International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 03, March 2017
www.ijates.com
ijates
ISSN 2348 - 7550

PROTECTED AUDITING AND DEDUPLICATING DATA IN CLOUD BY MEANS OF SECCLOUD

Mr. Avula Praveen Kumar , Mr. PaillaThiramalReddy , T.Madhu

[1]*Department of Computer Science & Engineering JNTUH,*
[2]*Department of Computer Science & Engineering JNTUH.*
[3]*Department of Computer Science &Engineering SRTI, Nalgonda,*

ABSTRACT

As the cloud computing technology develops during the last decade, outsourcing data to cloud service for storage becomes an attractive trend, which benefits in sparing efforts on heavy data maintenance and management. Nevertheless, since the outsourced cloud storage is not fully trustworthy, it raises security concerns on how to realize data deduplication in cloud while achieving integrity auditing.

In this work, we study the problem of integrity auditing and secure deduplication on cloud data. Specifically, aiming at achieving both data integrity and deduplication in cloud, we propose two secure systems, namely SecCloud and SecCloud$^+$. SecCloud introduces an auditing entity with a maintenance of a MapReduce cloud, which helps clients generate data tags before uploading as well as audit the integrity of data having been stored in cloud. Compared with previous work, the computation by user in SecCloud is greatly reduced during the file uploading and auditing phases. SecCloud$^+$ is designed motivated by the fact that customers always want to encrypt their data before uploading, and enables integrity auditing and secure deduplication on encrypted data.

I INTRODUCTION

Cloud storage is a model of networked enterprise storage where data is stored in virtualized pools of storage which are generally hosted by third parties. Cloud storage provides cus-tomers with benefits, ranging from cost saving and simplified convenience, to mobility opportunities and scalable service. These great features attract more and more customers to utilize and storage their personal data to the cloud storage: according to the analysis report, the volume of data in cloud is expected to achieve 40 trillion gigabytes in 2020.

Even though cloud storage system has been widely adopted, it fails to accommodate some important emerging needs such as the abilities of auditing integrity of cloud files by cloud clients and detecting duplicated files by cloud servers. We illustrate both problems below.

The first problem is integrity auditing. The cloud server is able to relieve clients from the heavy burden of storage management and maintenance. The most difference of cloud storage from traditional in-house storage is that the data

is transferred via Internet and stored in an uncertain domain, not under control of the clients at all, which inevitably raises clients great concerns on the integrity of their data. These concerns originate from the fact that the cloud storage is susceptible to security threats from both outside and inside of the cloud [1], and the uncontrolled cloud servers may Passively hide some data loss incidents from the clients to maintain their reputation. What is more serious is that for saving money and space, the cloud servers might even actively and deliberately discard rarely accessed data files belonging to an ordinary client. Considering the large size of the outsourced data files and the clients' constrained resource capabilities, the first problem is generalized as *how can the client efficiently perform periodical integrity verifications even without the local copy of data files.*

The second problem is secure deduplication. The rapid adoption of cloud services is accompanied by increasing volumes of data stored at remote cloud servers. Among these remote stored files, most of them are duplicated: according to a recent survey by EMC [2], 75% of recent digital data is duplicated copies. This fact raises a technology namely dedu-plication, in which the cloud servers would like to deduplicate by keeping only a single copy for each file (or block) and make a link to the file (or block) for every client who owns or asks to store the same file (or block). Unfortunately, this action of deduplication would lead to a number of threats potentially affecting the storage system [3][2], for example, a server telling a client that it (i.e., the client) does not need to send the file reveals that some other client has the exact same file, which could be sensitive sometimes. These attacks originate from the reason that the proof that the client owns a given file (or block of data) is solely based on a static, short value (in most cases the hash of the file) [3]. Thus, the second problem is generalized as *how can the cloud servers efficiently confirm that the client (with a certain degree assurance) owns the uploaded file (or block) before creating a link to this file (or block) for him/her.*

In this paper, aiming at achieving data integrity and dedu-plication in cloud, we propose two secure systems namely SecCloud and SecCloud$^+$. SecCloud introduces an auditing entity with a maintenance of a MapReduce cloud, which helps clients generate data tags before uploading as well as audit the integrity of data having been stored in cloud. This design fixes the issue of previous work that the computational load at user or auditor is too huge for tag generation. For completeness of fine-grained, the functionality of auditing designed in SecCoud is supported on both block level and sector level. In addition, SecCoud also enables secure deduplication. Notice that the "security" considered in SecCoud is the prevention of leakage of side channel information. In order to prevent the leakage of such side channel information, we follow the tradition of [3][2] and design a proof of ownership protocol between clients and cloud servers, which allows clients to prove to cloud servers that they exactly own the target data.

