Journal of Engineering Sciences

Vol 11, Issue 7,July/2020

ISSN NO:0377-9254

A SECURE STORAGE SCHEME TO AVOID EDOS ATTACKS

[1]I.SAPTHAMI, [2] THULLURU MAMATHA

[1]Assistant professor, Dept. of CSE, VEC, Kavali, AP, India.

[2]MCA, Dept. of MCA, PBR VITS, Kavali, AP, India.

Abstract – To ensure confidentiality, data owners outsource encrypted data instead of plaintexts. To share the encrypted files with other users, Ciphertext-Policy Attribute-based Encryption (CP-ABE) can be utilized to conduct fine-grained and owner-centric access control. But this does not sufficiently become secure against other attacks. Many previous schemes did not grant the cloud provider the capability to verify whether a downloader can decrypt. Therefore, these files should be available to everyone accessible to the cloud storage. A malicious attacker can download thousands of files to launch Economic Denial of Sustainability (EDoS) attacks, which will largely consume the cloud resource. The payer of the cloud service bears the expense. Besides, the cloud provider serves both as the accountant and the payee of resource consumption fee, lacking the transparency to data owners. These concerns should be resolved in real-world public cloud storage. In this paper, we propose a solution to secure encrypted cloud storages from EDoS attacks and provide resource consumption accountability. It uses CP-ABE schemes in a black-box manner and complies with arbitrary access policy of CP-ABE. We present two protocols for different settings, followed by performance analysis.

Keywords – Access Control, Privacy preserving, cloud computing.

I. INTRODUCTION

Cloud storage has many benefits, such as always-online, pay-as-you-go, and cheap [1]. During these years, more data are outsourced to public cloud for persistent storage, including personal and business documents. It brings a security concern to data owners [2]–[4]: the public cloud is not trusted, and the outsourced data should not be leaked to the cloud provider without the permission from data owners. Many storage systems use server-dominated access control, like password-based [5] and certificate-based authentication [6]. They overly trust the cloud provider to protect their sensitive data. The cloud providers and their employees can read any document regardless of data owners' access policy. Besides, the cloud provider can exaggerate the resource consumption of the file storage and charge the payers more without providing verifiable records [2], [7], [8], since we lack a system for verifiable computation of the resource usage.

Relying on the existing server-dominated access control is not secure. Data owners who store files on cloud servers still want to control the access on their own hands and keep the data confidential against the cloud provider and malicious users.

Encryption is not sufficient. To add the confidentiality guarantee, data owners can encrypt the files and set an access policy so that only qualified users can decrypt the document. With Ciphertext-Policy Attribute-based Encryption (CP-ABE) [9], [10], we can have both fine-grained access control and strong confidentiality [11]–[16]. However, this access control is only available for data owners, which turns out to be insufficient. If the cloud provider cannot authenticate users before downloading, like in many existing CP-ABE cloud storage systems [14], [15], the cloud has to allow everyone to download to ensure availability. This makes the storage system vulnerable to the resource-exhaustion attacks. If we resolve this problem by having data owners authenticate the downloaders before allowing them to download, we lose the flexibility of access control from CP-ABE. Here lists the two problems should be addressed in our work:

Problem I: resource-exhaustion attack. If the cloud cannot do cloud-side access control, it has to allow anyone, including malicious attackers, to freely download, although only some users can decrypt. The server is vulnerable to resource-exhaustion attacks. When malicious users launch the DoS/DDoS attacks to the cloud storage, the resource consumption will increase. Payers (in pay-as-you-go model) have to pay for the increased consumption contributed by those attacks, which is a considerable and unreasonable financial burden. The attack has been introduced as Economic Denial of Sustainability

(EDoS) [17]–[20], which means payers are financially attacked eventually. In addition, even files are encrypted, unauthorized downloads can reduce security by bringing convenience to to offline analysis and leaking information like file length or update frequency.

Problem II: resource consumption accountability. In the pay-as-you-go model, users pay money to the cloud provider for storage services. The fee is decided by resource usage. However, CP-ABE based schemes for cloud storage access control does not make online confirmations to the data owner before downloads. It is needed for the cloud service provider to prove to the payers about the actual resource usage. Otherwise, the cloud provider can charge more without being discovered.

