

## Research Article

# A Scheme for Verification on Data Integrity in Mobile Multicloud Computing Environment

**Laicheng Cao, Wenwen He, Xian Guo, and Tao Feng**

*School of Computer and Communication, Lanzhou University of Technology, Lanzhou 730050, China*

Correspondence should be addressed to Laicheng Cao; [caolaicheng@163.com](mailto:caolaicheng@163.com)

Received 24 February 2016; Accepted 25 July 2016

Academic Editor: Yuqiang Wu

Copyright © 2016 Laicheng Cao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In order to verify the data integrity in mobile multicloud computing environment, a MMCDIV (mobile multicloud data integrity verification) scheme is proposed. First, the computability and nondegeneracy of verification can be obtained by adopting BLS (Boneh-Lynn-Shacham) short signature scheme. Second, communication overhead is reduced based on HVR (Homomorphic Verifiable Response) with random masking and sMHT (sequence-enforced Merkle hash tree) construction. Finally, considering the resource constraints of mobile devices, data integrity is verified by lightweight computing and low data transmission. The scheme improves shortage that mobile device communication and computing power are limited, it supports dynamic data operation in mobile multicloud environment, and data integrity can be verified without using direct source file block. Experimental results also demonstrate that this scheme can achieve a lower cost of computing and communications.

## 1. Introduction

Cloud computing is envisioned as the future IT service paradigm, it has attracted tremendous attention from academia and industry [1], and mobile cloud model is the future development trend of cloud computing [2, 3]; mobile cloud computing requires cloud data security assurance mechanisms, which can prevent leakage and loss of user data [3, 4]. However, CSP (the cloud service providers) are usually not trustworthy. They may conceal the data loss or error from the users for their own benefit. Even more, they might delete rarely accessed user data for saving storage space [4, 5]. As a result, many users are still hesitant to use cloud storage due to security and confidentiality threats toward their outsourced data.

Therefore, how to verify the integrity of customer data under cloud storage environment is a serious problem. To solve this problem, [6] presented a data integrity verification scheme based on the RSA algorithm. This scheme made RSA exponentiation for the entire file, but it has the large computational overhead. Reference [7] proposed a scheme based on labeling, which can verify data integrity but does not support dynamic updates data. Reference [8] used homomorphic verify label scheme though to reduce communication overhead,

but it does not support dynamic updates data. Considering the limited computing and storage capacity of mobile devices in mobile cloud computing environment, the complexity of the data integrity verification process on the mobile terminal is inappropriate. Therefore, [9–11] proposed a scheme that is suitable for verification on data integrity in mobile cloud computing environment. But these programs are the circumstances under a single cloud environment.

For the general cloud storage case, owing to the security and control reasons, some data cannot be placed in the public cloud. In order to save costs, the user wishes to store the lower security level data on the public cloud; thereby it creates a hybrid cloud model. However, the mobile cloud users also face the issue that not all data can be placed in the public cloud. Therefore, a solution needs to be proposed to reduce the cost of mobile cloud computing and communications, but also to ensure safety and to apply for hybrid cloud environments. Based on these two needs identified above, we propose hybrid data integrity verification scheme for the mobile cloud environment. The scheme supports mobile end users to verify the cloud data integrity and update operations; what is more, it also supports third-party verification without the direct involvement of source file and saved authentication

