MB-DDIVR: A Map-Based Dynamic Data Integrity Verification and Recovery Scheme in Cloud Storage

Zizhou Sun[2], Yahui Yang[1](✉), Qingni Shen[1], Zhonghai Wu[1], and Xiaochen Li[1,2]

[1] School of Software and Microelectronics, Peking University, Beijing, China
{yhyang, qingnishen, wuzhu}@ss.pku.edu.cn
[2] MoE Key Lab of Network and Software Assurance, Peking University, Beijing, China
sunzz679@pku.edu.cn

Abstract. Outsourcing data to are remote cloud service provider allows organizations or individual users to store more data on the cloud storage than on private computer systems. However, a specific problem encountered in cloud storage is how to ensure user's confidence of the integrity of their outsourced data on cloud. One important approach is Proof of Retrievability (POR) which allows a verifier to check and repair the data stored in the cloud server. However, most of existing PORs can only deal with static data and provide one single recovery method which may lead to inefficiency and inflexibility. To address these cloud storage issues, we propose a map-based dynamic data integrity verification and recovery scheme in cloud storages. We first present two recovery methods with different granularity and introduce a new data structure. Relying on algebraic signature with homomorphism property, our integrity verification is highly efficient. Furthermore, our solution can prevent multiple cloud servers from colluding to fabricate consistent signatures.

Keywords: Cloud storage · Data integrity · Algebraic signature · Dynamic operation · Erasure code

1 Introduction

Since data is increasing exponentially, data owners rapidly increase their demand for cloud storage. Abilities like scalability, reliability, flexibility and security make cloud storage essentially a technology for future. Cloud service providers (CSP) offer users clean and simple distributed file-system interfaces, abstracting away the complexities of direct hardware management.

From the data security's point of view, which is always an concerned aspect of quality of service, however, cloud storage is confronted with new challenging security threats. Firstly, CSP may not be trustworthy, data owners lose direct control over their sensitive data. This problem brings data confidentiality and integrity protection issues. Secondly, cloud storage does not just store static data. The data stored in the cloud may be frequently modified and updated by users. CSP should have ability to demonstrate users' data is correctly update.

So data integrity and availability are important components of cloud storage, it is imperative for users to verify the integrity of their data and could recover the corrupted data anytime. One solution is the so-called Proof of Retrievability (POR), which was first introduced by Juels and Kaliski [1] and its subsequent versions are [2, 3]. PORs enable the server to demonstrate to the verifier whether the data stored in the servers is intact and available, and enables the clients to recover the data when an error is detected. Atenies et al. [4] introduced a Provable Data Possession (PDP) scheme, in which the user generates some information for a file to be used later for verification purpose through a challenge-response protocol with the cloud server. Based on this scheme, there are different variations of PDP schemes [5–8].

Main Contribution. In this paper, we propose an efficient and secure public auditing and recovering scheme, which also supports the dynamic data operations. Our contributions can be summarized as follows:

(1) To the best of our knowledge, our scheme is the first to support two data recovery methods at different levels of granularity.
(2) We design a data structure called map-based dynamic storage table, which provides better support for dynamic data operation and two different levels of recovery methods.
(3) We propose a solution to protect the security of algebraic signature verification process with little storage overhead against cloud servers colluded to offer fake data signatures and created parity for these signatures.

2 Scheme Overview

2.1 System Model

The system model of the MB-DDIVR scheme is depicted in Fig. 1:

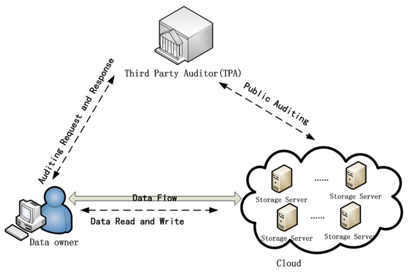


Fig. 1. System model of MB-DDIVR

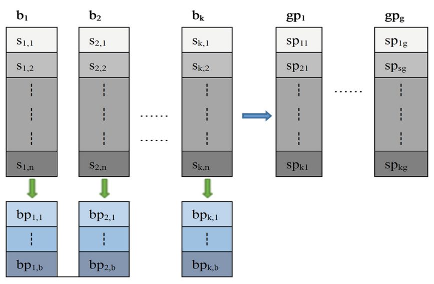


Fig. 2. One group encoding example

User: These entities have data to be stored in the cloud.

