SCIENCE CHINA
Information Sciences

• Supplementary File •

A blockchain-based data auditing scheme with key-exposure resistance for IIoT

Pan YANG[1] & Jingli REN[1*]

[1]*Henan Academy of Big Data, Zhengzhou University, Zhengzhou* 450052, *China*

Appendix A Preliminaries

The notations used in this paper are defined in Table A1.

Table A1 Notations

p	Large prime number	$SSig$	Signature algorithm
Z_p^*	$\{1,2,\cdots,p-1\}$	(ssk, spk)	Signing secret/public key pair
G_1, G_2	$p-$order multiplicative cyclic groups	(x, R)	The private/public key of the DO
$e: G_1 \times G_1 \to G_2$	The bilinear pairing	(β, g^β)	The private/public key of the MN
g, u	Generators of G_1	$para, v$	Public parameter
$H_1: \{0,1\}^l \to G_1$	Secure map-to-point hash function	$m_{i,j}$	Outsourced data blocks
$H_2: \{0,1\}^* \to G_1$	Secure map-to-point hash function	S	Addition on Z_p^*
$H_3: G_1 \to Z_p^*$	Full domain hash function	$M_{Z_p^*}, M_{G_1}, M_{G_2}$	Multiplication on Z_p^*, G_1, G_2
$f, h: \{0,1\}^* \to Z_p^*$	Hash function	E_{G_1} and E_{G_2}	Exponentiation on G_1, G_2
$\mathcal{F}: Z_p^* \times \{1,2,\cdots,n\} \to \{1,2,\cdots,n\}$	Pseudorandom permutation	$\mathcal{P}$	Pairing
n	Number of total blocks	H	Map-to-point hash operation
c	Number of challenged blocks	$\|p\|$	Element length in Z_p^*
s	Number of sectors	$\|G_1\|, \|G_2\|$	Element length in G_1, G_2

We recall some cryptographic primitives and security assumptions used in our scheme.

Bilinear map. Let G_1, G_2 be two multiplication cyclic groups of order p, p being a large prime integer. A bilinear mapping is defined as $e: G_1 \times G_1 \to G_2$, if it satisfies the following properties:

a) Bilinearity. For $\forall a, b \in Z_p^*$, $u, v \in G_1$, $e(u^a, v^b) = e(u, v)^{ab}$.

b) Non-degeneracy. There exist two generators $u, v \in G_1 (u \neq v)$, such that $e(u, v) \neq 1$.

c) Computability. There exists a efficient probabilistic polynomial time (PPT) algorithm to calculate $e(u, v)$ for $\forall u, v \in G_1$.

Computational Diffie-Hellman (CDH) assumption. Let G_1 be a multiplication cyclic group with order p, and g^a, g^b be two random elements in G_1 with unknown $a, b \in Z_p^*$. It is computationally intractable to calculate g^{ab} for any PPT algorithm, and the advantage ϵ of algorithm $\mathcal{A}$ in solving CDH problem is negligible, which is defined as follows:

$$Adv_{\mathcal{A}}^{CDH} = Pr[g^{ab} \leftarrow \mathcal{A}(g, g^a, g^b)] \leqslant \epsilon.$$

Discrete Logarithm (DL) assumption. Let G_1 be a multiplication cyclic group with order p, and g, g^a be two random elements in G_1 with an unknown $a \in Z_p^*$. It is computationally intractable to calculate a by any PPT algorithm, and the advantage ϵ of algorithm $\mathcal{A}$ in solving DL problem is negligible. It is defined as follows:

$$Adv_{\mathcal{A}}^{DL} = Pr[a \leftarrow \mathcal{A}(g, g^a)] \leqslant \epsilon.$$

* Corresponding author (email: renjl@zzu.edu.cn)

Appendix B Workflow and deployment of BAKER

We assume that the CSP has no incentive to expose its hosted data to external entities and has the ability to forge a proof to pass verification. The DO is semi-trusted and may deny the auditing result. The verifier has the ability to check the integrity of the data for the DO using the proof from CSP, but it may be curious about the original data. The CSP has responsibility to generate the random challenge. In order to guarantee the unpredictability and randomness of the challenge, the CSP periodically traverses the auditing tasks of all DOs. When a new auditing task is found, the CSP uses the public information τ as a random seed to generate the challenge. Note that τ consists of the timestamp and parameters of the current block header, which cannot be controlled by the CSP [4].

Appendix B.1 Workflow

Figure B1 shows the working process of BAKER.

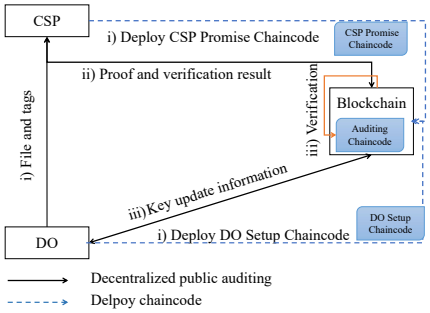

Figure B1 (Color online) Working process of the proposed scheme.

i) The DO first generates the corresponding authenticators for each data block of the file, and then sends tags and the file to CSP. The DO and the CSP deploy the DO Setup Chaincode and CSP Promise Chaincode on the blockchain, respectively.

ii) The CSP traverses the blockchain to find auditing tasks, generates the corresponding challenge and proof by using the public status information of the current block, and submits a transaction proposal containing the proof and auditing chaicode ID on the blockchain. The verifier invokes the Auditing Chaincode through the proof and public parameters. Then the verification result is packaged into a new block on the blockchain.

iii) To achieve key exposure resistance for decentralized data auditing, the DO constructs the authenticator for each data block of F using the auditing secret key SK. At each key update, the DO and the MN update their own private keys using a random value generated by the DO. The MN then generates an update message and sends it to the DO, which uses the update message to generate the latest auditing secret key. Since the random value is secret, it is impossible for an adversary to compute the latest auditing secret key, even if the adversary obtains the DO's private key for some previous time period. To prevent a malicious CSP from tampering with the authenticator, especially those uploaded during the key exposure period, the DO generates the authenticator update key auk and sends it to the CSP who uses the auk to update the specified authenticators.

Appendix B.2 Deployment of BAKER

In the IIoT, DOs choose different cloud storage spaces based on business requirements and operating costs, and sometimes multiple DOs have to jointly maintain a data ledger due to business distribution, supply chain, or other factors. Therefore, the consortium blockchain is more suitable for our scheme. Hyperledger Fabric[1)] is an open source consortium blockchain platform with distributed ledger technology. The different channels and permissioned chain maintain a high level of privacy in industrial activities, and the pluggable consensus satisfies the requirements under different trust models, which are not available in the public blockchain.