Motivated by the fact that customers always want to encrypt their data before uploading, for reasons ranging from personal privacy to corporate policy, we introduce a key server into SecCloud as with [4] and propose the SecCloud$^+$ schema. Besides supporting integrity auditing and secure deduplica-tion, SecCloud$^+$ enables the guarantee of file confidentiality. Specifically, thanks to the property of deterministic encryption in convergent

encryption, we propose a method of directly auditing integrity on encrypted data. The challenge of dedupli-cation on encrypted is the prevention of dictionary attack [4]. As with [4], we make a modification on convergent encryption such that the convergent key of file is generated and controlled by a secret "seed", such that any adversary could not directly derive the convergent key from the content of file and the dictionary attack is prevented.

This paper is organized as follows: In Section II, we review the related works on integrity auditing and secure deduplica-tion. In Section III, we introduce some background including the bilinear maps and convergent encryption. Section IV and Section V respectively proposes the SecCloud and SecCoud^{+} system. Section VI and Section VII respectively analyzes the security and efficiency of proposed systems. Finally Section VIII draws the conclusion of this paper.

III RELATED WORK

Since our work is related to both integrity auditing and secure deduplication, we review the works in both areas in the following subsections, respectively.

A. Integrity Auditing

The definition of provable data possession (PDP) was introduced by Ateniese et al. [5][6] for assuring that the cloud servers possess the target files without retrieving or downloading the whole data. Essentially, PDP is a probabilistic proof protocol by sampling a random set of blocks and asking the servers to prove that they exactly possess these blocks, and the verifier only maintaining a small amount of metadata is able to perform the integrity checking. After Ateniese et al.'s proposal [5], several works concerned on how to realize PDP on dynamic scenario: Ateniese et al. [7] proposed a dynamic PDP schema but without insertion operation; Erway et al. [8] improved Ateniese et al.'s work [7] and supported insertion by introducing authenticated flip table; A similar work has also been contributed in [9]. Nevertheless, these proposals [5][7][8][9] suffer from the computational overhead for tag generation at the client. To fix this issue, Wang et al. [10] proposed proxy PDP in public clouds. Zhu et al. [11] proposed the cooperative PDP in multi-cloud storage.

Another line of work supporting integrity auditing is proof of retrievability (POR) [12]. Compared with PDP, POR not merely assures the cloud servers possess the target files, but also guarantees their full recovery. In [12], clients apply erasure codes and generate authenticators for each block for verifiability and retrievability. In order to achieve efficient data dynamics, Wang et al. [13] improved the POR model by ma-nipulating the classic Merkle hash tree construction for block tag authentication. Xu and Chang [14] proposed to improve the POR schema in [12] with polynomial commitment for reducing communication cost. Stefanov et al. [15] proposed a

POR protocol over authenticated file system subject to frequent changes. Azraoui et al. [16] combined the privacy-preserving word search algorithm with the insertion in data segments of randomly generated short bit sequences, and developed a new POR protocol. Li et al. [17] considered a new cloud storage architecture with two independent

cloud servers for integrity auditing to reduce the computation load at client side. Recently, Li et al. [18] utilized the key-disperse paradigm to fix the issue of a significant number of convergent keys in convergent encryption.

B. Secure Deduplication

Deduplication is a technique where the server stores only a single copy of each file, regardless of how many clients asked to store that file, such that the disk space of cloud servers as well as network bandwidth are saved. However, trivial client side deduplication leads to the leakage of side channel information. For example, a server telling a client that it need not send the file reveals that some other client has the exact same file, which could be sensitive information in some case.

In order to restrict the leakage of side channel information, Halevi et al. [3] introduced the proof of ownership protocol which lets a client efficiently prove to a server that that the client exactly holds this file. Several proof of ownership protocols based on the Merkle hash tree are proposed [3] to enable secure client-side deduplication. Pietro and Sorniotti [19] proposed an efficient proof of ownership scheme by choosing the projection of a file onto some randomly selected bit-positions as the file proof.

