Enc}_m(m), C, C', C'', \forall i = 1 \to l; C_i, D'_{1,i}, D'_{2,i}),$$

if $\rho(i) = \tilde{x}_k$:

$$C_i = C_i'(\text{PK}'_{x_k})^{-t_i}\text{CUK}_{x_k}, D'_{1,i} = g^{-c_{r,s}t_i}\beta_k, D'_{2,i} = g^{-c_{r,s}t_i}\beta_k.$$  

We note that the transmitted $\text{CUK}_{x_k} = D'_{2,i} = h_{x_k}^{\text{AAUK}_{x_k}\beta_k}$, $\rho(i) = \tilde{x}_k$, and assume that the revoked user has not stored the previous CT. Then it is hard for the revoked users to calculate the exponent $h_{x_k}^{\text{AAUK}_{x_k}\beta_k}$, due to those revoked users’ blindness by the random number $\tilde{r}_i$ chosen by cloud servers, the component $\text{PK}'_{x_k}^{-t_i}$ cannot be canceled out by the revoked user itself. Therefore, even though the revoked user can obtain all involved communication information like $D_{2,i}, \text{CUK}_{x_k} \text{AAUK}_{x_k}$ in NEDAC-MACS, it still cannot stretch the updated previous CT’ back to the previous version CT the revoked user can properly decrypt.

2. For the newly outsourced ciphertext CT’:

$$CT' = (\text{Enc}_m(m), C, C', C'', \forall i = 1 \to l; C_i, D_{1,i}, D_{2,i}),$$

if $\rho(i) = \tilde{x}_k$:

$$C_i = C_i'(\text{PK}'_{x_k})^{-r_{i}}\beta_k, D_{1,i} = g^{-r_{i}\beta_k}, D_{2,i} = g^{-r_{i}\beta_k}.$$
The revoked user cannot construct \((D_{\gamma_2})^{\beta_k\text{AUK}_{\gamma_2}/y_k}\), since only the uncorrupted attribute authority AA_k, who supervises \(\bar{x}_k\) can calculate exponent \(\beta_k\text{AUK}_{\gamma_2}/y_k\). Therefore, the revoked user cannot transform the \(C_i = g^{a\lambda_i(PK_x)_{\gamma_i}}^{-r_i}\) into \(C_i = g^{a\lambda_i(PK_x)_{\gamma_i}}^{-r_i}\).

Overall, the revoked user cannot reverse any previously published ciphertext CT’ and the newly outsourced ciphertext CT’ back to nonrevoked state when \(U_i\) can properly decrypt the ciphertext. \(\square\)

Theorem 5 proves that our NEDAC-MACS can ensure the forward revocation security, which means when the attribute revocation period ended in NEDAC-MACS, each newly recruited user \(U_n\) has been assigned the attribute \(\bar{x}_k\) (suppose \(\bar{x}_k\) is revoked from user \(U_m, m \neq n\)), is able to decrypt any authorized ciphertext corresponding ciphertext CT. The proof of theorem 5 can be derived on the basis of the Lemma 1 which describes the correctness of our modification at the “Attribute Revocation” phase.

Lemma 1. In NEDAC-MACS, the attribute revocation phase is correct, and still retain the proper working of whole NEDAC-MACS.

Proof. At the step Secret Key Update by Nonrevoked Users of the attribute revocation in NEDAC-MACS, the secret attribute keys of the nonrevoked user \(U_i\) who was assigned the revoked attribute \(\bar{x}_k\), are updated to

\[SK_i^{\gamma_i} = (K_{\gamma_i}, R_{\gamma_i}, \forall x_k \in S_i^{\gamma_i}: K_{\gamma_i,x_k}, U_i^{\gamma_i,x_k}),\]

if \(x_k = \bar{x}_k\), then \(K_i^{\gamma_i,x_k} = v_i^{y_i}\cdot (g^{y_i(b^{-1})\cdot H(\bar{x}_k)}), U_i^{\gamma_i,x_k} = g^{y_i(b^{-1})}\)

