International Journal of Engineering & Technology, 7 (2.8) (2018) 13-17

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

Privacy preserving proof of ownership for data in cloud storage systems

B. Tirapathi Reddy [1, 2] *, M. V. P. Chandra Sekhara Rao [3]

[1] *Research Scholar, ANU College of Engineering, Acharya Nagarjuna University, Andhra Pradesh, India*
[2] *Associate Professor, Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, India 522502*
[3] *Professor, Department of CSE, RVR&JC College of Engineering, Guntur, Andhra Pradesh, India*
**Corresponding author E-mail: tirupathireddyb@gmail.com*

Abstract

Storing data in cloud has become a necessity as users are accumulating abundant data every day and they are running out of physical storage devices. But majority of the data in the cloud storage is redundant. Data deduplication using convergent key encryption has been the mechanism popularly used to eliminate redundant data items in the cloud storage. Convergent key encryption suffers from various drawbacks. For instance, if data items are deduplicated based on convergent key, any unauthorized user can compromise the cloud storage by simply having a guessed hash of the file. So, ensuring the ownership of the data items is essential to protect the data items. As cukoo filter offers the minimum false positive rate, with minimal space overhead our mechanism has provided the proof of ownership.

Keywords: *Cloud Storage; Proof of Ownership; Data Deduplication.*

1. Introduction

Cloud storage models available on payment basis, their scalability on demand, security and performance has attracted many of the computer users in recent times. With the increased interest in collecting and archiving huge volumes of data, using the cloud storage seems to be the only viable solution for the computer users [1]. As cloud computing offers an abstraction of unlimited storage with minimal effort, people are attracted towards the cloud storage. But mostly the data items uploaded to the cloud storage are duplicate in nature, which leads to wastage of storage space. Data deduplication mechanisms have been used extensively to avoid redundant storage of data items. Majority of the cloud storage service providers are adopting data deduplication schemes to minimize storage overhead, i.e, eliminated duplicate data items. Deduplication can happen at the user side which is referred as the client side data deduplication. If deduplication happens at the storage provider, it is called the server side data deduplication. Deduplicaton can take place at the file level or block level. Based on the setup and input data items deduplication ratios may vary from 4:1 to 50:1. [12][20]

Deduplication is a process to avoid redundant data items from the cloud storage by leveraging the advantage of convergent key encryption. Convergent key encryption [4][3] is a mechanism that produces same cipher text for the given plain text. In convergent key encryption the file is applied to the hash function and the resulting hash code is used as the key to encrypt the file. Due to hash function properties, when different users encrypt the file with hash of the file, it produces same cipher text. This process will help the cloud storage provider to verify the duplication of the data items, without knowing the corresponding original content of the file.

As several users are using the cloud storage to archive their data items [19] [20], ensuring privacy[2] [18] [9] of the data items and the users is essential. However, verification of duplicate data items leaks the privacy of the data items and the users, i.e process of verifying the redundant data items is not compatible with the conventional way of converting the data into secret form using symmetric key encryption. To make deduplication compatible with symmetric key encryption, convergent key encryption is used.

Client side data deduplication provides more benefits in terms of minimizing the upload cost and bandwidth consumption, as the process of verifying the duplication of data items takes place, prior to uploading the data items. The efficiency of this mechanism depends on how securely data is deduplicated and how it controls the access to the data items, i.e. how securely data is accessible to the users. To make deduplication compatible with symmetric key encryption, convergent key encryption is considered. Rather, convergent key encryption suffers from various drawbacks.

As the cloud storage provider maintains hashes of all the files, in order to gain access to the file client sends the hash of the file to verify whether same hash value exists in the database of the storage provider. If the hash exists in the database, storage server restricts the user from uploading the data as the file already exists and he/she will be included in the list of owners of that file. Whereas, if the hash value is not identified in the cloud storage provider, the file does not exist in the cloud storage server. Subsequently, the user can upload the file. After uploading the data, the user is made the owner of the file.