In this paper, we combine the cloud-side access control and the existing data owner-side CP-ABE based access control, to resolve the aforementioned security problems in privacy preserving cloud storage. Our method can prevent the EDoS attacks by providing the cloud server with the ability to check whether the user is authorized in CP-ABE based scheme, without leaking other information.

For our cloud-side access control, we use CP-ABE encryption/ decryption game as challenge-response. While upload an encrypted file, the data owner firstly generates some random challenge plaintexts and the corresponding ciphertexts. The ciphertexts are related to the same access policy with the specific file. For an incoming data user, the cloud server asks him/her to decrypt randomly selected challenge ciphertext. If the user shows a correct result, which means he/she is authorized in CP-ABE, the cloud-side access control allows the file download.

To make our solution secure and efficient in real world applications, we provide two protocols of cloud-side and data owner-side combined access control.

II. LITERATURE SURVEY

a) CP-ABE: Ciphertext-Policy Attribute-based-Encryption

CP-ABE is a public key encryption scheme with fine-grained access control. In CP-ABE, each user has some attributes and data owners encrypt their files with an access policy over attributes. Users in the system hold their own secret keys associated with their attribute sets. If and only if the user satisfies the access policy, the user can decrypt. Some useful definitions in CP-ABE are as follows: Attributes. Attributes depict the party's properties relevant to access control. For example, students in EE at Berkeley may have attribute set fEE; Berkeleyg and students in CS at USTC may have attribute set fCS; USTCg. Policy. A policy is a predicate over the attributes. For example, the policy (EE ν CS) allows those students above to access, but none of them satisfy the policy (CS ^ Berkeley).

Syntax. CP-ABE for security parameter λ€N and messages m€{0,1} $^{l(\lambda)}$ consists of PPT (probabilistic polynomial time) algorithms ABE = (Setup; KeyGen; Enc; Dec) as follows:

_ (mpk, msk) ←Setup 1^{λ} generates a master public key mpk and a master key msk.

_ sk_i KeyGen (msk, A_i). It takes the master secret key msk and the user's attribute set A_i as the input and generates a secret key ski associated with the attribute set A_i.

_ ct Enc (mpk, m, A). It takes the master public key mpk, the message m, and the access policy A as the input. It outputs the ciphertext ct.

_ m = Dec (sk_i; ct). It takes the ciphertext ct (encrypted with access policy A) and the secret key sk_i as input. If the attribute set A_i satisfies the access policy A, it outputs the message m. Otherwise, outputs ?.

Correctness, security and the construction. The definitions and formal proofs of correctness and security, and the construction of CP-ABE can be found in [9], [10]. CP-ABE achieves indistinguishability under chosen-plaintext attacks.

b) Authenticated Encryption with Associated Data

Authenticated Encryption with Associated Data (AEAD) is a symmetric-key encryption that provides both confidentiality and integrity, for example, AES-GCM [39]. Here gives the

syntax: Syntax. The symmetric-key encryption AEAD for the key k €{0,1} $^{(\lambda)}$ and any message, m€{0,1} $^{n(\lambda)}$ where n (λ) is a polynomial-bounded function, consists of two PPT algorithms AEAD = (Enc; Dec).

c ←Enc (k, m) outputs the ciphertext c for message m under key k.

m ← Dec (k; c) recovers the plaintext m from c under key k, but outputs ⊥ if the result is invalid.

c) Digital Signature

The system uses a public-key signature scheme for message integrity. Assumed the secure distribution of public keys, any data recipient can verify the message

integrity. For succinctness of signatures, we can use ECDSA:

Syntax SIG for the security parameter λ€N and any arbitrary message m€{0,1} $^{n(\lambda)}$ where n (λ) is a polynomial bounded function, consists of three PPT algorithms SIG =(Gen, Sign, Verify).