Then, at the step Ciphertext Update by Cloud, the \(\bar{x}_k\)-corresponding CT is updated to

\[CT^\gamma = (E_{\gamma_i}(m), C^\gamma, C^\gamma, \forall i = 1 \rightarrow l: C_{i,j}, D_{i,j}', D_{i,j}''),\]

If \(\rho(i) = \bar{x}_k\) we have:

\[C_i^\gamma = g^{a\lambda_i\cdot (PK_{\gamma_i})^{-(r_i+\gamma_i)}}, D_{i,j}' = D_{i,j}g^{y_i}, D_{i,j}'' = D_{i,j}g^{y_i}\cdot \frac{r_i y_i}{\gamma_i}\]

All above operations are equivalent to assigning a new random number \(r_i^\gamma = r_i + \gamma_i\) in \(Z_p\) to the ciphertext, since \(r_i\) is randomly chosen in \(Z_p\).

Then, if nonrevoked user has the attribute subset authorized in the above CT’, the result of token TK is

\[TK = \prod_{k \in x_k} e(g, g)^{aq\lambda_k} = \prod_{k \in x_k} e(g, g)^{aq\lambda_k}\]

Then the user \(U_i\) can obtain the plaintext \(m\):

\[m = C/TK^\gamma, D = e_{\gamma_i}(E_{\gamma_i}(m))\]

Therefore, these update operations of revocation still maintain the formal consistency of all parameters and algorithms in NEDAC-MACS. \(\square\)

Theorem 5. NEDAC-MACS characterizes forward security in context of attribute revocation.

Proof. The proof of NEDAC-MACS’s forward security is similar to Lemma 1, since, after the Attribute Revocation phase, the newly joined user’s secret keys and any ciphertexts on cloud servers are all corresponding to the latest version public key of the revoked attribute, just as nonrevoked \(U_j\) with revoked \(\bar{x}_k\) does in lemma 1. \(\square\)

6.3 Data Confidentiality

In NEDAC-MACS, even though the cloud servers learn user’s secret keys SK and perform the operation of outsourced decryption computation, the cloud servers cannot properly decrypt any ciphertext uploaded by data owners since the full decryption algorithm involves user’s global secret key GSKuid. Furthermore, at the ciphertext update step of Attribute Revocation phase, cloud servers update any corresponding ciphertext CT without the ability to decrypt them. Therefore, data confidentiality against the curious but honest cloud servers is guaranteed.

Invalid users who hold insufficient attributes to satisfy access policy, cannot receive proper Token TK from cloud servers for decryption. Due to the users’ blindness of the random numbers \(e_{i,j,k}\) according to theorem 2 and 3, the invalid user cannot fabricate and upload proper set of Secret Keys for decrypting objective ciphertext. Therefore, data confidentiality against invalid users is guaranteed.

6.4 Provable Security against Static Corruption of Authorities

Under the security model defined in 5.2, the NEDAC-MACS can enjoy the same provable security against static corruption of authorities as DAC-MACS, which is reduced to the hardness of the decisional \(q\)-parallel BDHE assumption [28, 29, 30].

Theorem 6. When the decisional \(q\)-parallel BDHE assumption holds, no polynomial time adversary can selectively break the NEDAC-MACS with a challenge matrix of size 1* \(n^*\) where \(n^* < q\).

Proof. We adopt proof by contradiction like DAC-MACS. Suppose there is an adversary algorithm \(A\) chooses a challenge matrix \(M^*\) with at most \(q-1\) columns and can selectively break the NEDAC-MACS with non-negligible advantage \(Adv_{a,\gamma}\) in the selective security game. Then, based on random oracle model, we can construct a simulator algorithm \(B\) that plays the decisional \(q\)-parallel BDHE with a nonnegligible advantage as follows.

Init: \(B\) takes as inputs \(\bar{y}\) and \(T\) of the decisional \(q\)-parallel BDHE problem. The adversary sends the challenge access structure \((M^*, \rho^*)\) to the \(B\), where \(M^*\) has \(n^* < q\) columns.

