







Enabling Hidden Frequency Keyword-Based Auditing on Distributed Architectures for a Smart Government

Jingting Xue^{1,2}(✉) , Shuqin Luo¹(✉) , Lingjie Shi¹ , Xiaojun Zhang^{1,3} ,
and Chunxiang Xu² 

¹ School of Computer Science, Research Center for Cyber Security, Southwest
Petroleum University, Chengdu 610599, China
jtxue@swpu.edu.cn, shuqin_luo0108@163.com

² School of Computer Science and Engineering, University of Electronic Science
and Technology of China, Chengdu 610097, China

³ School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai
200030, China

Abstract. Public auditing checks the integrity of outsourced data via random sampling and verifying sample data blocks. In practice, however, users do not pay attention to the entire data set but focus on the integrity of only the part of the data containing keywords of interest. Therefore, the keyword-based auditing paradigm is proposed; it depends entirely on the subjective choice or access habits, which makes it possible for malicious storage servers to analyze the auditing frequency, or reduce redundant backups. For government data, auditing frequency privacy leakage or corruption of any file could be catastrophic. In this paper, we propose a hidden frequency keyword-based auditing scheme for a smart government named HFKA, which is compatible with distributed storage architecture. HFKA leverages a Bloom filter, which adjusts the false positive rate to consider auditing files corresponding to specified keywords and auditing random files obtained via fuzzy matching. To obtain privacy-preserving fuzzy matching, HFKA constructs an index table embedded with update times to retrieve a wide range of files to be audited. This approach is secure against the replay attack and supports the index table update through structure iteration instead of recalculation. HFKA provides storage robustness, privacy protection of hidden frequencies, and data security. Additionally, HFKA can reduce audit computation overhead by 32.6% compared to the probabilistic public auditing.

Keywords: Keyword-based auditing · Frequency hiding · Distributed architecture · Smart government · Fuzzy matching

1 Introduction

With the emergence of information islands and the increasing volume of government data, a smart government [1] is being developed to facilitate the integration

of government management services, as evidenced by the realization of data sharing between government departments at all levels. The cloud [2] is the available intermediate carrier for data storage and sharing, and the public integrity auditing [3, 5–7, 9, 10] paradigm can be used to check the integrity of outsourced data to support its availability. However, for massive and targeted government data, the existing probabilistic public auditing (PPA) model directly applied to smart government scenarios has the following two limitations. (i) The PPA model does not support targeted auditing of government data. Specifically, civil servants in different departments pay attention to the integrity of only those files about their own departments rather than the entire outsourced database. The challenge information in the PPA model cannot completely cover the involved data, and therefore, cannot complete integrity auditing of the specified data. (ii) PPA, which evenly selects data to be audited, is not economical for government data verification. In each audit, PPA consumes unnecessary computation resources to check irrelevant randomly selected files in addition to the department files. In summary, for smart government, the proposed public keyword-based auditing (PKA) model [4] that determines the challenge information according to users' wishes could be a better choice.

In the PKA model, the third-party auditor (TPA) periodically verifies the users' data of interest integrity by retrieving specific keywords and performing verifications, thereby reducing the overall cost of auditing while satisfying their needs. Unfortunately, the PKA model cannot fully meet the security requirements of data auditing and sharing in smart government. In general, various departments' keyword setting is a long-term and business-related process to a certain extent, which results in the files selected for auditing being relatively fixed. The probability of these kinds of unrelated files being audited is negligible. In this case, malicious cloud servers can infer privacy contexts such as department information or file types, based on the auditing frequency and may even delete files that are rarely retrieved to save storage space. Undoubtedly, the value of government data is enormous. Either a privacy leak or a file corruption can expose a government to a major security crisis.