Another line of work for secure deduplication focuses on the confidentiality of deduplicated data and considers to make deduplication on encrypted data. Ng et al. [20] firstly intro-duced the private data deduplication as a complement of public data deduplication protocols of Halevi et al. [3]. Convergent encryption [21] is a promising cryptographic primitive for ensuring data privacy in deduplication. Bellare et al. [22] formalized this primitive as message-locked encryption, and explored its application in space-efficient secure outsourced storage. Abadi et al. [23] further strengthened Bellare et al's security definitions [22] by considering plaintext distributions that may depend on the public parameters of the schemas. Regarding the practical implementation of convergent encryp-tion for securing deduplication, Keelveedhi et al. [4] designed the DupLESS system in which clients encrypt under file-based keys derived from a key server via an oblivious pseudorandom function protocol.

As stated before, all the works illustrated above considers either integrity auditing or deduplication, while in this paper, we attempt to solve both problems simultaneously. In addition, it is worthwhile noting that our work is also distinguished with [2] which audits cloud data with deduplication, because we also consider to 1) outsource the computation of tag generation, 2) audit and deduplicate encrypted data in the proposed protocols.

IV PRELIMINARY

We now discuss some preliminary notions that will form the foundations of our approach.

A. Bilinear Map and Computational Assumption

Definition 1 (Bilinear Map): Let G and G_T be two cyclic multiplicative groups of large prime order p. A bilinear pairing is a map $e : G \times G \rightarrow G_T$ with the following properties:

- *Bilinear:* $e(g_1^a; g_2^b) = e(g_1; g_2)^{ab}$ for all $g_1; g_2 \in_R G$ and $a; b \in_R Z_p$;
- *Non-degenerate:* There exists $g_1; g_2 \in G$ such that $e(g_1; g_2) \neq 1$;
- *Computable:* There exists efficient algorithm to compute $e(g_1; g_2)$ for all $g_1; g_2 \in_R G$.

The examples of such groups can be found in supersingular elliptic curves or hyperelliptic curves over finite fields, and the bilinear pairings can be derived from the Weil or Tate pairings. For more details, see [24].

We then describe the Computational Diffie-Hellman prob-lem, the hardness of which will be the basis of the security

B. Convergent Encryption

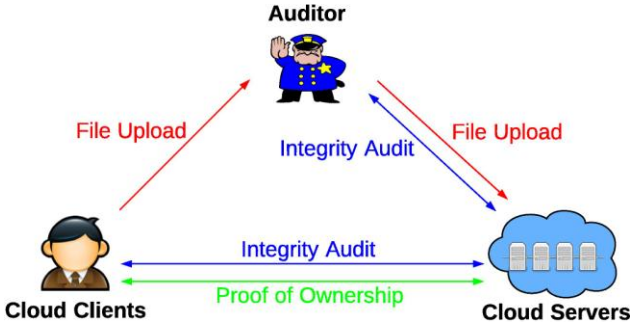
Convergent encryption [22][23][21] provides data confiden-tiality in deduplication. A user (or data owner) derives a convergent key from the data content and encrypts the data copy with the convergent key. In addition, the user derives a tag for the data copy, such that the tag will be used to detect duplicates. Here, we assume that the tag correctness property [22] holds, i.e., if two data copies are the same, then their tags are the same. Formally, a convergent encryption scheme can be defined with four primitive functions:

- KeyGen(F) : The key generation algorithm takes a file content F as input and outputs the convergent key ck_F of F;
- Encrypt(ck_F ; F) : The encryption algorithm takes the convergent key ck_F and file content F as input and outputs the ciphertext ct_F ;
- Decrypt(ck_F ; ct_F) : The decryption algorithm takes the convergent key ck_F and ciphertext ct_F as input and outputs the plain file F;
- TagGen(F) : The tag generation algorithm takes a file content F as input and outputs the tag tag_F of F. Notice that in this paper, we also allow TagGen(·) to generate the (same) tag from the corresponding ciphertext as with [22][18].

V SECCLOUD

In this section, we describe our proposed SecCloud system. Specifically, we begin with giving the system model of Sec-Cloud as well as introducing the design goals for SecCloud. In what follows, we illustrate the proposed SecCloud in detail.

3



Aiming at allowing for auditable and deduplicated storage, we propose the SecCloud system. In the SecCloud system, we have three entities:

- Cloud Clients have large data files to be stored and rely on the cloud for data maintenance and computation. They can be either individual consumers or commercial organizations;
- Cloud Servers virtualize the resources according to the requirements of clients and expose them as storage pools. Typically, the cloud clients may buy or lease storage capacity from cloud servers, and store their individual data in these bought or rented spaces for future utilization;
- Auditor which helps clients upload and audit their out-sourced data maintains a MapReduce cloud and acts like a certificate authority. This assumption presumes that the auditor is associated with a pair of public and private keys. Its public key is made available to the other entities in the system.

