2017 13th International Conference on Computational Intelligence and Security

An Approach to Verifying Data Integrity for Cloud Storage

Yindong Chen, Liping Li, Ziran Chen
College of Engineering
Shantou University
Shantou, China
e-mail: {ydchen, 15lpli, 17zrchen1}@stu.edu.cn

***Abstract*—The developments of cloud computing technology and reliable modern network have been motivating users to store data in cloud servers and rendering security of cloud computing a focus of attention. While data security can be ensured by data encryption technology, data integrity is secured by certain checking algorithm. A verification method of data integrity which can be performed repeatedly is proposed after the comprehensive analysis of the feature of static data stored in a cloud server and its security requirements. The client preprocesses the data file according to the times that the data is required checking, and then upload the data file with check metadata to cloud server, so that the client can spot check the data within limited times and judge the data integrity. Through the analysis of security and performance, we show that this scheme can efficiently verify the data integrity and resist the replay attack and MitM (Man-in-the-middle attack).**

Keywords—cloud storage; data integrity; MAC; data-storage security.

I. INTRODUCTION

With the boom in cloud computing, cloud-based storage systems based on massive amounts of data are starting to come out. Many internet service providers such as Amazon, Google and Microsoft are introducing cloud platforms and cloud services. They start with cloud storage service, which not only bring people convenience and benefit, but also save the social resources and energy. Cloud storage provides cheap and reliable storage service. But, once the enterprise or individual store their data to the cloud storage, the user has lost the absolute control of the personal data. This means that the client will always be in their outsourcing data security concerns. On the one hand, cloud service providers may make changes to user data for their own profits. On the other hand, the inevitable failure of the server or poor management of the server can result the data integrity in compromising. When faced with these troubles, in order to prevent the cloud service providers from concealing the fact that data corruption deliberately and assuring users that their data is still intact in the cloud, we need integrity checking periodically for the data stored in the cloud.

II. RELATED WORK

The research of data integrity verification mechanism is mainly attaching importance to two aspects in accordance with the fault-tolerant pretreatment of the data. One is PDP (Provable Data Possession) [1-4], the other is POR (Proofs of Retrievability) [5-6]. The difference between them is that the PDP can efficiently verify the completeness of the cloud data. And POR can not only recognize the incompleteness of data, but also recover the corrupted data.

Deswarte et al were first who used the HMAC (Hash-based Message Authentication Code) to realize the integrity checking of remote external data [7, 11]. It was using the HMAC to preprocess the files that would be checked before the data was stored in the cloud server. Then, the challenge-response was used to verify the integrity of the data file. But this method cannot resist replay attack. Later, Deswarte considered another PDP mechanism, which was using the homomorphism characteristics of RSA (RSA algorithm) signature to implement the data file validation. But this method needed to represent the file with a huge number, the modular exponentiation of the checked data may give a high rise to computational cost and consume an army of computing resources. Subsequently, Ateniese et al first proposed using the homomorphic verification label which used a probabilistic strategy which based on the RSA to realize integrity verification [1-10]. Although the homomorphic label technology could be applied to meet the needs of unlimited times' checks for the static data. But it would bring a huge checksum burden because of the larger computation with modular exponentiation. Xu et al put forward using CBF to realize the integrity verification of cloud storage and implemented the verification scheme that based on MAC successfully [8], but it had certain misjudgment rate and required more third party stores information. In consideration of delivering the user's inspection tasks to a trusted third party, the participation of third parties also increased the cost of storage and brought other security threats such as data breaches [9].

This paper proposes an integrity check algorithm based on MAC. Under the precondition that there is no third party, the local stores less information can be accurately realize data validation for many times.

III. A SOLUTION OF DATA INTEGRITY VERIFICATION IN CLOUD STORAGE ENVIRONMENT

A. Summary of Solution

The solution of the whole system scenario is summarized as follows. First, client and server make the mutual authentication; next, client generates public and private key pair for verification, and generates check metadata according to the times that the data needs to be verified. This check metadata is the unique identification to every data block. Besides, a random sequence is generated for each data block

0-7695-6341-4/17/31.00 ©2017 IEEE
DOI 10.1109/CIS.2017.00135

by a pseudo-random function. The check metadata is added to the location of the corresponding block according to the random sequence. Then, client applies for visiting the cloud storage system. When the two sides trust each other and confirm their identity, client uploads the whole file to the storage server. In the verification phase, the client initiates a verification challenge, the server finds out the data block which is needed to be verified according to the challenge information, and finds the corresponding location of the check metadata according to the times that the data has been verified. Then server encrypts the entire data after remove this check metadata to generate authentication data. Server returns the verification check metadata and verification data to the client. Client verifies the integrity according the local key. So far the process of data integrity verification in the cloud storage has completed.