(vk_i, sk_i) ←Gen (1^λ) outputs a signing key sk_i and the corresponding verifying key vk_i.

s ←Sign (sk_i, m) generates a digital signature for the message m.

b€{0,1} $^{n(\lambda)}$ ←Verify (vk_i,s,m) outputs whether s is a valid signature of message m.

d) Hybrid Encryption for CP-ABE

We illustrate the usage of hybrid encryption with the example of two CP-ABE ciphertexts with the same access policy and from the same data owner (the public key is vk_o). The cost can be reduced by encrypting an ephemeral key for both ciphertexts, described as follows:

$$\begin{pmatrix} ct_0\ ABE\ .Enc(mpk, m_1, A) \\ ct_1\ ABE.Enc(mpk, m_1, A) \\ s_0\ SIG.Sign\ (sk_i, ct_0) \\ s_0\ SIG.sign(sk_i, ct_1) \\ output(ct_0, ct_1, s_0, s_1) \end{pmatrix}$$

Transform

$$\Longleftrightarrow$$

$$\begin{pmatrix} k \leftarrow \mathbb{S}\{0,1\}^\lambda \\ ct \leftarrow ABE.Enc(mpk, k, A) \\ s \leftarrow SIG.Sign(sk_i, ct) \\ c_0 \leftarrow AEAD.Enc(k, 0 \parallel m_0) \\ c_1\ AEAD.Enc(k, 1 \parallel m_1) \end{pmatrix}$$

Necessity of signatures. Many existing cloud storage systems assume the cloud provider to be semi-honest, in which the ciphertext integrity is not a security concern. However, if we assume the cloud provider to be covert, we need to protect the ciphertext integrity, like using the signatures from the data owner. The signature should also sign on the file metadata, including the file name and the version.

Performance. The security and correctness of the two constructions are the same. But the latter has a benefit in performance – the CP-ABE encryption and decryption, which rely

on heavy pairing, are reduced to one. The two piece of the AEAD-encrypted messages can still be sent in arbitrary order.

e) Bloom Filter

Bloom filter [41] is an m-bit sequence for membership test that is reasonably accurate and space-efficient. The bloom filter BF of m-bit for strings in $\{0, 1\}^{poly(\lambda)}$ as follows.

bf ←Setup (m,λ) generates an empty m-bit bit array.

bf^l Insert (bf,e) inserts an element e by setting the following l positions of bf to 1: H(k; 1‖e), H(k; 2‖e), ,

H(k; l‖e), where H(k, .) is a keyed collision-resistant hash function and k is a security parameter.

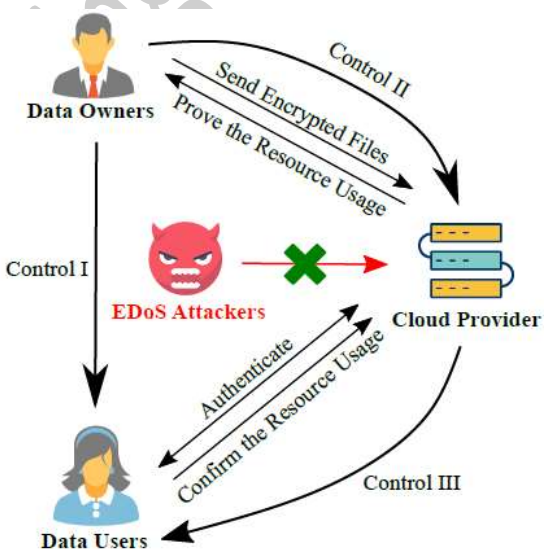
b ←Test (bf, e) checks whether the element e has been inserted to the bloom filter by checking whether all of these positions H(k; 1‖e);H(k; 2‖e); :::;H(k; l‖e) are 1.

Bloom filter has false positives but no false negatives. When bloom filter says that an element is in the set, it may be false.

According to the literatures [41]–[43], the false positive rate of a m-bit Bloom filter is:

$$fp \approx (1-(1-1/m^{ln}))^l$$

where n is the number of the existing members in a set, and l is the number of hash functions used in the bloom filter.



Control I: Data owners only allow authorized data users to decrypt the files.
Control II: Data owners verify the resource consumption records of the cloud provider.
Control III: The cloud provider verifies the data users before the download.

Fig. 1: System Overview

III. RESEARCH METHOD

In this section, we first describe the three-party model for cloud storage used in our construction and our proposed schema.

A) System Model

As shown in Fig. 1, the cloud storage system consists of three entities: data owners, data users, and the cloud provider.