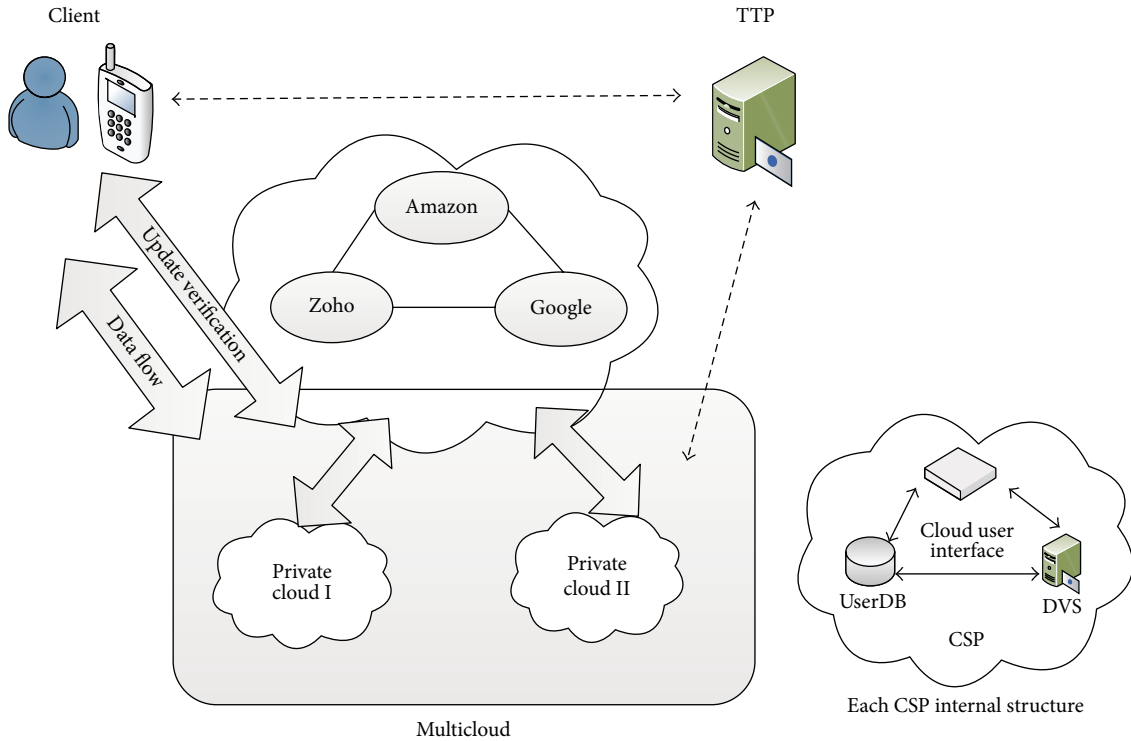


FIGURE 1: MMCDIV scheme system components.

information state. Our scheme is suitable for mobile multicloud computing environments oriented application data.

The rest of the paper is organized as follows. Section 2 presents architecture and techniques. Section 3 describes the proposed scheme in detail. We also provide security analysis and performance evaluation in Sections 4 and 5. Section 6 concludes the proposed scheme.

## 2. Architecture and Techniques

**2.1. The Architecture of System.** Generally, a mobile multicloud data integrity verification scheme consists of the following components:

- (i) *Client*, the owner of the data, who has a lot of data stored in the hybrid cloud;
- (ii) *CSP*, multiple cloud storage service providers that supply client with collaborative data storage services (there is an organizer, which manages cloud storage service providers and directly communicates with the verifier).
- (iii) *TTP*, public information that is stored when a trusted third-party verification is required.

The client is installed on a mobile device and user can interact with the CSP, send a request to the CSP, and accept the data which CSP return. CSP consist of the cloud user interaction interface, data validation services (DVS), and user database. DVS performs specific validation algorithm logic. User databases store user information. TTP (trusted third party) is used to verify the integrity of user data and reduce the burden

on the client computing. The system components of mobile multicloud data integrity verification (MMCDIV) scheme are shown in Figure 1.

**2.2. Short Signature Technology Based on Elliptic Curve Bilinear BLS.** In our scheme, elliptic curve bilinear pairing BLS (Boneh-Lynn-Shacham) short signature scheme [12] is used as the theoretical basis. BLS short signature scheme is based on bilinear mapping  $e: G_1 \times G_2 \rightarrow G_T$ . Among them, groups  $G_1$  and  $G_2$  are two Gap Diffie-Hellman (GDH) groups. Mapping  $e$  has three properties:

- (1) *Computability.* Mapping  $e$  is an efficient algorithm to calculate the bilinear mapping  $e$ .
- (2) *Bilinear.* For any  $h_1, h_2 \in G$ ,  $a, b \in Z_p$ ,  $Z_p$  is the nonnegative integers which is less than  $p$  and bilinear mapping  $e$  is

$$e(h_1^a, h_2^b) = e(h_1^b, h_2^a) = e(h_1, h_2)^{ab}. \quad (1)$$

- (3) *Nondegenerative.* Consider  $e(h_1^a, h_2^b) \neq 1$  and  $g$  is a generator of group  $G$ . When using BLS method, if the user client needs his message signature, he can make  $x$  as an any element of the collection of  $Z_p$ ; then his public key is  $g^x$ . To sign the message, client must map his/her own message for an element  $h$  of group  $G$  and then generates a message signature  $h^x$ . Another user wants to verify this message; he needs to check whether  $e(h, g^x) = e(\delta, g)$  is established. Only when the left is equal to the right of this formula, this user can verify this message.