Given that the verification algorithm in BAKER runs on Fabric, it is important to address issues about trust and malicious peers. For trust issues, the identity authentication mechanism based on member service providers (MSPs) and policies in Fabric can keep illegal nodes out of the network. Any node that joins the network, such as a peer, must be issued a digital certificate by a Certificate authority (CA), which provides identity authentication for nodes on the blockchain and defines their permissions over on-chain resources. Policies, such as access control, chaincode lifecycle, and endorsements, clarify how members agree or refuse the state update of channels, chaincodes, ledgers, and more. For the ledger fork caused by malicious peers that exist in most permissionless blockchains, the raft ordering service in Fabric defines an ordering node (orderer) to sort transactions, and with the deterministic consensus algorithm, "ledger fork" is prevented.

The consortium blockchain in our BAKER consists of the following components.

• MN: The MN, as a certificate authority, is responsible to process the registration and revocation of members, and provides for dispute resolution as an arbitrator concurrently.

1) https://hyperledger-fabric.readthedocs.io/en/release-2.3/index.html

- Client: The client is an application that accessesand updates the ledger via a peer connection on the Fabric gateway. Updating the ledger involves more complicated interactions between the client and various peers, including the submission of the transaction proposal, endorsement, and submission of the endorsed transaction.
- Peer: One peer can maintain multiple ledgers by joining different channels. There are four types of peers: the endorser simulates the transaction proposal and returns an endorsement response; the leader broadcasts the new transactions to other peers; the committer peer verifies the new block of transactions from the leader peer and updates the ledger; the anchor peer exchanges information with other organizations. Note that a peer may simultaneously play one or more roles and every peer can act as a committer to verify the new block.
- Orderer: The orderer node is responsible for sorting the endorsed transactions, packaging them into a new block and broadcasts the block to leader peer.
- Org: That is organization with several peers. It can be a department or an enterprise in IIoT.
- Channel: Peers in the same channel maintain a ledger.
- Chaincode: The chaincode contains one or more smart contracts and can be invoked in specific channels.

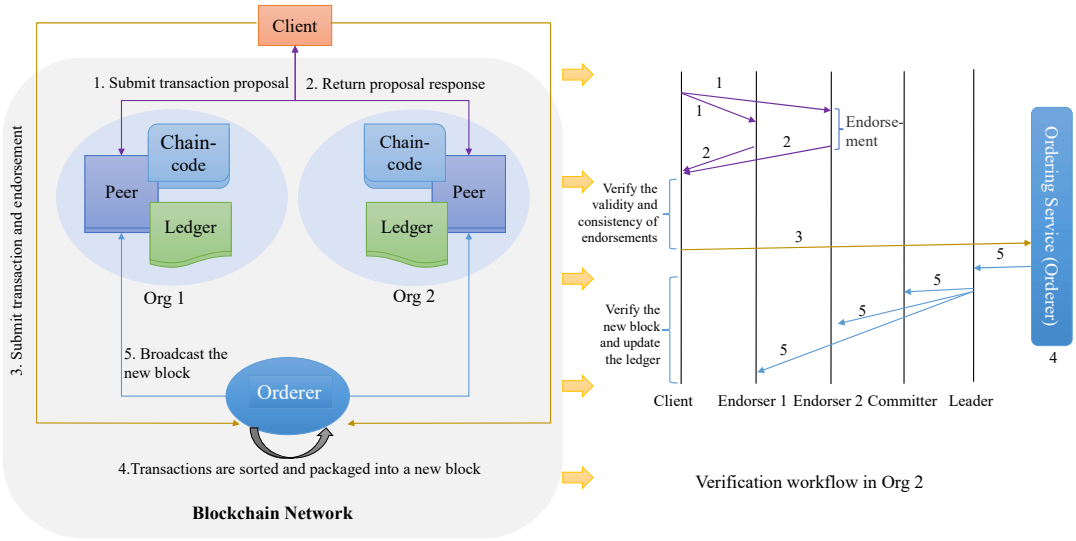

Figure B2 (Color online) Deployment of BAKER.

In order to implement decentralized public auditing, DOs and CSPs in the same channel should get the certificates from the MN before they are permitted to join the blockchain network as peers. All peers and the orderer are nodes in the Hyperledger Fabric, and their interaction process during the auditing phase is as follows (see Figure B2).

1. The CSP submits a transaction proposal, including the proof, auditing chaincode ID, and signature, to multiple endorsers through the client.
2. The endorser takes the transaction proposalas input to run the auditing chaincode and gets the verification result after validating that the signature on the transaction proposal is correct. The verification outcomes will then be signed and put into the proposal response. Note that this is a simulation of transaction execution and will not affect the status of the blockchain.
3. After receiving enough proposal responses (i.e., endorsements), the client checks their signatures and consistency before forwarding the endorsed transaction to the orderer. The endorsement strategy determines the number of endorsements. If there are inconsistent verification results in responses, the transaction is aborted, indicating that the verification failed.
4. The orderer sorts all transactions received in order and packages them into a new block. The orderer then broadcasts the new block to the leader of each Org within the channel.
5. Each leader checks the validity of transactions in the new block. The valid block will be broadcasted to committers within the organization and saved locally. The committer first verifies the transactions in the new block. The new block will be added to the blockchain only if the verification passes. That is, the current auditing task confirms the integrity of the data.

The ledger and data of all organizations within the same channel will be synchronized through their respective anchor peers.

Appendix C BAKER Design

Appendix C.1 The concrete construction of BAKER

The BAKER consists six algorithms: *KeyGen*, *TagGen*, *KeyUpdate*, *AuthUpdate*, *ProofGen*, and *Verification*.

Given a security parameter κ, it determines a finite field Z_p and a bilinear pairing map $e: G_1 \times G_1 \to G_2$, where G_1, G_2 are two multiplicative cyclic groups of a large prime order p. Let g, u be two different generators of G_1. Select a pseudorandom permutation $\mathcal{F}: Z_p^* \times \{1,2,\cdots,n\} \to \{1,2,\cdots,n\}$. Set five different collision resistant hash functions $H_1: \{0,1\}^l \to G_1$, $H_2: \{0,1\}^* \to G_1$, $H_3: G_1 \to Z_p^*$, $f: \{0,1\}^* \to Z_p^*$, $h: \{0,1\}^* \to Z_p^*$, where l is the length of time period t, f maps $\{0,1\}^*$ to the key space of $\mathcal{F}$. We assume that all the verifiers and CSPs have long-term private/public key pairs on the blockchain, and the public keys correspond to their addresses on the blockchain.