Cloud service provider (CSP): The CSP has major resources and expertise in building and managing distributed cloud storage service. The servers are managed and monitored by a CSP.

Third Party Auditor (TPA): This entity is delegated the responsibility to check the servers on behalf of the clients. The TPA is assumed to be trusted

2.2 Thread Model

In this work, we assume the TPA to be curious-but-honest. It performs honestly during the whole auditing procedure but it is curious about the received data. The CSP is not trusted, even though the file data is partially or totally losing, the CSP may try to deceive the user that he holds the correct file. All of storage servers may join up to implement collusion attack, i.e., they may forge the response of integrity challenge to deceive the user and TPA.

2.3 Design Goals

In this paper, we analyze the problem of ensuring the security and dependability for cloud storage and we aim to design efficient mechanisms for data verification, recovery and dynamic operation. We expect to achieve the following goals:

(1) Fast fault localization of data: to effectively and efficiently locate the specific error block when data corruption has been detected.
(2) Fast reparation of data: to use different recovery methods to ensure quick recovery of the data when errors are detected.
(3) Security: to enhance data availability against malicious data modification and server collusion attack.
(4) Support of dynamic data operation: to maintain the same level of storage correctness assurance even if users modify, delete or append their data files in the cloud.

3 Key Solutions on MB-DDIVR

This section presents the principium of our scheme. We start from explaining what two different granularity of recovery methods are, and then present the structure of our map-based dynamic storage table, at last we will show the security of our scheme.

3.1 Recovery Method

To make our recovery method easier to follow, we assume that the user wants to store a file **F** on the cloud server *S* which is a finite ordered collection of m blocks: **F** = $(b_1, b_2, \ldots, b_m)$ and every b_i is partitioned into n sections (section is the smallest unit of storage) denoted as $b_i = (s_{i1}, s_{i2}, \ldots, s_{in})_{(i \in \{1,\ldots,n\})}$.

- *Two Levels of Recovery Methods:* We firstly use a dynamic reliability group as an example to illustrate our encoding strategy, which is shown in Fig. 2. Every ***k*** blocks are grouped into a group which is named *dynamic reliability group*. In each of the group (k + g,g)-erasure code is used to encode group members as ***g*** parity blocks, which are called *group parity blocks*. For each of F_i, we also use (n + b,b)-erasure code over $GF(2^l)$ to encode $\{f_{ij}\}_{(i \in \{1,\ldots,n\})}$ sections as ***b*** small parity sections, which are named *block parity sections*.

  As mentioned, our scheme has two data recovery methods at different levels of granularity, there are coarse-grained *intra-group* and fine-grained *intra-block* recovery methods. Group parity blocks are used in intra-group recovery method which has higher fault tolerance but slower rate of recovery, but intra-block recovery method utilizes blocks' parity sections to restore corrupted sections whose feature is fast repair speed but lower data recovery ability.
- *Recovery Scenarios:* To be convenient for explanation, this section only consider the recovery scenario in one single dynamic reliability group. The way to recover multi groups is the same as a single group. When corrupted data is detected, verification function could tell us the number of damaged sections, which is denoted as ***d***. The following scenarios are possible:

Scenario A: If $\boldsymbol{d \le b}$, no matter whether these ***d*** damaged sections are included in a single block or not, intra-block recovery method, by the knowledge of erasure code, has ability to reconstruct those corrupted blocks. Intra-group recovery method, which has higher fault tolerance, can also recover data successfully in above case, but its rate of recovery is slower than intra-block's. Therefore, we take intra-block recovery method without using intra-group method in this case.

