Provision of Securing Data in Cloud through Auditing and Deduplication

Sandhya rani G.V [1] ,Naveen Ghorpade [2] , Dhananjaya V [3], K Prakash [4]
[1,4]Dept .of CSE, JNTUA, Aanatapur-01 , [2,3]Dept. of CSE, VTU, Bangalore-90,
Chintujob.blr@gmail.com ,naveen.ghorpade@gmail.com, csdhananjay@gmail.com, prakashkpmkp@gmail.com

Abstract— Outsourcing storage into the cloud is economically attractive for the cost and complexity of long-term large-scale data storage. At the same time, though, such a service is also eliminating data owners' ultimate control over the fate of their data, which data owners with high service-level requirements have traditionally anticipated. Specifically, aiming at achieving both data integrity and deduplication in cloud, we propose two secure systems, namely SecCloud and SecCloud+. SecCloud+ enables the guarantee of file confidentiality. The challenge of deduplication on encrypted is the prevention of dictionary attack. SecCloud introduces an auditing entity with maintenance of a MapReduce cloud. 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.

Keywords: SecCloud, SecCloud+, deduplication, MapReduce.

I. INTRODUCTION

Cloud storage provides customers with benefits, ranging from cost saving and simplified convenience, to mobility opportunities and scalable service. 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. 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. 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. 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. This fact raises a technology namely deduplication, 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).

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 deduplication in cloud, we propose two secure systems namely SecCloud and SecCloud+.

SecCloud introduces 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. SecCloud is supported on both block level and sector level. In addition, SecCloud also enables secure deduplication. A 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. SecCloud+ enables the guarantee of file confidentiality. The challenge of deduplication 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.

II. LITERATURE SURVEY

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. 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. 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]. Wang et al. [10] proposed proxy PDP in public clouds. Zhu et al. [11] proposed the cooperative PDP in multi-cloud storage. 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. Wang et al. [13] improved the POR model by manipulating 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, such that the disk space of cloud servers as well as network bandwidth are saved. Trivial client side deduplication leads to the leakage of side channel information. 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. 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.

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 introduced 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 encryption 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.

III. ALGORITHMS USED AND COMPUTATION

Some preliminary notions that will form the foundations of our approach.

A. Convergent Algorithm

Convergent encryption [22][23][21] provides data confidentiality 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 *ckF* of *F*.

• **Encrypt (*ckF; F*)**: The encryption algorithm takes the convergent key *ckF* and file content *F* as input and outputs the ciphertext ctF.

• **Decrypt (*ckF; ctF*)**: The decryption algorithm takes the convergent key *ckF* and ciphertext ctF 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 *tagF* of *F*. Notice that in this paper, we also allow TagGen (·) to generate the (same) tag from the corresponding ciphertext as with [22][18].

B. Bilinear Map and Computational Assumption

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

• **Bilinear:** $e(ga1 ; gb2) = e(g1; g2)ab$ for all $g1; g2 \in R$ G and $a; b \in R$ Zp.

• **Non-degenerate:** There exists $g1; g2 \in$ G such that $e(g1; g2) \neq 1$.

• **Computable:** There exists efficient algorithm to compute $e(g1; g2)$ for all $g1; g2 \in R$ G.

The examples of such groups can be found in super singular elliptic curves or hyper elliptic 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 problem, the hardness of which will be the basis of the security of our proposed schemes.

Definition 2 (CDH Problem): The Computational Diffie-Hellman problem is that, given $g; gx; gy \in$ G1 for unknown $x; y \in Z* p$, to compute gxy.

IV. SECCLOUD

In this section, we describe our proposed SecCloud system. we begin with giving the architecture of SecCloud as well as introducing the design goals for SecCloud. Aiming at allowing for auditable and deduplicated storage, we propose the SecCloud system. In the SecCloud system, we have three entities:

• Auditor which helps clients upload and audit their outsourced data maintains a Map Reduce cloud and acts like a certificate authority.

• Cloud Servers virtualize the resources according to the requirements of clients and expose them as storage pools.

• 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.