The SecCloud system supporting file-level deduplication includes the following three protocols respectively highlighted by red, blue and green in Fig. 1.

1) File Uploading Protocol: This protocol aims at allowing clients to upload files via the auditor. Specifically, the file uploading protocol includes three phases:

- *Phase 1* (cloud client → cloud server): client performs the duplicate check with the cloud server to confirm if such a file is stored in cloud storage or not before uploading a file. If there is a duplicate, another protocol called Proof of Ownership will be run between the client and the cloud storage server. Otherwise, the following protocols (including *phase 2* and *phase 3*) are run between these two entities.
- *Phase 2* (cloud client → auditor): client uploads files to the auditor, and receives a receipt from auditor.

- *Phase 3* (auditor → cloud server): auditor helps generate a set of tags for the uploading file, and send them along with this file to cloud server.

2) Integrity Auditing Protocol: It is an interactive protocol for integrity verification and allowed to be initialized by any entity except the cloud server. In this protocol, the cloud server plays the role of prover, while the auditor or client works as the verifier. This protocol includes two phases:

- *Phase 1* (cloud client/auditor → cloud server): verifier (i.e., client or auditor) generates a set of challenges and sends them to the prover (i.e., cloud server).
- *Phase 2* (cloud server → cloud client/auditor): based on the stored files and file tags, prover (i.e., cloud server) tries to prove that it exactly owns the target file by sending the proof back to verifier (i.e., cloud client or auditor).

At the end of this protocol, verifier outputs true if the integrity verification is passed.

3) Proof of Ownership Protocol: It is an interactive protocol initialized at the cloud server for verifying that the client exactly owns a claimed file. This protocol is typically triggered along with file uploading protocol to prevent the leakage of side channel information. On the contrast to integrity auditing protocol, in PoW the cloud server works as verifier, while the client plays the role of prover. This protocol also includes two phases

- *Phase 1* (cloud server → client): cloud server generates a set of challenges and sends them to the client.
- *Phase 2* (client → cloud server): the client responds with the proof for file ownership, and cloud server finally verifies the validity of proof.

Our main objectives are outlined as follows.

- Integrity Auditing. The first design goal of this work is to provide the capability of verifying correctness of the remotely stored data. The integrity verification further requires two features: 1) *public verification*, which allows anyone, not just the clients originally stored the

3) Proof of Ownership Protocol: The PoW protocol aims at allowing secure deduplication at cloud server. Specifically, in deduplication, a client claims that he/she has a file F and wants to store it at the cloud server, where F is an existing file having been stored on the server. The cloud server asks for the proof of the ownership of F to prevent client unauthorized or malicious access to an unowned file through making cheating claim. In SecCloud, the PoW protocol is similar to [3] and the details are described as follows.

Suppose the cloud server wants to ask for the ownership proof for file F. It randomly picks a set of block identifiers, say $I_F \subseteq \{1; 2; : : : ; s\}$ where s is the number of blocks in F, for challenge. Upon receiving the challenge set I_F , the client first computes a short value and constructs a Merkle tree. Note that only sibling-paths of all the leaves with challenged identifiers are returned back to the cloud server, who can easily verify the correctness by only using the root of the Merkle tree. If it is passed, the user is authorized to access this stored file.

VI SECCLOUD$^+$

We specify that our proposed SecCloud system has achieved both integrity auditing and file deduplication. However, it cannot prevent the cloud servers from knowing the content of files having been stored. In other words, the functionalities of integrity auditing and secure deduplication are only imposed on plain files. In this section, we propose SecCloud$^+$, which allows for integrity auditing and deduplication on encrypted files.

A. System Model

Compared with SecCloud, our proposed SecCloud$^+$ in-volves an additional trusted entity, namely key server, which is responsible for assigning clients with secret key (according to the file content) for encrypting files. This architecture is in line with the recent work [4]. But our work is distinguished with the previous work [4] by allowing for integrity auditing on encrypted data.

SecCloud$^+$ follows the same three protocols (i.e., the file uploading protocol, the integrity auditing protocol and the proof of ownership protocol) as with SecCloud. The only difference is the file uploading protocol in SecCloud$^+$ involves an additional phase for communication between cloud client and key server. That is, the client needs to communicate with the key server to get the convergent key for encrypting the uploading file before the phase 2 in SecCloud.