In either case the small hash value will be used as the proxy for the entire file. Any user who possesses that hash value is able to access the file. Because of this, several security problems may arise [11] [13] [15]. Any unauthorized user can persuade the cloud storage provider that he/she is the authorized user to access the file. To illustrate: Assume that there are two users A and B. B is a genuine and loyal user, uploads data items to the cloud and accesses them. If User A wants to know the data uploaded to the cloud by User B he tries to upload a file by suppling the same hash of file, which User B has uploaded. As cloud storage provid-

er identifies similar files based on the hash of the file, he assumes that Users A and B have uploaded the same file. In future, If user A wishes to download the file, he is asked to submit the hash of the file, and after submission, cloud authorizes him to download the file. This problem originates from the fact that any user can access the file if he/she has information like hash of the data item. Furthermore, they can persuade the cloud service provider that he /she actually owns the data item and can access the file. So, it is required to identify the actual owner to avoid many problems. Proof of ownership techniques is a way for the CSP to verify whether the client possesses the file or not.

2. Literature work

Benny Pinkas [5] et. al describes the drawbacks with client side data deduplication and provides solution to deal with the confidentiality and privacy problems. The biggest problem is the privacy factor. Any unauthorized user is able to access the contents and the cloud storage provider has become a data distribution centre by which the privacy of data is lost.

To address the problems identified in client side data deduplication Halevi et al [6] introduced the proof of ownership scheme, to authorize whether the client trying to upload a file really owns the file or not.. The author presented three different solutions to verify the ownership of the data items using merkle trees, constructed with the content of the original data items. In all the three techniques cloud server challenges the client to submit the sibling paths for Random subset of tree leaves. Both the storage server and the client will construct merkle tree and cloud storage provider maintains the root node only. To prove the ownership of the data item , the client has to submit the super logarithmic random number of leaf nodes of the constructed merkle tree. Cloud storage server computes the root of the tree based on the leaf nodes received and sibling paths, and is compared with the root node saved by the cloud storage server. If the computed root node is equivalent to the stored root node, proof of ownership is verified. In the first technique merkle tree is constructed from erasure code of the file and spreads unfamiliar blocks of data file over the number of data blocks of particular erasure coded version. But the erasure code is not proficient in terms of I/O, as it incurs more space overhead. The second technique makes use of hash functions without using erasure coding to generate the merkle tree. It makes use of a reduction buffer of 64 MB to avoid sharing of data items by compromised users. In Terms of space overhead, the second mechanism is advisable than the first mechanism. But, as the computation complexity is more another technique is proposed. In the third technique server performs a collection of reduction, mixes phases on contents of the file received from the user for the first time, produces a sparse linear file. The computational complexity at the client and server side is less when compared to the other two techniques.

Roberto Di Pietro et al [7] [10] introduced a technique considering the challenge - response mechanism. According to it, when a file is uploaded to the cloud storage server, the server computes the challenges for the data file and stores them . The server maintains a hash - map data structure, which maps strings of finite size to four tuples: containing a pointer to the file, an array of responses and indexes which keeps track of highest challenge used so far and a count of the number of challenges used so far respectively. Whenever the user wants to prove the ownership, he needs to send the identifier of the data file. Cloud server selects the unused challenge from the pre computed challenges based on the index and forwards to the user. The user has to compute the responses based on the knowledge of the data items, and then sends to the cloud storage server. If the computed responses match with the stored response, ownership of the file is verified. In addition to providing proof of ownership, several authors have proposed mechanisms to eliminate redundant data items form the cloud storage.