(1) *KeyGen*. The DO first generates a signing private/public key pair (ssk, spk), randomly picks $x \in Z_p$ as the private key. Then selects $\beta \in Z_p^*$ as the secret key of the MN, Computes $SK = H_1(t_0)^{\beta x}, g^\beta, R = g^{\beta x}$, sets $SK = H_1(t_0)^{\beta x}$ as the auditing secret key of the initial time period, and sends β to the MN in a secure channel. The public parameter tuple is $para = (H_1, H_2, H_3, f, h, \mathcal{F}, g, u, g^\beta, R, spk)$ and visible to everyone in this system, while the secret parameters is confidential.

(2) *TagGen*. For a file F to be uploaded to a cloud server in the system, the DO first divides the file into $n \times s$ sectors. To protect the confidentiality of IIoT data, each sector is encrypted and denoted as $m_{i,j},\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant s$, assuming a symmetric encryption algorithm used. Record each s-sector as a data block $m_i = (m_{i,1}, \cdots, m_{i,s})$ in order. Then he does the following:

- Select $s+1$ random secrets $\alpha, \alpha_1, \cdots, \alpha_s \in Z_p^*$, and compute $v = g^\alpha, u_j = u^{\alpha_j} \in G_1, j = 1, \cdots, s$, and the authenticator of the $i-th$ block as $\sigma_i = (h_i \cdot u^{\sum_{j=1}^s \alpha_j m_{i,j}})^\alpha \cdot SK$, where $h_i = H_2(name \| i \| C_{id})$, $name, C_{id}$ are the identifiers of the file and CSP, respectively.
- Set $F = \{m_i\}_{i\in[1,n]}, \Phi = \{\sigma_i\}_{i\in[1,n]}, U = \{u_1, \cdots, u_s\}$. Compute the file tag as

$$\mathrm{FT} = (name \| C_{id} \| t \| U \| v) \| SSig_{ssk}(name \| C_{id} \| t \| U \| v)$$

to ensure the correctness of the file identifier $name$ and time period t.

- The DO sends $\{F, \Phi, \mathrm{FT}\}$ to CSP who verifies whether they are valid once receiving data-tag pairs:

$$e(g, \sigma_i) = e(v, h_i \cdot \prod_{j=1}^{s} u_j^{m_{i,j}}) \cdot e(R, H_1(t)), \quad i = 1, \cdots, n. \tag{C1}$$

If it holds, the CSP stores these data blocks and $\Phi = \{\sigma_i | i = 1, \cdots, n\}$; otherwise, the storage transaction is aborted.

(3) *KeyUpdate*. At the beginning of time period t, the DO selects a random number $\rho \in Z_p^*$, updates her/his own private key to $x_t = x/\rho$, and sends ρ, t to MN through a secure channel. The MN updates the private key as $\beta_t = \beta \cdot \rho$, calculates the update message $H_1(t)^{\beta_t}$, and returns it to the DO. The DO computes $H_1(t)^{\beta_t x_t}$ and checks the correction of $H_1(t)^{\beta_t}$ as follows

$$e(g, H_1(t)^{\beta_t x_t}) = e(R, H_1(t)). \tag{C2}$$

If it holds, the DO sets $\widetilde{SK} = H_1(t)^{\beta_t x_t}$ as the auditing secret key of time period t.

(4) *AuthUpdate*. The authenticators will be updated after a new auditing secret key $\widetilde{SK}$ is generated at time period t. Concretely,

- The DO computes the authenticator update key $auk = \widetilde{SK}/SK$ and sends it to the CSP, where SK is the auditing secret key of the last time period $t-1$.
- The CSP verifies whether auk is valid through the equation below

$$e(g, auk) = e(R, H_1(t)/H_1(t-1)). \tag{C3}$$

Only if it holds, the CSP updates all block authenticators by computing $\widetilde{\sigma_i} = \sigma_i \cdot auk$, and displaces Φ with $\widetilde{\Phi}$.

(5) *ProofGen*. The CSP periodically traverses the auditing tasks of all DOs, and gets the public information τ. Then do the following.

- Choose a random $1 \leqslant c \leqslant n$ and compute $k = f(\tau)$. $\forall i \in [1, c]$, compute the challenge $s_i = \mathcal{F}(k, i) \in [1, n], v_i = h(\tau \| i) \in Z_p^*$. Let $I = \{s_1, s_2, \cdots, s_c\}$ be the index set of the challenged blocks.
- Select s random numbers $\eta_1, \cdots, \eta_s \in Z_p^*$, and compute $Q = \prod_{j=1}^s u_j^{\eta_j} \in G_1$, $\gamma = H_3(Q)$, $\mu_j = \eta_j + \gamma \sum_{i\in I} v_i \cdot m_{i,j}$ with $j = 1, \cdots, s$. Let $\mu = \{\mu_1, \cdots, \mu_s\}$. Calculate the aggregate authenticator $\sigma = \prod_{i\in I} \sigma_i^{v_i \gamma}$.
- Submit a transaction proposal containing the proof $P = (\mathrm{FT}, \tau, c, Q, \mu, \sigma)$ and auditing chaincode ID on blockchain.

(6) *Verification*. Verifiers first checks the validity of the file tag FT and verifies the authenticity of state information τ via blockchain. If any verification failed, it aborts. Otherwise, recover $name, C_{id}, U, v$, and compute $\gamma = H_3(Q), k = f(\tau), I = \{\mathcal{F}(k, i)\}$, $v_i = h(\tau \| i), h_i = H_2(name \| i \| C_{id})$ for $i = 1, \cdots, c$. Then verifiers check whether the following equation (C4) holds

$$e(v, Q) \cdot e(g, \sigma) = e(v, (\prod_{i\in I} h_i^{v_i})^\gamma \cdot \prod_{j=1}^{s} u_j^{\mu_j}) \cdot e(R, H_1(t)^{\gamma \sum_{i\in I} v_i}). \tag{C4}$$

If it holds, then return "True"; otherwise, return "False".

DO Setup Chaincode
Data owner's address: DO.addr
CSP's address: CSP.addr
System public parameters: para
Storage cycle: SC
Auditing tasks: AT
Deposit: D
Service fee: SF
……

Figure C1 DO Setup Chaincode.

CSP Promise Chaincode
CSP's address: CSP.addr
Data owner's address: DO.addr
Deposit: D
Compensation for the delay of auditing task: CD
Fine for data corruption: FD
if (The auditing task is not performed according to the DO Setup Chaincode:)
Execute: Transaction(from: CSP.addr, to: DO.addr, value: CD);
……

Figure C2 CSP Promise Chaincode.