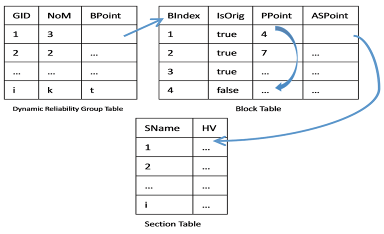
Scenario B: Damaged sections are included in a single block and ***d*** satisfies the inequality $\boldsymbol{b < d \le n}$**.** The *intra-block recovery method* is disable, and intra-group recovery method can play a role to reconstruct the block even this whole block is missed.

Scenario C: If the number of corrupted blocks—in each of which has at least b up to n of broken sections—is up to g, we must first use intra-block recovery method to repair the rest of blocks, and only then use intra-group recovery method to recover above-mentioned block sections.

3.2 Map-Based Dynamic Storage Table

The map-based dynamic storage table (MBDST) is a small dynamic data structure stored on the user(or TPA) side to validate the integrity and consistency of file blocks outsourced to the CSP. It is composed of three sub-tables, which are dynamic group table, block table and section table. An example about the data structure is given in Fig. 3.

As its name indicates, the **group table** is used for managing the information of dynamic reliability group, which consists of three columns. *NoM* is used to show how many original data blocks are in this group. In our scheme the minimum number of blocks that a group contains has been set, if the number of memberships in this group is



GID	NoM	BPoint
1	3	
2	2	…
…	…	…
i	k	t

BIndex	IsOrig	PPoint	ASPoint
1	true	4	
2	true	7	…
3	true	…	…
4	false	…	…

SName	HV
1	…
2	…
…	…
i	…

Fig. 3. Map-based dynamic storage table

less than the threshold we will merge a group into another, so we can judge whether this group should be merged or not via *NoM*. *BPoint* is a pointer that points to the block table belonging to this group.

Block table is the virtual table, in other words, the base unit of storage system for blocks and their parity data is section. Each row in a **block table** represents an virtual data block or a parity block, and it has four columns. *BIndex* indicates the logical position of block, and the index is numbered from 1. *IsOrig* is used to mark if this block is an original data block or a parity block. *PPoint* is a point that point to its parity blocks. *ASPoint* points to sections that belonged to this virtual block.

Every virtual block holds a section table, which contains metainformation about each section. There are only two columns: *Sname* and *HV*. *HV* stores the hash value of the block which is used for determining specified corrupted sections.

It is important to note that the verifier keeps only one structure for unlimited number of file sections, i.e., the storage requirement on the verifier side does not depend on the number of file blocks on cloud servers. For k sections of a data file of size$|F|$, the storage requirement on the CSP side is $\boldsymbol{O(|F|)}$, while the verifier's overhead is O(m + n + k) for the whole file (m is the number of dynamic groups, n is the number of blocks).

3.3 Collusion Attack

We apply algebraic signature to verify integrity of data in cloud storage, but that basic verification model may suffer from a drawback: many storage servers may make up signatures as long as they internally consistent. To solve this problem, we developed an improved model that allows a user or TPA to ensure the security of signature verification process with little storage overhead.

We simply blind the original data by pseudo-random stream. Firstly, we design a hash function ***H(block_id, user_id, secret_key)***, where block_id identifies a block, user_id varies for each use of the hash function, and secret_key is used to prevent storage servers from deriving the same function. The original data is then XORed with a pseudo-random stream generated by RC4, for example, seeded with a value derived from the above hash function; thus, the value stored for a block section would be $s_{ij} \oplus ps_{ij}$, where s_{ij} is the *j*th section of the block b_i, and ps_j is the *j*th value of the pseudo-random stream for b_j.

When the user sends a challenge to storage servers and receives algebraic signatures, he must remove the blinding factor from the signature of encrypted data sections (sig_i, for example). This can be done by computing the signature of the values in the pseudo-random stream and XORing it with the signature of encrypted section data. That is to say, let signature of the values in the pseudo-random stream is ps_i, the signature of original data section is $sig_i \oplus ps_i$. Then the user can utilize the method we mentioned in Sect. 4 to verification the correctness of his data. Because of hardly obtaining the hash function and secret key, CSP cannot colluded to offer fake data signatures, thus our scheme achieves the goal to defense collusion attack.