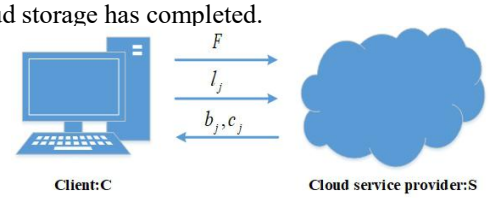

Figure 1. System model

B. Preparation Phase

At this phase, user wants to start storing the file, the file will be divided into n blocks first: $F = \{f_1, f_2, ..., f_n\}$.

TABLE I. CRYPTOGRAPHIC SYMBOLS

Symbol	*Implication*
$h_{key}(.)$	One-way hash function with key
$E_{key}(M)$	Encrypting the message M with key
$D_{key}(M)$	Decrypting the message M with key
$Sig_{key}(M)$	Signing the message M with key

For each data block $f_i, i \in [1,n]$, the client generates challenge key $b_j, j \in [1,t]$ according to the check times of the block. Client gets check metadata $c_j, j \in [1,t]$ by using the public key to encrypt those challenge keys.

The client generates a seed which is used to generate a sequence $\{a_{i1}, a_{i2}, a_{i3}.....\}$ for each data block by a pseudo-random function; next, inserts the verification tag into the specified location by the sequence according to the order of validation. We take the verification process of the first data block as an example. If the block can be verified three times, the structure of the data will be uploaded as follows:

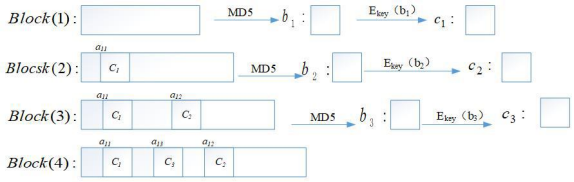

Figure 2. Structure of the data

C. Storage Phase

Granted that client and server have their own authentication key pairs and can obtain the public key of the other party. The client's key pair is (PK_C, SK_C) . The server's key pair is (PK_S, SK_S) . The methods of storage phase are as follows:

1) $C \to S: E_{PK_C}(r_c, S), Sig_C(h(r_c, S))$

2) $S \to C: E_{PK_C}(r_c, C, r_S), Sig_S(h(r_c, C, r_S))$

3) $C \to S: E_{PK_S}(S, r_S), Sig_C(h(S, r_S))$

4)

$C: F = \{f_1, f_2, \cdots, f_n\}$

$l_j = E_{sk_C}(i \| j), i \in [1,n], j \in [1,t]$

$A = \{A_1, A_2, \cdots, A_n\}$

$A_1 \Rightarrow \{a_{11}, a_{12}, \cdots, a_{1t}\},$

$A_2 \Rightarrow \{a_{21}, a_{22}, \cdots, a_{2t}\}, \cdots, An \Rightarrow \{a_{n1}, a_{n2}, \cdots, a_{nt}\}$

5) $C \to S$: $E_{PK_C}(Block(i))$

Description:

1) C selects r_C randomly, then encrypts the identifiers of r_C and S, and signs their hash values.

2) S verifies the hash value of the digital signature. If passed, S selects r_S randomly, encrypts the identifiers of r_S , r_C and C, and makes a signature of their hash values.

3) C verifies the signature and hash value to check whether the random number r_C is sent by itself. If passed, C encrypts the S's identifier and r_S , and signs their hash values. S verifies the information sent by C and authenticates that the connection is secure.

4) Client divides the file into n blocks, and generates challenge keys $l_j = E_{sk_c}(i \| j), i \in [1,n], j \in [1,t]$ according to the number of verification times t that is needed of each data block. First, a pseudo-random function is used to generate n random numbers which are depended on the system clock as a seed. And then, those n random numbers are as seeds also for each data block respectively to generate t random numbers. The client generates t validation metadata for each data block, and inserts the validation metadata at the specified location of the random numbers, the locations are found in the file f_i based on the size of the check metadata information.