- Data owners are the owner and publisher of files and pay for the resource consumption on file sharing. As the payers for cloud services, the data owners want the transparency of resource consumption to ensure fair billing. The data owners require the cloud provider to justify the resource usage. In our system, the data owner is not always online.
- Data users want to obtain some files from the cloud provider stored on the cloud storage. They need to be authenticated by the cloud provider before the download (to thwart EDoS attacks). The authorized users then confirm (and sign for) the resource consumption for this download to the cloud provider.
- Cloud provider hosts the encrypted storage and is always online. It records the resource consumption and charges data owners based on that record. The cloud is not public-accessible in our system as it has an authentication based access control. Only data users satisfying the access policy can download the corresponding files. The cloud provider also collects the proof of the resource consumption to justify the billing.

As shown in Fig. 1, we have three controls among three entities in our system:

- Control I. Data owners assign an access policy in the document, which controls the set of data users who have the privileges to decrypt the contents.
- Control II. Data owners verifies the resource consumption from the cloud provider, which controls the cloud provider not to exaggerate the resource usage.
- Control III. The cloud provider verifies whether the user can decrypt before the download, which controls the ability of a malicious user who launches DDoS/EDoS attacks.

Moreover, our system differs from previous cloud storage constructions, as we take into account the resource consumption. In practice, the cloud services are usually charged according to the resource consumption, which includes the resource spent on attackers. The DDoS/EDoS attacks will invariably succeed and raise the overhead, which is controlled in our system due to the introduction of the cloud-side access control.

B) Overview of our scheme

To achieve the security requirements, the scheme consists of two componetns: 1) A cloud-side access control to block users whose attribute set Ai does not satisfy the access policy A; 2) A proof-collecting subsystem where the cloud provider can collect the proofs of resource consumption from users, and present to the data owners later. In real-world scenarios, it is reasonable to specify an expected maximal download times, and data owners can remain offline unless it wants to increase this value. This leads to our first protocol: Partially Outsourced Protocol (POP) (V-B). In some other cases where the data owner cannot set an expectation of download times or would be offline for a long time, the data owner can delegate to the cloud. This leads to our second protocol: Fully Outsourced Protocol (FOP) (V-C).

C) Partially Outsourced Protocol (POP)

In this protocol, the data owner encrypts an ephemeral key in CP-ABE, which is then used for message encryption/ decryption and cloud-side access control. The data owner provides the cloud provider with N challenge ciphertexts fenchaligi2[N] and the hashed challenges fhashigi2[N]. The user proves the the legitimacy to the cloud provider by showing the decryption result chalj of the randomly selected unused challenge ciphertext enchalj is a preimage of hashj . If the user response is valid, the cloud provider stores the user response
for further resource consumption accounting. Furthermore, To boost the efficiency and together reduce the storage space, we introduce the bloom filter for data owners to store their
challenge plaintexts. This bloom filter can be stored locally or remotely on the cloud server. As the process of challenge update should be implemented on demand or periodically by the data owner, which cannot be outsourced to the cloud, we call the scheme as Partially Outsourced Protocol (POP).

The procedure of POP is described in detail as follows:

1) Encrypt and Upload (POP-EU): This operation is implemented by an individual data owner independently, which can be divided into the following four steps:

POP-EU-1: The data owner uses hybrid encryption to encrypt the message. The data owner randomly selects a symmetric key k $\leftarrow\$ \{0,1\}^{\lambda}$ and uses the key to encrypt the message M. Then the data owner encrypts that symmetric key k with CP-ABE under A:

$c_0 \leftarrow$AEAD.Enc (k, “message” ‖ M) ;

$c_1 \leftarrow$ ABE.Enc (mpk; k;A) ;

$c_2 \leftarrow$SIG.Sign ($sk_{owner}.c_1$) :

POP-EU-2: The data owner randomly generates N challenge plaintexts from the message space. They should be different with each other.