**2.3. Homomorphic Verifiable Response.** Reference [13] extended state verification label HVT (Homomorphic Verifiable Tags) to the same state authentication response HVR (Homomorphic Verifiable Response), which can aggregate responses from many different CSP into a response. HVR method can reduce communication overhead but also hide the location information of user data.

**2.4. Sequence-Enforced Merkle Hash Tree.** The sMHT (sequence-enforced Merkle hash tree) [14] is a hash tree structure to solve the problem that the original BLS cannot verify whether the data of the service provider returned is the challenge specified data. This method makes the hash value of the file block labels from left to right in the order correspond to the hash leaves node and then orderly links two-level hash sequence and finally calculates the hash value of the root node. The process of verifying the 2nd data block is the following.

To verify  $T_2$ 's value and position, root( $R, 4$ ) and  $\Omega_2 = \{(h_1, 1, 1), (h_b, 2, 0)\}$  are used:

- (1) Calculate rank of  $A$  as  $h_a = h(h(T_2 \parallel 1) \parallel h_1 \parallel 2)$ .
- (2) Calculate rank of root as  $R' = h(h_a \parallel h_b \parallel 4)$ .
- (3) Verify if  $R = R'$ .

Note:  $\Omega$  denotes auxiliary authentication information (AAI), 1 indicates the left sibling node, and ( ) indicates the right one on the proof path. Left( ) signifies all the rank of the left sibling node in  $\Omega$ . The process of verifying the 2nd data block is shown in Figure 2.

### 3. The Proposed Scheme

In our scheme, we split a file into  $n$  blocks; each of block has a sector which is notated as  $s$ , so a file  $F$  can be expressed as the file with  $n \times s$  section,  $F = \{m_{i,j}\}_{\substack{i \in [1,n] \\ j \in [1,s]}}$ . We use  $t$  to describe the private key of the organizer CSP and use  $c$  to express the number of cloud storage files.  $\delta$  describes the set of tags,  $\delta = \{\delta_i\}_{i \in [1,n]}$ .  $Q$  describes the set of index-coefficient pairs,  $Q = \{(i, v_i)\}$ .  $\rho$  describes the response for the challenge  $Q$ .  $C$  describes the set of CSP to store a file.

**3.1. Setup Phase.** The setup phase of data integrity verification parameter in MMCDIV scheme is shown as follows.

**Phase 1 (KeyGen( $l$ )).** Client takes a security parameter  $l$  as input and gets the return value of a public-secret key pair (spk, ssk) and then selects random  $\beta \in Z_p$ , at the same time selects  $s$  random elements  $\{u_1, u_2, \dots, u_s\} \subset G_1$ , calculates  $u = g^\beta$ , and finally makes  $sk = (\beta, ssk)$ ,  $pk = (u, spk, g, \{u_j\}_{1 \leq j \leq s})$ .

**Phase 2 (TagGen(sk,  $F, C$ )).** File  $F$  is divided into  $n \times s$  districts by client,  $F = \{m_{i,j}\}_{\substack{i \in [1,n] \\ j \in [1,s]}}$ , the label of  $F$  is  $t = \text{name} \parallel n \parallel \text{Sig}_{\text{ssk}}(\text{name} \parallel n)$ , and calculating file block of the signature is shown as follows:

$$\delta_i^k = \left( \{H(m_i)\} \cdot \prod_{j=1}^s u_j^{m_{i,j}} \right)^\beta \quad (1 \leq k \leq c, m_i \in C_k). \quad (2)$$

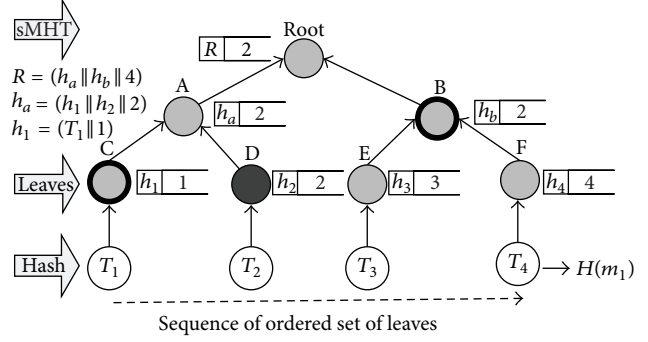


FIGURE 2: The process of verifying the 2nd data block.

