International Journal of Computer Applications (0975 – 8887)
Volume 115 – No. 6, April 2015

Optimizing the Algorithm for Secure and Dynamic Cloud Storage using MHT

Amit Kumar
Student,
Department of Computer Engineering,
GHRCEM, Savitribai Phule Pune University, India

S S Sambare
Associate Professor,
Department of Computer Engineering, PCCOE,
Savitribai Phule Pune University, India

ABSTRACT

With the consistently increasing demand of storing data backup online by millions of users, cloud computing - which provides resources both hardware and software as a service through a network (usually internet) has gained much momentum and attention. Storing data on cloud services relieves the user from the task of storing data locally. As beneficial as it sounds, cloud storage comes along with the quintessential need to protect the stored data from threats such as loss of confidential data and denial of service. The purpose of this research paper is provide a mechanism which uses tokens and distributed forward error correction coded data. The mechanism not only provides the assurance of a secure storage but also provides with the identification of error or the wrongful server simultaneously. Considering the user to store data dynamically, we have also felicitated the dynamic operations such as insertion, deletion and appending blocks. In comparison to previous works, our algorithm can be more flexible with the use of Merkle Hash Tree (MHT) rather than contiguous block data structure.

Keywords

Cloud storage, security of data, dynamic data, data integrity.

1. INTRODUCTION

Although cloud infrastructures are much more powerful and safe, the shift of computing platform has not been very well accepted. The main reason being that the user has no assurance of the integrity and availability of his data stored on the cloud since the copy is not retained on the local machines. This also emerges with the problem of the cloud service provider's misbehaviour. For instance to reduce the cost a service provider may remove rarely accessed data from the cloud without the permission of the client. Also, unlike a data warehouse, the client would not only like to access his data but also update them frequently. Hence, the need to incorporate operations like insert, update, delete and append becomes inevitable. In this paper, a scheme to ensure data integrity and dynamic support on cloud servers has been introduced. With ongoing research in the field of secure data storage on cloud our scheme stands out from the previous ones. Earlier schemes have only provided a binary status for error detection and mechanism for identifying the corrupted server. But our contribution to provide dynamic support to a similar secure algorithm is unique to the former mechanisms.

2. RELATED WORK

In previous works ,Juels et al. [1] defines the model known as the "proof of retrievability" (POR) model for ensuring the remote data integrity using spot-checking and error correcting code. The algorithm employs two important schemes: Error Correcting Code (ECC) and Message Authentication Code (MAC). Another algorithm was introduced by Shacham and Waters [3]. The validation is done on using authenticators which are stored on server. The client requests for these authenticators for several indexes and expects them to be valid. Similarly, Bowers-Juels-Oprea [1] and Dodis-vadhan-Wichs [4] proposed schemes based on the POR model to check integrity of data on remote servers. However, these schemes operated only on static data. It is natural for a client to modify his data stored on the cloud.

3. ARCHITECTURE

Typical network architecture for cloud storage service can have the following three network entities: User, Cloud Server (CS) and a Third Party Auditor (TPA), which has access to CS and can expose the risk involved in the cloud storage services. After reviewing some basic techniques, the token will be computed by a function which belongs to a hierarchy of universal hash function [6], which can preserve the homomorphic properties, which may be integrated along with the verification module of erasure-coded data [6]-[10]. As needed, a challenge-response mechanism for verifying the storage correctness and hence identifying any of the present misbehaving servers.

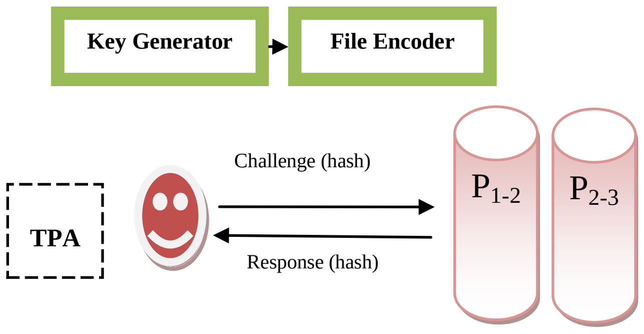

Fig 1 Cloud Storage Service Architecture [4], [5]

As plotted in the Fig 1, our aim is to ensure data integrity on cloud servers with the goals such as, Storage correctness-ensuring the users of the integrity of their data, fast error detection-effective identification of the compromised or misbehaving server, Using MHT, error detection can be eased for large sets of data. In case of ensuring data stored on cloud the first challenge is to detect any modification or deletion in the file. Second challenge is to exactly identify which server has been compromised or corrupted. MHT, originally

4. IMPLEMENTATION

The proposed algorithm can be summarized as:

1. Divide F into tokens.
2. Derive Encoded File and corresponding hash. Generate Merkle Hash Tree.

3. Pass MHT Hash Values and Nodes for challenge.
4. Derive and Compute the tokens from server using hashes.

In the current approach, during the file preparation stage, instead of storing the tokens directly, the same is stored in the signature of merkle hash tree. The tree will calculate the hashes as needed and store the tokens at the nodes. It will thus provide a virtual signature of actual data stored over the cloud. This will give the advantage of better data operations rather than working on the entire data as a whole. The structure of the merkle hash tree will appear as shown in Fig 2.