$\{chal_1; chal_2; : : : ; chal_N\}$; chal $\leftarrow\$ \{0; 1\}^{\lambda}$:

The data owner generates the hashes of these challenges:

$$hash_i = H(chal_i); \forall i \in [1,N];$$

where H(.) is a collision-resistant hash function. For each challenge plaintext $chal_i$, the data owner uses k to encrypt it with a fixed prefix "challenge". The prefix makes these challenges different from messages, which prevents the cloud from deceiving the users into decrypting messages instead of challenges. Here the encryption is also under the same hybrid encryption structure:

$$enchal_i = AEAD.Enc(k, \text{"}challenge\text{"} \| chal_i) :$$

Now, we have

$$c_3 = \{hash_i\}_{i\in[N]},$$
$$c_4 = \{chal_i\}_{i\in[N]}$$

POP-EU-3: The data owner creates a bloom filter to store the challenge plaintexts. We denote m as the size of the bloom filter.

$$bf \leftarrow BF.Setup(m, \lambda) ;$$
$$\forall i \in [N]; bf \leftarrow BF.Insert(bf; chal_i) :$$

And then the data owner encrypts the bloom filter:

$$c_5 = ABAE.Enc(k, bf) ;$$

where k is the data owner's secret key. Note that to avoid the cloud provider understanding the structure of the bloom filter, the data owner should use its own keyed hash functions in the element insertion and test. We assume that the data owner keeps the version number of the bloom filter to thwart rollback attacks.

POP-EU-4: The following tuple is uploaded to the cloud:

$$ct = (c0, c1, c2, c3, c4, c5) ;$$

2) Cloud-side Access Control: POP-CR.

POP-CR-1: The cloud provider selects one of the unused challenge $enchal_j$ and sends the following tuple to the user:

$$(c1; c2; enchal_j) :$$

The data user decrypts the ciphertexts and verifies the signature of the owner. The decryption of c1 requires the data user to satisfy the policy A:

$$\text{HALT if } SIG.Verify(vk_{owner}, c_2, c_1) = 0;$$
$$k \leftarrow ABE.Dec(sk_i. c_1) ;$$
$$chal^1{}_j \leftarrow AEAD.Dec(k, enchal_j) :$$

The data user sends $chal^1{}_j$ to the cloud provider.

POP-CR-2: The cloud checks $hash_j \stackrel{?}{=} H(chal^1{}_j)$. If true, the cloud sends c_0 to the data user, which can be decrypted with the session key k and meanwhile the challenge as used. Otherwise, the cloud aborts.

The user response $chal^1{}_j$ is the proof of the resource consumption accounting.

3) Challenge update (POP-SU): If the specified upper bound of download times (N) has not yet reached, there is no need to update. But if the data owner wants to provide additional challenges, either on-demand or periodically, both only needs to be online for a short period, it is also supported. The update process is the same as that in the phase of POP-EU-2 under the same key k. We assume the data owner keeps a record of session keys either in local storage or outsourced to cloud in an encrypted form. As the plaintext space for challenges is sufficiently large, we assume no duplicated challenge plaintexts are generated. The bloom filter (and its encryption form) introduced in POP-EU-3 will be reconstructed.

4) Resource Accounting (POP-RA): data owners and the cloud interactively implement this operation. The cloud sends back the encrypted bloom filter c_5 and m user responses $\{chal_i\}_{i=1,2,3,....}$. Given the probabilistic check rate, m responses are randomly selected for verification:

$$(chal^1 1, chal^1 2, :::; chal^1 m) \ \$ \ (chal_1; chal_2; :::; chal_m):$$

The data owner decrypts the bloom filter c_5, only if integrity holds and the version number indicates the freshness, the data owner can accept the resource consumption if:

$$\sum_{i=1}^{\beta.m} BF.Test(chal_i^1) = \beta.m$$

Though the bloom filter has some false positives, it is sufficient to achieve the covert security against a cloud provider.

D) Fully Outsourced Protocol (FOP)

If we cannot expect the file download times, we can outsource the challenge update to the cloud. In this section, we give a protocol based on signature algorithm, which has both the outsourced challenges generation/update and resource accounting without an external PKI, therefore we name it as Fully Outsourced Protocol (FOP).

Compared with POP, we have two main differences: 1) Instead of having the data owners generate the challenges fenchaligi2[N], the challenges are generated by the cloud; 2) The data owners generate a pair of signature keys (vk; sk) for every file, with which legitimate users sign a confirmation to prove the resource consumption.