Unlike SecCloud, another design goals of file confidentiality is desired in SecCloud$^+$ as follows. accessing the content of files. Specially, we require that the goal of file confidentiality needs to be resistant to "dictionary attack". That is, even the adversaries have pre-knowledge of the "dictionary" which includes all the possible files, they still cannot recover the target file [4].

B. SecCloud$^+$ Details

We introduce the system setup phase of SecCloud$^+$ as follows.

- System Setup. As with SecCloud, the auditor initializes the public key $pk = (g ; \{u_i\}^t_{i=1})$ and private key $sk =$, where $g; u_1; u_2; : : : ; u_t \in_R$ G. In addition, to preserve the confidentiality of files, initially, the key server picks a random key *ks* for further generating file encryption keys, and each client is assigned with a secret key *ck* for encapsulating file encryption keys.

Based on the initialized parameters, we then respectively describe the three protocols involved in SecCloud$^+$.

1) File Uploading Protocol: Suppose the uploading file F has s blocks, say $B_1; B_2; : : : ; B_s$, and each block B_i for $i = 1; 2; : : : ; s$ contains t sectors, say $B_{i1}; B_{i2}; : : : ; B_{it}$.

Client computes h_F = Hash(F) by itself. In addition,

for each sector B_{ij} of F where $i = 1; 2; : : : ; s$ and $j = 1; 2; : : : ; t$, client computes its hash $h_{B_{ij}} = \text{Hash}(B_{ij})$. Finally $(h_F ; \{h_{B_{ij}}\}_{i=1;:::;s;j=1;:::;t})$ is sent to key server for generating the convergent keys for F.

Upon receiving the hashes, the key server computes $ssk_F = f(ks; h_F)$ and $ssk_{ij} = f(ks; h_{B_{ij}})$ for $i = 1; : : : ; s$ and $j = 1; : : : ; t$, where ks is the convergent key seed kept at the key server, and $f(\cdot)$ is a pseudorandom function. It is worthwhile noting that, 1) We take advantage of the idea of convergent encryption [21][22][23] to make the deterministic and "content identified" encryption, in which each "content" (file or sector) is encrypted using the session key derived from itself. In this way, different "contents" would result in different ciphertexts, and deduplication works. 2) Convergent encryption suffers from dictionary attack, which allows the adversary to recover the whole content with a number of guesses. To prevent such attack, as with [4], a "seed" (i.e., convergent key seed) is used for controlling and generating all the convergent keys to avoid the fact that adversary could guess or derive the convergent key just from the content itself. 3) We generate convergent keys on sector-level (i.e., generate convergent keys for each sector in file F), to enable integrity auditing. Specifically, since convergent encryption is deterministic, it allows to compute homomorphic signatures on (convergent) encrypted data as with on plain data, and thus the sector-level integrity auditing is preserved.

Client then continues to encrypt F sector by sector and uploads the ciphertext to auditor. Specifically, for each sector B_{ij} of F, $i = 1; 2; : : : ; s$ and $j = 1; 2; : : : ; t$, client computes $ct_{B_{ij}} = \text{Enc}(ssk_{B_{ij}} ; B_{ij})$, and sends $(\text{ID}_F ; \{ct_{B_{ij}}\}_{i=1;:::;s;j=1;:::;t})$ to auditor, where $\text{Enc}(\cdot)$ is the symmetric encryption algorithm. The convergent keys ssk_{ij}

- File Confidentiality. The design goal of file confi- are encapsulated by client's secret key ck and directly stored dentiality requires to prevent the cloud servers from at the cloud servers.

2) Secure Deduplication: Similarly, we can also define a game between challenger and adversary for secure dedupli-cation below. Notice that the game for secure deduplication captures the intuition of allowing the malicious client to claim it has a challenge file F through colluding with all the other clients not owning this file.

- Setup Phase. A challenge file F with fixed length and minimum entropy (specified in system parameter) is randomly picked and given to the challenger. The challenger continues to run a summary algorithm and generate a summary sum_F .
- Learning Phase. Adversary F can setup arbitrarily many client accomplices not exactly having F and have them to interact with the cloud servers to try to prove the ownership of file F. Notice that in the learning

phase, the cloud server plays as the honest verifier with input sum sum_F and the accomplices could follow any arbitrary protocol set by A.

The verification for a different file F', it implies a collision of hash function used in the construction of Merkle Hash Tree. Based on the assumption of the collusion-free, this happens on with negligible probability.