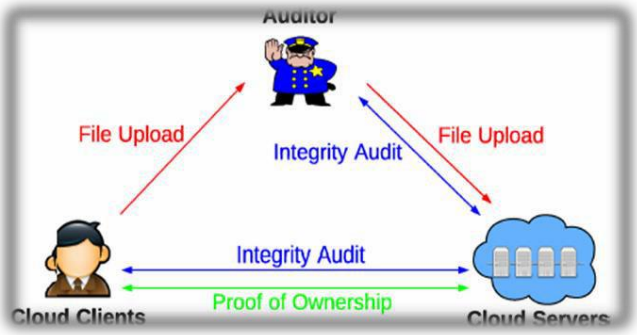

Fig. 1. SecCloud Architecture

The SecCloud system supporting file-level deduplication includes the following three protocols respectively highlighted in Fig. 1.

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

• Step 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 step 2 and step 3) are run between these two entities.

• Step 2 (cloud client → auditor): client uploads files to the auditor, and receives a receipt from auditor.

• Step 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.

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:

• Step 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).

• Step 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.

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 steps:

• Step 1 (cloud server → client): cloud server generates a set of challenges and sends them to the client.

• Step 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:

• **Cost-Effective**: The computational overhead for providing integrity auditing and secure deduplication should not represent a major additional cost to traditional cloud storage, nor should they alter the way either uploading or downloading operation.

• **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 file, to perform verification; 2) *stateless verification*, which is able to eliminate the need for state information maintenance at the verifier side between the actions of auditing and data storage.

• **Secure Deduplication**: The second design goal of this work is secure deduplication. In other words, it requires that the cloud server is able to reduce the storage space by keeping only one copy of the same file. Notice that, regarding to secure deduplication, our objective is distinguished from previous work [3] in that we propose a method for allowing both deduplications over files and tags.

B. SecCloud Details

In this subsection, we respectively describe the three protocols including file uploading protocol, integrity auditing protocol and proof of ownership protocol in SecCloud. Before our detailed elaboration, we firstly introduce the system setup phase of SecCloud, which initializes the public and private parameters of the system.

• **System Setup:** The auditor working as an authority picks a random integer _ ∈*R* Z*p* as well as random elements *g; u*1*; u*2*; : : : ut* ∈*R* G, where *t* specifies the maximum number of sectors in a file block. The secret key *sk* is set to be _ and kept secret, while the public key *pk* = (*g_; {ui}t i*=1) is published to other entities.

File Uploading Protocol: Based on the public and private parameters generated in system setup, we then describe the file uploading protocol. Suppose the uploading file *F* has *s* blocks: *B*1*;B*2*; : : : ;Bs*, and each block *Bi* for *i* = 1*;* 2*; : : : ; s* contains *t* sectors: *Bi*1*;Bi*2*; : : : ;Bit*. Let *n* be the number of slave nodes in the MapReduce cloud. The client runs the deduplication test by sending hash value of the file Hash (*F*) to the cloud server. If there is a duplicate, the cloud client performs Proof of Ownership protocol with the cloud server which will be described later. If it is passed, the user is authorized to access this stored file without uploading the file. Otherwise (in the second phase), the cloud client uploads a file *F* as well as its identity ID*F* to the distributed file system in MapReduce auditing cloud, and simultaneously sends an "upload" request to the master node in MapReduce, which randomly picks *{_i}ni* =1 such that Σ*n i*=1 *_i* = _ and assigns the *i*th slave node with *_i*. When each slave node (say the *i*th salve node) receives the assignment *_i*, it does two steps: 1) Pick up (ID*F;F*) in the distributed file system in MapReduce, and build a Merkle hash tree on the blocks *{Bj}sj*=1 of *F*. 2) Let *hroot* denote the hash of the root node of Merkle hash tree built on *F*. This slave node uses *_i* to sign *hroot* by computing *_i* = *h_iroot*. Finally, the signature *_i* is sent to the the slave node which is specified by master node for executing the *reducing* procedure. The specified slave node for reducing procedure gathers all the signatures *{_i} ni* =1 from the other slave nodes, and computes _ = Π*n i*=1 *_i*. The "reduced" signature _ is finally sent back to client as receipt of the storage of file *F*. In the third phase, the MapReduce auditing cloud starts to upload the file *F* to cloud server. To allow public auditing, the master node builds file tags of *F*. Specifically, master node firstly writes and arranges all the sectors of *F* in a matrix (we say *S*), and computes a homographic signature for each *row* of the matrix *S* (highlighted red in Fig. 3). Notice that the tag generation procedure also follows the computing paradigm with MapReduce. That is, for the *i*th (*i* = 1*;* 2*; : : : ; s*) row of *S*, the *j*th (*j* = 1*;* 2*; : : : ; n*) slave node computes _*ij* = [Hash(ID*F*||*Bi*) Π*t k*=1 *uBik k*]_*j* , where Σ*n j*=1 _*j* = _. Accordingly, all the signatures *{_ij} nj* =1 are then multiplied into the homomorphic signature _*i* = Π*n j*=1 _*ij* at a specified reducing slave node. The homomorphic signature allows us to in future aggregate the signatures signed on the sectors in the same *column* of *S* using multiplication. Finally, the master node uploads (ID*;F; {_i}si* =1) to cloud server.

