Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(4): 2577–2600.
DOI: 10.3934/era.2025115
Received: 04 November 2024
Revised: 27 February 2025
Accepted: 07 March 2025
Published: 27 April 2025

Research article

Cloud auditing for outsourced storage service in healthcare systems with static data transfer

Ge Wu[1,2], Longlong Cao[1], Hua Shen[3,4], Liquan Chen[1], Xitong Tan[1] and Jinguang Han[1,*]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China

[2] School of Big Data and Information Engineering, Xinjiang University of Technology, Hotan 848000, China

[3] School of Computer Science, Hubei University of Technology, Wuhan 430068, China

[4] Hubei Key Laboratory of Green Intelligent Computing Power Network, Hubei University of Technology, Wuhan 430068, China

* **Correspondence:** Email: jghan@seu.edu.cn.

Abstract: The rapid development and wide adoption of electronic healthcare systems provide huge conveniences for the users. The health records are normally outsourced to a server for centralized management. To ensure data integrity, auditing techniques are promising candidates to be applied. In practice, there is always a need to transfer data, such as when the user changes the hospital or medical center. Moreover, how to provide efficient data transfer and not bring complicated transfer process for auditing should be given attention. For traditional data transfer processes, the tags need to be recomputed or updated and then stored on the server. The scheme we proposed in this paper has the property that the tags remain the same during data transfer. This process is referred to as static transfer since the data and tags do not change. In addition, the basic construction is built upon Shacham-Waters auditing, which is the foundation of the homomorphic linear authenticator (HLA). Therefore, our scheme is compatible with other optimizations for HLA schemes, such as the evolution tag technique proposed recently to achieve a compact tag. To show the effectiveness of our scheme, we combined the evolution tag and our construction to achieve static data transfer functionality and compact tag efficiency. The improvement in our scheme preserves the underlying HLA structure and hence does not complicate the tag generation, challenge, and response phases while achieving static data transfer.

Keywords: cloud auditing; electronic health record; outsourced storage; static data transfer; computational efficiency

1. Introduction

Healthcare systems have the advantages of convenience and long-term data storage. In addition, the medical data can help doctors and health consultants analyze and keep track of the patients' health status. Healthcare data mining has many potential applications in not only theoretical studies, but also in various aspects, such as disease diagnosing, medicine development, and illness detection [1]. Different kinds of healthcare data may get involved in further analyses like health check data and daily body monitoring data. Moreover, many health statuses can be measured by various wearable devices. Taking the resource-limited property of these devices into consideration, the health data needs to be uploaded to medical centers or hospitals. For centralized management and to release the burden of establishing and maintaining data storage infrastructure in each medical center or hospital, these data can be stored at central servers. This can save costs for the data owners and guarantee conveniences for doctors and health consultants to reviewing data. The system users (patients, doctors, consultants, etc.) or data owners (hospitals, medical centers, etc.) do not need to store or maintain the data in their local devices.

Security is one of the issues people are most concerned about regarding outsourced data [2]. Integrity is one of the most fundamental security requirements in information systems. In particular, the precise value of healthcare data is important for diagnoses, data analyses, medicine evaluation etc. and can cause severe consequences if the integrity is not guaranteed. Many techniques exist to achieve data integrity. As for outsourced data, cloud auditing is the most commonly adopted method. In such a system (e.g., healthcare system), users' electronic health records (EHRs) are remotely stored on the server, and the data integrity can be verified through an auditing process. The auditing process is an interactive process that consists of a challenge phase and a response phase. It can be undertaken between the data owner and the storage server, or between a third-party auditor (TPA) and the server. As discussed before, an intended established party that deals with the auditing requests from data owners is preferred.

The basic idea of data auditing is composed of a challenge-response interaction between the cloud server and TPA. The data owner first generates a tag for each data block and uploads the data together with the tag to the server. Then, the TPA can launch an auditing task on behalf of the data owner by sending the server a challenge. Next, the server computes the correct response according to the challenge with valid data-tag pairs and returns the response. After receiving the response, the TPA verifies its validness and provides the data owner the results.

1.1. Data transfer in outsourced storage systems

In practice, the data can be transferred frequently in the system. For example, when a patient changes the hospital or a client signs a contract with new insurance company, the user's personal EHRs should be transferred from the original data owner to the new data owner. Taking all the aforementioned situations into consideration, the system framework of a cloud auditing scheme for healthcare systems consists of four basic components, namely the system user, data owner, cloud server, and TPA, as shown in Figure 1. The system user generates EHRs at the hospital or medical organizations and enjoys the services. The hospital or medical organizations control the users' data and serves as data owner on behalf of its clients according to relevant laws and regulations. The cloud server and TPA provide the outsourced storage and auditing functionality as traditional cloud platform. The core requirement in the system is how to maintain efficiency during data transfer process when the data is frequently transferred among data owners.

As for the real-world cases of transfer for healthcare data, examples include patient transmission, coordinating treatment, remote consultation, disease surveillance etc. Many public datasets or open sources are available, such as global data like Global Health Observatory (GHO), Global Burden of Disease (GBD), Johns Hopkins COVID-19 Dashboard, Global Health Data Exchange (GHDx); regional data like European Center for Disease Prevention and Control (ECDC), American CDC WONDER; and open source data like Kaggle Health Datasets. According to the public information, the data size of the aforementioned cases ranges from 1 GB to 30 PB. Even a single subset or the data of one indicator, e.g., a GBD subset for medical research may reach 10 GB. The transfer for such size data requires heavy costs. Efficient construction not only has theoretical benefits but also enjoys advantages in practical implementations, such as energy saving [3].

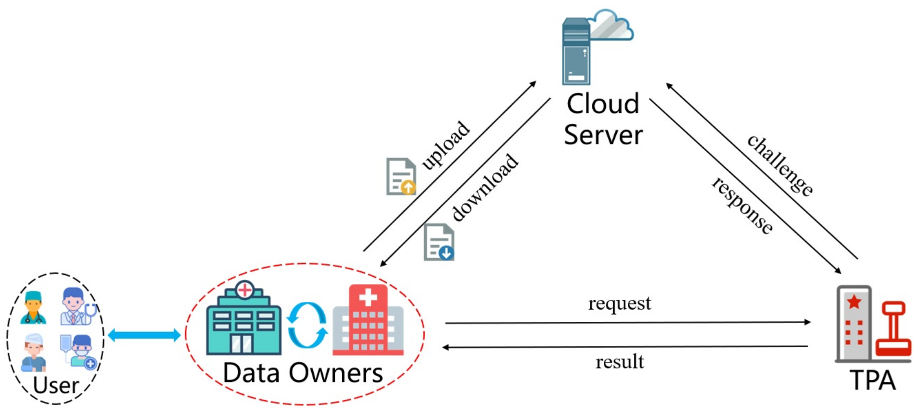

Figure 1. Framework of data auditing in healthcare systems.

A straightforward approach for cloud auditing with data transfer is to recompute the corresponding tags by the new data owner and conduct auditing tasks with the new data owner's public key after the transfer process. However, this causes heavy computational burden for the data owner and requires complete download and upload operations, which also brings heavy transmission costs. An alternative method is to move the computational operations to the server side. The original owner generates a token and uploads the token to the server, then the server can update the tag. The updated tag is equivalent to the tag that generated by the new owner. The basic idea in dealing with the tag is similar to proxy re-signature technique [4], and the remaining auditing processes are the same. The outline of traditional cloud auditing that supports data transfer is shown in Figure 2. The concept of data ownership transfer was proposed by Shen et al. [5] and extended by follow-up researches, including multi-ownership transfer [6], threshold transfer [7], and in the certificateless cryptography setting [8]. Other related schemes, such as designated ownership transfer [9], E-Health data transfer [10], block-chain based transfer [11], anonymous transfer [12], and in the IoT setting [13], are also available in the literature.

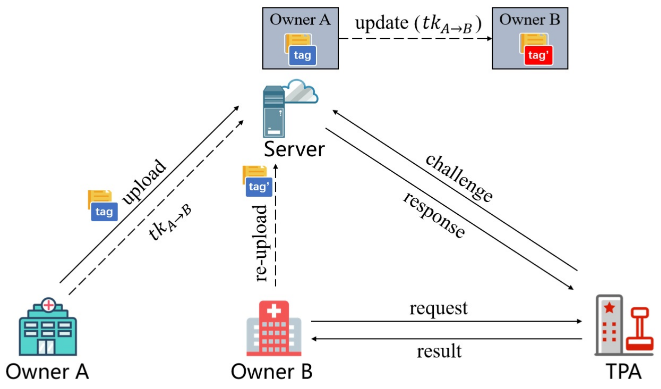

Figure 2. Traditional data transfer process in cloud auditing.

Despite the basic functionality provided by current transfer techniques, heavy computational cost occurs either at the owner side or the server side when the data transfer operation take places. If the system has frequent data transfer requirements, such as in a healthcare system, it brings heavy burden to the system. How to reduce the overall computational cost is an essential factor in improving the system efficiency.

Efficiency issues. As discussed, frequent data (together with the ownership) transfer can cause heavy computation and transmission burdens. In this paper, we attempt to remove these operations by keeping the tag unchanged during the transfer process. Therefore, both the aforementioned two costs can be relieved. To achieve this goal, we modify the transfer process as follows: The token is sent to the new data owner instead of the server, and the auditing request is generated by the token and new owner's public key. The TPA also updates the challenge with the auditing request from a new owner. Then, the server computes the response with the unchanged data and tag. As a result, the response computed with the updated challenge can be verified using the new owner's public key. Since there is no need to change the data or tag on both the owner and the server side, we refer to this process as static transfer. The technical overview is summarized in Figure 3. Please note that when the data owner changes during transfer, the challenge-response phases also need to be modified. Otherwise, the response computed by the same data and tag cannot be verified by the new owner's public key. According to Figures 2 and 3, the advantages of static data transfer can be summarized as Table 1.