Setup: The simulator runs the initialization algorithms CASetup and AASetup. The adversary specifies the corrupted authority set \(S_{a} \subset S\), and reveals \(S_{a}\) to the simulator. For each \(AA_k \in S_a - S_{a}\), the simulator randomly assigns the corresponding \(\alpha_k, \beta_k, \gamma_k\) to each \(AA_k \in S_a - S_{a}\) by letting \(\alpha_k = \alpha_k + a^{q+1}\) and \(e(g, g)^{a_k} = e(g^{a_k}, g^{a^{q+1}})\).

Let \(X = \{1 \mid \rho^*(1) = x\}\). The random oracle \(H\) is defined by simulator as

\[H(x) = g^{a^2_{M_{x}^*}} \prod_{x \in X} g^{x_{j}} \cdot g^{x_{j}} \cdot \ldots \cdot g^{x_{j}}\]

We note that the outputs of the random oracle are randomly distributed due to a randomly chosen value
where \( g^d \) and also note \( H(x) = g^x \) for \( X = \emptyset \). For each \( \mathbb{A}_k \in S_k - S_k^* \), the simulator randomly selects a version number \( v_{x_k} \in \mathbb{Z}_p \), then simulates the public key \( PK_k \) and the public attribute keys \( PK_{x_k} \) as:
\[
PK_k = \left( e(g, g)^{a_k}, g_{PK}, g_{\bar{PK}} \right),
\]
\[
PK_{x_k} = \left( \prod_{i \in x_k} g^{a_{\bar{M}^{l_2}}}, g^{a_{\bar{M}^{l_1}}}, g^{a_{\bar{M}^{s_2}+d_{x_k}}}, g^{a_{\bar{M}^{s_1}+d_{x_k}}} \right)^{y_k}.
\]
After assigning a user identity \( uid \) to the adversary \( \mathcal{A} \), the simulator \( \mathcal{B} \) randomly selects \( u_{uid}, z_{uid} \in \mathbb{Z}_p \) and then lets:
\[
GSK_{uid} = z_{uid}, u_{uid} = u_{uid} - a \cdot GSK_{uid},
\]
\[
GPK_{uid} = g_{uid} \cdot \left( g^{a_k} \right)^{-1/z_{uid}}.
\]
The simulator \( \mathcal{B} \) then sends the \((GPK_{uid}, GSK_{uid})\) to the adversary \( \mathcal{A} \).

**Phase 1:** The adversary \( \mathcal{A} \) refers \((uid, S_k)\) for the simulator to obtain secret keys and update keys. Thereinto \( S_k \) denotes attributes set from \( \mathbb{A}_k \in S_k - S_k^* \) and \( S_k^* \) does not satisfy \( M^* \) in combination with any keys of \( \mathbb{A}_k \in S_k^* \). Since \( S_k \) does not satisfy \( M^* \), a vector \( \bar{\omega} = (\omega_1, \omega_2, ..., \omega_n) \in \mathbb{Z}_p^n \) can be found by the simulator \( \mathcal{B} \) where \( \omega_1 = -1 \), and for each \( i, \omega^i = S_k : \bar{\omega} \cdot M_i^* = 0 \).