Auditing Chaincode
Promise: {
if (*Verification(para,P) == True*)
Execute: Transaction(from: DO.addr, to: CSP.addr, value: SF);}
……

Figure C3 Auditing Chaincode.

Appendix C.2 Supporting arbitration

Service fees and compensation. After successfully uploading the data to the cloud server, the DO deploys the DO Setup Chaincode (see Figure C1) on the blockchain. The chaincode contains information such as the DO's address on the blockchain (i.e., public key), the CSP's address on the blockchain, public parameters, the storage period, auditing tasks, and the storage service fee payable for each verification. The CSP deploys a chaincode to state that it will conduct integrity checks in accordance with the auditing task list. It promises to pay compensation if the check is not completed within the specified time or verification fails, see Figure C2. The verification algorithm is packaged into the auditing chaincode, see Figure C3, which triggers a service fee transaction when the verification passes, at which point the CSP can receive the service fee due for this period from the DO.

Arbitration. If a malicious verifier outputs "False", for example, the DO may frame the CSP for compensation. The framed CSP can apply for arbitration to the MN who selects s random numbers $\{v'_1, \cdots, v'_c\}$ from Z_p^* and sends them to the CSP. The CSP generates a new proof as follows:

- Compute $\mu'_j = \sum_{i\in I} v'_i \cdot m_{i,j}$ with $j = 1, \cdots, s$. Let $\mu' = \{\mu'_1, \cdots, \mu'_s\}$. Calculate the aggregate authenticator $\sigma' = \prod_{i\in I} \sigma_i^{v'_i}$.
- Submit a transaction proposal containing the proof $P' = (\mathrm{FT}, \tau, c, \mu', \sigma)$ and auditing chaincode ID on blockchain.

After the MN receives P', it first verifies whether the file tag FT and τ are valid. If any verification failed, it aborts. Otherwise, recover the identifiers $name, C_{id}$, and compute $k = f(\tau), I = \{\mathcal{F}(k, i)\},\ i = 1, \cdots, c$. Then check whether the following equation (C5) holds

$$e(g, \sigma') = e(v, \prod_{i\in I} h_i^{v'_i} \prod_{j=1}^{s} u_j^{\mu'_j}) \cdot e(R, H_1(t)^{\sum_{i\in I} v'_i}). \tag{C5}$$

If it holds, it means the data is stored correctly; otherwise, the data is corrupted, and the CSP has to pay compensation and fine to the DO.

Appendix D Security analysis

Theorem 1. (Correctness) If all challenged data blocks and authenticators are correct, then the proof from the CSP will pass verification.

Proof. Equation (1) is used to verify the correctness of the data blocks and Φ once the CSP receives the file and tag set.

$$\begin{aligned} e(g, \sigma_i) =& e(g, (h_i \cdot u^{\sum_{j=1}^{s} \alpha_j m_{i,j}})^{\alpha}) \cdot e(g, H_1(t)^{\beta x}) \\ =& e(v, (h_i \cdot \prod_{j=1}^{s} u^{\alpha_j m_{i,j}})) \cdot e(R, H_1(t)) \\ =& e(v, h_i \cdot \prod_{j=1}^{s} u_j^{m_{i,j}}) \cdot e(R, H_1(t)). \end{aligned}$$

The DO uses equation (2) to verify the validity of the update message from the MN in the *KeyUpdate* phase

$$e(g, H_1(t)^{\beta_t x_t}) = e(g^{\beta x}, H_1(t)) = e(R, H_1(t)).$$

The CSP verifies the correctness of the authenticator update key auk from DO using equation (3).

$$e(g, auk) = e(g, H_1(t)^{\beta_t x_t} / H_1(t-1)^{\beta_{t-1} x_{t-1}}) = e(g, (H_1(t)/H_1(t-1))^{\beta x}) = e(R, H_1(t)/H_1(t-1)).$$

Equation (4) is used to proved that the challenged data blocks are stored securely in the cloud. Using public parameters, $e(u,Q)\cdot e(g,\sigma)$ can be computed as follows

$$\begin{aligned}e(v,Q)\cdot e(g,\sigma)=&e(v,\prod_{j=1}^{s}u_j^{\eta_j})\cdot e(g,\prod_{i\in I}((h_i\cdot u^{\sum_{j=1}^{s}\alpha_j m_{i,j}})^{\alpha}\cdot SK)^{v_i\gamma})\\=&e(v,\prod_{i\in I}h_i^{v_i\gamma}\prod_{j=1}^{s}u_j^{\eta_j+\gamma\sum_{i\in I}v_i\cdot m_{i,j}})\cdot e(g,SK^{\gamma\sum\limits_{i\in I}v_i})\\=&e(v,(\prod_{i\in I}h_i^{v_i})^{\gamma}\cdot\prod_{j=1}^{s}u_j^{\mu_j})\cdot e(R,H_1(t)^{\gamma\sum\limits_{i\in I}v_i}).\end{aligned}$$

The MN uses the equation (5) to check if the data is stored correctly by CSP. Using the public parameters and the proof generated by CSP, it yields

$$\begin{aligned}e(g,\sigma')=&e(g,\prod_{i\in I}(h_i\cdot u^{\sum_{j=1}^{s}\alpha_j m_{i,j}})^{\alpha v_i'})\cdot e(g,\prod_{i\in I}SK^{v_i'})\\=&e(g,\prod_{i\in I}(h_i\cdot u^{\sum_{j=1}^{s}\alpha_j m_{i,j}})^{\alpha v_i'})\cdot e(R,H_1(t)^{\sum_{i\in I}v_i'})\\=&e(v,\prod_{i\in I}h_i^{v_i'}\prod_{j=1}^{s}u_j^{\mu_j'})\cdot e(R,H_1(t)^{\sum_{i\in I}v_i'}).\end{aligned}$$

This ends the proof.

In order to formally define the security of the proposed BAKER, we design a game composed of an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ to attack the security of the system:

(1) **Setup phase.** $\mathcal{C}$ runs *KeyGen* to generate the secret key x and the system public parameters *para*. Give *para* to $\mathcal{A}$.

(2) **Query phase.** $\mathcal{A}$ is allowed to perform a series of queries:

- Secret key queries. $\mathcal{A}$ can query the secret key in time period t. $\mathcal{C}$ returns the secret key x in time period t to $\mathcal{A}$.
- Authenticator queries. $\mathcal{A}$ can query the authenticator of any data block $m_i=(m_{i,1},\cdots,m_{i,s})$ in time period t. $\mathcal{C}$ returns the authenticator σ_i corresponding to (m_i,t).