Table 1. Cost comparison of data transfer techniques.

Transfer Technique	Transmission Cost		Computation Cost
	Between Owners	With Server	
Recompute & Reupload	-	download & upload	tag recomputation
Tag Update	-	token	tag update
Static Transfer	token	-	-

Download & upload represents the transmission cost between data owner and the server, the cost includes the size of both data and tag. Token represents the transmission cost which only includes the size of transfer token. Tag recomputation and tag update involve heavy computation costs at the data owner side and server side, respectively.

In particular, the improvement with respect to end-to-end system latency and throughput in large-scale systems can be quite substantial. For instance, the Global Health Observatory (GHO) database contains 10 GB total data amount with a 100 MB size single dataset for a certain research purpose. According to the simulation result (with a laptop computer) in Section 4.3, the size of a block is 64 Byte, the computation cost of tag generation, response, and verification for each block is around 0.04 s, 0.04 s, and 0.01 s, respectively. A single dataset (100 MB) consists of roughly 1.5 million blocks, and taking the challenge ratio as 1%, the number of challenged block is 15 k. The end-to-end system latency between the TPA and server (with respect to challenge-response interaction) takes more than 600 seconds, while the latency between new data owner and the server (with respect to recompute & reupload operations) takes more than 16 hours with additional double throughput occupation.

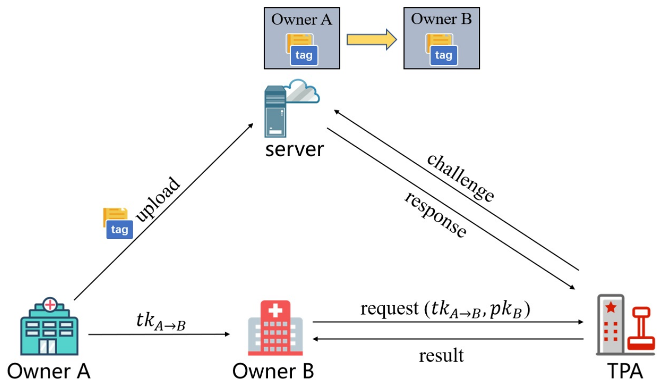

Figure 3. Cloud auditing with static data transfer.

1.2. Related work

There are two branches of data auditing following the above idea, namely the proof of data possession (PDP) by Ateniese et al. (ACM CCS 2007) [14] and proof of retrievability (PoR) by Juels and Kaliski (ACM CCS 2007) [15]. Later, Shacham and Waters (Asiacrypt 2008) [16] proposed an improved PoR scheme that supports arbitrary number of auditing. Many following studies [17–25] and other further improvements extended from these studies are currently available in the literature. In [18], Ateniese et al. first considered scalable PDP, which supports data update operations. Later, Erway et al. [26] introduced dynamic PDP that allows insert, update, and delete operations. Other dynamic auditing schemes can be found in [27–29]. Data auditing with multi-replicas [20] was proposed to increase data retrieving probability and further improved by Armknecht et al. [30]. As for the auditing schemes in the identity-based settings, [22, 31–33] are introduced with different motivations; Li et al. [34] presented a fuzzy identity-based auditing scheme; examples of attribute-based auditing are [35, 36]. Shen and Tzeng [37] achieved delegatable PDP, where the data owner can grant auditing power to an auditor it identifies. More efficient delegatable constructions [38, 39] with lightweight tag generations are proposed. Post-quantum auditing schemes can be found in [40–42].

As shown, two popular constructions that work as the underlying building blocks are available among most cloud auditing schemes. In [14], the tag is similar to a RSA-type signature, and the ratio of data size to tag size can be $n : 1$ for a proper parameter n. It has high storage efficiency; however, the size of response is not compact enough since it is mainly a RSA-type signature. On the other hand, the tag in [16] is similar to a BLS-type signature, and the size of the response is compact. Nevertheless, the tag cannot be compressed in a straightforward way and so the ratio of data size to tag size is 1 : 1. The storage efficiency can achieve 50% at most. Since these two constructions have been proposed, it seems that an inherent tradeoff between storage and response size is inevitable, and we can choose only one aspect to improve efficiency. Until recently, a revolutionary technique named evolution tag was introduced in [43] to solve the aforementioned problem. It achieved high storage efficiency and compact response simultaneously for the first time.

1.3. Motivation and contribution

Static data transfer has efficient transfer process since no change on data or tag is required. However, the BLS-type auditing structure faces the storage efficiency problem. Moreover, the evolution tag technique [43] can improve storage efficiency but cannot be directly applied to support data transfer. In the healthcare systems, a patient may often transfer from one hospital to another; or the experimental data needs to be transferred to the next medical center in a joint medicine development procedure. There is much data that is involved in transfers. On one hand, efficient transfer is preferred to relieve the system computational burden. On the other hand, the storage efficiency is another aspect that we should pay attention to. As illustrated above, static data transfer requires less computation during the transfer process; while evolution tag can improve storage efficiency by aggregating tags. Coincidentally, both the two techniques are based on the HLA structure. We can combine these two techniques to achieve static data transfer and aggregated tags simultaneously.

In this paper, we show how to combine the static data transfer with the evolution tag technique. It is feasible because the two structures both rely on the BLS-type structure. By elaborately designing the transfer token, the tags can be aggregated and then extracted as evolution tags before or after data transfer. There is no extra computations or transmissions during the challenge and response phases. The major contributions of this paper are summarized as follows.

1) We first present the basic scheme for cloud auditing that supports static data transfer. The transfer token is introduced and the challenge phase needs to update. Neither the data nor the tags need to change during data transfer process.
2) To further improve storage efficiency, we combine the static data transfer with evolution tag technique. The definitions of static data transfer and the corresponding threat model are proposed.
3) We put forward a concrete construction of static data transfer auditing. The efficiency with respect to public key, tag, challenge, and response is comparable with other schemes.

1.4. Organization

The rest of this paper is organized as follows. In Section 2, we review some mathematical preliminaries, including homomorphic linear authenticator (HLA) and hardness assumption. Then, we give the definition of static data transfer auditing and the corresponding threat model in Section 3. Next, our concrete construction is presented in Section 4, together with the security and efficiency analysis. Finally, the conclusion part comes in Section 5.

2. Preliminaries

2.1. Homomorphic linear authenticator (HLA)

The core construction of homomorphic linear authenticator (HLA) in [16] is as follows.
Parameter. The public parameter includes pairing groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, h_2)$ and hash function H, with $g_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2, e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The pairing has an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ and $g_1 = \psi(h_2)$.
Key structure. The public and secret key pair is $pk = h_2^a, sk = a$ for $a \in \mathbb{Z}_p$.
Tag structure. A message M with file name f, parse M into $(m_1, m_2, \dots)$, where $m_i \in \mathbb{Z}_p$ to be stored at index i and the tag is

$$t_i = (H(f\|i) \cdot g_1^{m_i})^a.$$

Challenge structure. Choose a set of indexes I, randomly pick $r_i \in \mathbb{Z}_p$ for each $i \in I$, and the challenge is $chal = \{(i, r_i)\}$.
Response structure. For each $i \in chal$, find the message m_i and tag t_i, then compute

$$\mu = \sum_{(i,r_i)\in chal} m_i \cdot r_i \bmod p, \sigma = \prod_{(i,r_i)\in chal} t_i^{r_i}.$$

The response is $res = (\mu, \sigma)$.

2.2. HLA with evolution tag

The core idea to improve storage efficiency in the homomorphic linear authenticator (HLA) is to aggregate tags into an aggregated tag; and then "recover" tags from the aggregated tag for computing response. Please note that we cannot simply recover the original tags from the aggregated tags since the entropy of an aggregated tag is much smaller than the original tags. This is the reason why evolution tag is introduced and the evolution tags are the real input to compute responses. The innovation of utilizing evolution tags paved the way to achieving high storage efficiency and does not sacrifice the size of response. The comparison between the traditional HLA and evolution tag technique is shown in Figure 4. The underlying structures are both BLS-type. The sketch of evolution tag technique is described as follows.
Parameter. The public parameter is the same as the scheme in [16] with an additional hash function $H_2 : \{0, 1\}^* \to \mathbb{Z}_p$ and a parameter n. This parameter determines the size of public and secret key pair and the storage efficiency.
Key structure. The public and secret key pair is

$$pk = (h_2^a, h_2^{a^2}, \dots, h_2^{a^n}, h_2^{a^{n+1}}, \dots, h_2^{a^{2n}}),\ sk = a$$

Tag structure. For a message M with file name f, parse M into $(m_1, m_2, \dots)$, where $m_i \in \mathbb{Z}_p$ to be stored at index i. First, compute $i = j \cdot n + k$, where $k \in [1, n]$.* The tag is

$$t_i = (H(f\|i) \cdot g_1^{m_i})^{a^k}.$$

*Please note the integer k must between 1 to n, which is similar to the modular operation but different when $n|i$ since k cannot be 0 but can be n. To be more specific, k can be defined as $k = \begin{cases} n, & \text{if } n|i, \\ i \bmod n, & \text{if } n \nmid i. \end{cases}$

Then, aggregate every n tags into one aggregated tag $T_1 = \prod_{l=1}^{n} t_l, T_2 = \prod_{l=n+1}^{2n} t_l, \ldots$ and these aggregated tags $\{T_j\}$ will be uploaded.

