SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Public verifiability for shared data in cloud storage with a defense against collusion attacks

WANG ZhongHua[1*], HAN Zhen[1] & LIU JiQiang[1]

[1]*School of Computer and Information Technology, Beijing Jiaotong University, Beijing* 100044, *China*

Appendix A Scheme construction

Appendix A.1 Basic scheme

The proposed scheme consists of five algorithms: **Setup**, **KeyGen**, **ReKeyGen**, **Sign**, **Verify**. Details of the scheme are described in Figure A1.

Appendix A.2 Data modification details

For the MHT demonstrated in Figure A2, m_1's $\Omega_i = \{h(H(m_2)), h(e), h(c)\}$. After receiving the responses from the cloud server, U_j first generates the root R with $\{H(m_i), \Omega_i\}$ to verify the correctness of $sig_{sk}(H(R))$ by checking $e(sig_{sk}(H(R)), g) \overset{?}{=} e(H(R), g^{\pi_A})$. If it's wrong, output $FALSE$; otherwise further computes the new root value based on $\{H(m_i{}'), \Omega_i\}$ and compares it with R'. If it's wrong, output $FALSE$; otherwise U_j computes $sig_{sk_j}(H(R'))$ with U_j's private key, and then sends it to the cloud server for update.

When U_j wants to modify $l(l \geqslant 2)$ data blocks simultaneously, first generates the corresponding signatures $\left\{\delta'_{i_l}\right\}_{i_l \in L}$, where L represents the indices of the modified data blocks, then sends update information $update = \left\{i_l, m_{i_l}{}', \delta_{i_l}{}'\right\}_{i_l \in L}$ to the cloud server. Upon receiving the messages, firstly the cloud server replaces m_{i_l} and δ_{i_l} with $m_{i_l}{}'$ and $\delta_{i_l}{}'$ respectively, then generates the value of new root R' at a time with the modified blocks, lastly sends P_{update}=$\left\{H(m_{i_l}), \Omega_{i_l}, sig_{sk}(H(R)), R'\right\}_{i_l \in L}$ to U_j, where Ω_{i_l} means the node siblings on the path from the leaves $\left\{h(H(m_{i_l}))\right\}_{i_l \in L}$ to the root R of the MHT. For example, as shown in Figure A2, $\left\{m'_1, m'_3, m'_5\right\}$'s $\Omega_{i_l}{}_{i_l \in \{1,3,5\}} = \{h(H(m_2)), h(H(m_4)), h(H(m_6)), h(g)\}$. After receiving the responses from the cloud server, U_j could generate the root R with $\left\{H(m_{i_l}), \Omega_{i_l}\right\}$ and the new root R' with $\left\{H(m'_{i_l}), \Omega_{i_l}\right\}$ only once. Consequently, the computation and communication overheads of U_j and the cloud server could be lowered significantly. Nevertheless, when multi-user intend to modify multi-block, the above method can not be adopted. That's because users may modify the same data block at some point. If the method is used, there will be an inconsistency of the relationship between the shared data and the root R. Most importantly, the revoked user could tamper with data arbitrarily, however, the tampered data is not detected by the TPA and other valid users.

Appendix B Security analysis

As defined in ref. [7], the threat model of our scheme mainly includes two types of attacks, called as external security attack and internal security attack.

Appendix B.1 Security of related works

As presented in letter, ref. [4–6] focus on the problem of integrity verification of the shared data with user revocation. Unfortunately, they have severe security issues when user revocation happens. A detailed analysis could be illustrated as follows.

Ref. [4] achieved the integrity of the shared data by utilizing the idea of proxy re-signatures. When the data owner has revoked a group general user, the cloud server needs to convert the revoked user's signatures into a valid group user's.

* Corresponding author (email: wangzhonghua@bjtu.edu.cn)

Setup. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order p, g be a generator of $\mathbb{G}_1$, e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map, ω be another generator of $\mathbb{G}_1$. The global parameters are ($\mathbb{G}_1$,$\mathbb{G}_2$,e, p, g, ω, H), where H is a hash function with $H: \{0,1\}^* \rightarrow \mathbb{G}_1$. h is a cryptographic hash function. The total number of blocks in the shared data is n, so the shared data could be represented as $M = (m_1, ..., m_n)$. The total number of users in the group is d.

KeyGen. Data owner A chooses $\pi_A \in \mathbb{Z}_p^*$ as private key SK_A randomly, and outputs public key $PK_A = g^{\pi_A}$. Group general user U_j also generates a random $\pi_j \in \mathbb{Z}_p^*$ as U_j's private key SK_j, and outputs public key $PK_j = g^{\pi_j}$, where $j \in \{1, ..., d\}$. A and U_j generate personal secret $\epsilon_l \in \mathbb{Z}_p^*$ and output $g^{\pi_l \epsilon_l}$ respectively, where $l \in \{0, d\}$. Suppose the secret produced by A is ϵ_0, A sends $g^{\pi_A \epsilon_0}$ to group user U_j and the TPA.

ReKeyGen. The cloud server generates a unidirectional re-signing key $rk_{j\rightarrow A}$ from U_j to A as follows: 1) the cloud server generates a random $y \in \mathbb{Z}_p^*$ and sends it to A; 2) A sends $\frac{\pi_A \epsilon_0}{y}$ to U_j; 3) U_j sends $\frac{\pi_A \epsilon_0}{\pi_j \epsilon_j y}$ to the cloud server; 4) the cloud server recovers $rk_{j\rightarrow A} = \frac{\pi_A \epsilon_0}{\pi_j \epsilon_j}$.

Sign. Given private key SK_A and secret ϵ_0, block $m_k \in \mathbb{Z}_p$, where $k \in [1, ..., n]$, A produces the signature on block m_k as: $\delta_k = (H(m_k)\omega^{m_k})^{\pi_A \epsilon_0} \in \mathbb{G}_1$, denotes the set of signatures by $\delta = \{\delta_k\}$. Then A generates a root R based on the construction of MHT, where the leave nodes of the tree are ordered set of BLS hashes of $H(m_k)$. Afterwards, A signs the root R with private key SK_A as: $sig_{sk_A}(H(R)) \leftarrow (H(R))^{\pi_A}$.

Verify. The TPA could verify the integrity of the outsourced shared data stored in the cloud server via a challenge-and-response protocol. The detailed process is as follows:

(1) Challenge message generation. Firstly, the TPA generates g^r, where r is a random in $\mathbb{Z}_p^*$. Next, the TPA picks a random c-element subset I of set [1,n] to locate the c selected random blocks. For each $i \in I$, the TPA chooses a random $v_i \in \mathbb{Z}_q^* \subseteq \mathbb{Z}_p$, where q is a prime. Lastly, the TPA sends $\{i, v_i\}_{i\in I}$ and g^r to the cloud server.