To construct a simulator, that given g; $\tilde{g} = g$, h, where is unknown, outputs h . In the setup phase, the simulator sets v as $\tilde{g}$, chooses two vectors of randomness $_1; : : : ; _t \in Z_p$ and $_1; : : : ; _t \in Z_p$, and sets $u_j = g^j h^j$ for $j = 1; : : : ; t$. It additionally initiates an empty hash tables H-table and simulates the random oracle queries as follows.

When a hash query of B_i comes and an entry $(B_i; r_i)$ exists in the hash-table for some random value r_i, the simulator just returns g_i^r. When a query of a new B_i that has not been queried, the simulator performs the following steps. Firstly, it randomly chooses a value r_i from Z_p and puts $(B_i; r_i)$ in the Hash table H-table and returns Hash$(M_i) = g^{r_i}$

The simulator also needs to simulate the *Uploading Oracle*. Specifically, for a query of file F to be uploaded, the simulator

- Challenge Phase. The exact proof of ownership pro- computes the hash values and constructs Merkle Hash Tree to col is executed. Specifically, the challenger outputs a root R from the file. The proof is very similar to [17] and challenge to A and A responses with a proof based on its learnt knowledge. If A's proof is accepted by the cloud server, we say A succeeds.

The security in terms of secure deduplication is achieved, if for all probabilistic polynomial-time adversaries A, the prob-ability that A succeeds in the above experiment is negligible.

B. Security Proof

Theorem 1: Assume that the CDH problem is a hard prob-lem. Then, the proposed public-verifiable PoR scheme satisfies the soundness. That is, no adversary could generate an integrity proof for any file such that the verifier accepts it with non-negligible probability.

Proof: We prove the soundness of the construction by reduction. Firstly, assume there is an adversary who can break the soundness with non-negligible probability. We show that how to construct a simulator to break the computational Diffie-Hellman problem through interacting with the adversary. During this phase, the simulator is required to answer all the queries as the real application.

In more details, the simulator has to answer the tag genera-tion and integrity proof queries from the adversary. After the simulation, if the adversary outputs a valid tag that is not from client, the simulator can use this algorithm to solve the CDH problem. Notice that the simulation for the n slave nodes can be reduced to just one node because of the assumption that all the slave nodes are honest-but-curious and they will not collude. More clearly, the master key can be split to n sub-keys by choosing $n - 1$ random values and assigned to slave nodes as the corresponding private keys, while the n-th node is assigned the key of minus the sum of these random values.

Suppose that there exists an adversary who can generate the correct description of a prover. Denote $F = (B_1; : : : ; B_t)$ as the file for integrity verification, $\Phi = \{ _i\}_{1\le i\le t}$ as the signatures of blocks, and the set $Q = \{(i; c_i)\}_{i\in I_F}$ as the query. Denote by R the root generated from the file F. If the adversary can generate a correct root R from F which passes omitted here. The adversary can also start the query for the integrity proof. When an oracle query of a file tag, the simulator just starts an honest protocol with the adversary for the simulation.

After the above simulation, the adversary outputs a forgery of a valid signature $'$ $\neq$ satisfying the verification. Similar to the security analysis in [17], the simulator can compute and

get the value h $= (' {}^{-1} v^{-\sum_{j=1}}$ $_j{}^{\Delta_j})^{\sum_{j=1}}$ $_{j\Delta j}$ as the solution to the given CDH instance.

■ *Theorem 2:* An extractor can be constructed to recover the file in time $O(n^2(s + 1) + (1 + n^2)n=!)$ for well behaved -admissible prover by running $O(n=!)$ interactions on a n- block file with $! =$ $- 1=p - (n=n - c + 1)^c$.

Actually, such an extractor can be constructed to get correct proof for the verification queries in the protocol. With the combinatorial techniques, we can easily get the result that a -fraction of encoded file blocks can be retrieved after at most $O(n=!)$ interactions. Based on the rate- error correcting codes, all the file blocks are able to be recovered. The security model for the integrity verification protocol is the same as in Shacham and Waters' PoR model. Thus, the simulation for extracting the original file is similar to that in [12][17], which is omitted here.

By combining Theorem 1 and Theorem 2, we can directly have the following theorem.

Theorem 3: The proposed PoR construction is (;)-sound for any -admissible prover where $= 1 - (1 - 1=p)^{\log n+1} + 1=p$.

Regarding the file confidentiality of SecCloud$^+$, we have the following theorem.