Challenge structure. Choose a set of indexes I, randomly pick $c \in \mathbb{Z}_p$ and $r_i \in \mathbb{Z}_p$ for each $i \in I$. Compute $(h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}})$ and the challenge is

$$chal = (\{(i, r_i)\}, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}}).$$

Please note the item $h_2^{ca^{n+1}}$ serves as the secret value and is not included in the challenge.

Response structure. For each $i \in chal$, find the message m_i and the aggregated tag T_{j+1} that includes t_i. Then, compute the evolution tag

$$e_i = \frac{e(T_{j+1}, h_2^{ca^{n+1-k}})}{\prod_{l=jn+1, l\neq i}^{(j+1)n} e(H(f\|l) \cdot g_1^{m_l}, h_2^{ca^{n+1-i+l}})} = e(H(f\|i) \cdot g_1^{m_i}, h_2^{ca^{n+1}}).$$

Next, compute $\mu = \sum_{(i,r_i)\in chal} m_i \cdot r_i \bmod p, \sigma = \prod_{(i,r_i)\in chal} e_i^{r_i}$ and the response is $res = (\mu, \sigma)$.

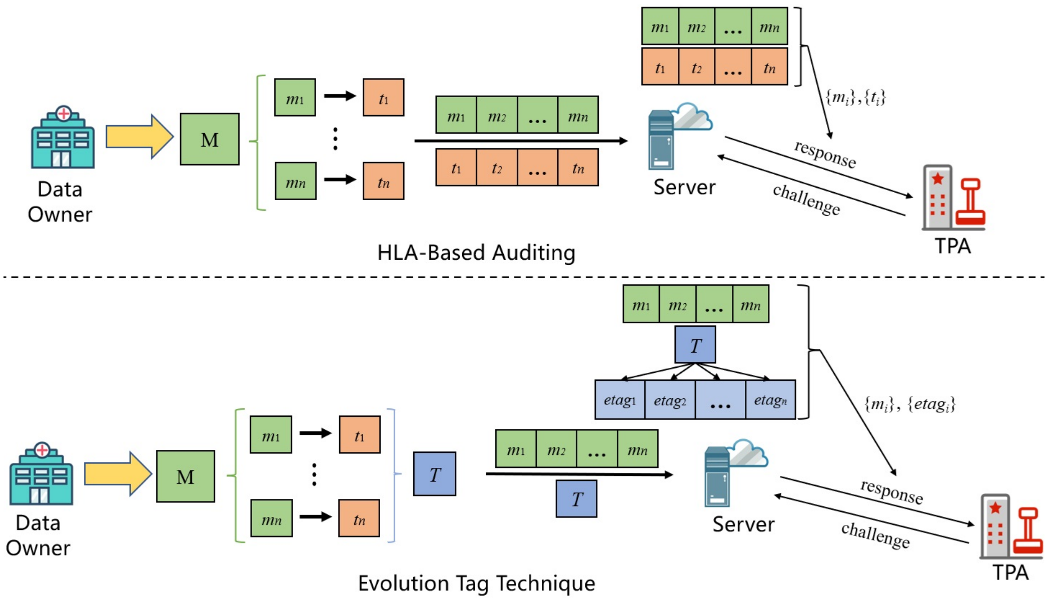

Figure 4. Comparison of the traditional HLA and evolution tag.

2.3. HLA with static data transfer

In this section, we first sketch the core idea to achieve static transfer. The data and tags remain "static" because the token is directly given from the original data owner to the new data owner. Moreover, the challenge algorithm is updated if the auditing request is launched by a new data owner. To achieve this goal, the token generation algorithm is added. Namely, the original owner A ($pk_A = h_2^a, sk_A = a$) computes the transfer token $tk_{A\to B}$ with its secret key a and the new owner B's public key $pk_B = h_2^b$, where the token $tk_{A\to B} = h_2^{\frac{b}{a}}$. Then, the original owner gives the token to the new owner. The new owner can send the auditing request with the token $tk_{A\to B}$ to the TPA. The token can also be verified by checking whether the equation $e(\psi(tk_{A\to B}), pk_A) = e(g_1, pk_B)$ holds. On the other hand, the challenge and response phases also need to update to support data transfer.

As analyzed, the static transfer process is guaranteed because the challenge phase is updated with the public key and token of the new data owner. Consequently, the difference between auditing before and after data transfer is implicitly conducted by distinguishing the challenge generation. Next, we give the description of the concrete construction as the basic scheme in this paper to illustrate how static transfer is obtained.

Parameter. The public parameter includes pairing groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, h_2)$ and hash function H, with $g_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2, e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The pairing has an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ and $g_1 = \psi(h_2)$.

Key structure. The public and secret key pair is $pk = h_2^a, sk = a$ for $a \in \mathbb{Z}_p$.

Tag structure. For a message M with file name f, parse M into $(m_1, m_2, \dots)$, where $m_i \in \mathbb{Z}_p$ to be stored at index i and the tag is

$$t_i = (H(f\|i) \cdot g_1^{m_i})^a.$$

Token structure. The transfer token $tk_{A\to B}$ from the original owner A ($pk_A = h_2^a, sk_A = a$) to new owner B ($pk_B = h_2^b$) is $tk_{A\to B} = h_2^{\frac{b}{a}}$.

Challenge structure. Choose a set of indexes I, randomly pick $r_i \in \mathbb{Z}_p$ For each $i \in I$, and the challenge is further divided as

- For data before transfer, pick a random $c \in \mathbb{Z}_p$, compute $cpk = h_2^c$ and $s = h_2^{ac}$ with $pk_A = h_2^a$;
- For data after transfer, pick a random $c \in \mathbb{Z}_p$, compute $cpk = h_2^{\frac{bc}{a}}$ and $s = h_2^{bc}$ with $tk_{A\to B} = h_2^{\frac{b}{a}}$ and $pk_B = h_2^b$.

The challenge is $chal = (\{(i, r_i)\}, cpk)$, and the value s is kept secret for verifying the response.

Response structure. For each $i \in chal$, find the message m_i and tag t_i, compute

$$\mu = \sum_{(i,r_i)\in chal} m_i \cdot r_i \bmod p, \sigma = e\left(\prod_{(i,r_i)\in chal} t_i^{r_i}, cpk\right).$$

The response is $res = (\mu, \sigma)$.

Verification structure. Given response $res = (\mu, \sigma)$ and the secret value s, check whether the equation $\sigma = e\left(\prod_{(i,r_i)\in chal} H(f\|i)^{r_i} \cdot g_1^{\mu}, s\right)$ holds.

Our basic scheme can be seen as the combination of the Shacham-Waters auditing scheme [16] with an additional token generation algorithm. To make it easier for comprehension, we give the comparison of scheme constructions in Figure 5.

Next, we summarize the core structures in HLA, evolution tag, and static data transfer in Table 2.

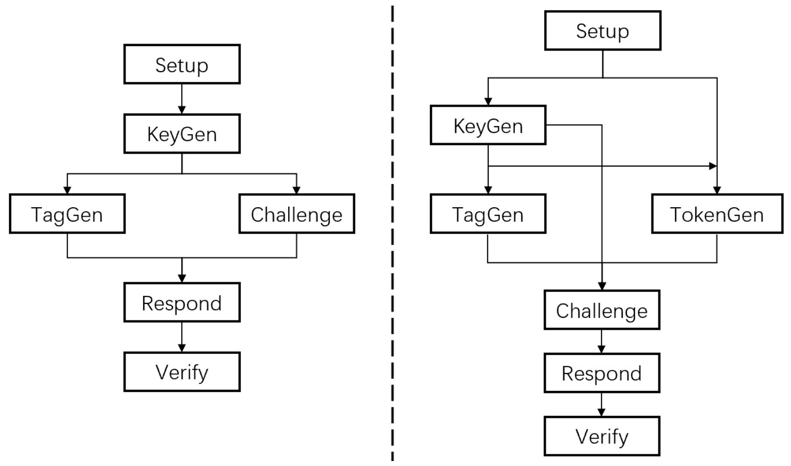

Figure 5. Comparison of traditional HLA (left) and our basic scheme (right).

Table 2. Comparison among HLA-based cloud auditing schemes.

	Shacham-Waters [16]	Static Transfer	Evolution Tag [43]
Key	$pk = h_2^a$, $sk = a$	$pk_A = h_2^a, pk_B = h_2^b$ $sk_A = a, sk_B = b$	$pk = (h_2^a, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, \ldots, h_2^{a^{2n}})$, $sk = a$
Tag	$t_i = (H(f\|\|i) \cdot g_1^{m_i})^a$	$t_i = (H(f\|\|i) \cdot g_1^{m_i})^a$	$t_i = (H(f\|\|i) \cdot g_1^{m_i})^{a^k}$
Token	-	$h_2^{\frac{b}{a}}$	-
Challenge	$chal = \{(i, r_i)\}$	$chal = (\{(i, r_i)\}, cpk)$, $cpk = h_2^c$ or $h_2^{\frac{bc}{a}}$	$chal = (\{(i, r_i)\}, cpk)$, $cpk = (h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}})$
Response	$\sigma = \prod_{(i,r_i)\in chal} t_i^{r_i} \in \mathbb{G}_1$	$\sigma = e\left(\prod_{(i,r_i)\in chal} t_i^{r_i}, cpk\right) \in \mathbb{G}_T$	$e_i = e(H(f\|\|i) \cdot g_1^{m_i}, h_2^{ca^{n+1}})$ $\sigma = \prod_{(i,r_i)\in chal} e_i^{r_i} \in \mathbb{G}_T$

2.4. Hardness assumption

In [44], Boneh et al. defined the Bilinear Diffie-Hellman Exponent (BDHE) problem for the pairing group $\mathbb{G}$ in a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. The detailed definition comes as follows.