The simulator \( \mathcal{B} \) then randomly selects a number \( r \in \mathbb{Z}_p \) and sets \( t \) as:
\[
t_{uid,k} = r + \omega_1 a^{q-1} + \omega_2 a^{q-2} + ... + \omega_n a^{q-n}.
\]
Then component \( R_{uid,k} \) and \( K_{uid,k} \) can be calculated as:
\[
R_{uid,k} = g^{ar} \cdot \prod_{i=1,2,...,n} \left( g^{a_{\bar{M}^{s_i}+d_{x_k}}} \right)^{y_i},
\]
\[
K_{uid,k} = g_{uid} \cdot \prod_{i=1,2,...,n} \left( g^{a_{\bar{M}^{s_i}+d_{x_k}}} \right)^{\omega_i}.
\]
In the NEDAC-MACS, the component \( K_{uid,x_k} \) and \( L_{uid,x_k} \) in the secret key are modified by adding some fractions. For those \( x_k \in S_k \) used in the access structure \( B, i.e. \) such that \( \rho^i() = x_k \), \( L_{uid,x_k} \) and \( K_{x_k,uid} \) can be constructed by the simulator as follows:
\[
V_{x_k} \in S_k:
\]
\[
L_{uid,x_k} = g^{a_{\bar{M}^{s_i}+d_{x_k}}} \cdot \prod_{i=1,2,...,n} \left( g^{a_{\bar{M}^{s_i}+d_{x_k}}} \right)^{\omega_i},
\]
\[
K_{uid,x_k} = \left( \prod_{i \in x_k} g^{a_{\bar{M}^{l_2}}} \right)^{y_k} \cdot \left( \left( g^{a_{\bar{M}^{s_2}+d_{x_k}}} \right)^{PK_{x_k}} \cdot \prod_{j \in 1,2,...,n} \left( g^{a_{\bar{M}^{s_1}+d_{x_k}}} \right)^{-PK_{x_k}y_k} \right)^{-PK_{x_k}y_k}.
\]
For those attributes \( x \in S_{uid} \) not used in the access structure, \( L_{uid,x_k} \) and \( K_{x_k,uid} \) can be constructed as:
\[
K_{uid,x_k} = \left( \prod_{i \in x_k} g^{a_{\bar{M}^{l_2}}} \right)^{y_k} \cdot \left( g^{a_{\bar{M}^{s_2}+d_{x_k}}} \right)^{PK_{x_k}} \cdot \prod_{j \in 1,2,...,n} \left( g^{a_{\bar{M}^{s_1}+d_{x_k}}} \right)^{-PK_{x_k}y_k}.
\]
The adversary can submit some pairs \(((uid, x_k))\) to query update keys. When \( uid \) is a nonrevoked user and \( x_k \) is assigned a new version key \( v_{x_k} \), the simulator then responds corresponding keys \( KUK_{uid,x_k}, LUK_{uid,x_k} \) to adversary:
\[
KUK_{uid,x_k} = u_{uid} \cdot PK_{x_k} \cdot \left( v_{x_k} \right)^{y_k},
\]
\[
LUK_{uid,x_k} = PK_{x_k} \cdot \left( v_{x_k} \right)^{y_k}.
\]
Otherwise, it sends "1" back.

**Challenge:** After receiving two equal length messages \( m_0, m_1 \) and a challenging access structure from the adversary, simulator \( \mathcal{B} \) randomly chooses a bit \( b \) in \( \{0,1\} \). It first generates:
\[
C = m \cdot \prod_{k \in x} e(g^s, g^{a_k}), C' = g^s, C'' = g^{a_k}.
\]
Randomly choosing \( y_1, y_2, ..., y_n \in \mathbb{Z}_p \), the simulator shares secret \( s \) by a vector \( \bar{v} = (s, sa + y_2, s \cdot a^2 + y_3, \ldots, s \cdot a^{n-1} + y_n) \in \mathbb{Z}_p^n \), then \( \mathcal{B} \) can simulate each share \( \lambda_i, i = 1,2, ..., n' \) of the secret \( s \) as:
\[
\lambda_i = s \cdot M_i \cdot \prod_{j \in 1,2,...,n'} (a^{j-1} + y_j) \cdot M_i^j
\]
For each \( i = 1,2, ..., n' \), let \( R_i = \{ t \neq i | \rho^i() = \rho^j() \} \). \( \mathcal{B} \) randomly chooses \( r_i \), \( r_i' \), and simulates the \( C_i \) as:
\[
C_i = \left( g^{y_1 \cdot (\bar{v})^0} \cdot \prod_{j=1,2,...,n'} g^{a_{\bar{M}^j} h(j)} \right)^{y_k} \cdot \left( g^{a_{\bar{M}^j} \cdot (\bar{v})^0} \cdot \prod_{j=1,2,...,n'} g^{a_{\bar{M}^j} h(j)} \right)^{y_k}.
\]
The remaining components of the challenge ciphertext \( CT' \) can be simulated as:
\[
D_{1,i} = \left( g^{y_1 \cdot (\bar{v})^0} \right)^{y_k}, D_{2,i} = \left( g^{y_1 \cdot (\bar{v})^0} \right)^{y_k}.
\]