According to  $H(m_i)$  constructing sMHT with a root  $R$ , client uses the private key  $\beta$  for signature for  $R$ :  $\text{Sig}_{\text{sk}}(H(R)) = (H(R))^\beta$ . Finally, he sends  $\{m_i, \delta_i^k\}_{m_i \in C_k}$  to each of the CSP, sends  $\text{Sig}_{\text{sk}}(H(R))$  to the organizer, and deletes the local file.

#### 3.2. Proof Phase

**3.2.1. Constructing Challenges of the Information.** Data integrity can be verified by client and also can be verified by TTP for reducing the computation and storage burden of client. Firstly, TTP requires organizers to send the label  $t$  of file to him and then uses public key  $pk$  of client to verify  $t$ . If this verification is not passed, the file label may be damaged. Otherwise, challenge is executed. TTP randomly selects subset  $I$  of set  $[1, n]$ , for each  $i \in I$ , and selects a random number  $v_i \in Z_p$  to construct a challenge  $Q = \{(i, v_i)\}_{i \in I}$ , which is sent to a organizer. Lastly, the organizer sends  $Q = \{(i, v_i)\}_{i \in I}$  to each of the CSP.

**3.2.2. Performing Gen Proof.** Each  $C_k$  obtains  $Q_k = \{(i, v_i)\}_{m_i \in C_k} \subseteq Q$ , selects  $s$  random elements  $r_{j,k} \in Z_p$ ,  $j \in [1, s]$ , then calculates  $\pi_{j,k} = (u_j)^{r_{j,k}}$ , computes corresponding value  $\mu'_{j,k} = \sum v_j m_{i,j}$  for each data block, and at the same time calculates

$$\begin{aligned} \mu_{j,k} &= \mu'_{j,k} + r_{j,k} h(\pi_{j,k}) \in Z_p, \\ \delta_k &= \prod_{(i, v_i) \in Q_k} (\delta_i^k)^{v_i}, \\ \pi_k &= \prod_{j \in [1,s]} \pi_{j,k}^{h(\pi_{j,k})}. \end{aligned} \quad (3)$$

Then  $C_k$  returns  $\rho_k = (\{\mu_{j,k}\}_{j \in [1,s]}, \delta_k, \pi_k)$  to organizer. After receiving response from each of the CSP, organizer selects a random number  $\gamma \in Z_p$  and calculates  $H = v^\gamma \in G_2$ . Next, organizer aggregates these evidences  $\mu_j = \gamma \sum_{C_k \in C} \mu_{j,k}$ ,  $\delta = (\prod_{C_k \in C} \delta_k)^\gamma$ , and  $\pi = \prod_{C_k \in C} \pi_k$ . Last, organizer sends  $C = (\{H(m_i), \Omega_i\}_{i \in I}, \text{Sig}_{\text{sk}}(H(R), \rho, H))$  to TTP.

**3.2.3. Executing Algorithms Verify Proof.** After receiving authentication information  $C$ , TTP uses  $\{H(m_i), \Omega_i\}_{i \in I}$  to

reconstruct hash root node  $R$  and make the following two verifications:

$$\text{Verification 1: } e(\text{Sig}_{\text{sk}}(H(R)), g) = e((H(R))^\beta, g). \quad (4)$$

*Proof.* If validation fails, it returns false. If the validation is successful, it indicates that the root node  $R$  and the signature  $\text{Sig}_{\text{sk}} = (H(R))$  are matched under the premise of root node  $\text{Sig}_{\text{sk}} = (H(R))$  given CSP. It can also show that the label of file block  $\{H(m_i)\}_{i \in I}$  is intact and proceed to the next verification:

$$\begin{aligned} \text{Verification 2: } e(\delta, g) \\ = e\left(\pi^{-1} \cdot \prod_{i \in I} H(m_i)^{v_i}, H\right) e\left(\prod_{j=1}^s u_j^{H_j}, v\right). \end{aligned} \quad (5)$$