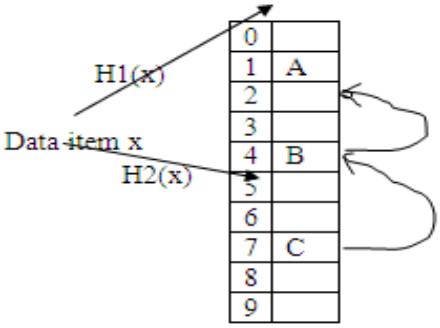
2.1. Cuckoo filter

It is considered as a probabilistic, high speed data structure which permits set membership operations. The advantage with cuckoo filter is that it is space efficient and requires less time to perform the operations. The main problem with the probabilistic data structures is that there is a possibility of false positivity, i.e element which doesn't exists in the set will be accepted as part of the set. The false positive rate in cuckoo filters is less compared to any than any other probabilistic data structures. Another advantage with cuckoo filter is, it allows addition and deletion of items and also consumes less space. All the other probabilistic data structures do not permit deletion of items dynamically but cuckoo filter permits deletion of items with minimal overhead.

A cuckoo filter contains cuckoo hash table that stores the fingerprint i.e., bit string retrieved from item to be inserted using the hash function. And it also contains an array of buckets, in which an item to be inserted has two candidate buckets calculated using the two independent hash functions H1(x),H2(x). Each bucket can accommodate varying number of fingerprints. Cuckoo filter in general is identified by its Bucket size and Fingerprints.

The cuckoo filter permits the operations like insert, delete and search. To insert a data item into the cuckoo filter, we have to obtain two indices by applying the data item to the two independent hash functions H1(x), H2(x) and calculates its fingerprint. Then place the elements into the hash table based on the indices. If the indices are not available, then find the alternate indices using the partial-key cuckoo hashing till the data item fits into the hash table. To avoid collisions, we make use of the perfect hash functions[20]. In order to search for the existence of a data item, we have to calculate the fingerprint of the data item and the candidate buckets. If the fingerprint existing in the candidate buckets does not match the cuckoo filter, it replies false, and in the same way if a match occurs, it replies true.

A) Before Item Is Inserted



B) After Data Item Is Inserted

0	
1	A
2	B
3	
4	C
5	
6	
7	X
8	
9	

Fig. 1: Illustration of Cuckoo Hashing.

The process of inserting elements into the cuckoo hash table is represented in the figure 1. Figure 1(a). It provides an example to insert a data item x into the hash table of size 10 buckets, in which the element x may be placed in buckets 1 or 7. If any of the x's candidate buckets are not occupied then x can be placed in bucket

1 or 7. If no candidate bucket is empty as in this case, the item picks any of the
candidate buckets, removes the item from the bucket and reinserts that item into its alternate location. The process is continued till it traces an empty location.

3. Objective of the proposed scheme

In a partially trusted cloud computing environment establishing the privacy of the data items is essential. To verify proof of ownership for the data items we consider interactions between the clients and the cloud storage service provider CSP. For example User A will have a unique identifier Id (A). When the user tries to upload the data items to the CSP, the interaction between the user and the CSP differs based on the data uploaded for the first time or subsequent uploading of data. If the user tries to upload a data item, server will accept the data item to be uploaded and initializes a cuckoo filter to be used further. But, if the same data item is uploaded by a different user or the same user uploading it again the upload is considered as a subsequent upload and the server challenges the user to prove knowledge about the data item. It verifies whether the response supplied by the user matches with the computed information.

3.1. Proposed scheme

Whenever user A wants to store a file 'f', hash of the file is calculated and forwarded to the server. Server initializes a cuckoo filter of certain size. The fingerprints of the data item are calculated based on the hash of the data item. The two candidate buckets are identified and fingerprints are stored into these candidate buckets. If there are collisions, the data items stored in these locations are forced to their alternate locations as per the Algorithm 1.