(2) Proof generation. Upon receiving $\{i, v_i\}_{i\in I}$ and g^r, the cloud server computes

$$\mu = \sum_{i\in I} v_i m_i, \quad \phi = \prod_{i\in I} {\phi_i}^{v_i} = \prod_{i\in I} e(\delta_i, g^r)^{v_i} = \prod_{i\in I} e((H(m_i)\omega^{m_i})^{\pi_A \epsilon_0}, g^r)^{v_i} \tag{1}$$

In addition, the cloud server will also provide the TPA with a small amount of auxiliary information $\{\Omega_i\}_{i\in I}$, which are the node siblings on the path from the leaves $\{h(H(m_i))\}_{i\in I}$ to the root R of the MHT. Finally, the cloud server generates proof $\boldsymbol{P}$=$\{\mu, \phi, \{H(m_i), \Omega_i\}_{i\in I}, sig_{sk}(H(R))\}$ and responds the TPA with it.

(3) Verify proof. Based on the responses from the cloud server, at the beginning the TPA generates root R with $\{H(m_i), \Omega_i\}_{i\in I}$ and authenticates $sig_{sk}(H(R))$ by checking:

$$e(sig_{sk_A}(H(R)), g) \overset{?}{=} e(H(R), g^{\pi_A}), \tag{2}$$

if it is failed, the TPA outputs $FALSE$ and reports it to the group users. Otherwise, the TPA checks

$$\phi \overset{?}{=} e(\prod_{i\in I}(H(m_i))^{v_i}\omega^{\mu}, g^{\pi_A \epsilon_0 r}). \tag{3}$$

If it is true, the TPA believes the correctness of the shared data; otherwise the shared data has been corrupted. Finally, the TPA reports the verification result to the group users.

Figure A1 Details of our proposed scheme.

However, as the re-signature key from U_i to U_j is $rk_{i\rightarrow j} = \pi_j/\pi_i$, if the revoked user U_i colludes with the cloud server, they will be able to obtain U_j's private key π_j by computing $\pi_i\pi_j/\pi_i$. Consequently, one of them could tamper with some block m_i to m_i^* and forge the signature $\delta_i^* = (H(id_i)\omega^{m_i^*})^{\pi_j}$ of m_i^*, here H is a hash function, id_i means the block index, ω is a generator of a group with prime order. Then the cloud server replaces the previous block m_i and signature δ_i with them respectively, but the TPA and other valid group users would believe that this operation is performed by U_j, because the signature δ_i^* is generated with U_j's private key.

Ref. [5] achieved the objective that the revoked user U_i and the cloud server could not obtain U_j's private key π_j even if they have collusion. Unfortunately, they are still able to deceive the TPA and other valid group users. Specifically, firstly U_i generates the signature $\delta_i^* = (u^{B_i^*} g^{m_i^* \alpha^2})^{\pi_i}$ of m_i^* with U_i's private key π_i, here B_i^* is a hash value of the block

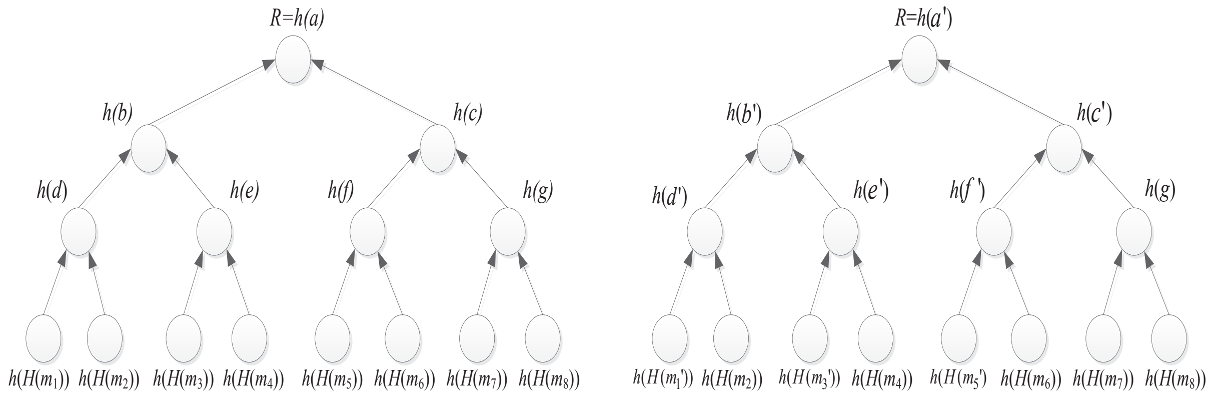

R=h(a)
h(b)
h(c)
h(d)
h(e)
h(f)
h(g)
h(H(m1)) h(H(m2)) h(H(m3)) h(H(m4)) h(H(m5)) h(H(m6)) h(H(m7)) h(H(m8))
R=h(a')
h(b')
h(c')
h(d')
h(e')
h(f')
h(g)
h(H(m1')) h(H(m2)) h(H(m3')) h(H(m4)) h(H(m5') h(H(m6)) h(H(m7)) h(H(m8))

Figure A2 MHT structure.

name and the time stamp, u and g are two generators of a group with prime order, α is a random number. During the verification process, the cloud server generates the proof for the selected signatures including δ_i^* by computing $e(\delta_i^*, g^{\frac{\pi_0 r}{\pi_i}}) = e((u^{B_i^*} g^{m_i^* \alpha^2})^{\pi_i}, g^{\frac{\pi_0 r}{\pi_i}}) = e((u^{B_i^*} g^{m_i^* \alpha^2}), g)^{\pi_0 r}$, which could be viewed as that generated by the master user U_0 (i.e. data owner) with U_0's private key π_0, here r is a random number. As the TPA merely utilizes the final value of proof from the cloud server to verify the integrity of the shared data, regardless of concrete information used for computing the proof, so the TPA could not find that m_i^* has been tampered with.

In ref. [6], the author claimed that secure group user revocation could be achieved with vector commitment and verifier-local revocation group signature. Nevertheless, the revoked user U_i could still forge a signature of the tampered data and pass the verification in the proposed scheme. The detailed explanation is presented as follows. Suppose U_i wants to modify some shared data m_i to m_i^*. Firstly, U_i generates the new commitment C^* with the old commitment C, m_i^* and m_i; next, U_i computes $T_1 = u^\alpha$ and $T_2 = A_i v^\alpha$, where u, v and α are three random numbers, and A_i means the revocation token corresponding to U_i's private key x_i; thirdly, U_i computes the challenge value c with C^*, T_1, T_2 and some random numbers; lastly, U_i outputs the signature of m_i^*, which is produced with T_1, T_2, c, x_i and some random numbers. When the TPA runs the integrity verification algorithm, firstly, the TPA verifies the validity of signature of m_i^* by comparing the received challenge value c with that computed locally. Secondly, by checking $e(T_2/A_i, u) \overset{?}{=} e(T_1, v)$ the TPA decides whether the signer of the signature is a revoked user or not. As the data utilized by the TPA for computing the challenge value c is the same as that utilized by U_i, so the signature of m_i^* is valid. However, as long as U_i modifies T_1 or T_2, $e(T_2/A_i, u) = e(A_i v^\alpha / A_i, u) \neq e(v^\alpha, u) \neq e(u^\alpha, v) \neq e(T_1, v)$. Consequently, the TPA will believe that the signer generating the signature of m_i^* has not been revoked.