Integrity Auditing Protocol: In the integrity auditing protocol, either the MapReduce auditing cloud or the client works as the verifier. Thus, without loss of generality, in the rest of the description of this protocol, we use verifier to identify the client or MapReduce auditing cloud. The auditing protocol is designed in a challenge-response model. Specifically, the verifier randomly picks a set of block identifiers (say *IF*) of *F* and asks the cloud server (working as prover) to response the blocks corresponding to the identifiers in *IF*. In order to keep randomness in each time of challenge, even for the same *IF*, we introduce a random coefficient for each block in challenge. That is, for each identifier *i* ∈*IF*, the coefficient *ci* for the block identified by *i* is computed as *ci* = *f* (*tm*||ID*F*||*i*), where *f*(·) is a pseudorandom function and *tm* is the current time period. Finally, *C* = *{*(*i; ci*)*}i*∈ *IF* is sent to cloud server for challenge.

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 files 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 *IF* ⊆*{*1*;* 2*; s}* where *s* is the number of blocks in *F*, for challenge. Upon receiving the challenge set *IF*, 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.

V. 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 Architecture

Compared with SecCloud, our proposed SecCloud+ involves 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.

• **File Confidentiality**: The design goal of file confidentiality requires to prevent the cloud servers from 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*_*; {ui}t i*=1) and private key *sk* = _, where *g; u*1*; u*2*; : : : ; ut* ∈*R* G. In addition, to preserve the confidentiality of files, initially, the key server picks a random key *k* 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+.

File Uploading Protocol: Suppose the uploading file *F* has *s* blocks, say *B*1*;B*2*; : : : ;Bs*, and each block *Bi* for *i* = 1*;* 2*; : : : ; s* contains *t* sectors, say *Bi*1*;Bi*2*; : : : ;Bit*.Client computes *hF* = Hash(*F*) by itself. In addition, for each sector *Bij* of *F* where *i* = 1*;* 2*; : : : ; s* and *j* = 1*;* 2*; : : : ; t*, client computes its hash *hBij* = Hash(*Bij*). Finally (*hF; {hBi}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 *sskF* = *f*(*ks; hF*) and *sskij* = *f*(*ks; hBij*) for *i* = 1*; : : : ; s* and *j* = 1*; : : : ; t*, where *ks* is the convergent key seed kept at the key server, and *f*(·) is a pseudorandom function. It is worthwhile nothing 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 cipher texts, 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 *Bij* of *F*, *i* = 1*;* 2*; : : : ; s* and *j* = 1*;* 2*; : : : ; t*, client computes *ctBij* = Enc(*sskBij ;Bij*), and sends (ID*F; {ctBij }i*=1*;:::;s;j*=1*;:::;t*) to auditor, where Enc(·) is the symmetric encryption algorithm. The convergent keys *sskij* are encapsulated by client's secret key *ck* and directly stored at the cloud servers.

Integrity Auditing Protocol: The integrity auditing protocol works in the same way of that in SecCloud, but imposed on encrypted data. Specifically, the verifier (could be either the client or the auditor) submits a set of pairs *{*(*i; ci*)*}i*∈ *IF* where *IF* ⊆ *{*1*;* 2*; : : : ; s}* and *ci* ∈ *R* Z. Upon receiving *{*(*i; ci*)*}i*∈*IF* , the cloud servers then computes *!j* = Σ *i*∈ *IF cictBij* for each *j* = 1*;* 2*; : : : ; t*, as well as the aggregated homomorphic signature _ = Π *i*∈*IF _ci i* . In addition, the cloud server constructs a Merkle hash tree on encrypted blocks *ctBi* of *F* and attempts to prove retrievability at block-level. Precisely, for each *i* ∈ *IF*, the cloud server computes a pair (Hash(*ctBi*)*;*Ω*i*), where *ctBi* = [*ctBi*1 *; : : : ; ctBit*] and Ω*i* includes the necessary auxiliary information for reconstructing the root node using *{ctBi }i*∈*IF* . Finally (_*; {!j}t j*=1*; {*(Hash(*ctBi*)*;*Ω*i*)*}i*∈*IF*) is sent to verifier for auditing.

