Scientific Research Publishing

Int. J. Communications, Network and System Sciences, 2017, 10, 274-282
http://www.scirp.org/journal/ijcns
ISSN Online: 1913-3723
ISSN Print: 1913-3715

Data Integrity Checking Protocol with Data Dynamics in Cloud Computing

Junjie Feng[1], Shigong Long[2*]

[1]College of Science, Guizhou University, Guiyang, China
[2]College of Computer Science and Technology, Guizhou University, Guiyang, China
Email: junjie91124@sina.cn, *ie.sglong@gzu.edu.cn

How to cite this paper: Feng, J.J. and Long, S.G. (2017) Data Integrity Checking Protocol with Data Dynamics in Cloud Computing. *Int. J. Communications, Network and System Sciences*, **10**, 274-282.
https://doi.org/10.4236/ijcns.2017.105B027

Received: May 10, 2017
Accepted: May 23, 2017
Published: May 26, 2017

Abstract

We introduce a model for provable data possession (PDP) which allows a client that has stored data at an un-trusted server to verify that the server possesses the original data without retrieving it. In a previous work, Ateniese *et al.* proposed a remote data integrity checking protocol that supports data partial dynamics. In this paper, we present a new remote data possession checking protocol which allows an unlimited number of file integrity verifications and efficiently supports dynamic operations, such as data modification, deletion, insertion and append. The proposed protocol supports public verifiability. In addition, the proposed protocol does not leak any private information to third-party verifiers. Through a specific analysis, we show the correctness and security of the protocol. After that, we demonstrate the proposed protocol has a good performance.

Keywords

Provable Data Possession (PDP), Cloud Storage, Data Dynamics, Public Verifiability, Data Integrity

1. Introduction

Recently, many works focus on providing remote data integrity checking protocols. Because storing data in the cloud has become a trend and an increasing number of clients store their important data in remote servers in the cloud, without leaving a copy in their local computers. Sometimes the data stored in the cloud is so important that the clients must ensure it is not lost or corrupted. Using a remote data integrity checking protocol, the client might be able to periodically verify that whether the data stored on the server side is complete. Any corruption will be noticed by the data owner who will be able to take immediate action.

Remote data integrity checking protocols have been proposed in the last few years and two recent results [1] Provable Data Possession (PDP) [2]-[14] and Proofs of Retrievability (POR) [15] have highlighted the importance of the problem and suggested two very different approaches. Ateniese *et al.* [2] first defined the notion of PDP, which allows a client to verify the integrity of its data stored at an un-trusted server without retrieving the entire file. Their scheme is designed for static data and used public key-based homomorphic tags for auditing the data file. Nevertheless, the pre-computation of the tags imposes heavy computation overhead that can be expensive for entire file. Subsequently, Ateniese *et al.* [3] constructed scalable and efficient schemes using symmetric keys in order to improve the efficiency of verification. This results in lower overhead than their previous scheme. The scheme partially supports dynamic data operations; however, it is not publicly verifiable and is limited in number of verification requests. Thereafter, several works were done following the models given in [2] [3]. Wang *et al.* [4] combined a BLS-based homomorphic authenticator with a Merkle hash tree to achieve a public auditing protocol with fully dynamic data. Recently, Mao Jian *et al.* [5] proposed the detection data of dynamic cloud using Merkle tree, the results show that this method is very effective. Erway *et al.* [6] proposed a fully dynamic PDP scheme based on rank-based authenticated dictionary. Unfortunately, their system is very inefficient. Hao Zhuo's protocols [7] support data dynamics and public verifiability, but the calculation of this protocol and the number of blocks have a direct relationship, the amount of computation is still relatively large. After that Wei Xu *et al.* [8] proposed a remote storage integrity checking protocol based on homomorphic hash function, but the disadvantage of this protocol is losing feature of public verifiability. Zhu *et al.* [9] created a dynamic audit service based on fragment structure, random sampling and index-hash table that supports timely anomaly detection. Recently, Chunming Tang *et al.* [10] propose a new publicly verifiable method based on linearly homomorphic cryptography.

As a summary of the outstanding protocols: a protocol which satisfies the following requirements ought to be called a perfect protocol: [11] [12]

1) The amount of communication, Storage and computation required by the protocol should be low.

2) It ought to be possible to run the verification an unlimited number of times.

3) It supports dynamic operations include modification, deletion, insertion and append.

4) It supports privacy protection.

5) It supports public verifiability.

6) The changed data can be recovered.

7) It is suitable for large data integrity detection.