(3) *Forgery phase.* $\mathcal{A}$ outputs the authenticator $\hat{\sigma_i}$ for a block $\hat{m_i}$ in time period $\hat{t}(\neq t)$, which is never queried in query phase.

Theorem 2. (Soundness) If the CDH assumption holds in group G_1, our proposed scheme is key exposure resistant and the authenticators generated during key-exposure time period are secure.

Proof. Given CDH instance $(g,G_1,\Upsilon=g^a,\Lambda=g^b)$, if $\mathcal{A}$ wins the game with non-negligible probability ϵ, then the challenger can solve the CDH problem. Suppose that the adversary makes q_s secret key queries and q_a authenticator queries.

(1) **Setup phase.** The challenger $\mathcal{C}$ initializes the time period as $t=0$, randomly selects $SK_{\mathcal{C}}=x\in Z_p^*$, and sets $PK_{MN}=\Upsilon, PK_{\mathcal{C}}=\Upsilon^x$, $SK_{MN}=a$ that $\mathcal{C}$ has no idea about it. Then $\mathcal{C}$ randomly selects $\alpha\in Z_p^*$ and s random values $(\alpha_1,\cdots,\alpha_s)\in Z_p^*$, and computes $u=g^{\alpha}$ and $u_j=g^{\alpha_j}(1\leqslant j\leqslant s)$. Finally, $\mathcal{C}$ sends the public parameters $(H_1,H_2,g,u,\{u_j\},PK_{MN},PK_{\mathcal{C}})$ to $\mathcal{A}$. Let t' be the challenged time period, that is, $\mathcal{A}$ cannot query the private key and tags during time period t'.

(2) **Query phase.** H_1 and H_2 are considered to be two different random oracles. There are four queries in this phase:

a) H_1 **query.** The challenger $\mathcal{C}$ creates a hash table H_1-table to record all queries and response about H_1 query. The H_1-table is empty at the beginning. When $\mathcal{A}$ queries H_1 at (t), $\mathcal{C}$ responds as follows:

- For input (t), if H_1-table contains a tuple (t,ω,h_1), $\mathcal{C}$ returns h_1.
- Otherwise, $\mathcal{C}$ randomly chooses $\omega\in Z_p^*$. If $t\neq t'$ computes g^{ω} and adds $(t,\omega,h_1=g^{\omega})$ to H_1-table. Otherwise, $\mathcal{C}$ computes Λ^{ω} and adds $(t,\omega,h_1=\Lambda^{\omega})$ to H_1-table. Finally, $\mathcal{C}$ responds with h_1.

b) H_2 **query.** The challenger $\mathcal{C}$ creates a hash table H_2-table to record all queries and response about H_2 query. The H_2-table is empty at the beginning. When $\mathcal{A}$ queries H_2 at $(name\|i\|C_{id})$, $\mathcal{C}$ responds as follows:

- For input $(name\|i\|C_{id})$, if H_2-table contains a tuple $(name\|i\|C_{id},\zeta,h_2)$, $\mathcal{C}$ returns h_2.
- Otherwise, $\mathcal{C}$ randomly chooses $\zeta\in Z_p^*$ and adds $(name\|i\|C_{id},\zeta,h_2=g^{\zeta})$ to H_2-table. $\mathcal{C}$ responds with h_2.

c) **The secret key query.** If $\mathcal{A}$ wants to query the private key $SK_{\mathcal{C},t}$ during time period t, $\mathcal{C}$ retrieves (t,ω,h_1) in H_1-table.

- If $t=t'$, $\mathcal{C}$ aborts.
- Otherwise, $\mathcal{C}$ randomly chooses $\rho_t\in Z_p^*$, computes $SK_{\mathcal{C},t}=x\rho_t$ as well as $SK_t=\Upsilon^{\omega SK_{\mathcal{C}}}$. Note that $SK_t=h_1^{SK_{MN}SK_{\mathcal{C}}}=g^{\omega SK_{MN}SK_{\mathcal{C}}}=\Upsilon^{\omega SK_{\mathcal{C}}}$. Finally, $\mathcal{C}$ responds with $SK_{\mathcal{C},t}$.

d) **The Authenticator query.** As for the query of $(t,name,i,C_{id})$, $\mathcal{C}$ first recovers (t,ω,h_1) from H_1-table and $(name\|i\|C_{id},\zeta,h_2)$ from H_2-table.

- If $t=t'$, $\mathcal{C}$ aborts.

• Otherwise, $\mathcal{C}$ computes $\sigma_i = (h_2 \cdot u^{\sum_{j=1}^{s} \alpha_j m_{i,j}})^{\alpha} \cdot SK_t = (g^{\zeta} \prod_{j=1}^{s} u_j^{m_{i,j}})^{\alpha} \cdot \Upsilon^{\omega SK_C}$, and responds with (t, σ_i).

(3) **Forgery phase.** The challenger $\mathcal{C}$ chooses a time period t in which $\mathcal{A}$ never made a key query and an authenticator query. It requests $\mathcal{A}$ to return a valid tag σ_i for challenged block $m_i = (m_{i,1}, \cdots, m_{i,s})$. $\mathcal{A}$ outputs (t, σ_i) and $\mathcal{C}$ recovers $(t, \hat{\omega}, \hat{h_1})$ from H_1-table.

a) If $t \neq t'$, $\mathcal{C}$ aborts.

b) If $t = t'$, $h_1 = \Lambda^{\hat{\omega}}$ and $h_2 = g^{\hat{\zeta}}$. If the forgery is valid, then

$$e(g, \sigma_i) = e(g, (h_2 \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^{\alpha} \cdot SK_t) = e(g, u^{\hat{\zeta} + \sum_{j=1}^{s} \alpha_j m_{i,j}}) \cdot e(g, g^{abx\hat{\omega}}).$$

$\mathcal{C}$ proceeds to calculate the value of g^{ab}. The result for the given CDH instance is $g^{ab} = (\sigma_i / u^{\hat{\zeta} + \sum_{j=1}^{s} \alpha_j m_{i,j}})^{(x\hat{\omega})^{-1}}$.

Let A_1 denote the secret key query phase, A_2 denote the authenticator query phase, A_3 denote the forge phase, and $\Omega = \frac{q^s + q^a}{q^s + q^a + 1}$. The probability of $\mathcal{A}$ solving the given CDH problem is

$$Pr[\neg A_1 \wedge \neg A_2 \wedge \neg A_3] \geqslant \Omega^{q_s} \Omega^{q_a} (1 - \Omega) = \left(\frac{(q^s + q^a)^{q^s + q^a}}{(q^s + q^a + 1)^{q^s + q^a + 1}} \right),$$

which is not negligible, and this contradicts the CDH assumption. Therefore it is not possible to forge a valid authenticator.