Definition 1 (Bilinear Diffie-Hellman Exponent (BDHE) Problem)**.** *Given the following instances*

$$\begin{pmatrix} g, h \in \mathbb{G}, \\ g^a, g^{a^2}, \ldots, g^{a^{n-1}}, g^{a^{n+1}}, \ldots, g^{a^{2n}} \in \mathbb{G} \end{pmatrix},$$

the BDHE problem is to compute $e(g, h)^{a^n} \in \mathbb{G}_T$.

Based on the above definition, the n-General Diffie-Hellman Exponent (n-GDHE) problem was introduced in [43] for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, h_2)$, where $g_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

Definition 2 (n-General Diffie-Hellman Exponent (n-GDHE) Problem)**.** *Given the following instances*

$$\begin{pmatrix} g_1, g_1^r \in \mathbb{G}_1, \\ h_2, h_2^c \in \mathbb{G}_2, \\ g_1^a, g_1^{a^2}, \ldots, g_1^{a^n} \in \mathbb{G}_1, \\ h_2^a, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, h_2^{a^{n+2}}, \ldots, h_2^{a^{2n}} \in \mathbb{G}_2, \\ h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, \quad , h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}} \in \mathbb{G}_2 \end{pmatrix},$$

the n-GDHE problem is to compute $e(g_1^r, h_2)^{ca^{n+1}}$.

We say an algorithm $\mathcal{A}$ has advantage ϵ in solving the n-GDHE problem if

$$\Pr[\mathcal{A} \rightarrow e(g_1^r, h_2)^{ca^{n+1}}] \geq \epsilon.$$

The n-GDHE assumption holds in pairing groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, if no P.P.T. algorithm that runs within time t has advantage ϵ in solving the n-GDHE problem.

3. Definition and security model

3.1. Scheme definition

A cloud auditing scheme with static data transfer is composed of the following seven algorithms.
Setup $(1^\kappa, n) \rightarrow params$: The setup algorithm is run by the system organizer. It inputs the security parameter 1^κ and a parameter n, which determines the data storage efficiency, and outputs the system public parameter $params$.
KeyGen $(params) \rightarrow (pk, sk)$: The key generation algorithm is run by a user in the system. It inputs the system public parameter $params$ and outputs a pair of public and secret key (pk, sk) for the user.
TokenGen $(params, sk_A, pk_B) \rightarrow tk_{A\rightarrow B}$: The transfer token generation algorithm is run by the original data owner. It inputs the system public parameter $params$, the original data owner's secret key sk_A, and the new owner's public key pk_B, and outputs a transfer token $tk_{A\rightarrow B}$.
TagGen $(params, sk, M) \rightarrow T$: The tag generation algorithm is run by the data owner. It inputs the system public parameter $params$, the data owner's secret key sk, and the message M, and outputs the aggregated tag T for the message.
Challenge $(params, I, pk_A/(tk_{A\rightarrow B}, pk_B)) \rightarrow (chal, s)$: The challenge algorithm is run by the third party auditor. It inputs the system public parameter $params$, a set of indexes I selected for data auditing, and can be further divided into the following two types.

- For auditing data before transferring, the challenge algorithm also takes the original data owner's public key pk_A as input;
- For auditing data after transferring, the challenge algorithm also takes the transfer token $tk_{A\rightarrow B}$ provided by the new owner and its public key pk_B as input.

Then, the challenge algorithm outputs the challenge $chal$ and a secret value s, which will be used in later verification.
Respond $(params, chal, \{m_i\}, \{T_j\}) \rightarrow res$: The response algorithm is run by the cloud server. It inputs the system public parameter $params$, challenge $chal$, and the message block set $\{m_i\}$ with aggregated

tag set $\{T_j\}$ (the tag t_i for message block m_i is aggregated into T_j) that is stored in the locations that correspond to the indexes indicated in the challenge, and outputs the response res.
Verify $(params, chal, res, s) \to 1/0$: The verification algorithm is run by the third-party auditor. It inputs the system public parameter $params$, challenge $chal$, response res, and the secret value s generated during the challenge phase, and outputs the verification result $1/0$. Specifically, if the response passes the verification, it outputs 1; otherwise, it outputs 0.
Correctness. The correctness requirement of a transferable data auditing scheme is defined as follows. If the auditing request is launched by the original data owner before transferring, it is required that, for any public and secret key pair (pk, sk) generated by the key generation algorithm KeyGen, any message M and the corresponding tag T generated by the tag generation algorithm TagGen with the secret key sk, any challenge $chal$ generated by the challenge algorithm Challenge with the public key pk, and any response res generated by the response algorithm Respond with valid message blocks and aggregated tags, the verification is valid, i.e.,

$$\Pr[\mathsf{Verify}(params, chal, res, s) = 1] = 1.$$

If the auditing request is launched by the transfer target user after transferring, it is required that, for any public and secret key pairs $(pk_A, sk_A), (pk_B, sk_B)$ generated by the key generation algorithm KeyGen, any transfer token $tk_{A\to B}$ generated by the transfer token generation algorithm TokenGen, any message M and the corresponding tag T generated by the tag generation algorithm TagGen with the secret key sk_A, any challenge $chal$ generated by the challenge algorithm Challenge with the transfer token $tk_{A\to B}$ and public key pk_B, and any response res generated by the response algorithm Respond with valid message blocks and aggregated tag, the verification is valid, i.e.,

$$\Pr[\mathsf{Verify}(params, chal, res, s) = 1] = 1.$$

3.2. Threat model

The threat model of a cloud auditing scheme with static data transfer is defined through the following game.
Setup. The challenger runs the Setup and KeyGen algorithms of the scheme to get the public parameter $params$ and two key pairs of data owners $(pk_A, sk_A), (pk_B, sk_B)$. Then, it runs the TokenGen algorithm to get a transfer token $tk_{A\to B}$. Next, it returns $(params, pk_A, pk_B)$ to the adversary.
Tag query. The adversary can adaptively query tags for a message M, and the challenger runs the TagGen algorithm with the secret key sk_A to get the tag T and returns T to the adversary.
Verification query. This query involves the interactions between the challenger and the adversary. The adversary can adaptively make challenge requests, including the challenges before data transfer and after data transfer. The challenger runs the Challenge algorithm to get the challenge $chal$ and secret s will be used later for verification. Then, it gives the challenge $chal$ to the adversary. Next, the adversary sends the response res to the challenger. Finally, the challenger runs the Verify algorithm to get the result and returns it to the adversary.
Challenge. The adversary indicates a challenge request for an auditing before or after data transfer. The challenger runs the Challenge algorithm according to the adversary's choice to get the challenge $chal^*$ and secret value s^* used for later verification. Then, it returns $chal^*$ to the adversary.

Forge. Finally, the adversary outputs a response res^*.

The adversary wins the game if the following conditions hold.

1) The response is valid, i.e., $\mathsf{Verify}(params, chal^*, res^*, s^*) = 1$;
2) The adversary never queries the tag for the message stored in the index used to generate the challenge $chal^*$.

We say adversary $\mathcal{A}$ has advantage ϵ in breaking the scheme if

$$\Pr\left[\mathbf{Verify}(params, chal^*, res^*, s^*) = 1 \middle| \begin{matrix}(params, pk_A, pk_B), chal^* \leftarrow C \\ res^* \leftarrow \mathcal{A}(O_{\mathsf{tag}}, O_{\mathsf{vrf}})\end{matrix}\right] \geq \epsilon.$$

Definition 3. *We say the cloud auditing scheme with static data transfer is* (t, ϵ)*-secure, if for any P.P.T. adversary that runs within time t, its advantage in winning the above game is at most* ϵ*.*

4. Concrete construction

4.1. Scheme description

In this section, we present the concrete construction of the cloud auditing scheme with static data transfer. The details of the seven algorithms are depicted as follows.

Setup $(1^\kappa, n)$. On input the security parameter 1^κ and another parameter n, the setup algorithm first generates pairing groups $\mathbb{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, h_2, e)$ with an admissible pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and generators $g_1 \in \mathbb{G}_1, h_2 \in \mathbb{G}_2$. Then, it selects cryptographic hash functions $H : \{0,1\}^* \rightarrow \mathbb{G}_1, H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ and publishes the system public parameter $params = (\mathbb{PG}, H, H_2, n)$.

KeyGen $(params)$. On input the system public parameter $params$, the key generation algorithm selects a random $a \in \mathbb{Z}_p$ and computes $(h_2^a, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, \ldots, h_2^{a^{2n}})$. Then, it outputs a pair of public key and secret key

$$pk = \left(h_2^a, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, \ldots, h_2^{a^{2n}}\right), sk = a.$$

$\mathsf{TokenGen}$ $(params, sk_A, pk_B)$. On input the system public parameter $params$, the original data owner's secret key $sk_A = a$, and the new owner's public key $pk_B = (h_2^b, h_2^{b^2}, \ldots, h_2^{b^n}, h_2^{b^{n+1}}, \ldots, h_2^{b^{2n}})$, the transfer token generation algorithm computes

$$(h_2^{b^{n+1}})^{\frac{1}{a}} = h_2^{\frac{b^{n+1}}{a}}, \ldots, (h_2^{b^{n+1}})^{\frac{1}{a^n}} = h_2^{\frac{b^{n+1}}{a^n}}, (h_2^{b^{n+1}})^a = h_2^{ab^{n+1}}, \ldots, (h_2^{b^{n+1}})^{a^n} = h_2^{a^n b^{n+1}}.$$

Then, it outputs the transfer token $tk_{A\rightarrow B} = \left(h_2^{\frac{b^{n+1}}{a}}, \ldots, h_2^{\frac{b^{n+1}}{a^n}}, h_2^{ab^{n+1}}, \ldots, h_2^{a^n b^{n+1}}\right)$.