To ensure the security requirements of government data in integrity auditing, we propose HFKA that combines PPA and PKA. Specifically, the contributions of this work are as follows:

- We propose the first hidden frequency keyword-based auditing scheme for a smart government, named HFKA. In HFKA, a Bloom filter is introduced to achieve fuzzy matching between user-specified keywords and files to be audited. According to the degree of confidentiality requirements, HFKA sets Bloom filter modes with different fuzziness levels. The generated verifiable challenge information can not only cover the files that users are interested in but also randomly select some low-frequency auditing files. HFKA is well-suited for scenarios where users perform differential targeted auditing on specific data in a shared dataset.
- We design an index table label (ITL) to support the implementation of fuzzy matching that can protect keyword contents from storage servers and resist

two lazy behaviors. The ITL aggregates the serial numbers of all files corresponding to a defined keyword, so the aggregated value determines whether a storage proof covers all the involved files in the challenge information. In addition, a global variable indicating the update time is embedded in the ITL. When a storage server does not perform update operations according to the storage protocol, HFKA can identify replay attacks on proofs in a new round of auditing.

- We demonstrate the storage robustness, privacy protection regarding auditing frequency and data security of HFKA. The performance evaluation shows that HFKA maintains the computation complexity and communication complexity at $O(1)$ in terms of the total number of files n during the integrity auditing phase. Furthermore, we design a comparative experiment to observe the auditing frequency of *Type-A* files (files corresponding to the predefined high-frequency keywords from all extracted keywords) and *Type-B* files (files corresponding to the other keywords). The experimental results show that it is negligible to distinguish user-specified high-frequency files from randomly selected low-frequency files. Our scheme achieves the auditing frequency hiding and satisfies the security requirements of government data.

Organization. The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 covers the system model, threat model and definitions, and briefly introduces design goals of HFKA. Section 4 presents a detailed construction of HFKA. Section 5 provides a security analysis, and Sect. 6 presents the performance evaluation. Finally, Sect. 7 concludes the paper and gives directions for future work.

2 Related Work

Data integrity auditing enables users to verify the integrity of outsourced data. In 2007, Juels et al. [6] proposed the proofs of retrievability (PoR) protocol to enable a user to recover complete data from partial data provided by a server. Ateniese et al. [5] introduced a probabilistic data possession (PDP) proof generation model that can ensure overall data integrity by randomly sampling certain data. Considering the computation overhead and economic burden of users to verify data in a cloud environment, Wang et al. [7] achieves public verifiability by introducing a TPA based on the PoR model to complete auditing work. Then, Wang et al. [9] implemented storage correctness assurance and error localization based on a homomorphic token and an erasure-coded approach [8]. In [10], to resist an honest but curious TPA with malicious behavior, based on a homomorphic linear authenticator with a masking technique, TPA was designed to audit user outsourced data integrity without knowing the data content. Similarly, masking was used to improve the system decentralization and storage efficiency in [13–15]. To protect conditional identity privacy in medical data, Zhang et al. [12] designed an identity-based aggregated signature to protect patients' real identity and used Ethereum blockchain to record TPA's auditing results, thereby preventing a dishonest TPA from performing malicious auditing behaviors. To

address the authenticity of big data streams in untrusted environments, a novel data structure, P-ATHAT, was constructed based on the BLS signature and Merkle hash tree to achieve real-time authentication of data streams in [16]. Li et al. [17] proved that the security of P-ATHAT is unable to resist forgery attacks on cloud servers. Shen et al. [19] focused on identity privacy using the user's biological data to verify their identity; the method can perform integrity auditing under the condition that there is no hardware token to store the private key. Zheng et al. [20] protected a user's private key by updating it. To reduce the burden of recalculating the private key, a TPA participates in generating part of the private key. Zhou et al. [11] discussed the use of certificateless signatures to avoid the management and computation problems of certificates in the case of multicopy storage. Notably, these schemes all use the probabilistic auditing model of PDP to audit the integrity of the user's private data.

For integrity verification of shared data, different auditing models should be selected depending on the specific scenario. Take smart government as an example. Users in different departments have different concerns about outsourced shared data, and the probabilistic auditing model cannot meet the individual needs. In 2021, Gao et al. [4] proposed a keyword-based auditing paradigm that determined the scope of audited data based on keywords selected by users. In addition, a feasible keyword-based auditing scheme was achieved using the privacy-preserving keyword-file index table designed by Ge et al. in [21] and referred to the trapdoors in [22]. As a result of the index table design, which adopted a linked-list structure with an index vector and generated an authentication label for each keyword, storage space is saved and it is easier for the user to detect malicious behavior of the cloud server based on the authentication label. The content in trapdoor in [22] is ciphertext, meaning that without access to user's encrypted key, no one can forge an efficient trapdoor or crack the information included in the trapdoor. It also supported multiple keywords submitted into a trapdoor.