**Phase 2:** Same as Phase 1.

**Guess:** The adversary \( \mathcal{A} \) finally ends Phase 2 and gives a guess \( b' \) of \( b \). If \( b' = b \), and the simulator \( \mathcal{B} \) outputs 0 to predict that \( T = e(g^s, g^{a_{\bar{M}^i}+d_{x_k}}) \in \mathbb{G}_T \); otherwise, it outputs 1 to indicate that it believes \( T \) is a random element in \( \mathbb{G}_T \).

When \( T \) results in a tuple, the simulator \( \mathcal{B} \) gives a perfect simulation and we have that:
\[
Pr[B(\bar{v}, T) = e(g^s, g^{a_{\bar{M}^i}+d_{x_k}})] = 1/2 + Adv_{\mathcal{A}}.
\]
When \( T \) results in a random group element in \( \mathbb{G}_T \), the message \( m_0 \) is completely hidden from the adversary \( \mathcal{A} \), and \( Pr[B(\bar{v}, T) = e(g^s, g^{a_{\bar{M}^i}+d_{x_k}})] = 1/2 \).

Therefore, the simulator \( \mathcal{B} \) can play the decisional \( q \)-parallel BDHE game with nonnegligible advantage.

### 6.5 Security Comparison

Table III details the comprehensive security comparison among schemes of S.Ruj et al.’s DACC [9], K.Yang et al.’s DAC-MACS [2] and our NEDAC-MACS in terms of collusion resistance, revocation security, data confidentiality and provable security against static corruption of authorities. Therein, \( \sqrt{\ } \) represents the scheme’s capability to achieve the corresponding index, whereas \( \times \) means the opposite.

#### 7 Performance Analysis

To validate the efficiency of our NEDAC-MACS, performance comparisons are carried out in terms of storage overhead, computation overhead and communication overhead among CP-ABE schemes of DACC [9], DAC-MACS [2] and our NEDAC-MACS.

**7.1 Storage Overhead**

Table IV details the storage comparison among the three
schemes, where \(|p|\) is the size of element in the groups \(G, G_T, Z_p\) with prime order \(p, t_u\) denotes the total number of attributes associated with a ciphertext, \(n_c\) denotes the total number of ciphertext on cloud, \(t_u\) denotes the total number of attributes of a user, \(x\) is the revoked attribute, \(n_{non,x}\) denotes the total number of nonrevoked users who have the revoked \(x\), \(n_{c,x}\) is the number of ciphertext associated with the revoked attribute \(x\), \(n_{a,k,uid}\) is the number of attributes assigned from AA\(_k\) to user \(u_{uid}\), \(n_{a,k}\) is the number of attributes managed by AA\(_k\), \(N_k\) is the number of AA involved in the system.

Table IV shows that the overall storage overhead of NEDAC-MACS is relatively same as that of DAC-MACS and has advantage over DACC when \(n_c\) the number of ciphertext or \(n_{c,x}\) the number of ciphertext associated with the revoked \(x\) is large in the system.

It is illustrated in Table IV that, on the authority side, DAC-MACS and NEDAC-MACS incur less storage overhead than DACC since both schemes require each attribute authority to store the version key of each held attribute and the secret authority key, whereas DACC needs to store the secret keys for all attributes. Moreover, the components need to be stored in NEDAC-MACS are similar to DAC-MACS except those added \(h_{i,k}\) need to be securely stored in users’ secret keys by the corresponding AA\(_k\) for each user. However, adding \(h_{i,k}\) results in a \(n_i|p|\) reducing of storage overhead on authority side comparing to that of DAC-MACS.