TagGen $(params, sk, M)$. On input the system public parameter $params$, secret key sk, and message M, the tag generation algorithm parses message M into elements in $\mathbb{Z}_p$, i.e., $M = (m_1, m_2, \ldots), m_i \in \mathbb{Z}_p$. Suppose the file name of the message is f, it first computes $i = j \cdot n + k$, for $k \in [1, n]$; then computes the tag t_i for m_i as

$$t_i = (H(f\|i) \cdot g_1^{m_i})^{a^k}.$$

Next, it aggregates every n tags into one aggregated tag as

$$T_1 = \prod_{l=1}^{n} t_l,\ T_2 = \prod_{l=n+1}^{2n} t_l, \ldots, T_{j+1} = \prod_{l=jn+1}^{(j+1)n} t_l.$$

The final tag T for message M consists of the aggregated tags $T = \{T_1, \ldots, T_{j+1}\}$.
Challenge $(params, I, pk_A/(tk_{A\to B}, pk_B))$. On input the system public parameter $params$, a randomly selected set of indexes I, the challenge algorithm picks a random $r_i \in \mathbb{Z}_p$ for each index i from set I and then proceeds as one of the following two branches.

- For auditing data before transferring, it additionally inputs the original data owner's public key $pk_A = (h_2^{a}, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, \ldots, h_2^{a^{2n}})$. Next, it picks a random $c \in \mathbb{Z}_p$ and computes

$$chal = \left(\{(i, r_i)\}, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}}\right).$$

  Please note the item $h_2^{ca^{n+1}}$ serves as the secret value s that will be used in later verification and is not included in the above tuple.
- For auditing data after transferring, it additionally inputs the transfer key

$$tk_{A\to B} = \left(h_2^{\frac{b^{n+1}}{a}}, \ldots, h_2^{\frac{b^{n+1}}{a^n}}, h_2^{ab^{n+1}}, \ldots, h_2^{a^{n-1}b^{n+1}}\right),$$

  and the transfer target user's public key $pk_B = (h_2^{b}, h_2^{b^2}, \ldots, h_2^{b^n}, h_2^{b^{n+1}}, \ldots, h_2^{b^{2n}})$. Next, it picks a random $c \in \mathbb{Z}_p$ and computes

$$chal = \left(\{(i, r_i)\}, h_2^{\frac{cb^{n+1}}{a}}, \ldots, h_2^{\frac{cb^{n+1}}{a^n}}, h_2^{cab^{n+1}}, \ldots, h_2^{ca^{n-1}b^{n+1}}\right).$$

  Please note that the item $h_2^{cb^{n+1}}$ (computed from $h_2^{b^{n+1}} \in pk_B$ and c) serves as the secret value s that will be used in later verification and is not included in the above tuple.

It outputs the challenge $chal$ and secret value s.
Respond $(params, chal, \{m_i\}, \{T_j\})$. On input, the system public parameter is $params$ and challenge is $chal$, and the response algorithm parses the challenge $chal$ to get the index set I and for each $i \in I$ computes $i = j \cdot n + k$, where $k \in [1, n]$. Then, it finds the message blocks m_i and the aggregated tags $\{T_{j+1}\}$ (contains t_i). Suppose the file name is f; then, it proceeds as one of the following two branches.

- For a challenge on the data before transferring, the challenge

$$chal = \left(\{(i, r_i)\}, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}}\right).$$

  Compute the evolution tag e_i from the aggregated tag $T_{j+1} = \prod_{l=jn+1}^{(j+1)n} t_l$ as

$$e_i = \frac{e\left(T_{j+1}, h_2^{ca^{n+1-k}}\right)}{\prod_{l=jn+1, l\neq i}^{(j+1)n} e\left(H(f\|l) \cdot g_1^{m_l}, h_2^{ca^{n+1-i+l}}\right)} = e\left(H(f\|i) \cdot g_1^{m_i}, h_2^{ca^{n+1}}\right).$$

- For a challenge on the data after transferring, the challenge

$$chal = \left(\{(i, r_i)\}, h_2^{\frac{cb^{n+1}}{a}}, \ldots, h_2^{\frac{cb^{n+1}}{a^n}}, h_2^{cab^{n+1}}, \ldots, h_2^{ca^{n-1}b^{n+1}}\right).$$

Compute the evolution tag e_i from the aggregated tag $T_{j+1} = \prod_{l=jn+1}^{(j+1)n} t_l$ as

$$e_i = \frac{e\left(T_{j+1}, h_2^{\frac{cb^{n+1}}{a^k}}\right)}{\prod_{l=jn+1, l\neq i}^{(j+1)n} e\left(H(f\|l) \cdot g_1^{m_l}, h_2^{ca^{l-i}b^{n+1}}\right)} = e\left(H(f\|i) \cdot g_1^{m_i}, h_2^{cb^{n+1}}\right).$$

Next, it computes $\mu = \sum_{(i,r_i)\in chal} m_i \cdot r_i \bmod p, \sigma = \prod_{(i,r_i)\in chal} e_i^{r_i}$ and outputs the response as $res = (\mu, \sigma)$.
Verify $(params, chal, res, s)$. On input, the system public parameter $params$, challenge $chal$, response res, and the secret value s, the verification algorithm parses $res = (\mu, \sigma)$ and checks whether the following equation holds:

$$\sigma = e\left(\prod_{i, r_i \in chal} H(f\|i)^{r_i} \cdot g_1^{\mu}, s\right).$$

If the above equation holds, it outputs 1; otherwise, it outputs 0.
Correctness. For simplicity, we consider one index i in the challenge and the general case that involves arbitrary number of indexes can be easily extended. For an auditing request before data transfer, the challenge is

$$chal = \left(i, r_i, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}}\right),$$

and the secret is $s = h_2^{ca^{n+1}}$. For message m_i and valid aggregated tag T_{j+1} stored at index i that includes tag t_i, the evolution tag $e_i = e(H(f\|i) \cdot g_1^{m_i}, h_2^{ca^{n+1}})$ is recovered during the response phase. Then, the response is computed as $\mu = m_i \cdot r_i \bmod p, \sigma = e_i^{r_i}$ and $res = (\mu, \sigma)$. Given the response and secret s, it is easy to see

$$\sigma = e_i^{r_i} = e\left(H(f\|i) \cdot g_1^{m_i}, h_2^{ca^{n+1}}\right)^{r_i} = e\left(H(f\|i)^{r_i} \cdot g_1^{\mu}), s\right).$$

For an auditing request after data transfer, the challenge is

$$chal = \left(i, r_i, h_2^{\frac{cb^{n+1}}{a}}, \ldots, h_2^{\frac{cb^{n+1}}{a^n}}, h_2^{cab^{n+1}}, \ldots, h_2^{ca^{n-1}b^{n+1}}\right),$$

and the secret is $s = h_2^{cb^{n+1}}$. Similarly, for message m_i and valid aggregated tag T_{j+1}, evolution tag $e_i = e(H(f\|i) \cdot g_1^{m_i}, h_2^{cb^{n+1}})$ is recovered during the response phase. Response $\mu = m_i \cdot r_i \bmod p, \sigma = e_i^{r_i}$ are computed and $res = (\mu, \sigma)$. It is easy to see that

$$\sigma = e_i^{r_i} = e\left(H(f\|i) \cdot g_1^{m_i}, h_2^{cb^{n+1}}\right)^{r_i} = e\left(H(f\|i)^{r_i} \cdot g_1^{\mu}), s\right).$$

Therefore, in both cases, we have Pr[Verify$(params, chal, res, s) = 1] = 1$. The correctness of the scheme holds.

4.2. Security analysis

In this section, we present the security analysis of our proposed scheme construction. As discussed, our scheme is built by combining the basic scheme with the evolution tag technique [43]. Next, we show that its security can also be reduced to the scheme in [43] and hence is based on the n-GDHE assumption.

Theorem 1. *If the cloud auditing scheme in [43] is (t, ϵ)-secure under the defined threat model, then our scheme is (t', ϵ')-secure under the threat model defined in Section 3.2, where $t' \approx t$ and $\epsilon' \approx \epsilon$.*

Please refer to the appendix for the details of proof.

4.3. Efficiency analyses

In this section, we give the performance comparison of our proposed scheme with related research in terms of computation and transmission costs. In addition to the underlying evolution tag technique in [43] that acts as the baseline, we also present comparisons with the scheme by Yan and Gui [45] of an identity-based cloud auditing, the scheme by Wang et al. [46] of a revocable cloud auditing, that are both designed for data sharing scenario, and the scheme by Shen et al. [5] that is proposed for data transfer purposes.

As for the computation costs, the detailed comparison in terms of tag generation, response, verification, and data transfer phases are presented in Table 3 as follows: From the table, we can see that there is not too much computational overhead in tag generation and verification phases of our scheme compared with other schemes that do not support data transfer or tag aggregation. Our scheme and [43] have a more complicated response phase since there is the need of computing the evolution tag. Nevertheless, these computations are conducted on the server side and do not involve any system user. Our scheme has an efficient transfer process, and $O(1)$ means it requires only simple and constant number operations on the server side. On the other hand, the schemes in [46] and [5] require linear number operations with respect to the size of the data to be transferred.

Table 3. Comparison of computation cost.