Theorem 4: The proposed SecCloud$^+$ achieves confiden-tiality of file with the assumption that the adversary is not allowed to collude with the key server.

Proof: In our construction, a key server is introduced to generate the convergent key and hash values for the duplicate check. Without the private key stored at the key server, no adversary can generate a valid convergent key for any file with

Non-negligible probability. Thus, for the cloud storage server, without the help of key server, it cannot launch the brute force attack because the underlying hash value over the file is a valid message authentication code. Furthermore, all the data has been encrypted before they are outsourced. The data is encrypted with the traditional symmetric encryption scheme and the key is generated by the key server. The convergent key is encrypted by another master key and stored in the cloud server. The convergent key has been computed from both the file and private key of the key server, which means that the convergent key is not deterministic only in terms of the file. Even if the file is

predictable, the adversary cannot guess the file with brute-force attack if the adversary is not allowed to collude with the key server. ■

Because we used the PoW technique, based on the assump-tion of secure PoW scheme, any adversary without the file can-not convince the cloud storage server to get the corresponding access privilege. Thus, our deduplication system is secure in terms of the security model.

VII. PERFORMANCE ANALYSIS

In this section, we will provide a thorough experimental evaluation of our proposed schemes. We build our testbed by using 64-bit t2.Micro Linux servers in Amazon EC2 platform as the auditing server and storage server. In order to achieve = 80 bit security, the prime order p of the bilinear group G and G_T are respectively chosen as 160 and 512 bits in length. We also set the block size as 4 KB and each block includes 25 sectors.

SIZE OF FILE (MB)

Fig. 5.Tag Generation

Fig. 5 shows the time cost of slave node in MapReduce for generating file tags. It is clear the time cost of slave node is growing with the size of file. This is because the more blocks in file, the more homomorphic signatures are needed to be com-puted by slave node for file uploading. We also need to notice that there does not exist much computational load difference between common slave nodes and the reducer. Compared with the common slave nodes, reducer only additionally involves in a number of multiplications, which is lightweight operation. It is worthwhile noting that, the procedure of tag generation (the phase 2 and 3 in file uploading protocol) could be handled in Preprocessing, and it is not necessary for client to wait until uploading file.

Before examine the time cost of file auditing, we need to firstly make analysis and identify the number of challenging blocks (i.e., $|I_F|$) in our integrity auditing protocol. According to [5], if fraction of the file is corrupted, through asking the proof of a constant m blocks of this file, the verifier can detect the misbehavior with probability = $1 - (1 -)^m$. To capture the spirit of probabilistic auditing, we set the probability confidence = 70%; 85% and 99%, and draw the relationships between and m in Fig. 6. It demonstrates that if we want to achieve low (i.e., 70%), medium (i.e., 85%) and high (i.e., 99%) confidence of detecting any small fraction of corruption, we have to respectively ask for 130; 190 and 460 blocks for challenge.

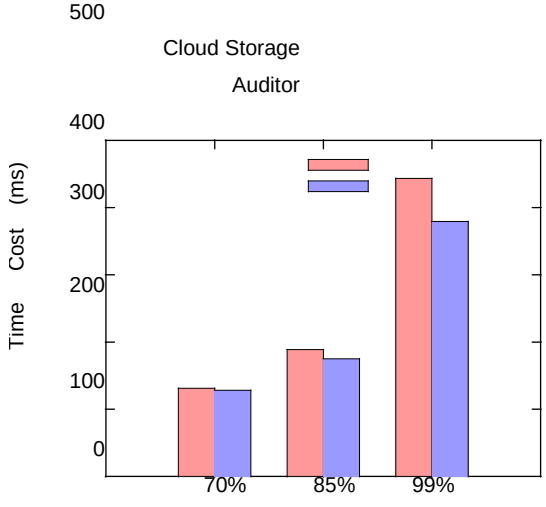


Fig. 7. File Auditing

Now, we come back to evaluate the time cost of file auditing in Fig. 7, which shows the time cost of auditing for detecting the misbehavior of cloud storage respectively with 70%*;* 85% and 99% confidence. Obviously, as the growth of the number of blocks for challenge (to guarantee higher confidence), the time cost for response from cloud storage server is increasing. This is because it needs to compute all the exponentiations for each challenge block as well as the coefficient for each column of *S*. Correspondingly, the time cost at auditor grows with the number of challenge blocks as well. But compared with cloud storage, the rate is slightly lower, because auditor only needs to aggregate the homomorphic signature of the challenged blocks.