The main procedure of FOP is described as follows:

1) Encrypt and Upload (FOP-EU):

FOP-EU-1: This operation is the same as POP-EU-1.

FOP-EU-2: The data owner generates a signing key pair:
(vk; sk) SIG:Gen(1);
We avoid the external PKI by only using the basic primitive of digital signatures directly. The signing key sk is encrypted under k:
c3 = AEAD:Enc (k; "signing" k sk) :
We provide the verifying key and the collision-resistant hash of k to the cloud:
c4 = H(k); c5 = vk:
FOP-EU-3: The following tuple is uploaded to the cloud:
ct = (c0; c1; c2; c3; c4; c5) :
2) Outsourced Challenge Generation (FOP-CG): In FOP, the cloud provider generates the challenges, which is different from POP. The generation can be done in advance or on demand. We choose the former.
The challenge is encrypted by c4 instead of k:
chali
$
f0; 1g; enchali = AEAD:Enc (c4; chali) :
3) Challenge-Response (FOP-CR). Data owners and the cloud run this operation, which can divided into the following 2 steps:
FOP-CR-1: The cloud provider selects an unused challenge enchalj and sends the following to the user:
(c1; c2; c3; enchalj) :
As the data user has the CP-ABE secret key ski, he/she can verify the signature and decrypts these ciphertexts:
HALT if SIG:Verify (vkowner; c2; c1) = 0; k ABE:Dec (ski; c1) ;
c4 H(k);
chal0
j AEAD:Dec (k; enchalj) ;
sk AEAD:Dec (k; c3) ;
And the data user generates the proof by making the signature with the signing key sk, which is generated by the data owner:
prof (SIG:Sign (sk; chalj k Info) ; chalj ; Info) ; where Info is an auxiliary information that consists of the timestamp and the filename.
FOP-CR-2: The cloud checks whether the user responses is the correct chalj and whether the signed record prof is valid. If true, the cloud sends c0 to the user, otherwise it aborts.
4) Resource Accounting (FOP-RA). This operation is interactively implemented by the data owner and the cloud. The data owner asks the cloud to send all signed records fprofigi=1;2;:::;m. Given the probabilistic check rate,
m responses are randomly chosen for verification.
(prof0 1; prof02; :::; prof0m) $ (prof1; prof2; :::; profm):
The data owner accepts the resource consumption if:
X
m i=1
SIG:Verify (vk; chali[0]; chali[1] k chali[2]) = m:
The probabilistic inspection remains sufficient.

IV. RESULT ANALYSIS

In this section, we analyze the two protocols on how they achieves several main security properties.

a) Security against EDoS Attacks

EDoS attackers are those that do not satisfy the access policy (i.e., unauthorized users) but want to trigger the cloud provider to send something through the network, as a result the resource consumption increases. To thwart such attacks, the cloud provider uses authentication. The protocols only send a constant amount of bytes to the data user before it passes the cloud-side access control. To succeed a EDoS attack in our definition, the attacker has to first pass the cloud-side access control.

b) Resource Consumption Accounting

As shown in Section VI-A, for a user whose attribute set Ai does not satisfy the access policy A: 1) The user cannot output a valid preimage chalj in POP; 2) The user cannot obtain the
signing key sk in FOP. Without the loss of generality, we assume that the cloud provider has never been granted any attributes and doesn't collude with any authorized users. The result above can also be applied to the cloud provider as follows:
In POP, to forge a proof, the cloud provider needs to decrypt enchali without the key k or find a preimage of hashi.
In FOP, to forge a proof, the cloud provider needs to generate a valid signed record prof on a message that has never been signed, without knowing the signing key sk.
Unforgeability in FOP. The input to the algorithm A is the verifying key vk. Due to the existential unforgeability of the signature scheme SIG (in Section III-C), no PPT algorithm can forge a signature on messages that have never been signed. Since it is difficult to forge a proof either in POP or FOP, we only need to consider the bllom filter and the probabilistic
check, which can reduce the overhead, but introduce a small possibility for the cloud provider to cheat without being caught.
Bloom filter (for POP). Note that the data owner uses a keyed collision-resistant hash function as described

in Section III-E, where the key is only known to the data owner. No adversary knows the mapping in the bloom filter. And the cloud provider does not know the elements in the bloom filter because it is encrypted. We want to show that even if the cloud provider knows N0 elements in the bloom filter, the cloud provider cannot forge an element that passes the test better than a random guess.