4 Implementation of MB-DDIVR

We propose a MB-DDIVR scheme allowing the user to update and scale the file outsourced to cloud servers which may be untrusted. Our scheme consists of four algorithms: file distribution preparation, integrity verification, data recovery and dynamic operation. In this section, we will give detailed implements descriptions of these for algorithms in proposed scheme.

4.1 File Distribution Preparation

The preparation operation is to encode files. As mentioned in Sect. 4, firstly we partition F into m distinct blocks $F = (b_1, b_2, \ldots, b_m)$, then every block b_i continues to be partitioned into n distinct sections $b_i = (s_{i1}, s_{i2}, \ldots, s_{in})(i \in \{1,\ldots,m\})$, where $n \le 2^p - 1$. Next, we group these file blocks into different dynamic reliability groups and compute parity data for dynamic reliability group and blocks. Using erasure code algorithm to generate parity blocks and parity sections for every dynamic reliability group DRG_i and every block, i.e., $(GroupP_{i1},\ldots, GroupP_{ig})$ and $(BlockP_{i1}, \ldots, BlockP_{ik})$, respectively.

We then generate hash for each block section and parity data. Generating secret keys and using pseudo-random stream mentioned in Sect. 4 to blind user's original data sections Σs_{ij} can defense storage servers' collusion attack. At last we should use relevant information to initialize our dynamic map-based storage table. The user only keeps secret keys in local and uploads the file and parity data to the cloud, and dynamic storage table to the TPA respectively.

4.2 Integrity Verification

Algebraic signature interact well with linear error and erasure correcting codes, thus, it may be used to verify that the parity and the data files are coherent without the need to obtain the entire data and parity, and users have no need of saving any validate tags. Specifically, corruption localization is a very important link in integrity verification in storage system. However, many previous schemes do not consider the problem of data error localization. Our scheme exceeds those by combining the integrity verification with error localization.

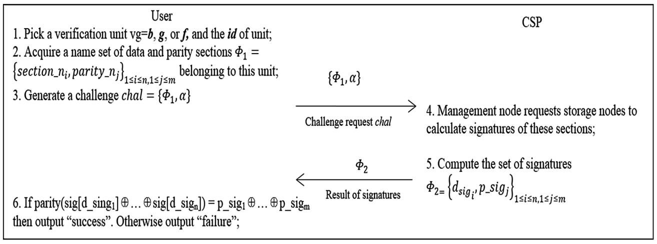

User
1. Pick a verification unit vg=b, g, or f, and the id of unit;
2. Acquire a name set of data and parity sections $\Phi_1 = \{section\_n_i, parity\_n_j\}_{1\le i\le n,1\le j\le m}$ belonging to this unit;
3. Generate a challenge chal = $\{\Phi_1, \alpha\}$
CSP
$\{\Phi_1, \alpha\}$
Challenge request chal
4. Management node requests storage nodes to calculate signatures of these sections;
5. Compute the set of signatures $\Phi_2 = \{d_{sig_i}, p\_sig_j\}_{1\le i\le n,1\le j\le m}$
Φ_2
Result of signatures
6. If parity(sig[d_sing1] ⊕ ... ⊕ sig[d_sign]) = p_sig1 ⊕ ... ⊕ p_sigm then output "success". Otherwise output "failure";

Fig. 4. Protocal of integrity verification

The verification protocol is illustrated in Fig. 4. Like recovery method, our verification method is at different levels of granularity, which are block level verification, group level verification and *file level* verification. We will use ***b***, ***g*** and ***f*** to represent above verification operations respectively. Our challenging index dataset is $\Phi = \{section\_n_i, parity\_n_j\}_{1 \le i \le n, 1 \le j \le m}$, where section_n$_i$ indicates the name of the i data section, and parity n$_j$ is the name of the j parity section. The user firstly select which level he want to verify, the index of group or block and the parameter of algebraic signature α, then sends the audition command to the CSP. A user sets a challenging index dataset to the TPA and delegates TPA to carry out the audit. Upon receiving the index dataset, the CSP let each corresponding storage server compute a short signature over the specified blocks and return them to the TPA. After receiving the set of signatures from CSP, the user can easily judge the integrity of data file.