Algorithm 1: Insertion of element into cuckoo hash table

Fi = Fiprint(x);
l1 = Hash(x);
l2 = l1 ⊕ Hash (Fi):
If Bucket [l1] or Bucket [l2] does not have an entry
Then insert Fi into that Bucket;
Return DONE;
l = Randomly pick l1 or l2;
for n = o; n < MaxNum Kicks; n=n+1 do
at random pick the entry e from Bucket[l];
Exchange Fi and Fprint available in entry e;
l = l⊕Hash (Fi);
If bucket[l] consists of no entry
Then ADD Fi to Bucket[l];
Return DONE;
Return FAILURE;

The CSP will maintain an Associative array AA, which maps elements of fixed size to 3-tuples: The array contains contents of the file, indexes of the cuckoo filter and identifiers of the list of owners.
This scheme works in two phases: In the first phase, when the user uploads a data item for the first time, the storage server will initialize a cuckoo filter and inserts the data items, i.e. fingerprints of the data items into the cuckoo hash table as per the Algorithm 1. The input file supplied by the user will be divided into a set of fixed size block, prepares token for each block of the file, and inserts the function of every token into the cuckoo file of the file. After the initialization of the cuckoo filter, if the same user or any other tries to upload the file F, the CSP will receive the hash code of the file from the user, considering the hash code as the look up query; it will check whether the same hash code exists in the hash table maintained by the CSP. If the received hash value matches with the stored hash value, it does mean that same file is already present at the CSP.
In the second phase it verifies whether the user actually owns the file or not, CSP challenges the user to upload some random number of tokens about the file to prove its knowledge. Upon receiving the tokens from the user the cloud storage provider searches for these tokens in the initialized cuckoo filter, as per the Algorithm 2. If the tokens are found in the candidate buckets then the cuckoo filter returns true otherwise it returns false. If the cuckoo filter returns false, it does mean that user does not have the file, he is not capable of proving the ownership of file to the cloud storage provider. Thus, he is restricted to access the file. If the cuckoo filter returns true then the ownership of the file is verified and user is allowed to access the file. The user can download the file by sending the hash of the file to cloud service provider; if the data item exists, at the cloud storage provider, the server checks whether the identifier of the owner belongs to the list of owners of the file, and sends the file to the owner upon successful verification.

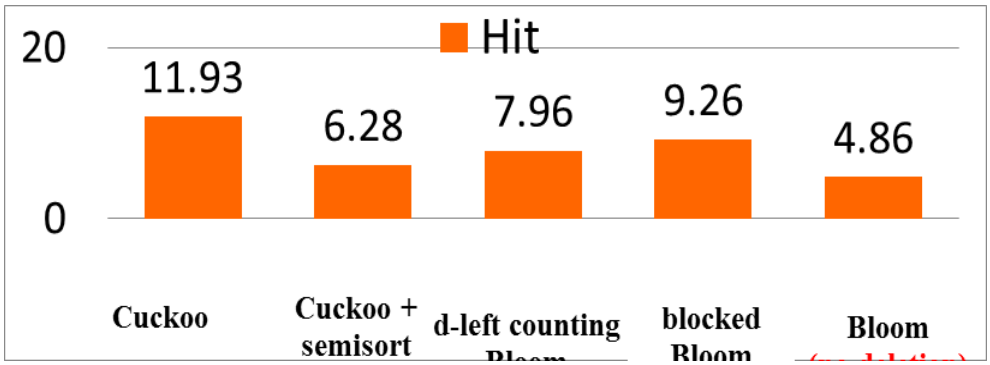
Algorithm 2: searching the element in cuckoo filter

Fi = Fprint(x);
l1 = Hash(x);
l2 = l1 ⊕Hash (Fi);
If Bucket [l1] or Bucket [l2] has Fi then
Return T;
Return F;