c) Performance Analysis

In this section, we give the experiment setup and analyze the computation overhead between original CP-ABE based storage (without covert security), POP, and FOP, respectively.

i. Experiment Setup

The experiment uses Microsoft Azure DS1 V2 (Single Core, 3.5GB Memory, SSD) running OpenSSL 1.1.0 and cpabe toolkit 0.11 [9].

The encrypted cloud storage has 220 files each of length 1MB. The cpabe library uses a1 elliptic curve, which has 2048-bit discrete-log-equivalent security. The access structure of CP-ABE is a 5-attribute-AND policy for illustration. The AEAD encryption is implemented with AES-GCM-128 and the secure hash function is implemented with AES-NI. The key lengths for ECDSA and RSA are 160 bit and 4096 bit, respectively.

The catching possibility is set to be = 10%, both for POP and FOP. In POP, we trade-off the verification time and the length of bloom filter by requiring the compression rate be

5% (i.e., k = 4 and m = 5021), and the POP = 11% The number of challenges is N = 1000 each file in POP.

FOP generates the challenge upon user request, and set the probabilistic check rate FOP = 10%.

ii. Computation Overhead

The experimental result in terms of computation overhead is given in Fig. 4(a), 4(b), 4(c), and 4(d). Since the CP-ABE has bilinear pairing, the encryption and the decryption costs 64ms

and 188ms in our experiment, respectively. The overhead of our construction over CP-ABE is from: 1) The encryption and hash for generating N = 1000 challenges and creating the bloom filter BL in POP; 2) The key generation, signature, and verification from ECDSA in FOP.

For the computation overhead, when the data owner uploads the file (as shown in Fig. 2), POP and FOP has 0:3ms and 0:1ms additional execution time, respectively. The addition is small compared with the original ABE (with owner's signature), as is <0:5% in Fig. 4(a).

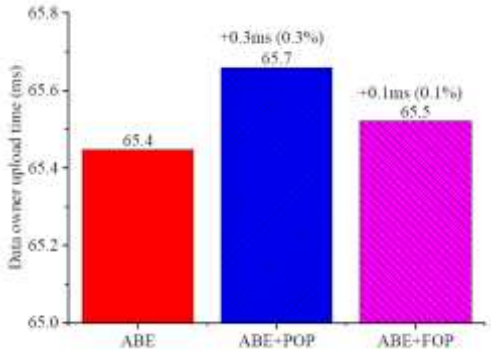


Fig 2: Data owner upload time

For the computation overhead, when the cloud provider authenticates a data user (as shown in Fig. 3), POP and FOP brings an additional overhead, 0.03s and 279.06s, respectively.

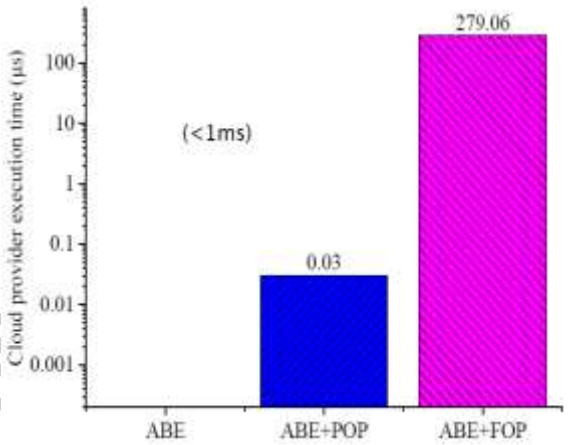


Fig 3: Cloud provider execution time

When an authorized data user retrieves a file (as shown in Fig. 4), the data user needs to solve the cloud provides's challenge. The challenge decryption can be done within several symmetric encryption and hashing, which is efficient both in POP and FOP. In FOP, the data user needs to generate a proof from ECDSA, which is small (<0:1%) compared with CP-ABE decryption. This shows the impact to honest users is small.