Appendix B.2 Security of proposed scheme

Theorem 1. Suppose CDH assumption holds in $\mathbb{G}_1$, this proposed scheme is existentially unforgeable under adaptive chosen message attack (EUF-CMA) in the *Random Oracle* model for an external adversary $\mathcal{A}_{out}$. Concretely, if $\mathcal{A}_{out}$ can generate a forgery of a signature against this proposed scheme with the time of t and advantage of ϵ after making at most q_H hash queries, q_{si} secret generation queries, q_s signing queries, q_{rs} re-signing queries, and requesting at most q_K public keys, then there exists a (t_1, ϵ_1)-algorithm $\mathcal{B}$ that can solve the CDH problem in $\mathbb{G}_1$ with $t_1 \leqslant t + q_H T_{\mathbb{G}_1} + q_s T_{\mathbb{G}_1} + 2q_{rs} T_p$ and $\epsilon_1 \geqslant \epsilon / q_H q_K$. Here $T_{\mathbb{G}_1}$ means one exponentiation time on $\mathbb{G}_1$ takes, T_p means one pairing operation time on $\mathbb{G}_2$ takes.

Theorem 2. Suppose DL assumption holds in $\mathbb{G}_1$, this proposed scheme is existentially unforgeable under adaptive chosen message attack (EUF-CMA) in the *Random Oracle* model for an internal adversary $\mathcal{A}_{in}$. Concretely, if $\mathcal{A}_{in}$ can generate a forgery of root R signature against this proposed scheme with the time of t' and advantage of ϵ' after making at most q_H hash queries, at most q_K public keys queries (includes uncorrupted key generation queries and corrupted key generation queries), q_{si} secret generation queries, q_s signing queries and q_{rk} re-signing key generation queries, then there exists a (t_2, ϵ_2)-algorithm $\mathcal{B}$ that can solve the DL problem in $\mathbb{G}_1$ with $t_2 \leqslant t' + q_s T_{\mathbb{G}_1}$ and $\epsilon_2 \geqslant \frac{1}{q_H}(\epsilon' - \frac{1}{q_K} - \frac{q_s}{2^{|p|}})$.

Proof. Suppose $\mathcal{B}$ is given as input a DL challenge tuple (g, g^a) with unknown $a \in \mathbb{Z}_p^*$, algorithm $\mathcal{B}$'s goal is to output a. algorithm $\mathcal{B}$ acts as the challenger and plays the EUF-CMA game with $\mathcal{A}_{in}$ in the following way.

Setup. Algorithm $\mathcal{B}$ gives public system parameters ($\mathbb{G}_1$,$\mathbb{G}_2$,e, p, g, ω, H) to $\mathcal{A}_{in}$, here Random Oracle H is controlled by $\mathcal{B}$. Algorithm $\mathcal{B}$ also maintains hash list H^{list} and key list K^{list} which are initially empty.

Hash oracle queries $\mathcal{Q}_{hash}$. At any time $\mathcal{A}_{in}$ can issue the random oracle queries H. On receipt of an H query on $X \in \mathbb{Z}_p$, if this query has appeared in the H^{list} with a tuple (X, α), returns the predefined value as the result of the query. Otherwise, chooses $\alpha' \in \mathbb{G}_1$ randomly, then adds the tuple (X, α') into the list H^{list} and responds with $H(X) = \alpha'$.

In this phase, $\mathcal{A}_{in}$ issues a series of queries as in the definition of the EUF-CMA game and $\mathcal{B}$ answers these queries for $\mathcal{A}_{in}$ as follows:

Uncorrupted key generation query (i) $\mathcal{Q}_{Ukeygen}$. $\mathcal{B}$ chooses $x_i \in \mathbb{Z}_p^*$ randomly and defines $pk_i = (g^a)^{x_i}, Y_i = 0$. Next, it adds the tuple (i, pk_i, x_i, Y_i) to K^{list} and returns pk_i to $\mathcal{A}_{in}$.

Corrupted key generation query (i) $\mathcal{Q}_{Ckeygen}$. $\mathcal{B}$ chooses $x_j \in \mathbb{Z}_p^*$ and defines $pk_j = g^{x_j}, Y_j = 1$. Next, it adds the tuple (j, pk_j, x_j, Y_j) to K^{list} and returns (pk_j, x_j) to $\mathcal{A}_{in}$.
Secret generation query (i) $\mathcal{Q}_{secinfo}$. $\mathcal{B}$ retrieves K^{list}, if $Y_i = 1$, $\mathcal{B}$ returns ϵ_i to $\mathcal{A}_{in}$, otherwise returns $a\epsilon_i \in \mathbb{Z}_p^*$ to $\mathcal{A}_{in}$.
Signing query (i) $\mathcal{Q}_{sign}$. Firstly, $\mathcal{A}_{in}$ inputs public key pk_i and message m. Afterwards, $\mathcal{B}$ retrieves K^{list} to obtain the corresponding private key x_i, and checks H^{list} whether it contains a tuple (m, α) or not. If it is true, returns signature $\delta = \alpha^{x_i}$ to $\mathcal{A}_{in}$; otherwise $\mathcal{B}$ chooses $\alpha' \in \mathbb{G}_1$ randomly and adds the tuple (m, α') to H^{list}, finally returns signature $\delta = (\alpha')^{x_i}$ to $\mathcal{A}_{in}$.
Re-signing key generation query (i, j) $\mathcal{Q}_{rekey}$. $\mathcal{B}$ retrieves K^{list}, if $Y_i = Y_j$, $\mathcal{B}$ returns $\frac{x_j\epsilon_j}{x_i\epsilon_i}$ to $\mathcal{A}_{in}$; if $Y_i = 0$, $Y_j = 1$, returns $\frac{ax_j\epsilon_j}{x_i\epsilon_i}$ to $\mathcal{A}_{in}$; if $Y_i = 1$, $Y_j = 0$, returns $\frac{x_j\epsilon_j}{ax_i\epsilon_i}$ to $\mathcal{A}_{in}$.
Forgery. $\mathcal{A}_{in}$ finishes the queries in the phase and outputs a target message m^* and a target public key pk^*. $\mathcal{B}$ responds as follows:

- Recovers tuple (pk^*, x^*, Y^*) from K^{list}, here $Y^* = 0, pk^* = (g^a)^{x^*}$.
- computes $\alpha^* = H(am^*) \in \mathbb{G}_1$ and outputs the target signature $\delta^* = (\alpha^*)^{ax^*}$.

$\mathcal{A}_{in}$ verifies the correctness of the forgery signature δ^* by comparing $e(\delta^*, g)$ with $e(\alpha^*, g^{ax^*})$, $\mathcal{B}$ outputs am^* from H^{list} as a response to the solution to the DL problem in $\mathbb{G}_1$.
Analysis. Next analyse the probability of solving the DL problem in $\mathbb{G}_1$ for $\mathcal{B}$ successfully. The event Ask_{H^*} denotes $\mathcal{A}_{in}$ queries am^* to H, the event Ask_{pubkey} denotes $\mathcal{A}_{in}$ queries pk^* to $\mathcal{Q}_{keygen}$, the event $signErr$ denotes $\mathcal{A}_{in}$ could generate valid signatures without querying H. As long as Ask_{H^*} does not happen, the simulation of H is perfect.