VIII. CONCLUSION

Aiming at achieving both data integrity and deduplication in cloud, we propose SecCloud and SecCloud$^+$. SecCloud in-troduces an auditing entity with maintenance of a MapReduce cloud, which helps clients generate data tags before uploading as well as audit the integrity of data having been stored in cloud. In addition, SecCoud enables secure deduplication through introducing a Proof of Ownership protocol and pre-venting the leakage of side channel information in data dedu-plication. Compared with previous work, the computation by user in SecCloud is greatly reduced during the file uploading and auditing phases. SecCloud$^+$ is an advanced construction motivated by the fact that customers always want to encrypt their data before uploading, and allows for integrity auditing and secure deduplication directly on encrypted data.

ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foun-dation of China (No.61100224 and No. 61472091), NSFC-Guangdong (U1135002) and Natural Science Foundation of Guangdong Province (Grant No. S2013010013671).

REFERENCES

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communication of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
2. J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *IEEE Conference on Communications and Network Security (CNS)*, 2013, pp. 145–153.
3. S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM, 2011, pp. 491–500.
4. S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Washington, D.C.: USENIX Association, 2013, pp. 179–194. [Online]. Avail-able: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bellare
5. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceed-ings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 598– 609.
6. G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 12:1–12:34, 2011.
7. G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, ser. SecureComm '08. New York, NY, USA: ACM, 2008, pp. 9:1–9:10.
8. Erway, A. Kupc¨ ,u,¨ C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 213–222.
9. F. Sebe,´ J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *IEEE Trans. on Knowl. and Data Eng.*, vol. 20, no. 8, pp. 1034–1038, 2008.
10. H. Wang, "Proxy provable data possession in public clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 551–559, 2013.

11. Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Trans-actions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231–2244, 2012.
12. H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '08. Springer Berlin Heidelberg, 2008, pp. 90–107.
13. Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Computer Security – ESORICS 2009*, M. Backes and P. Ning, Eds., vol. 5789. Springer Berlin Heidelberg, 2009, pp. 355–370.
14. J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: ACM, 2012, pp. 79–80.
15. E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 229–238.
16. M. Azraoui, K. Elkhiyaoui, R. Molva, and M. Onen, "Stealthguard: Proofs of retrievability with hidden watchdogs," in *Computer Security - ESORICS 2014*, ser. Lecture Notes in Computer Science, M. Kutyłowski and J. Vaidya, Eds., vol. 8712. Springer International Publishing, 2014, pp. 239–256.
17. J. Li, X. Tan, X. Chen, and D. Wong, "An efficient proof of retrievability with public auditing in cloud computing," in *5th International Con-ference on Intelligent Networking and Collaborative Systems (INCoS)*, 2013, pp. 93–98.
18. J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou, "Secure dedupli-cation with efficient and reliable convergent key management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, June 2014.
19. R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: ACM, 2012, pp. 81–82.
20. W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, 2012, pp. 441–446.
21. J. Douceur, A. Adya, W. Bolosky, P. Simon, and M. Theimer, "Reclaim-ing space from duplicate files in a serverless distributed file system," in *22nd International Conference on Distributed Computing Systems*, 2002, pp. 617–624.
22. M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryp-tion and secure deduplication," in *Advances in Cryptology – EURO-CRYPT 2013*, ser. Lecture Notes in Computer Science, T. Johansson and P. Nguyen, Eds. Springer Berlin Heidelberg, 2013, vol. 7881, pp. 296–312.

23. M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. Garay, Eds. Springer Berlin Heidelberg, 2013, vol. 8042, pp. 374–391.

	Mr. Avula Praveen KumarReceived the B.Tech degree in Computer Science and Engineering and M.Tech degree in Computer Science and Engineering from J N T U Hyderabad University. He is Currently working as a Assistant Professor in Swami RamanandaTirtha Institute of Science and technology, Nalgonda, Telangana, India. He has having 9 years of teaching experience. . His Research interests include in Cloud Computing and Data Mining.
	Mr. PaillaThiramalReddy Received the B.Tech degree in Computer Science and Engineering and M.Tech degree in Computer Science and Engineering from J N T U Hyderabad University. He is Currently working as a Assistant Professor in Swami RamanandaTirtha Institute of Science and technology, Nalgonda, Telangana, India. He has having 4 years of teaching experience. His Research interests include in Data Mining and Cloud Computing.
	T.Madhu is working as a Associate Professor in Swami RamanandaTirtha Institute of Science and technology, Nalgonda,Telangana, India