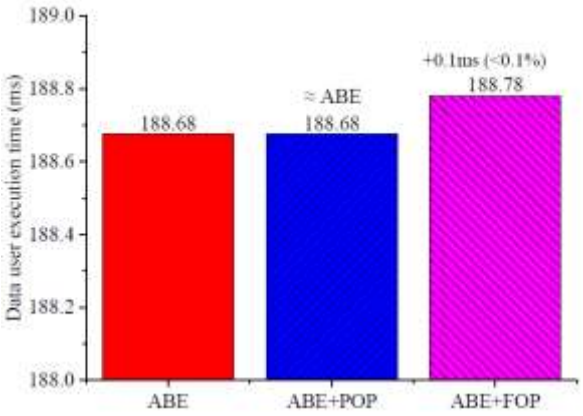


Fig 4: Data user execution time

For the resource consumption accounting (as shown in Fig. 5), the time of verification is less than 100ms, for verifying a total of 1000 challenges. This is only necessary when the data owner who wants to account the resource consumption.

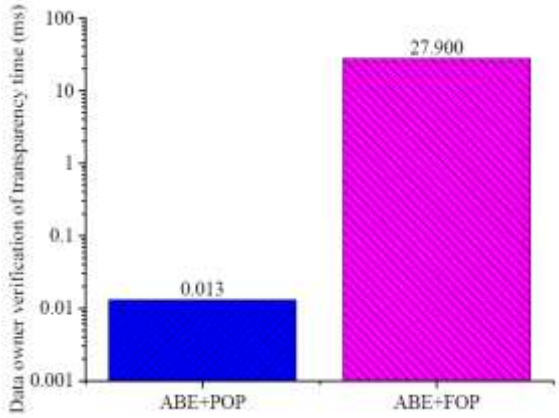


Fig 4: Verification of transparency time

V. CONCLUSION

In this paper, we propose a combined the cloud-side and data owner-side access control in encrypted cloud storage, which is resistant to DDoS/EDoS attacks and provides resource consumption accounting. Our system supports arbitrary CP-ABE constructions. The construction is secure against malicious data users and a covert cloud provider. To make use of the covert security, we use bloom filter and probabilistic check in the resource consumption accounting to reduce the overhead. Performance analysis shows that the overhead of our construction is small over existing systems.

REFERENCES

1. Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," Journal of Internet Services and Applications, vol. 1, no. 1, pp. 7–18, 2010.
2. K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," IEEE Internet Computing, no. 1, pp. 69–73, 2012.
3. L. Zhou, Y. Zhu, and A. Castiglione, "Efficient k-NN query over encrypted data in cloud with limited key-disclosure and offline data owner," Computers & Security, vol. 69, pp. 84–96, 2017.
4. S. Hu, Q. Wang, J. Wang, Z. Qin, and K. Ren, "Securing SIFT: Privacy-preserving outsourcing computation of feature extractions over encrypted image data," IEEE Transactions on Image Processing, vol. 25, no. 7, pp. 3411–3425, 2016.
5. H.-M. Sun, Y.-H. Chen, and Y.-H. Lin, "oPass: A user authentication protocol resistant to password stealing and password reuse attacks," IEEE Transactions on Information Forensics and Security, vol. 7, no. 2, pp. 651–663, 2012.
6. L. Harn and J. Ren, "Generalized digital certificate for user authentication and key establishment for secure communications," IEEE Transactions on Wireless Communications, vol. 10, no. 7, pp. 2372–2379, 2011.
7. V. Sekar and P. Maniatis, "Verifiable resource accounting for cloud computing services," in Proceedings of the 3rd ACM workshop on Cloud computing security workshop. ACM, 2011, pp. 21–26.
8. C. Chen, P. Maniatis, A. Perrig, A. Vasudevan, and V. Sekar, "Towards verifiable resource accounting for outsourced computation," in ACM SIGPLAN Notices, vol. 48, no. 7. ACM, 2013, pp. 167–178.
9. J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute based encryption," in 2007 IEEE Symposium on Security and Privacy (SP'07). IEEE, 2007, pp. 321–334.
10. B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in Public Key Cryptography– PKC 2011. Springer, 2011, pp. 53–70.