As described before, $\mathcal{A}_{in}$ issues at most q_K public keys queries, so $Pr[Ask_{pubkey}] = 1/q_K$. In the signing simulation, if a signature δ is *valid*, Ask_H denotes $\mathcal{A}_{in}$ queries X to H, we have $Pr[valid|\neg Ask_H] \leqslant \frac{1}{2^{|p|}}$, where $|p|$ denotes the length of element in $\mathbb{G}_1$. $\mathcal{A}_{in}$ issues at most q_s signing queries, so $Pr[signErr] \leqslant \frac{q_s}{2^{|p|}}$. Thus, *succeed* could be represented as the event $Ask_{H^*} \vee Ask_{pubkey} \vee signErr$, which denotes $\mathcal{A}_{in}$ succeeds in forging the target signature δ^*. Then

$$\epsilon' = Pr[succeed] = Pr[Ask_{H^*} \vee Ask_{pubkey} \vee signErr] \leqslant Pr[Ask_{H^*}] + Pr[Ask_{pubkey}] + Pr[signErr],$$

Since $Pr[Ask_{pubkey}] = 1/q_K$ and $Pr[signErr] \leqslant \frac{q_s}{2^{|p|}}$, so we have

$$Pr[Ask_{H^*}] \geqslant \epsilon' - Pr[Ask_{pubkey}] - Pr[signErr] \geqslant \epsilon' - \frac{1}{q_K} - \frac{q_s}{2^{|p|}},$$

Meanwhile, if Ask_{H^*} happens, $\mathcal{B}$ will be able to solve the DL problem in $\mathbb{G}_1$, then

$$\epsilon_2 \geqslant \frac{1}{q_H} Pr[Ask_{H^*}] \geqslant \frac{1}{q_H}(\epsilon' - \frac{1}{q_K} - \frac{q_s}{2^{|p|}}).$$

From the description of the simulation, $\mathcal{B}$ requires one extra exponentiation on $\mathbb{G}_1$ for each $\mathcal{Q}_{sign}$, so its running time t_2 is $\mathcal{A}_{in}$'s running time t' plus $q_s T_{\mathbb{G}_1}$.

According to what we discussed above, we give the security analyses of external attack and internal attack with Theorem 1 and Theorem 2 respectively. Specifically, as shown in Theorem 1, if the advantage of an external adversary(i.e. an attacker outside the system) for generating a forgery of a valid signature under the external security game is non-negligible, then we can find an algorithm to solve the CDH problem in $\mathbb{G}_1$ with a non-negligible probability($t_1 \leqslant t + q_H T_{\mathbb{G}_1} + q_s T_{\mathbb{G}_1} + 2q_{rs}T_p$, $\epsilon_1 \geqslant \epsilon/q_H q_K$), which contradicts to the assumption that the CDH holds in $\mathbb{G}_1$. Similarly, as stated in Theorem 2, if the advantage of an internal adversary(i.e. a revoked user) for generating a forgery of root R signature under the internal security game is non-negligible, then we can find an algorithm to solve the DL problem in $\mathbb{G}_1$ with a non-negligible probability($t_2 \leqslant t' + q_s T_{\mathbb{G}_1}$, $\epsilon_2 \geqslant \frac{1}{q_H}(\epsilon' - \frac{1}{q_K} - \frac{q_s}{2^{|p|}})$), which contradicts to the assumption that the DL holds in $\mathbb{G}_1$. Therefore, it is computationally infeasible to generate a forgery of a signature in our proposed scheme under the CDH and the DL assumptions. □

Appendix C Overhead analysis

We use $Exp_{\mathbb{G}_1}$ and $Mul_{\mathbb{G}_1}$ to denote the complexity of one exponentiation operation and one multiplication operation on group $\mathbb{G}_1$ respectively, use $Exp_{\mathbb{G}_2}$ and $Mul_{\mathbb{G}_2}$ to denote the complexity of one exponentiation operation and one multiplication operation on group $\mathbb{G}_2$ respectively, use $Hash_{\mathbb{G}_1}$ to denote the complexity of one hash operation on group $\mathbb{G}_1$, use *Pairing* to denote one pairing operation on $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, use n to denote the number of the whole shared data blocks, use c to denote the number of the shared data selected by the TPA, use η to denote the number of group users that modify the selected shared data, use Y to denote the number of data blocks last modified by the revoked users.

Appendix C.1 Computation overhead

Appendix C.1.1 *Computation overhead of related works*

To verify the integrity of the selected shared data, ref. [1] costs the TPA $2Pairing + Exp_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + cF_{prp}$ operations, where only F_{prp} operation is related to the number of the selected shared data. That's because the identifier of every selected shared data needs performing this operation. For the cloud server, not only every selected shared data but also the corresponding signatures need performing $Exp_{\mathbb{G}_1}$ and $Mul_{\mathbb{G}_1}$ operations.

In ref. [4], when the TPA verifies the integrity of the shared data uploaded by the data owner, the TPA first performs

$cHash_{\mathbb{G}_1} + cExp_{\mathbb{G}_1} + (c-1)Mul_{\mathbb{G}_1}$ operations to generate the proof related to the identifier of every selected shared data, then the TPA multiplies the result with the shared data block proof from the cloud server. Besides the block proof, the cloud server also performs $cExp_{\mathbb{G}_1} + (c-1)Mul_{\mathbb{G}_1}$ operations to generate the proof for the signatures. Finally, the TPA outputs the verification result by $2pairing$ operations. When the data owner revokes some group users, the cloud server only re-signs the signatures generated by the revoked users, the total cost of proof generation is the same as before. However, the TPA needs to consider the number of group users modifying the selected shared data, which has an effect on the proof generated by the cloud server, because the number of the subproof is proportionate to it.

In the verification process described in ref. [5], the TPA transfers plenty of complex operations on group $\mathbb{G}_1$ to the cloud server. When the data owner revokes some group users, the TPA performs $\eta Exp_{\mathbb{G}_1}$ operations to generate a challenge set for every group user modifying the selected shared data. Besides generating the proof, the cloud server still performs $Exp_{\mathbb{G}_1}$ operation for every selected data block last modified by the revoked users.

In ref. [6], at the beginning, the TPA verifies the correctness of signature produced with the shared data commitment and some group user's private key, which only costs the TPA constant times operations, but the verification cost of the selected shared data is linear in its number. The cloud server performs $(c-1)Exp_{\mathbb{G}_1} + (c-2)Mul_{\mathbb{G}_1}$ operations to generate the proof for every selected shared data. For the user revocation, the TPA performs more $2pairing$ operations on $\mathbb{G}_2$ and one division operation on $\mathbb{G}_1$ to authenticate the identity of the signer, the cloud server has no changes.