On the data owners side, DAC-MACS and NEDAC-MACS incur the same storage overhead better than that of DACC when \(n_c\) is large in the system. The reason is that DACC requires the data owners to hold the encryption secret for each ciphertext, whereas in DAC-MACS and NEDAC-MACS, public keys of attribute and AA\(_k\) are mainly needed to be stored.

On each user side, the storage overheads of DAC-MACS and NEDAC-MACS also stay identical and both require less overhead than that of DACC when \(n_{c,x}\) is large in the system. This is due to the reason that the storage overhead in DAC-MACS and NEDAC-MACS mainly comes from the global secret keys and the secret keys of users, whereas DACC requires each user to store both the secret keys issued by all the AAs and the ciphertext components which are associated with the revoked attribute.

The three schemes require almost the same storage overhead on the cloud server side since the storage mainly comes from the ciphertext, where we do not consider the plaintext size encrypted by symmetric keys.

### 7.2 Computation Overhead

Table V details the computation overhead comparison among the schemes and it indicates that NEDAC-MACS incurs less computation overhead than DACC and is comparable to DAC-MACS. DACC needs one pairing computation to encrypt each plaintext and requires more for decryption so that it incurs the largest amount of computation overhead both in encryption on data owners and decryption on user side. Moreover, since the computationally intensive and storage-demanding jobs of decryption process (TKGen) in DAC-MACS and NEDAC-MACS scheme are partitioned and offloaded on traditional cloud resources, it can greatly reduce the workload level on user side. However, DACC requires the data owners to change all stored ciphertext containing \(x\) \(\in I_u\), thus incurs a heavy computation overhead for attribute operations off cloud due to the huge amount of involved ciphertext.

The computation overhead is also conducted by simulating the whole architectures of DACC, DAC-MACS, and NEDAC-MACS with PBC library version 0.5.12 [27], on an Ubuntu system 14.04 with a 2.5 GHz processor and 2G RAM. We adopt the ordinary symmetric elliptic curve (type D internals) with elliptic curve group size 159-bit and embedding degree 6. Each value in Figures 2, 3, 4 is the mean of 10 simulation trials.

As shown in Fig.2, Fig.3, and Fig.4, the consuming time comparison of both encryption and decryption are conducted according to two parameters: the number of authorities and the number of attributes per authority. The revocation computation is based on the number of revoked attributes.

In Fig.2, suppose each user holds the same number of assigned attributes from each attribute. In Fig.2, we set 10 as the involved number of attributes from each attribute authority, and also the involved number of authority. Fig.2 illustrates that the three schemes nearly have the same efficiency in encryption time for data owners, since they are all based on CP-ABE.

In Fig.3 a), we set 10 as the number of involved attributes of user from each AA, and the number of involved authorities linked to the ciphertext is also set to be 10 in Fig.3 b). Fig.3 shows that NEDAC-MACS incurs less computation overhead than DACC and is relatively same as DAC-MACS in efficiency of decryption time for users. The reason is the most computation-consuming job of decryption is offloaded on cloud server in DAC-MACS and NEDAC-MACS scheme, which greatly reduces the workload level on user side. Moreover, the secret keys of users in in NEDAC-MACS and DAC-MACS systems can all be available in public for the cloud servers, which enhances the computation efficiency at the Data Decryption phase when comparing with the DACC.
TABLE V

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Crypt-Computation Encryption Time Cost(s)</th>
<th>Decryption Time Cost(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DACC</td>
<td>(t_p + (4t_r + 1)t_m)</td>
<td>2t_p t_m + 2t_r t_m</td>
</tr>
<tr>
<td>DAC-MACS</td>
<td>(t_p +</td>
<td>I_a</td>
</tr>
<tr>
<td>NEDAC-MACS</td>
<td>(t_p +</td>
<td>I_a</td>
</tr>
</tbody>
</table>

Fig. 2. Comparison of Encryption Time on Data Owners.