5) Finally, the data blocks that are containing authentication metadata are encrypted and stored in the S . Block(i) contains data block numbers and data.

D. Verification Phase

In the verification phase, the client sends a challenge message to the server, and the server starts dealing with challenge according to the challenge information. The methods of verification phase are as follows:

1) $C \to S : E_{PK_C}(r_c, S), Sig_C(h(r_c, S))$

2) $S \to C : E_{PK_C}(r_c, C, r_S), Sig_S(h(r_c, C, r_S))$

3) $C \to S : E_{PK_S}(S, r_S), Sig_C(h(S, r_S))$

4) $C \to S : E_{PK_C}(l_j), Sig_C(h(l_j))$

5) $S \to C : E_{PK_S}(b_j, c_j), Sig_S(h(b_j, c_j))$

6) C: if$((E_{key}(b_j) == c_j))$

return: Success;

if$(E_{key}(b_j) != c_j)$

return: Failure;

Description:

1) C selects r_C randomly, and encrypts the identifiers of r_C and S, and signs their hash values.

2) S verifies the hash value of the digital signature. If passed, it selects r_S randomly, encrypts the identifiers of r_S, r_C and C, and makes a signature of their hash values.

3) C verifies the signature and hash value to check whether the random number r_C is sent by itself. If passed, C encrypts the S's identifier and r_S, and signs their hash values. S verifies the information sent by C and authenticates that the connection is secure.

4) C selects the block of data that needs to validate, gets the challenge key l_j, and makes a signature of its hash value. At the same time, according to the seed corresponding to the block data, the *realNum(a_{i1})* function is called to generate the true location of the corresponding checksum in the data. If the real location of the a_{i1} is to be found, the function is described as follows:

```
realNum (ai1){
   for (j=0; j<t; j++){
      if (ai1 > aij)
      ai1 ++;
   }
}
```

5) S verifies the challenge information. After validating, the corresponding chunk's ID i is found in the challenge key. Besides, S finds the authentication information c_j that is required for this validation according to the challenge key j, and encrypts all the block data other than the validation information then obtains the b_j. At last, S encrypts b_j, c_j and sign their hash values.

6) C verifies the return information. After passing the verification, C calculates $E_{key}(b_j)$. If $(E_{key}(b_j) == c_j)$, C indicates that validation is passed and data is not destroyed. Conversely, if $(E_{key}(b_j) != c_j)$, it proves that the data has been changed.

IV. Security and Performance Analysis

A. Replay Attack

The first 3 steps of the validation phase are used to authenticate that are countering attacks by malicious third party. And the latter 3 steps verify the integrity of the data. It assumes that the communication environment is unsafe, the public key of each role is public, the private key is known only by himself, and the random number is different each time. The analysis is as follows:

1) If the malicious third party intercepted the message $E_{PK_C}(r_c, S), Sig_C(h(r_c, S))$ that C sent to S.

2) And the messages $E_{PK_C}(r_c, S)$ and $Sig_C(h(r_c, S))$ were intercepted by malicious third party that C sent to S. And S found that the messages had been received, so, the messages are judged to be replayed by the malicious third party. In the same way, C can determine whether the messages are sent by S.

3) The authentication information which is based on the reply value sent back from S is decrypt by C with the private key and is compared with the checked data. If it is inconsistent, the data had been changed.

B. MitM Attack

Man-in-the-middle attack (MitM) is an indirect attack, which means that malicious users cut into C and S through various means to launch an attack to tamper with information and steal information. In the design of this scheme, the transmission of each information has the step of signing the hash value of the message. The signature of the message is the authentication of the message sender. Thus this way prevents the illegal access of the malicious user and resists the Man-in-the-middle attack.

C. Algorithm Security Analysis

Each piece of data in the scheme can be verified several times, and the verification data of each verification is guaranteed to be the result of the calculation of the whole data. Thus, it completely avoids the data cheating behavior of the server.

Taking into account the data in the cloud storage space is not long-term static, some data migrates the data storage space because of the update or the user selects the new cloud service provider, lest the data exhibits the dynamic nature of the data. Therefore, in a certain storage period, as long as the number of times required to meet the test, client can safely assume that the program is feasible.