Appendix C.1.2 *Computation overhead of our scheme*

As shown in Section Appendix A, when the TPA verifies the shared data uploaded by the data owner initially, the TPA performs one $Exp_{\mathbb{G}_1}$ operation to generate the challenge message at the beginning, and then the cloud server performs $c(Pairing{+}Exp_{\mathbb{G}_2}){+}(c\text{-}1)Mul_{\mathbb{G}_2}$ operations to generate the proof information. To verify the integrity of the shared data, the TPA needs to perform $c(\log_2 n)Hash_{\mathbb{G}_1}{+}3Pairing{+}(c{+}2)Exp_{\mathbb{G}_1}{+}cMul_{\mathbb{G}_1}$ operations. To revoke a group user, the data owner A only needs to perform one $Exp_{\mathbb{G}_1}$ operation to generate the information used for verifying data integrity. If the valid group users or A wants to verify the integrity of the shared data after revoking a group user, firstly the TPA performs $2Exp_{\mathbb{G}_1}$ operations to generate the challenge message, and then the cloud server conducts at most $c(Pairing{+}Exp_{\mathbb{G}_2}{+}Exp_{\mathbb{G}_1}){+}(c\text{-}1)Mul_{\mathbb{G}_2}$ operations to generate the proof information, lastly the TPA performs the same operations as before to verify the integrity of the shared data.

Appendix C.1.3 *Comparison and analysis*

Now firstly we compare our proposed scheme with the ref. [1,4–6] in terms of computation overhead, the result is summarized in Table C1. As shown in Table C1, as the cloud server in ref. [6] needs to performs $c(c-1)Exp_{\mathbb{G}_1}+c(c-2)Mul_{\mathbb{G}_1}$ operations for the selected data, obviously, our proposed scheme is better than ref. [6]. However, compared with ref. [1,4,5], our proposed scheme may be not perfect in the total computation overheads of the TPA and the cloud server, the reason is that the TPA in our proposed scheme needs to perform $c(\log_2 n)Hash_{\mathbb{G}_1}$ operations to verify the value of root R in MHT based on the selected data(as illustrated in Section Appendix D.1, the computation time of $Hash_{\mathbb{G}_1}$ is the most longest one, which is about 2.7 times as long as that of $Pairing$). Nevertheless, it should be noted that the extra computation is necessary to achieve the correctness of integrity verification of the shared data, especially in the case of data modification and user revocation. Generally, in the process of data sharing, the group users modify the shared data frequently and the data owner often revokes the group users. Thus, the TPA rarely verify the integrity of the shared data which is uploaded by the data owner initially. Furthermore, when the TPA runs the verification process in the case of user revocation, η plays an important role in the process of ref. [4, 5], however, which is neglected in our proposed scheme. Besides, the verification time of the TPA in our scheme is related to the value of n. On the assumption of keeping c unchanged, if the size of each data block is increased in our proposed scheme, the verification cost of our scheme will be reduced significantly. So the extra computation of the verification process is not a severe issue in the cases of secure user revocation and data block of large size.

When a group user U_j modifies a data block, to prevent U_j tamper with the shared data maliciously without being detected by the TPA, U_j and the cloud server need to perform a series of operations containing data computing and information transmission. Thus, with a gradual increasement of the number of data blocks modified by the group user, it will bring a heavy overhead of computation and communication to the user and the cloud server. Consequently, we have to admit that it has an important effect on the proposed scheme for data modification. However, we could adopt a method to reduce the overhead of computation and communication. In particular, as described in Section Appendix A.2, suppose a group user U_j intends to modify l blocks simultaneously, firstly U_j needs to perform $l(2Exp_{\mathbb{G}_1}{+}Hash_{\mathbb{G}_1}{+}mul_{\mathbb{G}_1})$ operations to generate the signatures of new data blocks; afterwards, the cloud server merely performs $(\log_2 n{+}1)Hash_{\mathbb{G}_1}$ operations to generate the response to the update information; in the end, U_j performs $2(\log_2 n{+}1)Hash_{\mathbb{G}_1}{+}2Pairing{+}Exp_{\mathbb{G}_1}$ operations to check the value of new root R'. If we use the method of single data block modification, the cloud server will perform $l(\log_2 n{+}1)Hash_{\mathbb{G}_1}$ operations to generate the response totally, and U_j needs to perform $2l(\log_2 n{+}1)Hash_{\mathbb{G}_1}{+}2Pairing{+}Exp_{\mathbb{G}_1}$ operations to check the value of R'. Therefore, with the method of a single user updating multi-block discussed in Section Appendix A.2, the computation efficiency of the user and the cloud server can be improved drastically. Nevertheless, it should be noted that this method is not appropriate to the case of multi-user updating the data blocks.

$Fprp$ denotes a pseudorandom permutation in ref. [1], n is the number of the shared data blocks, $*$ denotes the operation actually is one division operation on $\mathbb{G}_1$ or $\mathbb{G}_2$(could be viewed as $Mul_{\mathbb{G}_1}$ and $Mul_{\mathbb{G}_2}$), $\varnothing$ denotes user revocation is not supported in ref. [1].