Current protocols can not meet all the above requirements. In this paper, the proposed protocol satisfies the above mentioned (1, 2, 3, 4, 5, 7). We have the following main contributions:

1) We propose a remote data integrity checking protocol for cloud storage, and the proposed protocol inherits the support of data dynamics from [3].

2) We give a security analysis of the proposed protocol which shows that it is secure against the un-trusted server

3) We have theoretically examined the performance and the results demonstrate that our protocol is efficient.

The rest of this paper is organized as follows: The new data possession checking protocol is described in Section2. Security analysis of the proposed protocol is presented in Section3. In section 4 we describe the support of data dynamics of the proposed protocol. Analysis of the proposed protocol is presented in Section5. Section 6 is a conclusion.

2. The New Data Integrity Checking Protocol

We consider a cloud storage system in which there is a client and an un-trusted server. The client stores her data in the server without keeping a local copy. Hence, it is of critical importance that the client should be able to verify the integrity of the data stored in the remote un-trusted server. If the server modifies any part of the client's data, the client should be able to detect it; furthermore, any third-party verifier should also be able to detect it. In case a third-party verifier verifies the integrity of the client's data, the data should be kept private against the third-party verifier.

In this scheme, we add the agreement stage between the client and the server, thus the use of probability select detection data reduces the computation both of the server and the verifier. The proposed protocol consists of five phases: *Setup*, *Giggen*, *Agreement*, *Challenge and Verification.*

Notice:

$H(\cdot), h(\cdot)$—cryptographic hash function. In practice, we use standard hash functions.

$f(\cdot)$—pseudo-random function.

P_X—the probability that can ensure the detection of the changed data blocks.

t—assume that the number of data blocks that have been changed.

N_a—random number.

Setup: Firstly, given the security parameter *k*, client run the key generation algorithm and returns a pair of matching public key *pk* and the secret key *sk*. *pk* is public to everyone, while *sk* is kept secret by the client. Then, the client selects random number generation function $f(\cdot)$ and a authentication index *r*. The *m* denote the file that will be stored in the un-trusted server, which is divided into *n* blocks of equal lengths: $m = \{m_1, m_2, \ldots, m_n\}$

SigGen: The client computes the verification tag for each block:

$V_i = H(h(m_i), r)$, $1 \le i \le n$, then using the secret key sk to encrypt the tag $\sigma_i = (V_i)_{sk}$, $\phi = \{\sigma_i\}$, $1 \le i \le n$ represents a collection of all tags and sends $\{m, (i, \sigma_i)\}$ to the server.

Agreement: The client sends the probability information of the data blocks to be detected to the verifier: $\{V, n, P_X, t, N_a\}_{C^{-1}}$. The verifier receives the infor-

mation and then uses the public key of the client to decrypt information and calculate to be detected for a data block c after decryption, then sends $\{C, n, P_X, t, N_a+1, c\}_{V^{-1}}$ to the client. The client received the message and decried it, so he can ensure that the verifier had received the message from him.

Challenge: The verifier selects c data blocks to be detected, and at the same time, the random challenge nonce $C_i = f(i), 1 \le i \le n$ is calculated for each data block. Then the verifier sends $chal = \{c, C_i\}$ to the server.

Verification: Having received the message from verifier, server computes $h(m_j), 1 \le j \le c$ and

$$K = C_1 h(m_1) + C_2 h(m_2) + \ldots + C_c h(m_c)$$

The server then retrieves σ_i and returns $\{K, \sigma_i (1 \le i \le c)\}$ to verifier who, in turn, decrypt σ_i where $V_i = (\sigma_i)_{pk}$, then compute $R = C_1 V_1 \oplus C_2 V_2 \oplus \ldots \oplus C_c V_c$ and $R' = H(K, r)$. The verifier checks whether $R = R'$. If the check succeeds, the function outputs "success", otherwise the function outputs "failure".

The following is the protocol flow **Chart 1**.

3. Security Analysis

Security means that the protocol is secure against the un-trusted server and is private against third-party verifiers. Client should not be able to pass verification unless it has access to complete unaltered version of m. Firstly, we think the remote server is not trusted; he can intentionally or unintentionally change the client's data.