4.3 Dynamic Data Operation

In cloud storage, there are many scenes where data stored in the cloud is dynamic, like log files. Therefore, it is crucial to consider the dynamic cases, where users may perform update, insertion and deletion operation at the block level. However, for users, CSP is untrusted and they do not wish these dynamic operations to be dominated by CSP, but performed by delegated TPA instead. In this section we design dynamic operation to reduce overhead and in practice, this cast is perfectly acceptable to users.

- *Update Operation:* A basic data modification operation refers to replace specified blocks with new ones. Let a file **F** = {b_1, b_2, …, b_m}, suppose the user wants to modify a block b_j with b_j'. He sends the block b_j' to TPA, and delegate the TPA to execute update operation. The TPA gets the reliability group RG$_i$ which contains block b_j, then obtains all block sections belong to group RG$_i$ from CSP. After updating b_j to b_j', and re-encoding group parity blocks and block parity sections, whose new values are GPB_i' and BPS_j', the TPA partitions the new block b_j' into n sections $\{s_{j,i}'\}_{i \in (1,n)}$, and update the table list entry. At last the TPA replaces original data in cloud with new data.
- *Delete Operation:* A delete operation on a file means deleting few file blocks from the file. To delete a specific data block b_j, the TPA gets the reliability group RG$_i$ which contains block b_j, then the user sends download commands to CSP to only

obtain data sections belong to group RG_i excepts $\{s_{j,i}\}_{i\in(1,n)}$, then re-computes group parity blocks, whose new value is $GPB_i^{'}$ and updates the table list entry. The user sends commands to CSP to replaces $RGEC_i$ with $RGEC_i^{'}$ in the cloud storage, and removes local data that just downloaded and generated.

- *Append Operation:* Block appending operation means adding a new block at the end of the outsourced data. To simplify the presentation, we suppose the user want to append block $b^{'}$ at the end of file **F**. The TPA looks in the **MBDST** to get the information about last reliability group RG_i and downloads data sections. If the group RG_i contains too much blocks after adding the block $b^{'}$ (this cloud be judged regarding a threshold), go to (a), or go to (b).

(a) RG_i will be partitioned into two dynamic reliability group RG_i and RG_{i+1}, and $b^{'}$ is merged into group RG_{i+1}. And then calls ***GPBGen(RG_i)*** and ***GPBGen(RG_{i+1})*** functions to re-encode group parity blocks, whose new values are $GPB_i^{'}$ and $GPB_{i+1}^{'}$. TPA updates the table list entry, and replace GPB_i with new value, then uploads $b^{'}$ and $GPB_{i+1}^{'}$ to the cloud storage.

(b) $b^{'}$ is merged into group RG_i, TPA re-encodes group parity block $GPB_i^{'}$, then updates the table list entry and replace GPB_i with new value, then uploads $b^{'}$ to the cloud storage.

5 Experiment and Evaluation

We have implemented our scheme in C and JAVA language. We conducted several experiments using the local 32-bit Centos operation system with an Intel Core 2 processor running at 1.86 GHz, 2048 MB of RAM, and run Hadoop 2.7.0. Most PDP and POR schemes who support dynamic data operation are based on an authenticated data structures – Merkle Hash Tree (MHT) [12] (short for MHTPOR-scheme), and we also construct a scheme based on that structure which is used as a reference model for comparing the proposed MB-DDIVR scheme. In [12] the MHT is explained in detail.