However, since users have attention preferences for certain data, semihonest cloud servers can infer user privacy by considering how often certain data are audited. Therefore, we need a new paradigm to address the challenge of auditing frequency privacy in this particular scenario. Bringer et al. [23] first used the locality-sensitive hash (LSH) algorithm to allow fuzzy search. LSH has been used in many schemes as a basic technique for fuzzing keywords [25–28]. Li et al. [24] chose to use edit-distance to measure the similarity between keywords for imprecise fuzzy search. Furthermore, the Bloom filter is a widely used fuzzy tool. In 2021, Indra et al. [30] designed a two-dimensional bloom matrix to achieve fast matching of similar words. Tong et al. [31] designed a twins Bloom filter with LSH in 2022 that combined the above approaches. The coin has two sides, and the fuzzy operations of the above schemes aim to narrow down the trapdoor search and improve the keyword search accuracy, which is the opposite of our intended effect of amplifying the hidden keyword frequency in the search results. Only the application of a Bloom filter by Gervais et al. [32] to fuzzy user's address in the process of simple payment verification fits the intended purpose perfectly.

There has recently been a high frequency of centralized storage accidents, but in smart government, any corruption of shared data is unacceptable. Consequently, distributed storage architecture is available. The distributed storage system, Hyperledger Fabric, is an ideal architecture for data storage designed by Cachin et al. [39]. Furthermore, Xu et al. [18] proved the feasibility of Hyperledger Fabric architecture when discussing how to achieve distributed storage integrity auditing. More privacy-preserving schemes are summarized in [40].

3 Preliminaries

3.1 System Model

On the basis of previous work [4], combined with the *Hyperledger Fabric* architecture¹, HFKA introduces the *Fabric certificate authority* and optimizes the traditional single cloud storage server into a scalable distributed storage structure. In addition, HFKA introduces a retrieval server to modularize the entity’s functionality.

The system model of HFKA involves five entities as illustrated in Fig. 1: *Fabric Certificate Authority (FCA)*, *User (U)*, *Third-Party Auditor (TPA)*, *Retrieval Server (RS)*, *Storage Node (SN)*.

- *Fabric Certificate Authority.* The *FCA* is a trusted institution that generates and determines digital certificates on the basis of public key infrastructure (PKI). According to the auditing requirements of smart government, the *FCA*

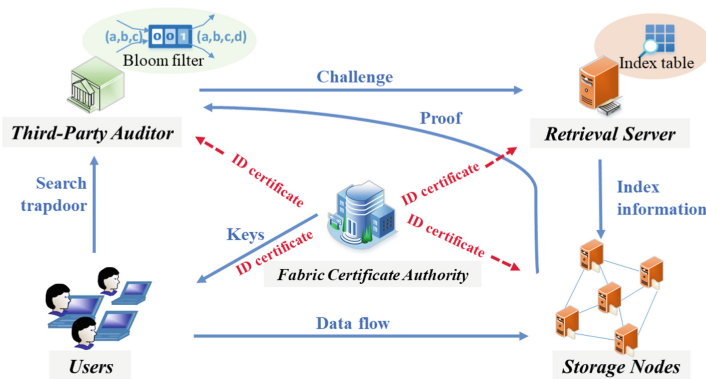


Fig. 1. System model of HFKA

¹ Hyperledger Fabric architecture is a scalable architectural design, an open interface style, and pluggable components. It first introduced authority management into the blockchain field, and its authority management function was completed by its independent Fabric certificate authority module. Hyperledger Fabric provides an important architectural reference for the design and implementation of distributed platforms.

mainly issues identity certificates of various entities and generates the signature key pairs for \mathcal{U} .