Further we will show that the proof P is also unforgeable. The above analysis implies the aggregation tag σ cannot be forged. Let $P = (\hat{t}, Q, \mu, \sigma)$ be the correct proof for a challenge $\{(i, v_i) | i \in I\}$ in time period $\hat{t}$. If $\mathcal{A}$ outputs a proof $P' = (\hat{t}, Q, \mu', \sigma)$ satisfying the equation (4), then we have

$$\sum_{j=1}^{s} \alpha_j (M'_j - M_j), \quad M'_j = \sum_{i \in I} v_i m'_{i,j}, M_j = \sum_{i \in I} v_i m_{i,j}.$$

Assuming that there exist d ($d \geqslant 1$) forged M'_j ($\neq M_j$), the probability of selecting the proper $\{\alpha_j\}$ is $\frac{p^{d-1}}{p^d} = \frac{1}{p}$, which is a negligible value. Thus a valid proof cannot be forged.

Similarly, we can prove that the proof P' in the Section 4.3 is also unforgeable.

From the above analysis, under the CDH assumption, our proposed scheme is key exposure resistant and the authenticators generated during key-exposure time period are secure.

Theorem 3. In our scheme, it is difficult to forge auditing secret key even if the DO's private key was exposed.

Proof. In each time period t, the auditing secret key $SK = H_1(t)^{x\beta}$ contains two secret values: the DO's private key and the MN's private key. Since the MN's private key is secret, it is impossible to successfully forge a valid auditing secret key under the DL assumption, even if an adversary obtains the DO's private key in the current period. Furthermore, the DO's private key and the MN's private key are updated using a random value, and it is impossible for an adversary to derive the DO's new private key since the random value is secret. Therefore, the auditing private key in our scheme is secure.

Theorem 4. (Privacy preserving) Any verifier cannot recover the original data simply by the proof P from CSP.

Proof. We prove this theorem from the following cases.

(1) Note that $\sum_{i \in I} v_i m_{i,j}$ is blinded by η_j and γ in equation $\mu_j = \eta_j + \gamma \sum_{i \in I} v_i m_{i,j}$. Verifiers cannot learn $\sum_{i \in I} v_i m_{i,j}$ from $\mu_j = \eta_j + \gamma \sum_{i \in I} v_i m_{i,j}$ without the random secret η_j. It is impossible to get η_j from Q under the assumption of CDH.

(2) Verifiers cannot learn any information on $\sum_{i \in I} v_i m_{i,j}$ from σ. Note that $\sigma = \prod_{i \in I} ((h_i \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^{\alpha} \cdot H(t)^{x\beta})^{\gamma v_i} = \prod_{i \in I} (h_i^{\alpha} \cdot H(t)^{x\beta})^{\gamma v_i} \prod_{j=1}^{s} u_j^{\gamma \sum_{i \in I} v_i m_{i,j} \alpha}$. It is hard to compute $\prod_{i \in I} (h_i^{\alpha} \cdot H(t)^{x\beta})^{\gamma v_i}$ under the CDH assumption, so $u_j^{\gamma \sum_{i \in I} v_i m_{i,j} \alpha}$ cannot be derived.

Therefore, the proposed scheme guarantees data privacy.

Theorem 5. (Detectability) Suppose that the file F stored in the cloud server is divided into n blocks, and the number of corrupted data blocks is a. If there are c challenged data blocks, the proposed scheme is $(\frac{a}{n}, 1 - (\frac{n-a}{n})^c)$ detectable.

Proof. We use X as the corrupted blocks in challenged blocks. Since each block m_i can be selected repeatedly for challenge, the detection probability of the corrupted blocks is

$$P\{X \geqslant 1\} = 1 - P\{X = 0\} = 1 - (\frac{n-a}{n})^c.$$

Thus our scheme is $(\frac{a}{n}, 1 - (\frac{n-a}{n})^c)$ detectable.

Fairness. In the proposed scheme, we design chaincodes to realize fair payments between DOs and CSPs based on consortium blockchain. Specifically, the details and trigger conditions of the transactions are embedded in the chaincodes. When the CSP honestly provides proof of data possession, it can get the service fee. However, when the CSP refuses to provide proof of possession or the verification reslut shows that the data is corrupted, the DO can initiate arbitration to obtain compensation. When a malicious node frames a CSP, the CSP can initiate arbitration to prove possession of the correct data. All actions of the auditing process are recorded by the blockchain, which can provide enough evidence to ensure fair transactions. Therefore, the proposed scheme achieves payment fairness.

Appendix E Performance evaluation

A functionality comparison is given between the proposed scheme and several related schemes [1–6]. See Table E1.

Table E1 Comparison of Public Auditing Schemes

Schemes	Decentralized Data Auditing	Key Update	Authenticator Update	Key-Exposure Resistance	Fair Payment
[1]	×	✓	×	✓	×
[2]	×	✓	✓	✓	×
[3]	×	✓	✓	×	×
[4]	✓	×	×	×	✓
[5]	✓	×	×	×	✓
[6]	×	✓	×	✓	×
Our scheme	✓	✓	✓	✓	✓

Appendix E.1 Theoretical result

Computational overhead. We discuss the computational overhead in the five phases: key update, authenticator generation, authenticator update, proof generation, and verification. A comparison of our computational overhead with $[1,2,6]$ is shown in Table E2. In the key update phase, the computational overhead in our scheme is $2M_{G_1} + 2E_{G_1} + H$, which is less than $[1,2,6]$ because the computational overhead of a H operation is higher than that of E_{G_1} and M_{G_1}. During the authenticator update phase, the DO only needs to compute one M_{G_1} to generate the authenticator update key, whereas $M_{G_1} + 2E_{G_1}$ are required in [2]. [1,6] do not discuss the authenticator update. Since we divide the file with each data block containing s sectors, whereas the other three schemes do not, we have $n' = n * s$ for a file of size $|F|$. Obviously, when $s > 2$, the computational overhead to generate authenticators for F in our scheme is significantly less than that in $[1,2,6]$. Our scheme is also lower in terms of computational costs for the proof and verification phases. For example, assuming the file F has a corrupted data blocks. From the Theorem 5, it needs $c \geqslant 44$ to achieve a detection probability of 99% if we set $a = 20, n = 200$ and $s = 50$. While for n', it requires $c' \geqslant 2301$. As can be seen from Table E2, for the same size of file, the computational cost of our scheme is less compared to $[1,2,6]$.