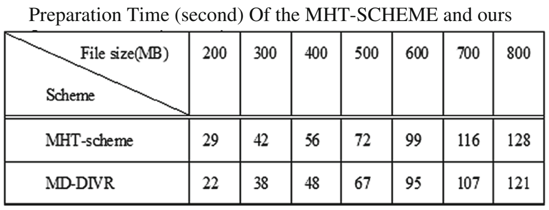
In order to compare the performance of the two schemes mentioned above, the stored files are chosen from 200 MB to 800 MB. We set up the size of every data block is 126 MB, and each of them will be partitioned into 6 sections. Every dynamic reliability group contains 3 blocks. One encoded parity block is generated for each of groups and two encoded parity sections are generated for every data block.

5.1 Time Analysis

In this section, we compare the MHTPOR scheme with our proposed scheme in file distribution preparation time and verification time.

- File Distribution Preparation Time
  The experimental results are shown in Table 1. It is can be seen from the result that the computation costs grow with the increase of file size linearly. Note that MHT-scheme need not to generate group encoded blocks, and the run time of our scheme is less than that of the MHT-scheme. That is because the creation of MHT

Table. 1 Preparation Time (second) of the MHT-SCHEME and ours



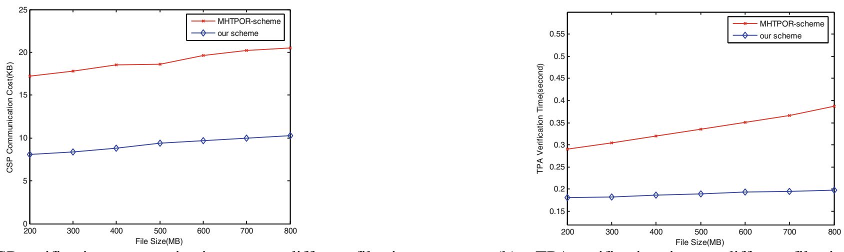
Preparation Time (second) Of the MHT-SCHEME and ours

File size(MB) / Scheme	200	300	400	500	600	700	800
MHT-scheme	29	42	56	72	99	116	128
MD-DIVR	22	38	48	67	95	107	121

and the operation on it is much more complicated than our designed dynamic storage table.

- Integrity Verification Cost

  In this experiment we select to verification the integrity of whole file. Figure 5(a) the communication cost on the CSP side when audit different size of file. The result shows that the cost is almost constant versus file size, i.e., the file size has no remarkable influence on the CSP's cost. Figure 5(b) also indicates our verification time on TPA side is also almost constant versus file size. That is due to the fact that algebraic signature has feature of homomorphism, and only need XOR operation, which has quick operating speed than regular integrity verification of PORs.



(a) CSP verification communication cost on different file size (b) TPA verification time on different file size

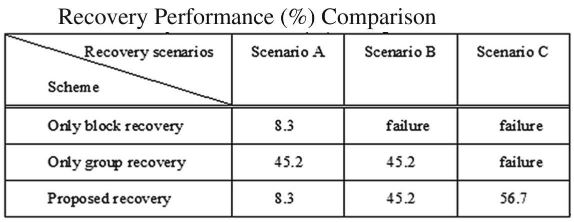
Fig. 5. Integrity verification cost.

5.2 Recovery Performance Analysis

We will use the experimental 620 MB data file to analyze the performance of our proposed data recovery methods under the same experimental background mentioned above. From the aspect of theoretic analysis, as we discuss in section recovery method in III, the worst-case scenario is the scenario C. In our experiment, each of the two groups misses one block (i.e., six sections, and every section is 21 MB), and each of the rest blocks misses two sections. So the fault tolerance rate in this experimental scheme is (2 * 126 MB + 3 * 2 * 21 MB)/620 MB = 60.9 %.