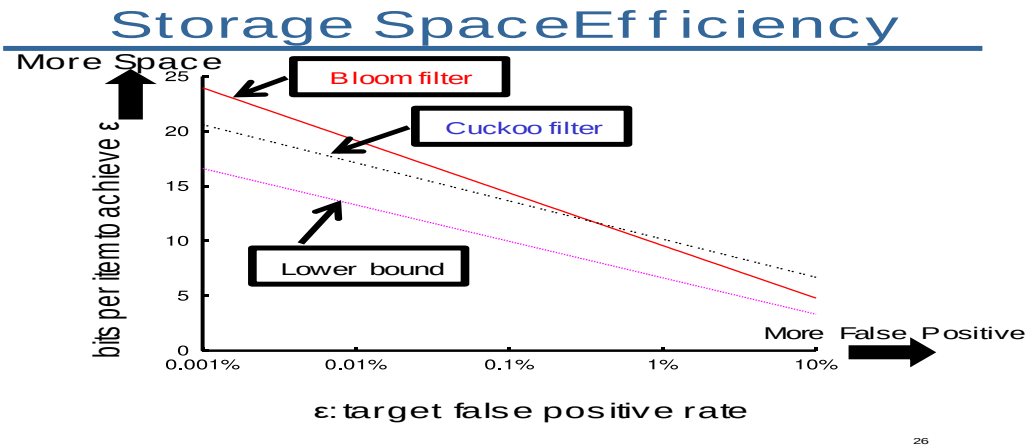
As the false positive rates with the cuckoo filter are very less when compared to any other probabilistic data structures[8] the process of verifying the ownership of the file produces accurate results. There are many other probabilistic data structures which offer membership queries on the large collection of data items, out of which cuckoo filter offers better performance in terms of space and search time.

4. Experimental setup

The researcher has implemented the scheme using C++. To implement all cryptographic operations OpenSSL Crypto Library is used. [14] Secure hash algorithm SHA-1 is used to implement the Hash (x). The scheme is implemented in Intel Xeon E5520 Nehalem 2.26 GHz processor with 20 Giga bytes of Random Access Memory running RHEL Server version 6.0. The input file size varies from 1 Mega bytes to 4 Giga Bytes.
The probability that unauthorized user knows the block of the file is considered as {0.6, 0.75, 0.9}, and the token size is considered as {64, 256, 512, 1024} bytes.
The Search performance of the cuckoo filter is compared with various types of blooms filters, out of which the cuckoo filter has produced better results than any other kind of probabilistic data structures. In terms of space efficiency cuckoo filter is far better than the Bloom filter. All the other probabilistic data structures could not allow the deletion of data items whereas, cuckoo filter offers deletion of data items and offers better performance.
The performance of cuckoo filter is compared with the Blooms filter [16] and the storage efficiency of cuckoo filter over bloom filter is shown below.



Search performance in terms of MOPS



When the false positive rate is low the cuckoo filter consumes less space than the blooms filter. As the target false rate increases, there is a slight overhead in terms of storage space with the cuckoo filter. Another advantage with cuckoo filter is, it permits deletion of items from the filter, whereas bloom's filter does not permit deletion of data items. The researcher compared the discussed mechanism with the mechanisms proposed by halevi et al and by R. Di Pietro et al. The current mechanism incurs less computational overhead when compared with the other mechanisms serving the same purpose.

5. Conclusion and future scope

To sum up, as majority of the computer users are adopting cloud storage for archiving their data items, providing privacy preserving cloud storage has become the need of the day. The researcher has provided the mechanism which preserves the privacy of the data owners and data items and proves the ownership of the data The probabilistic data structure helps in effectively executing the membership queries over the large collecton of data items. Cuckoo filter offers the best false positive rate among all the probabilistic data structures. In terms of space complexity and computational overhead the current mechanism is proved to be the best among all the mechanisms available. It ensures the proof of ownership for the data items in cloud storage. A thorough analysis of all security issues has been considered and a provable ownership has been achieved .Further research proceeds in making the cloud storage more secure and minimize space overhead.

References

[1] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The method and tool of cost analysis for cloud computing," in Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, 2009, pp. 93–100. https://doi.org/10.1109/CLOUD.2009.84.