Table E2 Computational Overhead

Schemes	[1]	[2]	[6]	Our scheme
KeyUpdate	$M_{G_1} + 2E_{G_1} + 2H$	$2(S + E_{G_1} + H) + 2M_{G_1}$	$2M_{G_1} + E_{G_1} + 2H$	$2M_{G_1} + 2E_{G_1} + H$
AuthUpdate	\	$M_{G_1} + 2E_{G_1}$	\	M_{G_1}
TagGen	$n'(M_{Z_p^*} + 2M_{G_1} + 2E_{G_1} + H)$	$n'(2M_{G_1} + 3E_{G_1} + H)$	$n'(M_{Z_p^*} + M_{G_1} + 2E_{G_1} + H)$	$n((s-1)S + sM_{Z_p^*} + 2M_{G_1} + 2E_{G_1} + H)$
ProofGen	$(c'-1)(S + M_{G_1}) + cM_{Z_p^*} + P + (c'+1)E_{G_1}$	$c'S + (c'+1)M_{Z_p^*} + (c'-1)M_{G_1} + c'E_{G_1} + E_{G_2} + P + H$	$(c'-1)(S + M_{G_1}) + c'M_{Z_p^*} + (c'+1)E_{G_1}$	$s(cS + (c+1)M_{Z_p^*}) + (c+s-2)M_{G_1} + (s+c+1)E_{G_1} + H$
Verification	$(c'-1)S + (c'+1)M_{G_1} + (c'+2)E_{G_1} + M_{G_2} + 3\mathcal{P} + (c'+1)H$	$(c'-1)S + M_{Z_p^*} + (c'+1)M_{G_1} + 2M_{G_2} + (c+4)E_{G_1} + 3\mathcal{P} + c'H$	$(c'-1)S + cM_{Z_p^*} + 2E_{G_1} + 2M_{G_2} + 4\mathcal{P} + 2H$	$(c-1)S + M_{Z_p^*} + (s+c-1)M_{G_1} + 2M_{G_2} + (s+c+2)E_{G_1} + 4\mathcal{P} + cH$

Communication overhead. The communication overhead in the key update phase comes from two parts: the DO sends the secret ρ to the MN, and the MN sends back the update information $H_1(t)^{\beta_t}$. So the communication overhead is $|p| + |G_1|$, which costs the same as in [2]. Although the communication overhead is $|G_1|$ in [1,6], their schemes only update the auditing secret key but not the DO's private key. In the authenticator update phase, only the DO needs to send an authenticator update key auk to the CSP with a communication overhead of $|G_1|$, which is identical to [2], while [1,6] do not discuss the authenticator update. The proposed scheme uses blockchain and smart contracts to achieve decentralized public auditing. Thus, there is no interaction during the challenge process. While in the other three schemes $[1,2,6]$, the communication overhead is $c|p|$, $c|p|$, and $|p|$, respectively.

Table E3 Communication Overhead

Schemes	[1]	[2]	[6]	Our scheme
KeyUpdate	$\|G_1\|$	$\|p\| + \|G_1\|$	$\|G_1\|$	$\|p\| + \|G_1\|$
AuthUpdate	\	$\|G_1\|$	\	$\|G_1\|$
Challenge	$c\|p\|$	$c\|p\|$	$\|p\|$	0

We analyze the communication and computational overhead of *KeyUpdate*, *AuthUpdate*, *ProofGen* and *Verification* in BAKER and $[1,2,6]$. See Table E3 and E2.

Note that the file in BAKER is divided into n blocks containing s sectors, whereas $[1,2,6]$ do not, so we set $n' = n * s$ for a file of size $|F|$. Obviously, when $s > 2$, the computational overhead to generate authenticators for F in our scheme is

significantly less than that in [1,2,6]. Our scheme is also lower in terms of computational costs for the proof and verification phases. For example, assuming the file F has a corrupted data blocks. From the Theorem 5, it needs $c \geqslant 44$ to achieve a detection probability of 99% if we set $a = 20, n = 200$ and $s = 50$. While for n', it requires $c' \geqslant 2301$. As can be seen from Table E2, for the same size of file, the computational cost of our scheme is less compared to [1, 2, 6].

Appendix E.2 Off-chain experimental result

In order to intuitively observe the performance of our proposed scheme, we carry out experimental simulation on algorithms: *TagGen*, *KeyUpdate*, *AuthUpdate*, *ProofGen* and *Verification*. All algorithms are written by Python language with the Charm-Crypto library [2)]. The time consumption of these algorithms is simulated on Ubuntu 20.04.3 LTS with 4GB RAM. The bilinear mapping is obtained by using type A elliptic curve with 160 bits. In our experiment, the selected file sizes are 200KB, 400KB, 600KB, 800KB, and 1MB. Set the size of each data block to 20 bytes in [1, 2, 6]. Set each data block to consist of 50 sectors of size 20 bytes in our scheme.

First, the simulation result of *TagGen* is shown in Figure E1. The time cost for generating the authenticator ranges from 1.00 s to 4.57 s as the file size increases. Compared with [1, 2, 6], the proposed scheme has a much lower time cost in computing authenticators for outsourced data blocks. This means that as the outsourced files grow, the proposed scheme is more friendly to the DO.

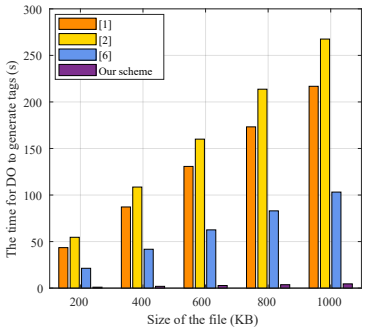

Figure E1 (Color online) Time cost for *TagGen*.