- *User*. Each \mathcal{U} is a collator and uploader of some government outsourced data. According to the construction goal of the smart government platform, \mathcal{U} s of each department outsource files to a distributed storage network, involving many \mathcal{SN} s. To ensure the recoverability and confidentiality of outsourced data, each \mathcal{U} completes data preprocessing, such as redundancy and encryption. During the auditing phase, each \mathcal{U} generates authenticators and index information.
- *Third-Party Auditor*. The \mathcal{TPA} performs audit tasks on behalf of \mathcal{U} s, with expertise and corresponding computing resources. The \mathcal{TPA} completes the fuzzy matching of the specified keywords, challenges the \mathcal{RS} in the auditing phase, and finally checks the validity of the proof information fed back by \mathcal{SN} s.
- *Retrieval Server*. The \mathcal{RS} assists the \mathcal{TPA} in completing the generation of the audited files set, specifically including saving index information and retrieving corresponding files according to the keyword trapdoor.
- *Storage Node*. Each \mathcal{SN} is a storage unit in the distributed storage architecture that stores redundantly processed data blocks and calculates storage proofs based on challenge information.

Here, we briefly describe the relationships among the various entities in the system model of HFKA. To free up local storage space and realize data sharing, the involved \mathcal{U} s outsource the preprocessed (encrypted and redundantly processed) data to a distributed storage architecture with multiple \mathcal{SN} s. The \mathcal{TPA} is authorized to perform periodic integrity auditing tasks, aiming to determine the integrity of outsourced data with the least amount of user-side communication and computational overhead. The \mathcal{RS} assists \mathcal{SN} s with completing the serial number confirmation of the audited data during the challenge phase. The \mathcal{FCA} creates signature keys for \mathcal{U} s and offers ID certificate management for each entity to assist the security of distributed storage systems.

3.2 Threat Model

In the threat model, \mathcal{RS} and \mathcal{U} are assumed to be completely trusted, but \mathcal{SN} is considered to be semihonest and \mathcal{TPA} is considered to be curious. The above security assumptions are consistent with the situation of the interests in the actual smart government.

Semihonest \mathcal{SN} . The \mathcal{SN} may detect data privacy by utilizing keyword-based auditing or a replay storage proof with exited proof to reduce the computational overhead of data updating. The details of the malicious behavior of a semihonest \mathcal{SN} are as follows: (i) *Frequency analysis attack*. In the challenge generating phase, the \mathcal{TPA} and the \mathcal{RS} select the auditing target data based on the search trapdoor input by the \mathcal{U} . \mathcal{U} 's subjectivity makes the auditing distribution concentrated in a small number of files. A curious \mathcal{SN} can analyze the auditing frequency of certain data and delete those data that are rarely audited to save

storage space. Additionally, a curious \mathcal{SN} can combine these data with external information (such as \mathcal{U} 's department) to guess the value and types of files with high auditing frequency. (ii) *Replay attack*. When the \mathcal{U} updates the outsourced data, a lazy \mathcal{SN} does not perform the updating operation. The generation of a storage proof is computed by the \mathcal{SN} completely, even though the \mathcal{SN} does not generate storage proofs based on the challenge information. Because the verified proof has the verification logic, it can pass the \mathcal{TPA} 's verification. In fact, the random number selection range of the \mathcal{TPA} is also limited. Although it is extremely unlikely that two groups of challenge information are exactly the same, the probability becomes nonnegligible after multiple rounds of auditing. It is more difficult to be distinguished in this case. Once the attack occurs, whether due to hardware/software failures or to reduce computational costs, the behavior violates cloud storage principles.

Curious TPA. A curious \mathcal{TPA} does not initiate attacks, but will eavesdrop on all kinds of private data. (iii) *Privacy speculation*. In HFKA, a curious \mathcal{TPA} may guess the type of outsourced files and further infer a \mathcal{U} 's identity based on the keywords in the \mathcal{U} 's search trapdoor. As the number of auditing rounds increases, the \mathcal{TPA} learns more information. Based on the \mathcal{U} 's identity and the keywords entered in the search trapdoor, the \mathcal{TPA} can detect the general content of the files. This level of confidentiality leakage is unacceptable for government data.

Based on the above discussion, we provide three relevant definitions for the security of HFKA.