Table C1 Comparison between our scheme and ref. [1, 4–6] in computation overhead

	Basic scheme		Scheme with user revocation	
	Comp.cost(TPA)	Comp.cost(cloud server)	Comp.cost(TPA)	Comp.cost(cloud server)
Ref. [1]	$2Pairing+Exp_{\mathbb{G}_1}$ $+Mul_{\mathbb{G}_1}+cF_{prp}$	$2cExp_{\mathbb{G}_1}$ $+2(c\text{-}1)Mul_{\mathbb{G}_1}$	$\varnothing$	$\varnothing$
Ref. [4]	$(c+1)Exp_{\mathbb{G}_1}$ $+c(Mul_{\mathbb{G}_1}+Hash_{\mathbb{G}_1})$ $+2pairing$	$cExp_{\mathbb{G}_1}+(c\text{-}1)Mul_{\mathbb{G}_1}$	$(\eta-1)(Mul_{\mathbb{G}_1}+Mul_{\mathbb{G}_2})$ $+(\eta+1)pairing$ $+(c+1)Exp_{\mathbb{G}_1}$ $+c(Mul_{\mathbb{G}_1}+Hash_{\mathbb{G}_1})$	$YExp_{\mathbb{G}_1}$ $+cExp_{\mathbb{G}_1}+(c\text{-}1)Mul_{\mathbb{G}_1}$
Ref. [5]	$6Exp_{\mathbb{G}_1}+3Pairing$ $+2Mul_{\mathbb{G}_2}+Mul_{\mathbb{G}_1}$	$c(Exp_{\mathbb{G}_1}+Exp_{\mathbb{Z}_q})$ $+(c\text{-}1)(Mul_{\mathbb{G}_1}+Mul_{\mathbb{G}_2})$ $+c(Pairing+Exp_{\mathbb{G}_2})$	$(\eta+7)Exp_{\mathbb{G}_1}+Mul_{\mathbb{G}_1}$ $+3Pairing+2Mul_{\mathbb{G}_2}$	$(Y+c)Exp_{\mathbb{G}_1}+cExp_{\mathbb{Z}_q}$ $+(c\text{-}1)(Mul_{\mathbb{G}_1}+Mul_{\mathbb{G}_2})$ $+c(Pairing+Exp_{\mathbb{G}_2})$
Ref. [6]	$(c+5)Exp_{\mathbb{G}_1}+4Exp_{\mathbb{G}_2}$ $+(2c+5)pairing$ $+(c^*+1)Mul_{\mathbb{G}_1}$ $+3Mul_{\mathbb{G}_2}+Mul_{\mathbb{G}_2}{}^*$	$c(c-1)Exp_{\mathbb{G}_1}$ $+c(c-2)Mul_{\mathbb{G}_1}$	$(c+5)Exp_{\mathbb{G}_1}+4Exp_{\mathbb{G}_2}$ $+(2c+7)pairing$ $+(c+1)^*Mul_{\mathbb{G}_1}+Mul_{\mathbb{G}_1}$ $+3Mul_{\mathbb{G}_2}+Mul_{\mathbb{G}_2}{}^*$	$c(c-1)Exp_{\mathbb{G}_1}$ $+c(c-2)Mul_{\mathbb{G}_1}$
Our scheme	$c(\log_2 n)Hash_{\mathbb{G}_1}$ $+(c+3)Exp_{\mathbb{G}_1}$ $+3Pairing+cMul_{\mathbb{G}_1}$	$c(Pairing+Exp_{\mathbb{G}_2})$ $+(c\text{-}1)Mul_{\mathbb{G}_2}$	$c(\log_2 n)Hash_{\mathbb{G}_1}$ $+(c+4)Exp_{\mathbb{G}_1}$ $+3Pairing+cMul_{\mathbb{G}_1}$	$c(Pairing+Exp_{\mathbb{G}_2})$ $+YExp_{\mathbb{G}_1}+(c\text{-}1)Mul_{\mathbb{G}_2}$

Appendix C.2 Communication overhead

Appendix C.2.1 *Communication overhead of related works*

In ref. [1], when the TPA verifies the integrity of the shared data, the challenge message contains $\{i, v_i\}$, whose total length is $c(|n|+|q|)$, $|n|$ is the size of an element of set [1,n], $|q|$ is the size of an element of $\mathbb{Z}_q$. The length of proof information from the cloud server is $2|p|$, where $|p|$ is the size of an element of $\mathbb{Z}_p$.

In ref. [4], the length of challenge message from the TPA is $c(|n|+|q|)$, the length of proof from the cloud server is $2|p|$; when some group users are revoked, in addition to the proofs for the selected data block and the signatures, the cloud server also sends the block identifiers, so the length of the total proof is $2\eta|p|+c|id|$, where $|id|$ is the size of a block identifier.

In ref. [5], the communication cost for the integrity verification mainly comes from the *log* records, the challenge message and the proof information. To verify the integrity of the selected shared data, the length of challenge message from the TPA is $c|n|+|p|+|q|$, the proof generated by the cloud server contains a random set based on the indices of the select blocks and the computation results related to the data blocks and the signatures, so the length is $(c+3)|p|$. When some group users are revoked, the length of challenge message is $c(|n|+|log|)+(\eta+2)|p|+|q|$, where $|log|$ is the length of a log file used for recording the event of updating data. The length of proof generated by the cloud server is still $(c+3)|p|$.

In ref. [6], the challenge message contains the indices of the selected data blocks and several random numbers, whose length is $c|n|+8|p|$; the cloud server generates the corresponding proof based on the indices, which includes every committed message and auxiliary information, whose length is $(2c+9)|p|$. When some group users are revoked, the communication overheads of the TPA and the cloud server remains unchanged.

Appendix C.2.2 *Communication overhead of our scheme*

In this proposed scheme, the communication cost for the integrity verification mainly refers to the challenge message generated by the TPA and the proof information produced by the cloud server. As shown in Section Appendix A, the challenge message of basic scheme contains $\{i, v_i\}$ and g^r, whose total length is $c(|n|+|q|)+|p|$, where $|n|$ is the size of an element of set [1,n], $|q|$ is the size of an element of $\mathbb{Z}_q$, $|p|$ is the size of an element of $\mathbb{Z}_p$; the proof information $\boldsymbol{P}=\{\mu, \phi, \{H(m_i), \Omega_i\}_{i\in I}, sig_{sk}(H(R))\}$, whose total length is $3|p|+c(|p|+\log_2 n|p|)$. Considering the scheme supporting user revocation in the letter, the challenge message contains $\{i, v_i\}$, $g^{r'}$ and $g^{(\frac{\epsilon_0+\rho}{\epsilon_0})r'}$, whose total length is $c(|n|+|q|)+2|p|$; the length of the proof information is the same as that of the basic scheme.

$|log|$ denotes the length of the log file used for recording data update in ref. [5], $|id|$ denotes the size of a block identifier in ref. [4].

Table C2 Comparison between our scheme and ref. [1, 4–6] in communication overhead

	Basic scheme		Scheme with user revocation	
	Comm.cost(TPA)	Comm.cost(cloud server)	Comm.cost(TPA)	Comm.cost(cloud server)
Ref. [1]	$c(\|n\|+\|q\|)$	$2\|p\|$	$\varnothing$	$\varnothing$
Ref. [4]	$c(\|n\|+\|q\|)$	$2\|p\|$	$c(\|n\|+\|q\|)$	$2\eta\|p\|+c\|id\|$
Ref. [5]	$c\|n\|+\|p\|+\|q\|$	$(c+3)\|p\|$	$c(\|n\|+\|log\|)$ $+(\eta+2)\|p\|+\|q\|$	$(c+3)\|p\|$
Ref. [6]	$c\|n\|+8\|p\|$	$(2c+9)\|p\|$	$c\|n\|+8\|p\|$	$(2c+9)\|p\|$
Our scheme	$c(\|n\|+\|q\|)+\|p\|$	$3\|p\|+c(\|p\|+\log_2 n\|p\|)$	$c(\|n\|+\|q\|)+2\|p\|$	$3\|p\|+c(\|p\|+\log_2 n\|p\|)$

Appendix C.2.3 *Comparison and analysis*

Now we compare our proposed scheme with ref. [1, 4–6] in terms of communication overhead, the result is summarized in Table C2. As shown in Table C2, compared with ref. [1, 4–6], the increased communication overhead of our proposed scheme is mainly the size of verification information related to MHT, which is an important shortcoming of this scheme. However, ref. [4–6] have the common security problem caused by the collusion between the revoked user and the cloud server, to make secure user revocation, our scheme needs the extra communication overhead. In addition, as illustrated in Section Appendix D, an element of $\mathbb{G}_1$ and $\mathbb{Z}_p$ is $|p|$=160 bits, assume the size of the shared file is 1GB, the size of each block is 1MB, the extra communication overhead in our scheme is $c\log_2 n|p| \approx$ 90KB. Compared with the block size, it is negligible. Meanwhile, with an increasement of the size of the shared data block, the extra communication overhead in our proposed scheme could be reduced significantly. So the extra communication overhead of the verification process is not a severe issue in the cases of secure user revocation and data block of large size.