Scheme	TagGen	Respond	Verify	Data Transfer (tag calculating)
Susilo et al. [43]	$2nT_e$	$cnT_p + cnT_e$	$T_p + (c+1)T_e$	–
Yan & Gui [45]	$2nT_e$	$(c+1)T_p + 2cT_e$	$2T_p + (c-1)T_e$	–
Wang et al. [46]	$2nT_e$	cT_e	$3T_p + (c+2)T_e$	nT_e
Shen et al. [5]	$(3n+2)T_e$	cT_e	$3T_p + (c+3)T_e$	$4nT_e$
Our Scheme	$2nT_e$	$cnT_p + cnT_e$	$T_p + (c+1)T_e$	$O(1)$

T_e and T_p represent the time cost for pairing and exponentiation in $\mathbb{G}_1$, respectively. n is the number of tags that will be aggregated, which is an important index of storage efficiency in our scheme and [43]. c denotes the number of blocks that are selected in challenge. For fair comparison, the tag generation shows the computation complexity of generating n tags. The multiplication in group $\mathbb{G}_1$ is omitted since the exponentiation is the dominant factor that mainly determines the computational cost.

As for transmission costs, the detailed comparison in terms of data upload, challenge, response, and data transfer phases are presented in Table 4 as follows: From the table, we can see that the storage efficiency and cost during data upload phase of our scheme is the same as [43], which is more efficient compared with other schemes. The challenge phase requires more bandwidth since the public key and token consist of multiple elements, which is linear with parameter n. This can also be further improved using the same random $c \in \mathbb{Z}_p$ during the challenge phase and denoting the tuple $(h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^{2n}})$ as tpk used by TPA for every challenge. For data transfer, only the token needs to be delivered and is done once and for all. This means the cost is independent of the size of data or the time involved in the transfer.

Table 4. Comparison of transmission cost.

Scheme	Data Upload	Challenge	Response	Data Transfer
Susilo et al. [43]	$\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{Z}_p\rvert$	$2c\lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{Z}_p\rvert$	–
Yan & Gui [45]	$(n+1)\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{Z}_p\rvert + \lvert SIG\rvert$	$3\lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{Z}_p\rvert$	–
Wang et al. [46]	$n\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{Z}_p\rvert$	$2c\lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{G}_1\rvert + (2c+2)\lvert\mathbb{Z}_p\rvert$	$3\lvert\mathbb{Z}_p\rvert$
Shen et al. [5]	$n\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{Z}_p\rvert + \lvert SIG\rvert$	$2c\lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{Z}_p\rvert$	$3\lvert\mathbb{Z}_p\rvert$
Ours	$\lvert\mathbb{G}_1\rvert + n\lvert\mathbb{Z}_p\rvert$	$2c\lvert\mathbb{Z}_p\rvert + (2n-1)\lvert\mathbb{G}_2\rvert$	$2\lvert\mathbb{Z}_p\rvert$	$2n\lvert\mathbb{G}_2\rvert$

$|\mathbb{G}_1|$, $|\mathbb{G}_2|$, and $|\mathbb{Z}_p|$ denote the size of an element in group $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{Z}_p$, respectively. $|SIG|$ represents the size of the signature such as the BLS signature or RSA signature. For simplicity, we assume the message M contains exactly n blocks. Please note in the description of our scheme that we use the traditional HLA construction with respect to the response ($\mu \in \mathbb{Z}_p, \sigma \in \mathbb{G}_T$) and it can be easily improved to achieve the same size as [43] by adding a hash function and setting $res = (\mu, H(\sigma))$.

Next, we give the simulation results of our scheme together with the aforementioned schemes. The implementation details are as follows: The experiment is conducted on a laptop with Intel(R) Core(TM) i7-9750H CPU 2.60 GHz and a 16.0 GB RAM. All the algorithms are implemented with Java language with IDEA2024 compiler and 2.0.0 JPBC library. We use the symmetric bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with 80-bit security, which is known as the Type A curves in JPBC library. The curve is defined in equation $y^2 = x^3 + x \ (mod\ p)$, where p is a prime of 512 bits, and the order of the group is of $rBits = 160$ bits, with the embedding degree $k = 2$. We run each algorithm 50 times to get the average running time.

To show the effectiveness of the **TagGen**, **Respond**, and **Verify** algorithms, the simulation results of our scheme, [45], and [43] are presented, and their time costs are shown in Figures 6–8, respectively. The number of data blocks is incremented from 50 to 500. In the tag generation phase, we assume the message is composed of exactly the number of data blocks; in response and verification phases, we also set the number of challenged blocks from 50 to 500. The costs of tag generation, response, and verification increase linearly with the number of data blocks. The cost of scheme [45] is slightly higher since the scheme is in the shared data scenario. From Figures 6–8, we can see that our scheme and [43] behaves almost the same in terms of the three phases, which means that there is very little extra computation burden that gets involved to achieve static data transfer.

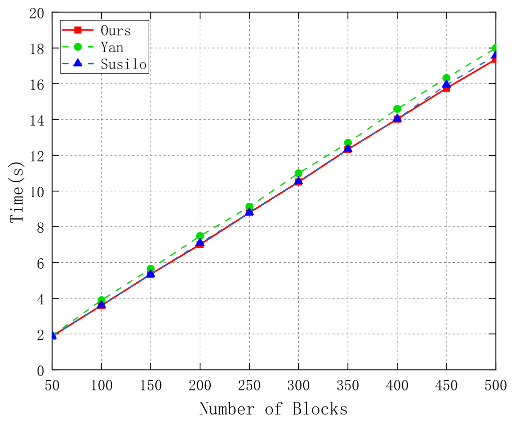

Figure 6. Time cost of tag generation.

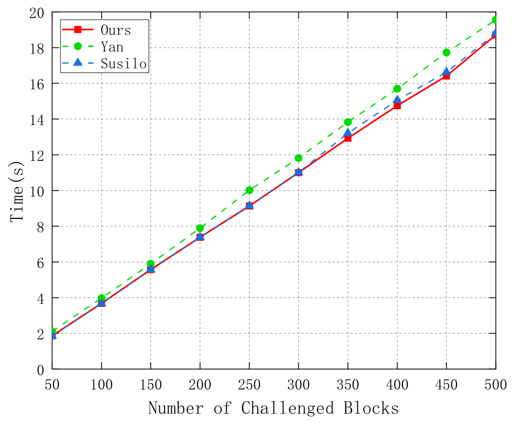

Figure 7. Time cost of response.

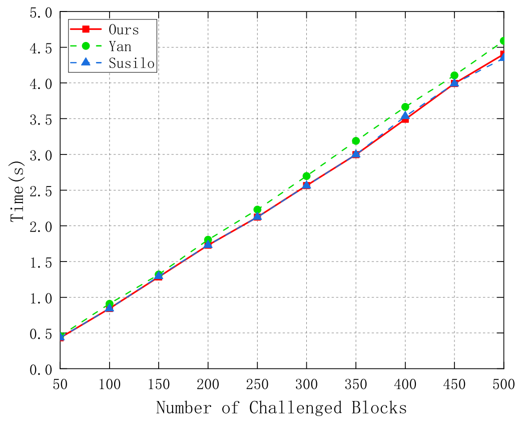

Figure 8. Time cost of verification.

To demonstrate the advantage of efficient data transfer in the static way, we give the simulation result of data transfer among our scheme, [46], and [5]. The number of data blocks is incremented from 500 to 5000. From Figure 9, we can see that the costs of scheme [46] and [5] increase linearly with the number of data blocks while it remains constant, and almost no computation operation is needed in our scheme. Due to static transfer property, efficiency is guaranteed.

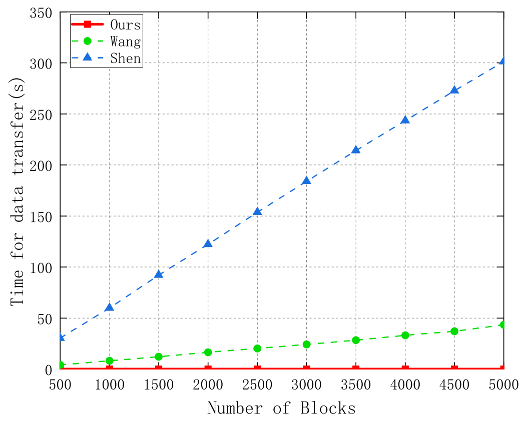

Figure 9. Cost of data transfer.

5. Conclusions

In this paper, we focused on how to achieve cloud auditing that supports efficient data transfer. We first introduced the notion of static transfer and defined the threat model. Since the data and tags remain unchanged during the transfer process, which is referred to as "static", the computation and transmission efficiency is guaranteed. A basic scheme was proposed to demonstrate the core technique. Furthermore, to pursue storage efficiency, we combined the evolution tag technique, which is also built upon the BLS-type. By elaborately designing the key, token, and challenge structure, we presented our complete cloud auditing scheme with static data transfer. According to the experimental results, the key, tag, challenge, and response phases remained the same, without computation or transmission efficiency. Moreover, the storage efficiency can also be improved since the tags are aggregated.