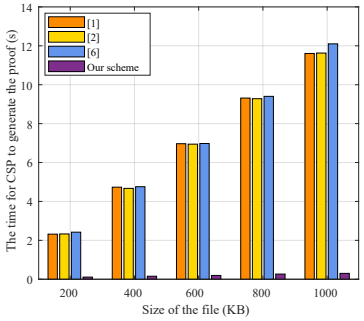

Figure E2 (Color online) Time cost for *ProofGen*

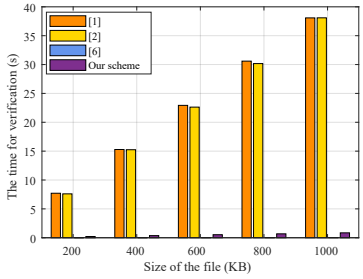

Figure E3 (Color online) Time cost for *Verification*

Then, we test the time cost of generating proofs and completing verification in the case that the number of corrupted data blocks is 20 and the detection probability of the corrupted blocks is 99%. Under this premise, files of sizes 200 KB, 400 KB, 600 KB, 800 KB, and 1 MB require a minimum number of challenge blocks of 44, 90, 134, 182, and 228 in BAKER respectively, while in [1,2,6] the corresponding minimum number of challenge blocks is respectively 2301, 4603, 6902, 9208, and 11511. The time required to generate the proof P and verify P is shown in Figure E2 and Figure E3. Figure E2 reflects that the time consumption of *ProofGen* in our scheme increases linearly with the size of the file, which varies from about 0.11 s to 0.30 s. Compared to the scheme of [1, 2, 6], our scheme has higher efficiency. The verification incurs higher time costs because the verification process involves more exponential and pairing operations than the proof generation process. As shown in Figure E3, the time for verification increases as the file size increases, which varies from 0.21 s to 0.85 s in Our BAKER, while in [1, 2] the time cost is over 38 s for a 1 MB file. In [6], although the verification takes less time, ranging from 0.01 s to 0.04 s, it requires more computational resources to generate the proof.

In addition, we implement algorithms for key update and authenticator update while considering the range of time periods from 1 to 10, see Figure E4. In our scheme, there are one multiplication and two exponential operations when updating the auditing key, which are independent of the time period. As shown in Figure E4, the key update time is approximately 3.77 ms in any time period from 1 to 10 in our scheme, which is less than [1, 2, 6]. This means that it does not have to worry that the time cost of updating the auditing secret key will increase significantly with the time period. Since the DO only needs to compute a division in the *AuthUpdate* phase, the time consumption is independent of time period. The time costs of *AuthUpdate* in our scheme and [2] are approximately 0.0034 ms and 2.1915 ms on DO's side respectively, which maintain at a low level as shown in Figure E5.

Finally, we numerically analyze the detectability of the proposed scheme. As shown in Figure E6, given the probabilities 60%, 80%, 90% ,95%, 99% that 50 corrupted blocks are detected, there is a linear relationship between the number of challenged blocks and total blocks. When the number of data blocks is 10000, about 920 challenged blocks are required to achieve a 99% detectable probability. Figure E7 shows that the ratio of challenged data blocks to the total number is inversely proportional to that of corrupted data blocks to the total number, where the total number of data blocks ranges from 2000 to 10000. Moreover, as the total number of data blocks grows, the ratio of challenged blocks decreases.

2) https://pypi.org/project/charm-crypto/

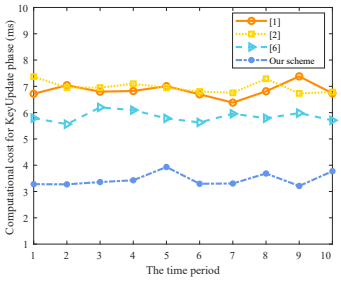

Figure E4 (Color online) Time cost for *KeyUpdate*.

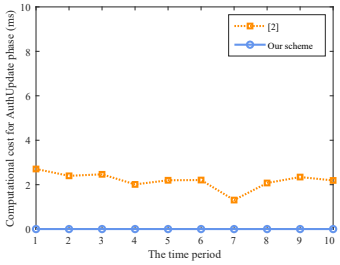

Figure E5 (Color online) Time cost for *AuthUpdate*

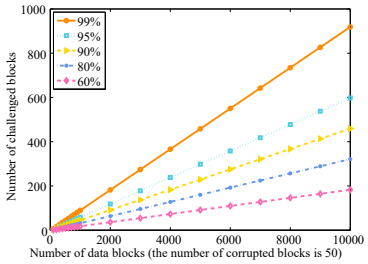

Figure E6 (Color online) Ratio of challenged blocks VS. ratio of corrupted blocks

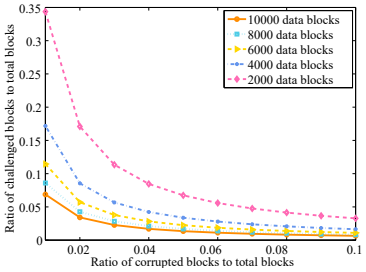

Figure E7 (Color online) Ratio of challenged blocks VS. ratio of corrupted blocks.

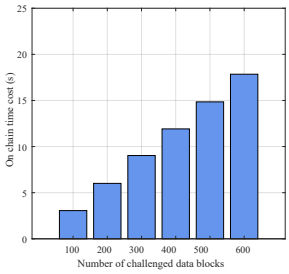

Figure E8 (Color online) Time cost of on-chain verification

Appendix E.3 On-chain experimental result

We develops three chaincodes to realize decentralized public auditing and fair payment. The codes are written by Java program language with the JPBC library[3)], and packaged into the chaincode. We deploy and invoke the chaincodes in Hyperledger Fabric 2.4 version test network on CentOS 7 with 4GB RAM.

The bilinear mapping is obtained by using a type A elliptic curve with 160 bits. In our on-chain experiment, the number of challenged data blocks ranges from 100 to 600. Each block consists of 50 sectors with a size of 20 bytes. We focus on testing the time consumption of on-chain verification. The time cost of on-chain verification is positively linearly related to the number of challenged blocks, as shown in Figure E8, ranging from 3.06 s to 17.85 s.

References

1 Yu J, Wang H Q. Strong key-exposure resilient auditing for secure cloud storage. IEEE Transactions on Information Forensics and Security, 2017, 12: 1931–1940

2 Xu Y, Sun S, Cui J, et al. Intrusion-resilient public cloud auditing scheme with authenticator update. Information Sciences, 2020, 512: 616–628

3 Zheng W Y, Lai C F, He D B, et al. Secure storage auditing with efficient key updates for cognitive industrial iot environment. IEEE Transactions on Industrial Informatics, 2021, 17: 4238–4247

4 Wang H, Qin H, Zhao M H, et al. Blockchain-based fair payment smart contract for public cloud storage auditing. Information Sciences, 2020, 519: 348–362

5 Li Z W, Xin Y, Zhao D, et al. A noninteractive multireplica provable data possession scheme based on smart contract. Security and Communication Networks, 2022, 2022: 6268449

6 Nithya S, Uthariaraj V R. Identity-based public auditing scheme for cloud storage with strong key-exposure resilience. Security and Communication Networks, 2020, 2020: 4838497

3) http://gas.dia.unisa.it/projects/jpbc/