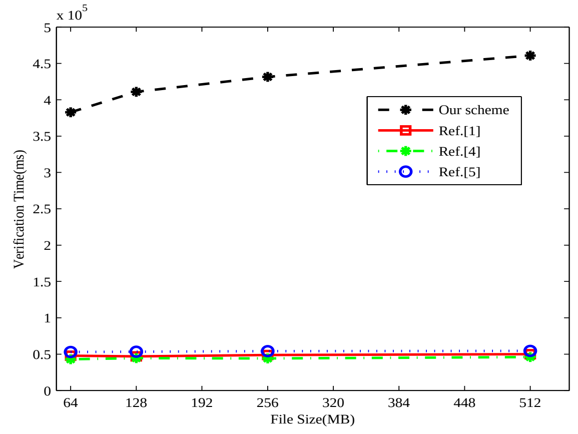
when some group user U_j modifies the shared data, to prevent U_j from tampering with the shared data maliciously without being detected by the TPA, U_j and the cloud server need to perform the process of related information transmission. As presented in Section Appendix C.1.3, we could employ a method to reduce the communication overhead when the group user intends to modify multi-block simultaneously. In particular, as described in Section Appendix A.2, suppose a group user U_j intends to modify l blocks simultaneously, first of all, the length of update information produced by U_j is $l(2|p|+|m|)$, where $|m|$ is the length of every data block. Secondly, the length of response from the cloud server is $(n+2)|p|$ at the most. Suppose the set of leaf nodes corresponding to the l blocks is represented as $Node_l$, $(n+2)|p|$ means that every element in $Node_l$ is any one of two leaf nodes, which have the same parent node. For example, as depicted in Figure A2, $Node_l$=$\{h(H(m_1)), h(H(m_3)), h(H(m_5)), h(H(m_7))\}$. At last, U_j needs to send $sig_{sk_j}(H(R'))$ to the cloud server, whose length is only $|p|$. However, if we use the method of single data block modification, the total length of information produced by U_j is $l(3|p|+|m|)$, and the length of response from the cloud server is $l(\log_2 n|p|+3|p|)$. When $l=\frac{n}{2}$, the length of response of the former method has its maximum value, so the total communication cost of U_j and the cloud server is $(2n+3)|p|+\frac{n}{2}|m|$. However, the total communication cost of the latter method is $(3n+\frac{n}{2}\log_2 n)|p|+\frac{n}{2}|m|$. Obviously, with the method of a single user updating multi-block discussed in Section Appendix A.2, the total communication cost of the user and the cloud server could be reduced significantly. Nevertheless, as discussed in Section Appendix C.1.3, this method is not appropriate to the case of multi-user updating the shared data.

Appendix D Experimental evaluation

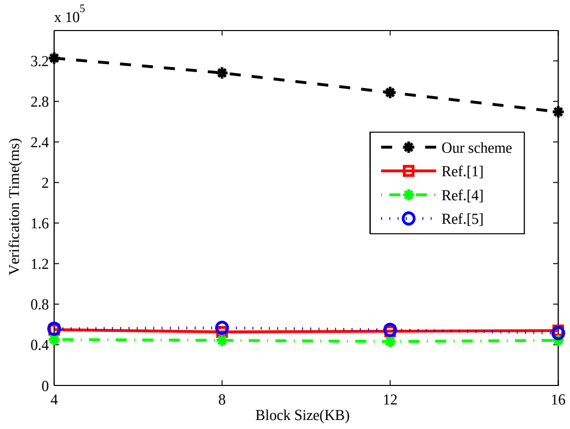
We evaluate the performance of our proposed scheme and the schemes proposed in ref. [1, 4–6]. We utilize java Pairing Based Cryptography (jPBC 1.2.1) library to implement cryptographic operations, and utilize Hadoop Distributed FileSystem (HDFS) and MapReduce to implement upload/ download operations for file blocks and proof generation respectively, which come from a famous open source cloud computing system hadoop2.2.0 64bit. All the experiments are tested under ubuntu with an Intel(R) Xeon(R) E5620 @2.40GHz processor and 64GB memory over 50 times. In the following experiments, we assume the size of an element of $\mathbb{G}_1$ and $\mathbb{Z}_p$ is $|p|$=160 bits, the size of an element of $\mathbb{Z}_q$ is $|q|$=80 bits, the size of each block is 4KB. For the simplicity of the construction of MHT and evaluating the influence of file size on the verification performance, the size of the selected shared files is 64MB, 128MB, 256MB and 512MB. To evaluate the influence of group size, we varies the revoked user number from 50 to 450.

Appendix D.1 Experimental evaluation on basic scheme

Next we give the comparison of verification time cost of the basic scheme at first. As shown in Figure D1, after the data owner uploads the shared data to the cloud server initially, when the TPA runs the verification process with the cloud server, the computation cost for them in ref. [1,4,5] is almost equal, the reason is that the computation cost is only linear in the number of the selected shared data, regardless of the shared files size. As shown in Table C1, the cloud server in ref. [6] performs $c(c-1)Exp_{\mathbb{G}_1}+c(c-2)Mul_{\mathbb{G}_1}$ operations to generate the proof for the selected shared data, so if we make the experiment evaluation for ref. [6] with the same number of the selected shared data, the experimental result is very imperfect. Based on the experiment, when the selected block number c is 460 and the selected shared file size is 64MB, the time of verification process in ref. [6] is the same as that of our proposed scheme with $c \approx 128$. So the experiment results related to

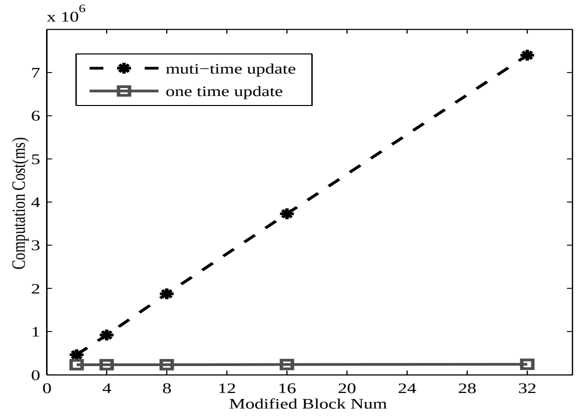


(a) Block Size=4KB, c=460

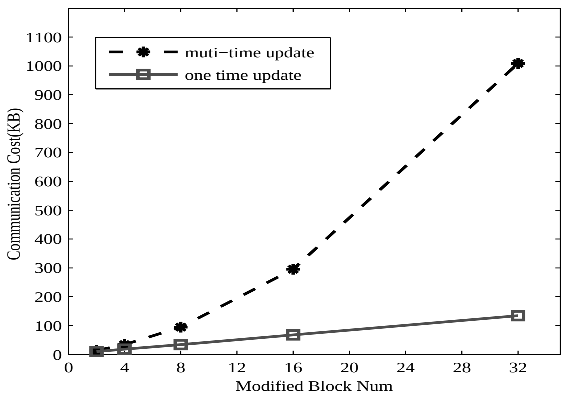


(b) File Size=8MB, c=460

Figure D1 Computation cost involved in the TPA and the cloud server in basic scheme. (a)data block of the same size; (b)data blocks of different size.