D. Storage Cost Analysis

Messages that C needs to be stored are identifier, public key of the S, the public and private key for authentication, and the check identifier for each piece of data. The traditional integrity check protocol stores the data's check information in the client's locality. This scheme embeds the check information into the data and stores data in the cloud server. From this point of view, the scheme greatly reduces the customer's storage costs. The performance indicators as

shown in Table Ⅱ by comparing this algorithm with other representative algorithms.

TABLE II. COMPARISONS BETWEEN THE ALGORITHMS' PERFORMANCE

	S-PDP[2]	Ateniese[1]	Xu[8]	This algorithm
The sizes of metadata	$O(n)$	$O(n)$	$O(n)$	$O(n)$
The costs of communication	$O(1)$	$O(1)$	$O(1)$	$O(1)$
The costs of server calculation	$O(c)$	$O(n)$	$O(c)$	$O(n)$
The costs of user calculation	$O(n)$	$O(n)$	$O(n)$	$O(1)$

At the same time, the user can still completely control the integrity of the data without controlling the data. Moreover, the scheme has no third-party required, and the whole verification process is simple and efficient.

V. CONCLUSION

Integrity validation is necessary to ensure that the user's data is properly stored in the cloud server. A MAC based integrity checking scheme is proposed in this paper. There is no third party under the precondition. The local can realize data validation for many times accurately only stores less information. When the user has verification demand, just needs to send the verification tag which include block number and the count of the block has checked to the cloud server. After that, you can know whether your data is integrate accurately.

Through analyzing the security and performance, the program can check integrity effectively and resists replay attacks and Man-in-the-middle attacks. Nevertheless, in this solution, once the data block is verified, unwanted message is generated, these messages not only waste storage space, but also give users the trouble of cleaning data. So the future work will be focused on how to dismiss the redundant data to propose a program which will have higher performance in all aspects of the solution.

ACKNOWLEDGMENT

This paper is supported by the National Natural Science Foundation of China (No. 61103244, 61702318), the Guangdong Natural Science Foundation (No. 2015A030313433), the Characteristic Innovation project in Higher Education of Guangdong (No. 2015KTSCX036, 2016KQNCX056), the Science and Technology Planning Project of Guangdong Province (No. 2016B010124012, 2016B090920095), the Engineering and Technology Research Center of Guangdong Higher Education Institutes (No. GCZX-A1306).

REFERENCES

[1] Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores[C]//CCS07: Proceedings of the 14th ACM Conference on Computer and Communications Security. New York: ACM Press, 2007: 598-609.

[2] Ateniese G, Pietro R D, Mancini L V, et al. Scalable and efficient provable data possession [C]//SecureComm08: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. New York: ACM Press, 2008: 1-10.

[3] Ateniese G, Kamara S, Katz J. Proofs of storage from homomorphic identification protocols [C]//ASIACRYPT'09: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology. Berlin: Springer-Verlag, 2009: 319-333.

[4] Curtmola R, Khan O, Burns R, et al. MR-PDP: Multiplereplica provable data possession [C]//The 28th International Conference on Distributed Computing Systems. Piscataway: IEEE, 2008: 411-420.

[5] Shacham H, Waters B. Compact proofs of retrievability [C]. LNCS 5350: Proc of ASIACRYPT ' 08, Berlin: Springer, 2008: 90-107.

[6] Wang Q, Wang C, Renk, et al. Enabling public auditability and data dynamics for storage security in cloud computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(5): 847-859.

[7] Deswarte Y, Quisyuater J J, Saïdane A. Remote integrity checking [C]//Proceedings of the IFIP TC11/WG11.5 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS). Lausanne, Switzerland, 2004: 1-11.

[8] XU Guang-wei, SUN Zhi-feng, CHEN Chun-lin1, et al. Verification of Dynamic Data Integrity Using Counting Bloom Filter in Cloud Storage[J]. Journal of Chinese Computer Systems, 2014, 35(10): 2778-2283. (in Chinese).

[9] CAO Xi, XU Li, CHEN Lanxiang. Data integrity verification protocol in cloud storage system[J]. Journal of Computer Applications, 2012(01): 8-12. (in Chinese).

[10] TAN Shuang, JIA Yan, HAN Wei-Hong. Research and Development of Provable Data Integrity in Cloud Storage[J]. Chinese Journal of Computers, 2015(01): 164-177. (in Chinese).