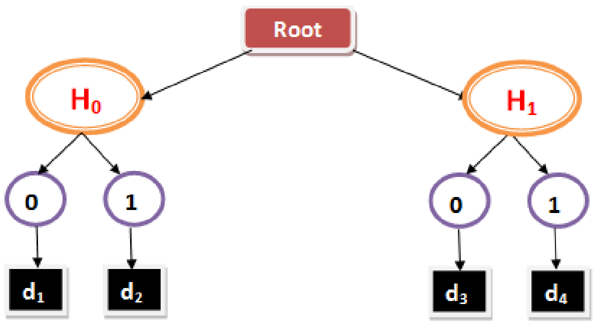

Fig 2 Client side signature token formed using merkle hash tree

At server, the file is stored w.r.t the corresponding index. In addition the hash value can be integrated where each node has a data field, a hash value for to the next node along and a sequence number. For an insert operation, the server appends the file with the required node sequence and the hashes are reconstructed.

The Hash values will be calculated for every data block till root. So, rather than token itself, MHT will be now stored by user. Then the algorithm will disperse all the byte streams $w_i^{(j)}$ across k servers. For our purpose we rely on using the Tiger Tree Hash (TTH), a common variation of the merkle hash tree. Although TTH is complex in terms of hashing algorithm, it proves to be efficient with respect to speed and security. It uses a Time Memory Trade-Off with the help of 4 small lookup-tables or S-Boxes, similar to DES or other block ciphers. The data is first padded to 512-bits. First by appending a single 1 bit, followed by 0s. Then the data is padded out to 448 (mod 512) bits. This is followed by a little-endian 64-bit representation of the length. The data is then divided into 512-bit (64-byte) blocks, each of which is processed individually. Hence in the algorithm, all the three used carry words will be affected by the corresponding value of the key after every round. In the next step, using key scheduling the input bits are rotated in accordance with the three passes. The final stage of this algorithm helps in propagating the hashed bits between the 512-bit blocks of the message.

Our scheme relies on generating homomorphic tokens, where every hash covers a random subset of data blocks. For the user to check whether his data is maintained correctly on the cloud, he challenges the cloud by providing randomly chosen hash for which the cloud computes small signatures and returns them to the user. These signatures should then match the values of the hash which have been pre-computed and stored on the machine locally with the user. Following is an overview to show the process of generating hash challenges:

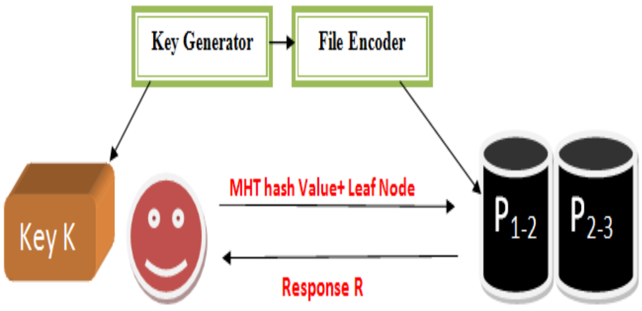

Fig 3 Challenge with a set of randomly generated block hashes

Earlier schemes have only provided us with binary status of whether an error was found or not. However, our scheme realizes the importance of error localization or identifying the misbehaving server using MHT for dynamic data operations. Then the user performs an equality check to determine if the received values remain valid to the secret hash value (R):

$$(R(1)^i, \ldots, R(m)^i) \times H ?= (R`(1)^i, \ldots, R`(m)^i).$$

If the above equation holds true then the data stored is correct otherwise the user knows that among these r rows an error has occurred. To identify which server has been compromised we rely on our pre-computed tokens. Since the response generated by server has been computed in the same manner as we computed our tokens we can use the following n equations to find the server.

$$\text{Rhash}^{(j)}{}_i ?= \text{vhash}^{(j)}{}_i, j \in \{1, \ldots, n\}.$$

If the above equation is not true then server j is known to be corrupted or compromised. In such condition the server cannot be capable of providing data to the user. Hence the server is identified and the problem has to be rectified accordingly. To recover from the error user can ask for the r rows stored on server j subsequently remove the server and then resend the recovered blocks to other servers. In-order to get support for dynamic data operations, an address field is induced with the data blocks to store the reference of corresponding hash.