(c) File Size=8MB, Block Size=4KB



(d) File Size=8MB, Block Size=4KB

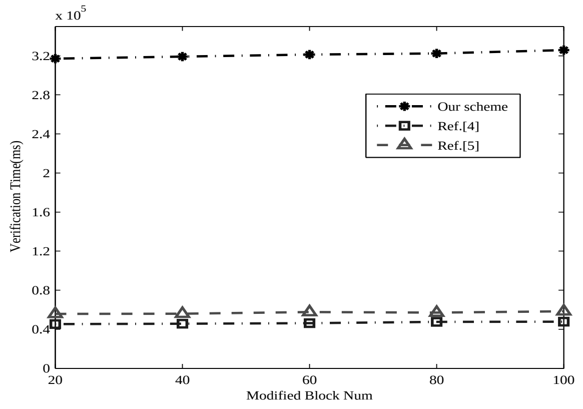
Figure D2 Performance cost involved in multi-block modification. (a) Computation cost; (b) Communication cost.

ref. [6] is not shown here and in Section Appendix D.2. With regard to our scheme, in Figure D1(a), the verification time is about 7 to 9 times as long as that of ref. [1,4,5]. The reason is that the TPA needs to perform a lot of $Hash_{\mathbb{G}_1}$ operations to verify the value of root R in MHT based on the selected data blocks. Besides, the computation time of $Hash_{\mathbb{G}_1}$ and $Pairing$ is more than that of $Exp_{\mathbb{G}_1}$,$Mul_{\mathbb{G}_1}$,$Exp_{\mathbb{G}_2}$ and $Mul_{\mathbb{G}_2}$(the computation time of $Exp_{\mathbb{G}_1}$,$Mul_{\mathbb{G}_1}$,$Exp_{\mathbb{G}_2}$,$Mul_{\mathbb{G}_2}$,$Hash_{\mathbb{G}_1}$ and $Pairing$ is about 18ms,0.1ms,2ms,14ms,57ms and 21ms in our experiments). However, the computation cost of this scheme is not only related to the number of the selected shared data, but also the size of the shared data blocks. So if the size is increased, the verification time will be shortened. As depicted in Figure D1(b), with the size of the data block increased, the computation cost in our scheme could be declined significantly. Therefore, the results shown in Figure D1 has confirmed the analysis in Section Appendix C.1.3.

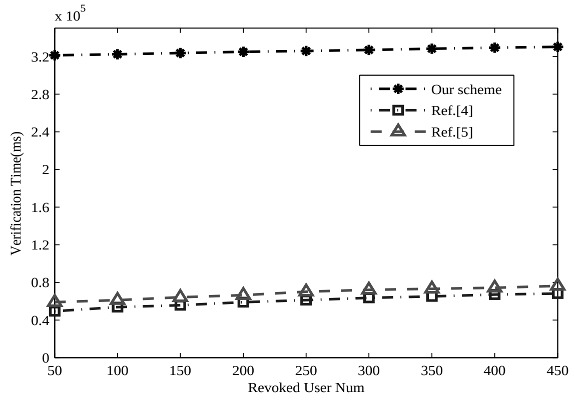
Appendix D.2 Experimental evaluation on dynamic scheme

In this section, to begin with, the experiment result of data modification process is demonstrated. As shown in Figure D2, when single group user modifies multi-block, the costs of computation and communication are depicted in Figure D2(a) and Figure D2(b) respectively. It can be seen that the costs of the user and the cloud server in the data modification process could be reduced significantly with the method proposed in Section Appendix A.2. Thus, the results have confirmed the analyses in Section Appendix C.1.3 and Section Appendix C.2.3.

Next we show the results of verification cost in the case of single revoked user modifying multi-block between ref. [4,5] and our proposed scheme. Afterwards, we give the experiment analysis of verification cost in the case of multi-user modifying multi-block, where each group user is supposed to modify different data blocks. As shown in Figure D3(a), after the data owner revokes a group user, the running time of integrity verification process is related to the number of the shared data modified by the revoked user. In particular, as Shown in Table C1, when the TPA intends to check the integrity of the shared data in the case of one user revocation, the cloud server needs to perform the extra operation($YExp_{\mathbb{G}_1}$) between ref. [4, 5] and our proposed scheme. Consequently, with the increasement of the number of the shared data modified by the revoked user, the verification time of ref. [4, 5] and our proposed scheme will get longer. However, it should be noticed



(e) File Size = 8MB, Block Size=4KB, c=460



(f) File Size = 8MB, Block Size=4KB, c=460

Figure D3 Computation cost involved in the TPA and the cloud server in dynamic scheme. (a) Single revoked user modifies multi-block; (b) multi-user modify multi-block.

that the verification time in our proposed scheme is the longest, which is a problem we have to face. When the data owner revokes multi-user, the running time of integrity verification process is demonstrated in Figure D3(b). Obviously, the running time of ref. [4, 5] and our proposed scheme will still grow longer with the increasement of the number of revoked users. Nevertheless, the growth rate of our proposed scheme is the least(from 50 users to 450 users, our growth rate is 2.7%, ref. [4] and ref. [5] are 38% and 29% respectively). As analyzed in Section Appendix C.1.3, the number of revoked users is not related to the verification time in our scheme. Therefore, we believe that the verification time in ref. [4, 5] will grows rapidly with the increasement of the number of revoked users.

References

1 Kwon O, Koo D Y, Shin Y J, et al. A secure and efficient audit mechanism for dynamic shared data in cloud storage. The Scientific World Journal, 2014, 2014: 1-16

2 Wang B Y, Li B C, Li H. Oruta: privacy-preserving public auditing for shared data in the cloud. IEEE Transactions on Cloud Computing, 2014, 2: 43-56

3 Wang B Y, Li B C, Li H. Knox: privacy-preserving auditing for shared data with large groups in the cloud. In: Proceedings of ACNS, Singapore, 2012. 507-525

4 Wang B Y, Li B C, Li H. Panda: public auditing for shared data with efficient user revocation in the cloud. IEEE Transactions on Services Computing, 2013, 8: 92-106

5 Yuan J W, Yu S C. Public integrity auditing for dynamic data sharing with multi-user modification. In: Proceedings of IEEE INFOCOM, Toronto, 2014. 2121-2129

6 Jiang T, Chen X F, Ma J F. Public integrity auditing for shared dynamic cloud data with group user revocation. IEEE Transactions on Computers, 2015.

7 Ateniese G, Hohenberger S. Proxy re-signatures: new definitions, algorithms and applications. In: Proceedings of ACM CCS, New York, 2005. 310-319