Nevertheless, there are limitations and possible improvements for the scheme. First, the parameter that indicates how many tags can be aggregated in evolution tag technique is settled when the system is setup. If the parameter can be adaptively updated, then the tradeoff between storage efficiency and computation costs can be more flexible. Second, when the data is transferred to the new owner, the new owner needs to store the transfer token for later use. This modifies the scheme from stateless to stateful. Although the size of the token is much smaller than the data or lower than the transmission cost, this may bring a passive influence to practical applications. Third, when the transfer is complete, the original owner may be able to grant a transfer token with its secret key. This situation can be avoided by introducing supervision mechanisms and taking responsibility for token abuse. Overall, the aforementioned extensions of the current construction are future research directions.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This paper is supported by National Natural Science Foundation of China (Nos. 62372103 and 62002058), Natural Science Foundation of Jiangsu Province (No. BK20200391), and Fundamental Research Funds for the Central Universities (No. 2242021R40011).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. G. Andrews, N. Titov, Treating people you never see: Internet-based treatment of the internalising mental disorders, *Aust. Health Rev.*, **34** (2010), 144–147. https://doi.org/10.1071/ah09775
2. P. Musiat, P. Goldstone, N. Tarrier, Understanding the acceptability of e-mental health-attitudes and expectations towards computerised self-help treatments for mental health problems, *BMC psychiatry*, **14** (2014), 1–8. https://doi.org/10.1186/1471-244x-14-109
3. S. Padmanaban, M. Zand, M. A. Nasab, M. Shahbazi, H. Nourizadeh, *Smart and Power Grid Systems–Design Challenges and Paradigms*, 1st edition, River Publishers, New York, 2022. https://doi.org/10.1201/9781003339557
4. M. Blaze, G. Bleumer, M. Strauss, Divertible protocols and atomic proxy cryptography, in *Advances in Cryptology-EUROCRYPT'98*, Springer, (1998), 127–144. https://doi.org/10.1007/BFb0054122
5. J. Shen, F. Guo, X. Chen, W. Susilo, Secure cloud auditing with efficient ownership transfer, in *Computer Security–ESORICS 2020*, Springer, (2020), 611–631. https://doi.org/10.1007/978-3-030-58951-6_30
6. J. Shen, X. Chen, J. Wei, F. Guo, W. Susilo, Blockchain-based accountable auditing with multi-ownership transfer, *IEEE Trans. Cloud Comput.*, **11** (2023), 2711–2724. https://doi.org/10.1109/TCC.2022.3224440
7. Y. Wang, W. You, Y. Zhang, A. Ye, L. Xu, Cloud EMRs auditing with decentralized (t, n)-threshold ownership transfer, *Cybersecurity*, **7** (2024), 53. https://doi.org/10.1186/s42400-024-00246-4
8. Y. Huang, W. Shen, J. Qin, Certificateless cloud storage auditing supporting data ownership transfer, *Comput. Secur.*, **139** (2024), 103738. https://doi.org/10.1016/j.cose.2024.103738
9. W. T. Liew, K. Tsai, J. Luo, M. Yang, Novel designated ownership transfer with grouping proof, in *IEEE Conference on Dependable and Secure Computing–DSC 2017*, (2017), 433–440. https://doi.org/10.1109/DESEC.2017.8073863
10. S. F. Aghili, H. Mala, M. Shojafar, P. Peris-Lopez, LACO: Lightweight three-factor authentication, access control and ownership transfer scheme for E-Health systems in IoT, *Future Gener. Comput. Syst.*, **96** (2019), 410–424. https://doi.org/10.1016/j.future.2019.02.020
11. R. Kumar, R. Tripathi, Data provenance and access control rules for ownership transfer using blockchain, *Int. J. Inf. Secur. Privacy*, **15** (2021), 87–112. https://doi.org/10.4018/IJISP.2021040105

12. M. Badakhshan, G. Gong, Privacy-preserving ownership transfer: Challenges and an outlined solution based on zero-knowledge proofs, in *9th IEEE World Forum on Internet of Things - WF-IoT 2023*, (2023), 1–9. https://doi.org/10.1109/WF-IoT58464.2023.10539396

13. M. Kiran, B. Ray, J. Hassan, A. Kashyap, V. Y. Chandrappa, Blockchain based secure ownership transfer protocol for smart objects in the internet of things, *Internet Things*, **25** (2024), 101002. https://doi.org/10.1016/j.iot.2023.101002

14. G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, et al., Provable data possession at untrusted stores, in *Proceedings of the 2007 ACM Conference on Computer and Communications Security*, (2007), 598–609. https://doi.org/10.1145/1315245.1315318

15. A. Juels, B. S. Kaliski, Pors: Proofs of retrievability for large files, in *Proceedings of the 2007 ACM Conference on Computer and Communications Security*, (2007), 584–597. https://doi.org/10.1145/1315245.1315317

16. H. Shacham, B. Waters, Compact proofs of retrievability, in *Advances in Cryptology - ASIACRYPT 2008*, Springer, (2008), 90–107. https://doi.org/10.1007/978-3-540-89255-7_7

17. F. Armknecht, J. Bohli, G. O. Karame, Z. Liu, C. A. Reuter, Outsourced proofs of retrievability, in *Proceedings of the 2014 ACM Conference on Computer and Communications Security - CCS 2014*, (2014), 831–843. https://doi.org/10.1145/2660267.2660310

18. G. Ateniese, R. D. Pietro, L. V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in *4th International ICST Conference on Security and Privacy in Communication Networks*, (2008), 1–10. https://doi.org/10.1145/1460877.1460889

19. K. D. Bowers, A. Juels, A. Oprea, Proofs of retrievability: Theory and implementation, in *Proceedings of the 1st ACM Cloud Computing Security Workshop*, (2009), 43–54. https://doi.org/10.1145/1655008.1655015

20. R. Curtmola, O. Khan, R. C. Burns, G. Ateniese, MR-PDP: Multiple-replica provable data possession, in *28th IEEE International Conference on Distributed Computing Systems*, (2008), 411–420. https://doi.org/10.1109/ICDCS.2008.68

21. D. Cash, A. Küpçü, D. Wichs, Dynamic proofs of retrievability via oblivious RAM, in *Advances in Cryptology - EUROCRYPT 2013*, Springer, (2013), 279–295. https://doi.org/10.1007/978-3-642-38348-9_17

22. Y. Yu, M. H. Au, G. Ateniese, X. Huang, W. Susilo, Y. Dai, et al., Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage, *IEEE Trans. Inf. Forensics Secur.*, **12** (2017), 767–778. https://doi.org/10.1109/TIFS.2016.2615853

23. K. Yang, X. Jia, An efficient and secure dynamic auditing protocol for data storage in cloud computing, *IEEE Trans. Parallel Distributed Syst.*, **24** (2013), 1717–1726. https://doi.org/10.1109/TPDS.2012.278

24. J. Yuan, S. Yu, Proofs of retrievability with public verifiability and constant communication cost in cloud, in *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, (2013), 19–26. https://doi.org/10.1145/2484402.2484408

25. J. Yuan, S. Yu, Pcpor: Public and constant-cost proofs of retrievability in cloud, *J. Comput. Secur.*, **23** (2015), 403–425. https://doi.org/10.3233/JCS-150525

26. C. C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, Dynamic provable data possession, in *Proceedings of the 2009 ACM Conference on Computer and Communications Security*, (2009), 213–222. https://doi.org/10.1145/269990

27. D. Cash, A. Küpçü, D. Wichs, Dynamic proofs of retrievability via oblivious RAM, *J. Cryptology*, **30** (2017), 22–57. https://doi.org/10.1007/s00145-015-9216-2

28. C. C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, Dynamic provable data possession, *ACM Trans. Inf. Syst. Secur.*, **17** (2015), 1–29. https://doi.org/10.1145/2699909

29. E. Stefanov, M. van Dijk, A. Juels, A. Oprea, Iris: A scalable cloud file system with efficient integrity checks, in *28th Annual Computer Security Applications Conference*, (2012), 229–238. https://doi.org/10.1145/2420950.2420985

30. F. Armknecht, L. Barman, J. Bohli, G. O. Karame, Mirror: Enabling proofs of data replication and retrievability in the cloud, in *25th USENIX Security Symposium-USENIX 2016*, (2016), 1051–1068.

31. W. Shen, J. Qin, J. Yu, R. Hao, J. Hu, Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage, *IEEE Trans. Inf. Forensics Secur.*, **14** (2019), 331–346. https://doi.org/10.1109/TIFS.2018.2850312

32. Y. Wang, Q. Wu, B. Qin, W. Shi, R. H. Deng, J. Hu, Identity-based data outsourcing with comprehensive auditing in clouds, *IEEE Trans. Inf. Forensics Secur.*, **12** (2017), 940–952. https://doi.org/10.1109/TIFS.2016.2646913

33. Y. Yu, L. Xue, M. H. Au, W. Susilo, J. Ni, Y. Zhang, et al., Cloud data integrity checking with an identity-based auditing mechanism from RSA, *Future Gener. Comput. Syst.*, **62** (2016), 85–91. https://doi.org/10.1016/j.future.2016.02.003

34. Y. Li, Y. Yu, G. Min, W. Susilo, J. Ni, K. R. Choo, Fuzzy identity-based data integrity auditing for reliable cloud storage systems, *IEEE Trans. Dependable Secure Comput.*, **16** (2019), 72–83. https://doi.org/10.1109/TDSC.2017.2662216

35. J. R. Gudeme, S. K. Pasupuleti, R. Kandukuri, Attribute-based public integrity auditing for shared data with efficient user revocation in cloud storage, *J. Ambient Intell. Hum. Comput.*, **12** (2021), 2019–2032. https://doi.org/10.1007/s12652-020-02302-6

36. Y. Yu, Y. Li, B. Yang, W. Susilo, G. Yang, J. Bai, Attribute-based cloud data integrity auditing for secure outsourced storage, *IEEE Trans. Emerging Top. Comput.*, **8** (2020), 377–390. https://doi.org/10.1109/TETC.2017.2759329

37. S. Shen, W. Tzeng, Delegable provable data possession for remote data in the clouds, in *Information and Communications Security*, Springer, (2011), 93–111. https://doi.org/10.1007/978-3-642-25243-3_8