□

*Proof.* We just prove that left is equal to right (as shown in (6)):

$$\begin{aligned} \text{Left} &= e\left(\left(\prod_{C_k \in \mathcal{C}} \left(\prod_{(i, v_i) \in Q_k} \delta_i^k\right)^{v_i}\right), g\right) \\ &= e\left(\left(\prod_{C_k \in \mathcal{C}} \left(\prod_{(i, v_i) \in Q_k} \left(\{H(m_i)\} \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^\beta\right)^{v_i}\right), g\right) \\ &= e\left(\prod_{i \in I} \left(H(m_i) \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^{v_i}, g^\beta\right) \\ &= e\left(\prod_{i \in I} (H(m_i))^{v_i}, v\right) e\left(\prod_{j=1}^s u_j^{\sum_{C_k \in \mathcal{C}} v_j m_{i,j}}, v\right) \\ \text{Right} &= e\left(\prod_{C_k \in \mathcal{C}} \pi_k^{-1} \cdot \prod_{i \in I} T_i^{v_i}, v^\gamma\right) e\left(\prod_{j=1}^s u_j^{\sum_{C_k \in \mathcal{C}} H_{j,k}}, v\right) \\ &= e\left(\prod_{i \in I} (H(m_i))^{v_i}, v\right) \cdot e\left(\prod_{C_k \in \mathcal{C}} \prod_{j=1}^s \pi_{j,k}^{-h(\pi_{j,k})}, v\right) \\ &\cdot e\left(\prod_{j=1}^s u_j^{\sum_{C_k \in \mathcal{C}} v_j m_{i,j}} \cdot \prod_{j=1}^s u_j^{\sum_{C_k \in \mathcal{C}} \gamma_{j,k} h(\pi_{j,k})}, v\right) \\ &= e\left(\prod_{i \in I} (H(m_i))^{v_i}, v\right) e\left(\prod_{j=1}^s u_j^{\sum_{C_k \in \mathcal{C}} v_j m_{i,j}}, v\right) \end{aligned} \quad (6)$$

Thus, Left = Right.

If the validation fails, it shows that documentation block is corrupt. It returns false. On the other hand, it shows that documentation block is complete. □

### 3.3. Implementation of Dynamic Operation

(i) *Constructing Updates Information.* The  $i$  block data  $m_i$  is updated to  $m_i^* (m_{*1}, m_{*2}, \dots, m_{*s})$  by client; the steps of constructing updates information are shown as follows.

*Step 1.* Client calculates  $H(m_i^*)$  (the hash value of  $m_i^*$ ) and computes  $\delta_i^{*k}$  (label of  $m_i^*$ ).

*Step 2.* Client constructs update information (update =  $(M, I, H(m_i^*))$ ) and sends it to organizers.

*Step 3.* Client sends  $(M, I, m_i^*, \delta_i^{*k})$  to the corresponding CSP.

(ii) *Executing Update.* After the updates information is constructed, client will execute update data; the process is described as follows.

*Step 1.* CSP replace blocks  $m_i^*$  and  $\delta_i^*$ , update the hash tree, generate a new node  $R'$  and updated authentication information  $C_{\text{update}} = (\Omega_i, H(m_i), \text{Sig}_{\text{sk}}(H(R)), H(R'))$ , and then send it to the TTP.

*Step 2.* TTP uses  $(\Omega_i, H(m_i))$  to reconstruct the root node  $R$  and verifies  $e(\text{Sig}_{\text{sk}}(H(R)), g) = e(H(R), v)$ . If verification is passed, client continues to check whether CSP execute update data option and then uses  $(\Omega_i, H(m_i^*))$  to structure updated value. Client sends  $m_i^*$  to TTP and compares  $m_i^*$  with  $R'$ . If it is consistent, it shows that client completes the update operation.

*Step 3.* Client uses his private key  $\text{sk}$  to calculate the label  $(\text{Sig}_{\text{sk}}(H(R')))$  of root node  $R'$  and then sends it to the CSP.

*Step 4.* CSP perform the insert operation. CSP first input the location of original file and the original file block label and request  $(F, \delta, C_{\text{insert}})$  that client sends, and then produce outputs  $(F', \delta', C_{\text{insert}})$ .

*Step 5.* CSP perform the removal operation. CSP first input the deleted request  $C_{\text{delete}}$  information which client sends and then produce outputs  $C_{\text{delete}}$  operations information.

## 4. Security Analysis

This MMCDIV scheme is based on the absence of an effective security algorithm; it can solve elliptic curve discrete logarithm problem and characteristics of GDH (Gap Diffie-Hellman).