Fig. 3. Comparison of Decryption Time on Users.

7.3 Communication Overhead

The communication overhead comparison is conducted among the three schemes regardless of the common fields (M, p) overhead in the ciphertext. Table VI details the communication overhead comparison.

It is easy to find that the three schemes incur almost the same communication overhead at both Encryption and Decryption phase since they all need to send the ciphertext in the two phases. At Attribute Revocation phase, when the ciphertext is reencrypted in DACC, some of its components related to the revoked attributes should be sent to each nonrevoked user who holds the revoked attributes, which increases the overhead of communication compared with DAC-MACS and NEDAC-MACS. We note that in NEDAC-MACS, L_{uid, x_{aid}} of secret keys of U_{uid} are linked with attribute x_{aid}, thus it requires the transmitted update message L_{UK} for updating when x_{aid} of U_{uid} is revoked from A\_A_{uid}, which results in corresponding re-

TABLE VI

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Attribute Revocation Time Cost(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DACC</td>
<td>N/A (n_{c,x}n_{non,x} + 1)</td>
</tr>
<tr>
<td>DAC-MACS</td>
<td>n_{non,x}</td>
</tr>
<tr>
<td>NEDAC-MACS</td>
<td>2n_{non,x}</td>
</tr>
<tr>
<td>DAC-MACS</td>
<td>n_{non,x}</td>
</tr>
<tr>
<td>NEDAC-MACS</td>
<td>2n_{non,x}</td>
</tr>
</tbody>
</table>

8 CONCLUSION

In this paper, we first give two attacks on DAC-MACS and EDAC-MACS for their backward revocation security. Then, a new effective data access control scheme for multi-authority cloud storage systems (NEDAC-MACS) is proposed to withstand the two vulnerabilities in section 3 and thus to enhance the revocation security. NEDAC-MACS can withstand the two vulnerabilities even though the nonrevoked users reveal their received key update keys to the revoked user. In NEDAC-MACS, the revoked user has no chance to decrypt any objective ciphertext even if it actively eavesdrop to obtain an arbitrary number of nonrevoked users’ Key Update Keys LUK or collude with some nonrevoked users or obtain any transmitted information such as Ciphertext Update Keys CUK. Then, formal cryptanalysis of NEDAC-MACS is presented to prove its improved security. Finally, the performance simulation shows the overall storage, computation, and communication overheads of the NEDAC-MACS are superior to that of DACC and relatively same as that of DAC-MACS.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (NO.61202448), and the Key Laboratory Program of Information Network Security of Ministry of Public Security (No.C14610).

REFERENCES


Rui Jiang is now an associate Professor at Southeast University, China. He major in information security, and mainly engages in cloud storage security protocols research.

Bharat Bhargava is a Professor of Computer Science at Purdue University. He is conducting research in security and privacy issues in distributed systems and sensor networks. This involves identity management, secure routing and dealing with malicious hosts, adaptability to attacks, and experimental studies. His recent work involves attack graphs for collaborative attacks. Professor Bhargava has received five best paper awards in addition to the technical achievement award and golden core award from IEEE, and is a fellow of IEEE. He received Outstanding Instructor Awards from the Purdue chapter of the ACM in 1996 and 1998. He has graduated the largest number of Ph.D students in CS department and is active in supporting/mentoring minority students. In 2003, he was inducted in the Purdue's Book of Great Teachers. He has graduated the largest number of women Ph.D students and the first African American student Ph.D in CS department. He is editor-in-chief of three journals and serves on over ten editorial boards of international journals. Professor Bhargava is the founder of the IEEE Symposium on Reliable and Distributed Systems, IEEE conference on Digital Library, and the ACM Conference on Information and Knowledge Management. Bhargava has worked extensively at research laboratories of Air Force and Naval. He has successfully completed several Darpa and Navy STTR proposals. He is working with General Motor Corporation in analyzing use of sensors in cars and other vehicle. He has organized an NSF workshop on V2V wireless network.