SYSTEM INFORMATION AND DEDUPLICATIONDATA IN CLOUD

Mrs.C.SELVARATHI
Assistant Professor / CSE,
M.Kumarasamy College of Engineering,
Karur –639 113
selvarathic.cse@mkce.ac.in

Mr.J.MOHAMMED NIVAZUDDIN,
Final Year / CSE,
M.Kumarasamy College of Engineering,
Karur – 639 113.
pappunivaz@gmail.com

Mr.P.MANIGANDAN,
Final Year / CSE,
M.Kumarasamy College of Engineering,
Karur–639113.
pmanigandancs05@gmail.com

Abstract--Cloud computing technology develops during the last decade, 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.Specifically, aiming at achieving both data integrity and deduplication in cloud.We propose secure systems, namely SecCloud and SecCloud+

Index Terms--Cloud computing,Integrityauiditing,SecCloud and SecCloud+

1 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 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 server.

In this paper, aiming at achieving data integrity and dedu-plication in cloud, we propose two secure systems namely SecCloud and SecCloud$^+$.

2 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.

2.1 Integrity Auditing

The definition of provable data possession (PDP) was introduced by Ateniese etal.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, several works concerned on how to realize PDP on dynamic scenario: Ateniese et al.proposed a dynamic PDP schema but without insertion operation; Erway et al. improved Ateniese et al.'s work and supported insertion by introducing authenticated flip table; A similar work has also been contributed.Nevertheless, these proposals suffer from the computational overhead for tag generation at the client. To fix this issue, Wang et al proposed proxy PDP in public clouds. Zhu et al proposed the cooperative PDP in multi-cloud storage.

Another line of work supporting integrity auditing is proof of retrievability (POR). Compared with

PDP, POR not merely assures the cloud servers possess the target files, but also guarantees their full recovery. In, clients apply erasure codes and generate authenticators for each block for verifiability and retrievability. In order to achieve efficient data dynamics, Wang et al. improved the POR model by manipulating the classic Merkle hash tree construction for block tag authentication. Xu and Chang proposed to improve the POR schema in with polynomial commitment for reducing communication cost. Stefanovetal.proposed a POR protocol over authenticated file system subject to frequent changes. Azraoui et al. 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. 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. Utilized the key-disperse paradigm to fix the issue of a significant number of convergent keys in convergent encryption.

2.2 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.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 to enable secure client-side deduplication. Pietro and Sorniottiproposed 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.Firstly intro-duced the private data deduplication as a complement of public data deduplication protocols of Halevi et al. Convergent encryption is a promising cryptographic primitive for ensuring data privacy in deduplication. Bellare et al formalized this primitive as message-locked encryption, and explored its application in space-efficient secure outsourced storage. Abadietal.Further strengthened Bellare etal's security definitions 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.Designed the DupLESS system in which clients encrypt under file-basedkeys derived from a key server via an oblivious pseudorandom function protocol.

3 PRELIMINARY

3.1 Convergent Encryption

Convergent encryption 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 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 function.

- KeyGen(F) : The key generation algorithm takes a file content F as input and outputs the convergent key $ckf$of F
- Encrypt($ck_F;F$) : The encryption algorithm takes the
- convergent key ck_F and file content F as input and outputs the ciphertextct_F ;
- Decrypt($ck_F;$ ct_F) : The decryption algorithm takes the convergent key ck_F and ciphertextct_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($\cdot$) to generate the (same) tag from the corresponding ciphertext.

4 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

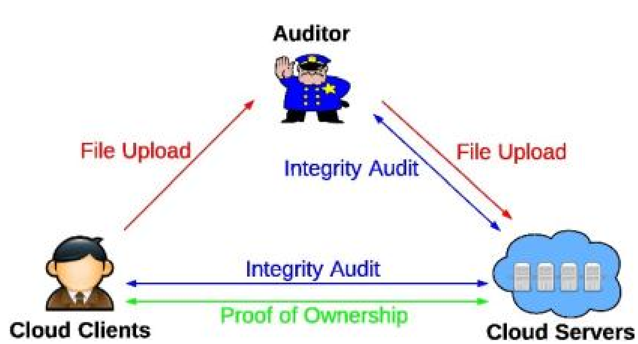

Fig. 1. SecCloud Architecture

A. System Model

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 allowingclients to upload files via the auditor. Specifically, the file uploading protocol includes three phases:

- **Phase 1**(cloud client→cloud server): client performsthe 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 tothe auditor, and receives a receipt from auditor.

- **Phase 3** (auditor→cloud server): auditor helps generatea 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 protocolfor 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 onthe 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 protocolinitialized 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 generatesa set of challenges and sends them to the client.
- **Phase 2**(client→cloud server): the client responds withthe 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 al-lows 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 in that we propose a method for allowing both deduplication over files and tags.

- **Cost-Effective**. The computational overhead for provid-ing 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.

5 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.

5.1 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. But our work is distinguished with the previous work 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 confident 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.

6 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.

6.1 Security Definitions

Based on the paradigm of SecCloud and SecCloud$^+$, we define the security definitions, adapting to the integrity au-diting 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 soundif 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'. 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 pa-rameter 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 IntegrityVerify*{A C*(*pk; tag*)*}* with *A*.
- Forgery. *A* outputs a file tag tag' and the description of a proverP_t.

2) Secure Deduplication: Similarly, we can also define agame 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*.

7 CONCLUSION

Aiming at achieving both data integrity and deduplication in cloud, we propose SecCloud and SecCloud+.SecCoud enables secure deduplication through introducing a Proof of Ownership protocol and preventing the leakage of side channel information in data deduplication.SecCloud+ is an advanced construction motivated by the fact that customers always want to encrypt their data before uploading

REFERENCE

1. 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
2. J. Li, X. Tan, X. Chen, and D. Wong, “An efficient proof of retrievability with public auditing in cloud computing,” in 5th International Conference on Intelligent Networking and Collaborative Systems (INCoS), 2013, pp. 93–98
3. 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.
4. 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