38. J. Xu, A. Yang, J. Zhou, D. S. Wong, Lightweight delegatable proofs of storage, in *Computer Security-ESORICS 2016*, Springer, (2016), 324–343. https://doi.org/10.1007/978-3-319-45744-4_16

39. A. Yang, J. Xu, J. Weng, J. Zhou, D. S. Wong, Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage, *IEEE Trans. Cloud Comput.*, **9** (2021), 212–225. https://doi.org/10.1109/TCC.2018.2851256

40. H. Li, L. Liu, C. Lan, C. Wang, H. Guo, Lattice-based privacy-preserving and forward-secure cloud storage public auditing scheme, *IEEE Access*, **8** (2020), 86797–86809. https://doi.org/10.1109/ACCESS.2020.2991579

41. Z. Liu, Y. Liao, X. Yang, Y. He, K. Zhao, Identity-based remote data integrity checking of cloud storage from lattices, in *3rd International Conference on Big Data Computing and Communications - BIGCOM 2017*, (2017), 128–135. https://doi.org/10.1109/BIGCOM.2017.29

42. Y. Zhang, Y. Sang, Z. Xi, H. Zhong, Lattice based multi-replica remote data integrity checking for data storage on cloud, in *Parallel Architectures, Algorithms and Programming*, Springer, (2019), 440–451. https://doi.org/10.1007/978-981-15-2767-8_39

43. W. Susilo, Y. Li, F. Guo, J. Lai, G. Wu, Public cloud data auditing revisited: Removing the tradeoff between proof size and storage cost, in *Computer Security - ESORICS 2022*, Springer, (2022), 65–85. https://doi.org/10.1007/978-3-031-17146-8_4

44. D. Boneh, X. Boyen, E. Goh, Hierarchical identity based encryption with constant size ciphertext, in *Advances in Cryptology-EUROCRYPT 2005*, Springer, (2005), 440–456. https://doi.org/10.1007/11426639_26

45. H. Yan, W. Gui, Efficient identity-based public integrity auditing of shared data in cloud storage with user privacy preserving, *IEEE Access*, **9** (2021), 45822–45831. https://doi.org/10.1109/ACCESS.2021.3066497

46. B. Wang, B. Li, H. Li, Panda: Public auditing for shared data with efficient user revocation in the cloud, *IEEE Trans. Serv. Comput.*, **8** (2015), 92–106. https://doi.org/10.1109/TSC.2013.2295611

Appendix (Proof of Theorem 1)

Proof. Suppose there exists an algorithm $\mathcal{A}$ that has advantage ϵ' in breaking our scheme. We show how to construct an efficient algorithm $\mathcal{B}$ that has advantage $\epsilon \approx \epsilon'$ in breaking the scheme in [43]. The algorithm $\mathcal{B}$ interacts with algorithm $\mathcal{A}$ as follows.

Setup phase. Upon receiving the public parameter $params$ and public key pk of the scheme in [43], where $pk = (h_2^a, h_2^{a^2}, \ldots, h_2^{a^n}, h_2^{a^{n+1}}, \ldots, h_2^{a^{2n}})$, $\mathcal{B}$ sets $pk_A = pk$, picks a random $t \in \mathbb{Z}_p$, and sets $pk_B = ((h_2^a)^t, \ldots, (h_2^{a^n})^{t^n}, \ldots, (h_2^{a^{2n}})^{t^{2n}})$. Then, $\mathcal{B}$ returns $(params, pk_A, pk_B)$ to algorithm $\mathcal{A}$. Please note that the secret key $sk_B = b$ for data owner B is implicitly set as $b = a \cdot t$, which is also not known to $\mathcal{B}$.

Tag query phase. When $\mathcal{A}$ submits a tag query on a message block $m \in \mathbb{Z}_p$, $\mathcal{B}$ transmits the tag query to the challenger C and receives the tag t. Then, $\mathcal{B}$ returns t to $\mathcal{A}$. It is easy to see that $\mathcal{B}$ can always correctly answer tag queries without abort.

Verification query phase. When $\mathcal{A}$ submits a challenge request for verification query,

- If the challenge is before data transfer, $\mathcal{B}$ transmits the request to C and receives the challenge $chal$. Then, it gives $chal$ to $\mathcal{A}$ and receives the response res. Next, it transmits res to C and receives the result. Finally, $\mathcal{B}$ returns the result to $\mathcal{A}$. $\mathcal{B}$ can always correctly answer verification queries before data transfer.
- If the challenge is after data transfer, $\mathcal{B}$ transmits the request to C and receives the challenge $chal$, where $chal = (i, r_i, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}})$. Then, $\mathcal{B}$ computes

$$\left((h_2^{ca})^{t^{n+1}}, (h_2^{ca^2})^{t^{n+1}}, \ldots, (h_2^{ca^n})^{t^{n+1}}, (h_2^{ca^{n+2}})^{t^{n+1}}, \ldots, (h_2^{ca^{2n}})^{t^{n+1}}\right),$$

and it is easy to see that all the components required in the challenge can be reformulated as

$$h_2^{\frac{cb^{n+1}}{a}} = h_2^{\frac{ca^{n+1}\cdot t^{n+1}}{a}} = h_2^{ca^n\cdot t^{n+1}}, \ldots, h_2^{\frac{cb^{n+1}}{a^n}} = h_2^{\frac{ca^{n+1}\cdot t^{n+1}}{a^n}} = h_2^{ca\cdot t^{n+1}},$$

$$h_2^{cab^{n+1}} = h_2^{caa^{n+1}\cdot t^{n+1}} = h_2^{ca^{n+2}\cdot t^{n+1}}, \ldots, h_2^{ca^{n-1}b^{n+1}} = h_2^{ca^{n-1}a^{n+1}\cdot t^{n+1}} = h_2^{ca^{2n}\cdot t^{n+1}}.$$

Next, $\mathcal{B}$ sets $chal_B = \left(i, r_i, h_2^{ca\cdot t^{n+1}}, \ldots, h_2^{ca^n\cdot t^{n+1}}, h_2^{ca^{n+2}\cdot t^{n+1}}, \ldots, h_2^{ca^{2n}\cdot t^{n+1}}\right)$ and returns $chal_B$ to $\mathcal{A}$. On receiving the response res from $\mathcal{A}$, which contain the item $e_i = e(H(f\|i)\cdot g_1^{m_i}, h_2^{cb^{n+1}}) = e(H(f\|i)\cdot g_1^{m_i}, h_2^{ca^{n+1}\cdot t^{n+1}})$, $\mathcal{B}$ first computes $e_i^{1/t^{n+1}} = e(H(f\|i)\cdot g_1^{m_i}, h_2^{ca^{n+1}})$ and then transmits the updated item $e_i^{1/t^{n+1}}$ to C. Finally, $\mathcal{B}$ receives the result from C and returns the result to $\mathcal{A}$. The validness of the results to both $\mathcal{B}$ and $\mathcal{A}$ are the same and hence $\mathcal{B}$ can always correctly answer verification queries after data transfer.

Challenge phase. $\mathcal{A}$ submits the challenge request on an index i^* that it never makes a tag query before and identifies the challenge is before or after data transfer.

- If the challenge is before data transfer, $\mathcal{B}$ transmits the request to C and receives the challenge $chal_A$. Then, $\mathcal{B}$ sets the challenge $chal^*$ as $chal^* = chal_A$;
- If the challenge is after data transfer, $\mathcal{B}$ transmits the request to C and receives the challenge $chal_A$. Similar to the above phase, $\mathcal{B}$ parses $chal_A$ as $(i^*, r_i^*, h_2^{ca}, h_2^{ca^2}, \ldots, h_2^{ca^n}, h_2^{ca^{n+2}}, \ldots, h_2^{ca^{2n}})$, computes and sets $chal^* = \left(i^*, r_i^*, h_2^{ca^n\cdot t^{n+1}}, \ldots, h_2^{ca\cdot t^{n+1}}, h_2^{ca^{n+2}\cdot t^{n+1}}, \ldots, h_2^{ca^{2n}\cdot t^{n+1}}\right)$.

Then, it returns the challenge $chal^*$ to $\mathcal{A}$.

Forge phase. $\mathcal{A}$ outputs a valid response $res^* = (\mu^*, \sigma^*)$.

- If $\mathcal{A}$ chooses the challenge before data transfer, then res^* is also valid for the challenge $chal_A$ received from challenger C. $\mathcal{B}$ transmits res^* to C;
- If $\mathcal{A}$ chooses the challenge after data transfer, then from the analysis above, we have

$$\sigma^* = e\left(H(f\|i^*)\cdot g_1^{m_{i^*}}, h_2^{cb^{n+1}}\right) = e\left(H(f\|i^*)\cdot g_1^{m_{i^*}}, h_2^{ca^{n+1}\cdot t^{n+1}}\right).$$

  Next, $\mathcal{B}$ computes $(\sigma^*)^{1/t^{n+1}} = e(H(f\|i^*)\cdot g_1^{m_{i^*}}, h_2^{ca^{n+1}})$, updates the response as $res^* = \left(\mu^*, (\sigma^*)^{1/t^{n+1}}\right)$ with the above item, and transmits res^* to C.

Since res^* outputted by $\mathcal{A}$ is valid, the updated response submitted by $\mathcal{B}$ is also valid and hence $\mathcal{B}$ also wins the game against the challenger of scheme in [43].

During the above phases, $\mathcal{B}$ can answer the queries and there is no abort. Therefore, we construct an algorithm that runs within polynomial time t' and has an advantage ϵ', which is approximately to ϵ. This completes the proof of Theorem 1.