Lemma 1. This probability is negligible for a secure hash function, the attacker X is able to successfully find m' which with known file m has a function of the same value and is different from m:

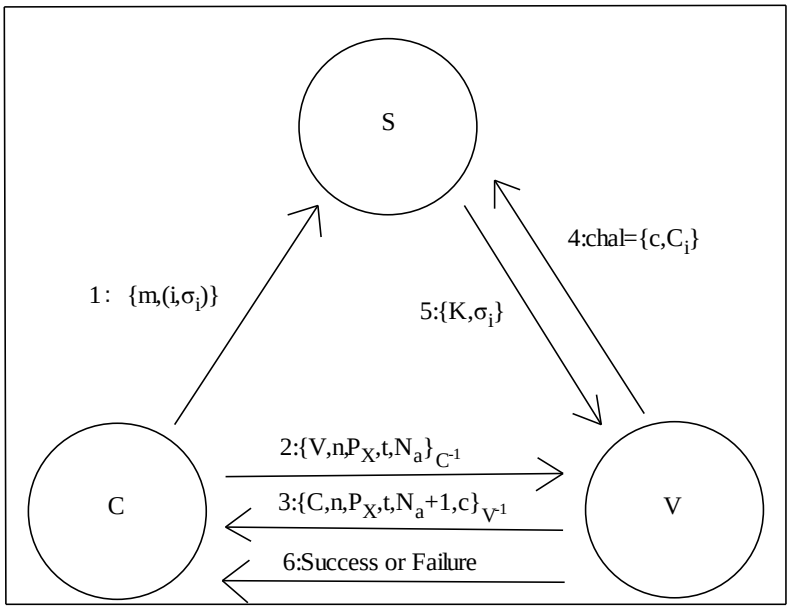


Chart 1. Protocol flow chart.

$$prob\left[(y) \leftarrow X(\mathrm{x}, \mathrm{h}(x)) : \mathrm{y} \neq \mathrm{x}, \mathrm{h}(y) = h(x)\right] < \varepsilon$$

Theorem 1. The proposed protocol uses secure hash functions $H()$, $h()$ and secure private key encryption scheme, according to the *lemma* 1, it is known that the attacker X can tamper with the data and the probability of success is negligible.

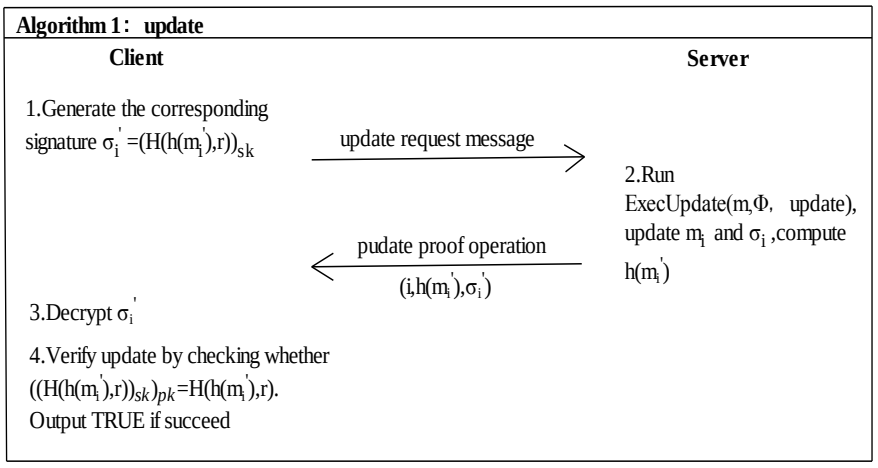
Proof. Assume that the un-trusted server has taken the m_i tampering with $m_i^{'}$ in the query results, then he computes $h(m_i^{'})$, $\because m_i^{'} \neq m_i$, according to the above lemma $prob\left[h(m_i^{'}) = h(m_i)\right] < \varepsilon$. So it is the same as $prob\left[R^{'} = R\right] < \varepsilon$ by the nature of the hash function. Therefore, the attacker can tamper with the data and the probability of success is negligible.

It is important to note that in the verification phase of the verifier decrypts σ_i, the probability of obtaining the message m_i is negligible, thus this protocol support third-party security verification

Then for the security of the agreement phase, the message is sent through the private key signature and the probability of the adversary's forgery signature is very small. Furthermore the message does not involve secret information. Agreement stage is only to ensure that the certification is received by the main body of the test will, so as to carry out data integrity testing.

4. Data Dynamics

Now we show how our scheme explicitly and efficiently handle fully dynamic data operations including data modification (*M*), data insertion (*I*), data deletion (*D*) and data append (*A*) for cloud data storage. Note that in the following descriptions for the protocol design of dynamic operation, we assume that the file m and the signature ϕ have already been generated and properly stored at server. The update operation is illustrated in Algorithm 1.



4.1. Data Modification

We start from data modification, which is one of the most frequently used operations in cloud data storage. A basic data modification operation refers to the replacement of specified blocks with new ones.