(1) *Resisting Forgery Attack.* Suppose the CSP to forge  $\delta_{\text{csp}}$  select the same  $s$  random elements  $\{u_1, u_2, \dots, u_s\} \subset G_1$  and a random number  $\gamma \in Z_p$ ; they can calculate (7) in accordance with (5) and (6):

$$\begin{aligned} \text{Left}_{\text{csp}} = e(\delta_{\text{csp}}, g) &= e\left(\left(\prod_{C_k \in \mathcal{C}} \left(\prod_{(i, v_i) \in Q_k} \delta_{i, \text{csp}}^k\right)^{v_i}\right), g\right) \\ &= e\left(\left(\prod_{C_k \in \mathcal{C}} \left(\prod_{(i, v_i) \in Q_k} \left(\{H(m_i)\} \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^\beta\right)^{v_i}\right), g\right), \end{aligned}$$

$$g) = e\left(\prod_{i \in I} \left(H(m_i) \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^{\gamma v_i}, g^{\beta_{\text{csp}}}\right)$$

$$\text{Right}_{\text{csp}} = e\left(\prod_{i \in I} (H(m_i))^{\gamma v_i}, v\right) \cdot e\left(\prod_{j=1}^s u_j^{\gamma \sum_{C_k \in C} v_j m_{i,j}}, v\right).$$
(7)

Equation (8) is obtained by (6) as follows:

$$\text{Left} = e\left(\prod_{i \in I} \left(H(m_i) \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^{\gamma v_i}, g^{\beta}\right)$$

$$= e\left(\prod_{i \in I} (H(m_i))^{\gamma v_i}, v\right) \cdot e\left(\prod_{j=1}^s u_j^{\gamma \sum_{C_k \in C} v_j m_{i,j}}, v\right)$$

$$= \text{Right}.$$
(8)

Because the right of (8) is equal to the right ( $\text{Right}_{\text{csp}}$ ) of (7), the forgery attack is going to be carried out; only (9) is established:

$$e\left(\prod_{i \in I} \left(H(m_i) \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^{\gamma v_i}, g^{\beta_{\text{csp}}}\right)$$

$$= e\left(\prod_{i \in I} \left(H(m_i) \cdot \prod_{j=1}^s u_j^{m_{i,j}}\right)^{\gamma v_i}, g^{\beta}\right).$$
(9)

So (10) is obtained by (9):

$$g^{\beta_{\text{csp}}} = g^{\beta}. \quad (10)$$

The prerequisite of (10) establishment is  $\beta_{\text{csp}} = \beta$ ; because of the hardness problem of GDH, the CSP cannot calculate the private key  $\beta$  of the client, so they cannot carry out the forgery attack.

(2) *Resisting Replacing Attack.* According to (2), we suppose  $\text{MAC} = H(m_i)$  ( $1 \leq k \leq c$ ,  $m_i \in C_k$ ), where the MAC is message authentication code, and the MAC of MD5 (Message Digest Algorithm v5) is 128 bit; namely,  $|\text{MAC}| = 128$  and the CSP replace  $m_i$  as  $m'_i$ .

Because the hash function has WCR (Weak Collision Resistance) and SCR (Strong Collision Resistance), thus

$$\Pr[H(m'_i) = H(m_i)] = 2^{-|\text{MAC}|} = 2^{-128}$$

$$\Pr[H(f(m'_i)) = H(m_i)] = 2^{-|\text{MAC}|} = 2^{-128}$$

$$(1 \leq k \leq c, m_i \in C_k), \quad (11)$$

where  $\Pr[x]$  is the probability of  $x$  and  $f(x)$  is a transformation of  $x$ .

Equation (11) shows the probability of the CSP replacing  $m_i$  as  $m'_i$  is very small, so it is almost impossible that the CSP carry out replacing attack.

## 5. Performance Evaluation

In order to solve the problem that the original BLS cannot verify whether the data of the service provider returned is the challenge specified data, MMCDIV scheme adopts the sMHT; meanwhile, the computability and nondegeneracy of verification are obtained by using BLS short signature scheme.