VI. SECURITY ANALYSIS

In this section, we attempt to analyze the security of our proposed both schemes. Before this, we firstly formalize the security definitions our schemes aim at capturing.

A. Security Definitions

Based on the paradigm of SecCloud and SecCloud+, we define the security definitions, adapting to the integrity auditing and secure deduplication goals. Our both definitions capture the philosophy of game-based definition. Specifically, we define two games respectively for integrity auditing and secure deduplication, and both of the games are played by two players, namely adversary and challenger. The adversary (the role of which is worked by semi-honest cloud server and cloud client respectively in integrity auditing and secure deduplication definition) is trying to achieve the goal condition explicitly specified in the game. Having this intuition, we give our security definitions as follows.

1) Integrity Auditing: An integrity auditing protocol is sound if any cheating cloud server that convinces the verifier that it is storing a file *F* is actually storing this file. To capture this spirit, we define its game based on Proof of Retrievability (PoR). The security model called Proof of Retrievability (PoR) was introduced by Shacham and Waters' in [12]. This security model captures the requirement for integrity auditing, whose basic security goal is to achieve proof of retrievability. In more details, in this security model, if there exists an adversary who can forge and generate any valid integrity proofs for any file *F* with a non-negligible probability, another simulator can be constructed who is able to extract *F* with overwhelming probability. The formal definition for the above model can be given by the following game between a challenger and an adversary *A*. Note that in the following security game, the challenger plays the role of auditing server while the adversary *A* acts as the storage server.

• **Setup Phase**: The challenger runs the setup algorithm with required security parameter and other public parameter as input. Then, it generates the public and secret key pair (*pk; sk*). The public key *pk* is forwarded to the adversary *A*.

• **Query phase**: The adversary is allowed to query the file upload oracle for any file *F*. Then, the file with the correct tags are generated and uploaded to the cloud storage server. These tags can be publicly verified with respect to the public key *pk*.

• **Challenge Phase**: *A* can adaptively send file *F* to the file tag *tag* comes, *C* runs the integrity verification protocol Integrity Verify *{A* C (*pk; tag*)*}* with *A*.

• **Forgery**: *A* outputs a file tag *tag′* and the description of a prover *Pt*.

Secure Deduplication: Similarly, we can also define a game between challenger and adversary for secure deduplication 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 *sumF*.

• **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 *sumF* and the accomplices could follow any arbitrary protocol set by *A*.

• **Challenge Phase**: The exact proof of ownership protocol is executed. Specifically, the challenger outputs a 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.

VII. PERFORMANCE ANALYSIS

In this section, we will provide a thorough experimental evaluation of our proposed schemes. We build our test bed 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. 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 computed 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 pre-processing, 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., $|IF|$) 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 misbehaviour 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.

VIII. CONCLUSION

Aiming at achieving both data integrity and deduplication in cloud, we propose SecCloud and SecCloud+. SecCloud introduces 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 preventing the leakage of side channel information in data deduplication. 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 Foundation of China (No.61100224 and No. 61472091), NSFCGuangdong (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, G. 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: Serveraided 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]. Available:https://www.usenix.org/conference/usenixsecurity 13/technicalsessions/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 *Proceedings 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, Z. 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] C. Erway, A. K¨upc¸ ¨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. Seb´e, 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 Transactions 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. O¨ nen, “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 Conferenc 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 deduplication 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, “Reclaiming 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 encryption and secure deduplication,” in *Advances in Cryptology – EUROCRYPT 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.
[24] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” in *Advances in Cryptology — CRYPTO 2001*, ser. Lecture Notes in Computer Science, J. Kilian, Ed. Springer Berlin Heidelberg, 2001, vol. 2139, pp. 213–229.