Suppose the client wants to modify the i-th block m_i to m_i', first the client generates the corresponding signature $\sigma_i' = \left(H\left(h\left(m_i'\right), r\right)\right)_{sk}$ .Step completely in accordance with the above update algorithm, the update request message here is "$update = \left(M, i, m_i', \sigma_i'\right)$" and sends it to the server, where M denotes the modification operation.

Upon receiving the request, the server runs

$ExecUpdate(\mathrm{m}, \phi, \mathrm{update})$. Specifically, the server replaces the block m_i with m_i' and σ_i with σ_i', then computes $h\left(m_i'\right)$. Finally, the server responses the client with a proof for this operation $\left(i, h\left(m_i'\right), \sigma_i'\right)$. After receiving the proof for modification operation from server, the client first decrypts σ_i', then checks whether

$\left(\left(H\left(h\left(m_i'\right), r\right)\right)_{sk}\right)_{pk} = H\left(h\left(m_i'\right), r\right)$. If it is not true, output FALSE, otherwise output TRUE.

4.2. Data Insertion

Data is inserted into the existing block in the operation are exactly the same with data modification. For the insertion of a single data block, this paper considers that the data files in a data block can be assigned to one or more existing data blocks and also can perform data modification operations.

4.3. Data Deletion

Data deletion is just the opposite operation of data insertion. Suppose the server receives the update request for deleting block m_i, it will replace the block m_i with DBlock, which DBlock is a fixed special block that represents deleted blocks. The details of the protocol procedures are similar to that of data modification and insertion, which are thus omitted here.

4.4. Data Append

Compared to data modification, data append does not change steps. The difference of additional data and data update is that you need to add new data blocks we call it m_j where $j = n + k, k \in N^*$, so the corresponding update request message is "$update = \left(\mathrm{A}, \mathrm{j}, m_j, \sigma_j\right)$", where A denotes the modification operation. Then the verification process behind the validation process and the data modification operation is exactly the same, so here it is no longer duplicated.

5. Analysis of the Proposed Protocol

Our scheme is based entirely on un-symmetric key cryptography. In this section we list the features of our proposed scheme and make a comparison of our scheme and state-of-the-art. It is worth noting that we denote pseudo random number generation and addition in the corresponding fields as *prng and add*.

5.1. Computational Cost

Server side. During verification, the server computes c hash integers $h\left(m_i\right), 1 \le i \le c$. Then, it computes the value

$K = C_1 h(m_1) + C_2 h(m_2) + \ldots + C_c h(m_c)$. The computation of each $C_i h(m_i)$ corresponds to the product of two integers being t and h bits long. So we obtain the upper bound on the server's computation time:

$$|c| time_{hash}(l) + |tc| time_{add}(th)$$

Verifier side. Except for additional pseudorandom num-ber generations corresponding to the challenge, the cost analysis of computing R is similar to that on the client side. Therefore, the verifier computation time is upper bounded by

$$|c| time_{prng}(t) + |tc| time_{add}(th) + time_{hash}(th, r)$$

The computation cost for server and verifier, though slightly higher, is still very reasonable. This does not include the time for the verifier of the data blocks to be detected, since the computation of the c requires only a very small mathematical calculation.

5.2. Storage Cost

Notice that each token is the output of a cryptographic hash function, so its size is small.

Server side. In line with the purpose of the protocol, the server has to store the complete data m and (i, σ_i), whose bitlength is $|m| + \lceil |m|/l \rceil |N|$ bits

Verifier side. The storage requirements for the verifier are only a *pk*, which was used in the verification phase.

5.3. Communication Cost

The communication cost consists of the challenge sent by the verifier to the server, with constant bitlength $2|c|$ bits, and the response sent by the server to the verifier, with constant bitlength $\lceil |m|/l \rceil |c|$ bits. It is critical to notice that the communication consumption which is ignored here in the agreement phase is very small.

5.4. Comparison with Selected Previous Protocols

We now compare our scheme with some existing schemes in terms of communication cost, computation cost on ser- ver side, computation cost on verifier side and storage cost on verifier side. For simplicity, here we denote them as Comm.cost, Comp.cost(S), Comp.cost(V), Storage.cost(V), respectively. Data dynamics and public verifiability are also listed in the table. **Table 1** indicates that the proposed protocol is really minimal overhead. You should note that n denotes all the blocks.

6. Conclusion

In this paper, we propose a new remote data integrity checking protocol for cloud storage. We extend the PDP model [3] by changing the generation method of the signature. The proposed protocol supports data level dynamics and public verifiability. Meanwhile, it is desirable to minimize the file block accesses, the

Table 1. Comparisons between the proposed protocol and previous protocols.