By contrast, we design the other two recovery methods, and each of them has only one recovery grain: intra-block and intra-group recovery methods. As can be seen in Table 2, the efficiency and capacity of recovery is not high if we use intra-block or

Table. 2 Recovery Performance (%) Comparison



Recovery Performance (%) Comparison

Scheme \ Recovery scenarios	Scenario A	Scenario B	Scenario C
Only block recovery	8.3	failure	failure
Only group recovery	45.2	45.2	failure
Proposed recovery	8.3	45.2	56.7

intra-group recovery methods alone. We offer flexible and powerful solutions to reconstruct corrupted data, it owns more powerful recovery capability to combine the intra-block and intra-group recovery methods together.

6 Related Works

To check the cloud server, researchers proposed the POR protocol [1–3] that enables the servers to demonstrate to the verifier whether the data stored in the servers is intact and available. But existing POR scheme can only deal with static data. Zheng and Xu [9] proposed 2-3 tree as the data structure on top of POR. This scheme introduced a new property, called fairness, which is necessary and also inherent to the setting of dynamic data. The related concept of PDP was introduced by Ateniese et al. [4], which was the first model for ensuring the possession of files on untrusted storages. PDP uses homomorphic tags for auditing outsourced data without considering dynamic data storage. Subsequent works [5–7] proposed data update protocols in the PDP model. Wang et al. [10] used Merkle Hash Tree to detect data integrity and their scheme supports dynamic data operations, however, this scheme did not encrypt the data and was only useful for a single copy. Curtmola et al. [6] proposed multiple-replica PDP scheme where the data owner can verify that several copies of a file are stored by the CSP.

Algebraic signature was proposed by Schwarz and Miller [11], which detects for sure any change that does not exceed n-symbols for an n-symbol signature. This scheme primarily utilizes one such characteristic homomorphism: taking a signature of parity gives the same result as taking the parity of the signatures. For example, assume that we have an erasure correcting code that calculates k parity containers $\mathbf{P_1, \ldots, P_k}$ from the m data buckets $\mathbf{D_1, D_2, \ldots, D_m}$ as $P_i = \mathcal{P}_i(\mathbf{D_1, D_2, \ldots, D_m})$. So there is:

$$sig_\alpha(\mathrm{p}_i(\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_m)) = \mathrm{P}_i(sig_\alpha(\mathrm{D}_1), sig_\alpha(\mathrm{D}_2), \ldots, sig_\alpha(\mathrm{D}_m))$$

7 Conclusion

In this paper, we proposed a distributed integrity verification and recovery scheme to ensure users' outsourced data with explicit dynamic data operation support. We utilize erasure coding to guarantee data confidentiality and dependability. To ensure the integrity of data files, an efficient dynamic data integrity checking scheme is constructed based on the principle of algebraic signatures.

Acknowledgment. This work is supported by the National High Technology Research and Development Program ("863" Program) of China under Grant No. 2015AA016009, the National Natural Science Foundation of China under Grant No. 61232005, and the Science and Technology Program of Shen Zhen, China under Grant No. JSGG2014051 6162852628.

References

1. Juels, A., Kaliski, B.S. Jr.: PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM (2007)
2. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
3. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security. ACM (2009)
4. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM (2007)
5. Erway, C.C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. ACM Trans. Inf. Syst. Secur. (TISSEC) **17**(4), 15 (2015)
6. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: multiple-replica provable data possession. In: The 28th International Conference on Distributed Computing Systems, 2008. ICDCS 2008. IEEE (2008)
7. Hao, Z., Yu, N.: A multiple-replica remote data possession checking protocol with public verifiability. In: Proceedings of 2nd International Symposium Data, Privacy, E-Commerce, pp. 84–89 (2010)
8. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. ACM (2008)
9. Zheng, Q., Xu, S.: Fair and dynamic proofs of retrievability. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy. ACM (2011)
10. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: INFOCOM, 2010 Proceedings IEEE, pp. 1–9. IEEE (2010)
11. Schwarz, T.S.J., Miller, E.L.: Store, forget, and check: using algebraic signatures to check remotely administered storage. In: 26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006. IEEE (2006)
12. Merkle, R.C.: Protocols for public key crytosystems. In: Proceedings of IEEE Symposium Security and Privacy, p. 122 (1980)