The tree signature will be updated accordingly as per the transactions. This approach gives the flexibility of reporting any modifications made over the cloud server to the users. Data insertion is possible as only the values of corresponding hashes has to me modified. The changes occurred due to the insertion operation has also to be updated along the signature token tree stored by the client. So the new MHT is sent to the client. The same approach can be applied for deletion of data at block level. In that case the difference will be in the arithmetic operation where subtraction will be performed instead of addition.

5. RESULTS

The proposed approach takes into account the need of data integrity and data availability along with the support for dynamic data operations. The merkle hash tree introduced on the user side is an added advantage which provides block to block data interpretation in the server. Our proposed algorithm has considerably less overhead when there is a need for data recovery in misbehaving servers as only the affected files can be extracted specifically using the signature hash stored on the client side. Since the need of array type data structure is minimized hence the file recovery time from the misbehaving servers will gradually decrease.

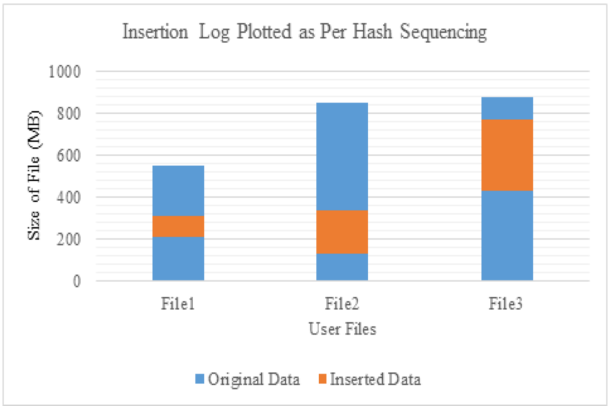

Fig 4 Insertion Operation

We implemented our proposed algorithm on multimedia files for testing insertion operation (Fig 4). Direct insertion is not feasible as this operation requires shifting of all subsequent blocks by relative position and hence massive computation to renumber.

6. CONCLUSION

The objective of our ongoing research is to provide a model in which several parameters will be assured to the client while storing data on a cloud server. Our scheme is based on calculating signature hashes before a file is distributed on multiple servers on a cloud. These tokens are used to challenge the server whenever the user wants to make sure that his data is available and maintained correctly. In addition this mechanism also gives error localization or the identity of the compromised server and thus subsequent recovery for the corrupted data. We also realize the need for dynamically updating of data by user and thus encourage the need to incorporate dynamic data operation support such as insert, delete and append. However, the size of secret key can be reduced in future implementations which will lead to less overhead and better throughput.

7. REFERENCES

[1] Kevin D. Bowers, Ari Juels, and Alina Oprea, Proofs of retrievability: theory and implementation, In Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW 2009, pages 43-54, New York, NY, USA, 2009. ACM.

[2] Muntes-Mulero V, Nin J. Privacy and anonymization for very large datasets. In: Chen P, ed. Proc of the ACM 18th Int'l Conf. on Information and Knowledge Management, CIKM 2009. New York: Association for Computing Machinery, 2009.

[3] Hovav Shacham and Brent 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, ASIACRYPT 2008, pages 90-107, Berlin, Heidelberg, 2008.Springer-Verlag.

[4] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs,Proofs of retrievability via hardness amplification, In Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC 2009, pages 109-127, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] Cong Wang, Qian Wang, Kui Ren, Ning Cao and Wenjing Lou, Towards secure and dependable storage services on Cloud Computing, IEEE Transactions on Services Computing, Vol. 5, No. 2, pp. 220-232, April-June 2012, 2012/07/12.

[6] L. Carter and M. Wegman, Universal hash functions,Journal of Computer and System Sciences, vol. 18,no. 2, pp. 143-154, 1979.

[7] T. Schwarz and E. L. Miller, Store, forget, and check: Using algebraic signatures to check remotely administered storage, in Proc. of ICDCS 2006, 2006, pp.12-12.

[8] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows,andM. Isard, A cooperative internet backup scheme, in Proc. of the 2003 USENIX Annual Technical Conference (General Track), 2003, pp. 29-41.

[9] M. Castro and B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Transaction on Computer Systems, vol. 20, no. 4, pp. 398-461, 2002.

[10] J. Hendricks, G. Ganger, and M. Reiter, Verifying distributed erasure-coded data, in Proc. of 26th ACM Symposium on Principles of Distributed Computing, 2007, pp. 139-146.

[11] Chris Erway, Alptekin, Charalampos Papamanthou,and Roberto Tamassia, Dynamic provable data possession, In Proceedings of the 16th ACM conference on Computer and communications security, CCS 2009, pages 213-222, New York, USA, 2009 ACM.