SchemeMetric	[6]	[4]	[7]	[3]	[10]	Our scheme
Data dynamics	Yes	Yes	Yes	Yes	Yes	Yes
Public verifiability	No	Yes	Yes	No	Yes	Yes
Comp.cost(*S*)	O(logn)	O(logn)	O(n)	O(1)	O(logn)	O(1)
Comp.cost(*V*)	O(logn)	O(logn)	O(n)	O(1)	O(logn)	O(1)
Comm.cost	O(logn)	O(logn)	O(1)	O(1)	O(logn)	O(1)
Storage cost(*V*)	O(1)	O(1)	O(n)	O(1)	O(1)	O(1)

computation on the server and the client-server communication. We expect that the salient features of our scheme make it attractive for realistic applications.

Acknowledgements

This work was supported by the Natural Science Foundation of China (61163001).

References

[1] Tan, S., Jia, Y. and Han, W.H. (2015) Research and Development of Provable Data Integrity in Cloud Storage. *Chinese Journal of Computers*, **38**, 164-177.

[2] Ateniese, G., Bruns, R. and Curtmola, R. (2007) Provable Data Possession at Untrusted Stores. *Proceedings of the* 14*th ACM Conference on Computer and Communications Security*, Alexandra, 598-609. https://doi.org/10.1145/1315245.1315318

[3] Ateniese, G., Pietro, R.D., Mancini, L. and Tsudik, G. (2008) Scalable and Efficient Provable Data Possession. *Proceedings of the* 4*th International Conference on Security and Privacy in Communication Networks*, Istanbul, 1-10. https://doi.org/10.1145/1460877.1460889

[4] Wang, Q., Wang, C., Li, J. and Lou, W.J. (2009) Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing. *Proceedings of the* 4*th European Symposium on Research in Computer Security*, Saint Malo, 355-370. https://doi.org/10.1007/978-3-642-04444-1_22

[5] Mao, J., Zhang, Y., Li, P., Wu, Q.H. and Liu, J.W. (2015) A Position-Aware Merkle Tree for Dynamic Cloud Data Integrity Verification. *Soft Computing*. https://doi.org/10.1007/s00500-015-1918-8

[6] Erway, C., Kupcu, A., Papamathou, C. and Tamassia, R. (2009) Dynamic Provable Data Possession. *Proceedings of the* 16*th ACM Conference on Computer and Communications Security*, Chicago, 213-222. http://doi.org/10.1145/1653662.1653688

[7] Hao, Z., Zhong, S. and Yu, N.H. (2011) A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability. *IEEE Transactions on Knowledge and Data Engineering*, **23**, 1432-1437. https://doi.org/10.1145/1653662.1653688

[8] Xu, W., Feng, D. and Liu, J.N. (2012) Remote Data Integrity Checking Protocols from Homomorphic Hash Functions. *Proceedings of* 2012 *IEEE* 14*th International Conference on Communication Technology*, Chengdu, 604-608 https://doi.org/10.1109/TKDE.2011.62

[9] Zhu, Y., Ahn, G.-J., Hu, H.X., Yau, S.S., An, H.G. and Hu, C.-J. (2013) Dynamic Audit Services for Outsourced Storages in Clouds. *IEEE TSC*, **6**, 227-238.

https://doi.org/10.1109/ICCT.2012.6511277

[10] Tang, C.-M. and Zhang, X.J. (2015) A New Publicly Verifiable Data Possession on Remote Storage. *Journal of Supercomputing*, 1-15. https://doi.org/10.1007/s11227-015-1556-z

[11] Hu, D.M. and Yu, X. (2014) Dynamic Cloud Storage Data Integrity Verifying Method Based on Homomorphic Tags. *Application Research of Computers*, **31**, 1362-1395. https://doi.org/10.1007/s11227-015-1556-z

[12] Hu, D.M. and Yu, X. (2014) Dynamic Data Integrity Detection Method in Cloud Storage Service. *Application Research of Computers*, **31**, 3056-3060.

[13] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Khan, O., Kissner, L., Peterson, Z. and Song, D. (2011) Remote Data Checking Using Provable Data Possession. *ACM Transactions on Information and System Security*, **14**, 1-34. https://doi.org/10.1145/1952982.1952994

[14] Li, A.P., Tan, S. and Jia, Y. (2016) A Method for Achieving Provable Data Integrity in Cloud Computing. *Journal of Supercomputing*, 1-17. https://doi.org/10.1145/1952982.1952994

[15] Juels, A. and Kaliski, B.S. (2007) PORs: Proofs of Retrievability for Large Files. *Proceedings of the* 14*th ACM Conference on Computer and Communications Security*, Whistler, 584-597. https://doi.org/10.1145/1315245.1315317