In integrity verification stage,  $e(\text{Sig}_{\text{sk}}(H(R)), g) = e(H(R), v)$  is verified to determine whether CSP returns the original authentication information by comparing the root node  $R$  with  $R'$  which is stored in TTP. Our scheme does not directly use the BLS procedure to calculate  $\delta_i^k$  but uses  $(\{H(m_i)\} \cdot \prod_{j=1}^s u_j^{m_{i,j}})^{\beta}$  to calculate  $\delta_i^k$ . This is done, because the CSP may delete user data, leaving only  $H(m_i)$ ; they are not related when client verifies file blocks, so in case of the label intact, verification can still be carried out. But the actual data blocks may be deleted, so the file blocks must be generated as parameters into the calculation of the signature file blocks. In addition, MMCDIV supports outsourcing verification; client and TTP do not verify any results returned but can directly use, even if TTP and CSP deceit together client, user data and private information will not be leaked to the TTP; therefore  $e(\text{Sig}_{\text{sk}}(H(R)), g) = e(H(R), v)$  and  $e(\delta, g) = e(\pi^{-1} \cdot \prod_{i \in I} H(m_i)^{v_i}, H) e(\prod_{j=1}^s u_j^{m_{i,j}}, v)$  are the guarantee of the TTP's reliability in our MMCDIV scheme.

In client end, client needs to establish complete hash tree operations, signature for nonleaf nodes, and the root filling and uses  $n$  to describe unfilled leaf node (the number of uploaded file blocks). When constructing hash tree,  $(2n - 1)$  times operations need to calculate the hash value. The signature number is  $(n + 1)$ , so total time is  $T_{\text{total}} = (2n - 1) \cdot T_{\text{hash}} + (n + 1)T_{\text{tag}}$ . For CSP, in GenProof stage, when traversing a hash tree, it needs to calculate  $T_{\mu}$  and  $T_{\delta}$ . So total time is  $T_{\text{total}} = c(2^{L+1} - 1) \cdot T_{\text{hash}} + T_{\delta} + T_{\mu}$  and  $c$  is the number of AAI nodes and  $L$  represents the average height of the  $c$  node. Client (or TTP) requires reconstructing root and completing two verifications in verifying proof stage, so total time AAI is  $T_{\text{total}} = (\text{AAI}_{\text{num}} \cdot (2^{L-L'+1} - 1) + c'(2^{L'+1} - 1))T_{\text{hash}} + T_{\text{test1}} + T_{\text{test2}}$ , num is the number of nodes AAI,  $L$  is the hash tree height, and  $L'$  represents the average height of the node where AAI is located.  $C'$  is the number of sibling nodes in the same subtree when challenge appears. Figures 3 and 4 denote the increase trends of server-side program (Intel Core i5 2.5 GHz, 8 G memory, Windows 7 operating system) and the client (armeabi-v7a, Android 2.3.3 system) with the actual running time file size which is 1024 bytes. As shown, when file size is 2 kb, 4 kb, 5 kb, 10 kb, 20 kb, 30 kb, 40 kb, and 50 kb, the constructing authentication information time (CAI-Time) steadily rises with rapid increase of the uploading and processing time (UP-Time), comparing Sheng [9] and Yan scheme [10]; the time complexity of MMCDIV scheme is much lower. In this process, the CAI-Time in client is much lesser than the UP-Time.

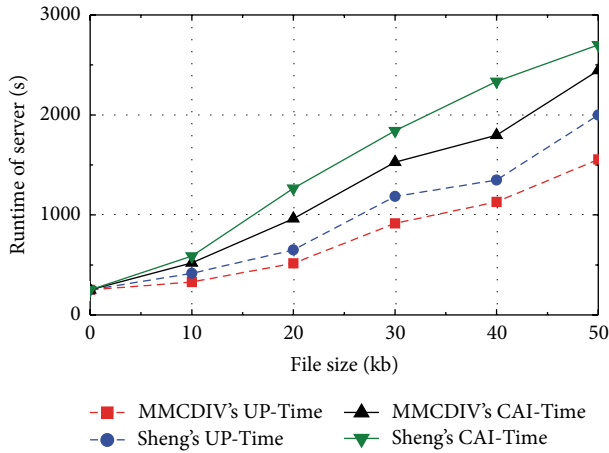


FIGURE 3: Runtime of server.