[2] Xu, Jia, and Jianying Zhou. "Leakage-resilient proofs of ownership in cloud storage, revisited." International Conference on Applied Cryptography and Network Security. Springer International Publishing, 2014. https://doi.org/10.1007/978-3-319-07536-5_7.

[3] M. Bellare, S. Keelveedhi, and T. Ristenpart. Dupless: Server aided encryption for deduplicated storage. In USENIX Security Symposium, 2013

[4] 4. J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management. In IEEE Transactions on Parallel and Distributed Systems 2013.

[5] Benny Pinkas, Danny Harnik, Alexandra Shulman-Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage", IEEE Security & Privacy, vol. 8, no. , pp. 40-47, November/December 2010, https://doi.org/10.1109/MSP.2010.187.

[6] 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, pages 491–500. ACM, 2011. https://doi.org/10.1145/2046707.2046765.

[7] 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. ACM, 2012, pp. 81–82.

[8] Fan, Bin, et al. "Cuckoo filter: Practically better than bloom." Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. ACM, 2014. https://doi.org/10.1145/2674005.2674994.

[9] B. Tirapathi Reddy, U.Ramya, Dr. M. V. P. Chandra Sekhara Rao, "A comparative study on data deduplication techniques in cloud storage", IJPT| Sep-2016 | Vol. 8 | Issue No.3 | 18521-18530.

[10] Di Pietro, Roberto, and Alessandro Sorniotti. "Proof of ownership for deduplication systems: a secure, scalable, and efficient solution." Computer Communications 82 (2016): 71-82. https://doi.org/10.1016/j.comcom.2016.01.011.

[11] Xu, Jia, Ee-Chien Chang, and Jianying Zhou. "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage." In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp. 195-206. ACM, 2013.

[12] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik, "Estimation of deduplication ratios in large data sets," in IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20,2012, Asilomar Conference Grounds, Pacific Grove, CA, USA, 2012,pp. 1–11. https://doi.org/10.1109/MSST.2012.6232381.

[13] Is Convergent Encryption really secure? http://bit.ly/Uf63yH

Belazzougui, Djamal, Fabiano C. Botelho, and Martin Dietzfelbinger. ”Hash, displace, and compress.” In Algorithms-ESA 2009, pp. 682-693. Springer Berlin Heidelberg, 2009.

[14] Perttula. Attacks on convergent encryption. http://bit.ly/yQxyvl.

B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.

[15] B.Tirapathi Reddy,Dr.M.V.P.Chandra Sekhara Rao, “Performance evaluation of various data deduplication schemes in cloud storage”, IJPAM| Sep-2016 | Vol. 116 | Issue No.5 | 175-180

[16] Adya, Atul, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. ”FARSITE: Federated, available, and reliable storage for an incompletely trusted environment.” ACM SIGOPS Operating Systems Review 36, no. SI (2002): 1-14.

[17] “Bitcasa Infinite Storage,” https://www.bitcasa.com/.

[18] https://en.wikipedia.org/wiki/Dynamic_perfect_hashing.

[19] S.V.Manikanthan and V.Rama“Optimal Performance Of Key Predistribution Protocol In Wireless Sensor Networks” International Innovative Research Journal of Engineering and Technology ,ISSN NO: 2456-1983,Vol-2,Issue –Special –March 2017.

[20] T. Padmapriya and V. Saminadan, “Inter-cell Load Balancing Technique for Multi- class Traffic in MIMO - LTE - A Networks”, International Conference on Advanced Computer Science and Information Technology , Singapore, vol.3, no.8, July 2015.

[21] S Nazeer Hussain, K Hari Kishore "Computational Optimization of Placement and Routing using Genetic Algorithm” Indian Journal of Science and Technology, ISSN No: 0974-6846, Vol No.9, Issue No.47, page: 1-4, December 2016.