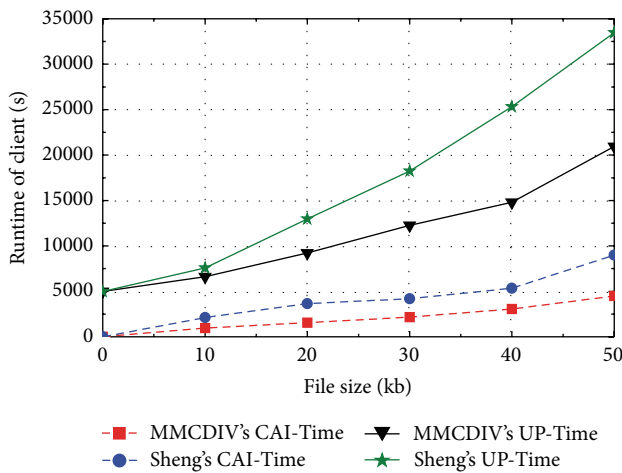


FIGURE 4: Runtime of client.

## 6. Conclusion

We use BLS short signature algorithm, homomorphic response technology, and sMHT to propose a MMCDIV scheme for verification on data integrity in mobile multicloud computing environment. The scheme improves shortage that mobile device communication and computing power are weak. Experimental results also demonstrate that this scheme can achieve a lower cost of computing and communications. At the same time, it also has the following advantages: (1) it supports third-party verification and reduces the computational burden of storage mobile terminal; (2) it is suitable for hybrid cloud environments; (3) it can be applied to verification without source files; no authentication information saved state; (4) it supports dynamic data update operations.

## Competing Interests

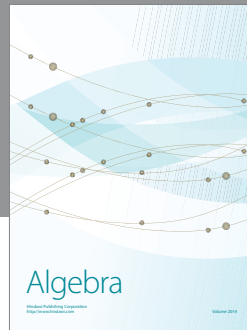
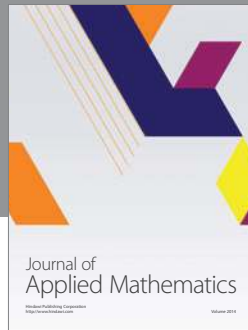

The authors declare that they have no competing interests.

## Acknowledgments

This work was supported by the National Nature Science Foundation of China (no. 61562059 and no. 61461027).

## References

- [1] D.-G. Feng, M. Zhang, Y. Zhang, and Z. Xu, "Study on cloud computing security," *Journal of Software*, vol. 22, no. 1, pp. 71–88, 2011.
- [2] P. Mell and T. Grane, *The NIST Cloud Computing Definition*, NIST Special, New York, NY, USA, 2011.
- [3] J. Yang, H. Wang, J. Wang et al., "Provable data possession of resource constrained mobile devices in cloud computing," *Journal of Networks*, vol. 6, no. 7, pp. 1033–1040, 2011.
- [4] A. N. Khan, M. L. Mat Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: a survey," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1278–1299, 2013.
- [5] Q. Wang, C. Wang, J. Li et al., "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proceedings of the 14th European Conference on Research in Computer Security (ESORICS '09)*, pp. 355–370, Springer, 2009.
- [6] D. L. G. Filho and P. S. L. M. Baretto, "Demonstrating data possession and uncheatable data transfer," in *IACR Eprint Archive*, International Association for Cryptologic Research, 2006.
- [7] A. Juels and B. S. Kaliski Jr., "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 584–597, Alexandria, Va, SA, November 2007.
- [8] G. Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 598–609, ACM, Scottsdale, Ariz, USA, November 2007.
- [9] Z.-D. Shen, C. Lin, and Q. Tong, "A method for lightweight verification on data integrity in mobile cloud computing environment," *Journal of Northeastern University*, vol. 36, no. 11, pp. 1563–1566, 2015.
- [10] L. Yan, R.-H. Shi, H. Zhong, J. Cui, S. Zhang, and Y. Xu, "Integrity checking protocol with identity-based proxy signature in mobile cloud computing," *Journal on Communications*, vol. 36, no. 10, pp. 278–286, 2015.
- [11] Z. Xiulong, *A Data Integrity Verification Scheme Suitable for Mobile Cloud Computing Environment: Design and Prototype Implementation*, JILIN University, Jilin, China, 2014.
- [12] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings*, vol. 2248 of *Lecture Notes in Computer Science*, pp. 514–532, Springer, London, UK, 2001.
- [13] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231–2244, 2012.
- [14] R. Merkle, *Security, Authentication, and Public Key Systems*, UMI Research, Stanford, Calif, USA, 1982.

**Hindawi**

Submit your manuscripts at  
<http://www.hindawi.com>