Definition 2 (*Storage Robustness.*) *Storage robustness means that HFKA can resist \mathcal{SN} 's replay attack and frequency analysis attack.*

Definition 3 (*Privacy Protection.*) *Privacy protection means that not only outsourced data context privacy, as well as keywords specified by user, index and keyword-file relation privacy can be guaranteed.*

Definition 4 (*Data Security.*) *Data security means that any adversary cannot detect original files in a direct or indirect manner, such as via brute force attack or ciphertext-chosen attack.*

3.3 Design Goals

To achieve a secure and efficient application of keyword-based auditing for smart government, HFKA aims to achieve the following goals: (i) *Privacy preserving*. HFKA hides information about files, keywords, keyword-file index table and trapdoor using encryption and other processing methods so that \mathcal{SN} s and \mathcal{RS} cannot obtain sensitive information about files. Even if the data are intercepted by an adversary in the middle of network transmission, no information can be leaked, even the correspondence between files and keywords. (ii) *Even auditing*. Even auditing makes it impossible for adversaries to capture file auditing regularity, which is an important feature to protect data integrity. For the traditional PPA auditing model, the probabilistic auditing framework aims to achieve even auditing. However, the considerable auditing overhead is unacceptable for smart government due to its absolute evenness. HFKA overcomes the limitations of

the above models and achieves relatively even auditing of the files that are and are not of concern to \mathcal{U} and is suitable for the smart government scenario. (iii) *Replay attack resistance*. To further ensure data security, HFKA resists replay attacks, focusing on resisting the replay of stored proofs of updated data. The \mathcal{TPA} can verify whether the storage proofs come from the updated data instead of the old data, which prevents \mathcal{SN} s from not updating the data to reduce computational overhead or because of hardware/software failures. Meanwhile, the calculation of the design for replay attack resistance is constant.

4 The Proposed Scheme

An \mathcal{FCA} , multiple \mathcal{U} s, a \mathcal{TPA} , an \mathcal{RS} and multiple \mathcal{SN} s in the distributed storage architecture are involved in HFKA. For readability, we show a \mathcal{U} and an \mathcal{SN} that interact with other entities, as shown in Fig. 2.

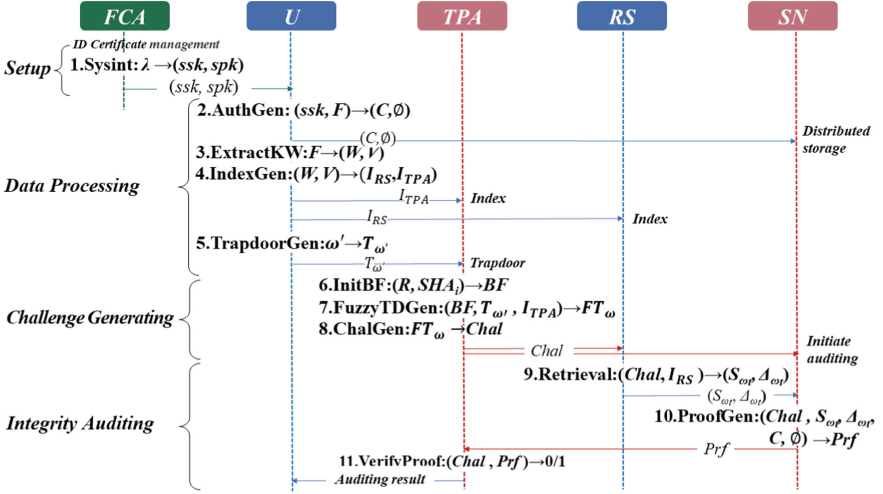


Fig. 2. Procedure of HFKA

Setup phase.

(1) **SysInit**. Given a secure parameter λ , \mathcal{FCA} chooses the public parameters and issues identification certificates for all entities and generates signature key pairs for \mathcal{U} .

- Initialize the public parameters: choose two multiplicative cyclic groups of q -order $G_1(g), G_2(u)$; a bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$; three hash functions $H_1, H_2, H_3 : \{0, 1\}^* \rightarrow G_1$; secure hash functions $SHA_i : \{0, 1\}^* \rightarrow Z_q^*$, $i \in \{1, 2, \dots, \mathcal{R}\}$; a symmetric encryption algorithm: $Enc(\cdot, key)$; a pseudo random permutation PRP: $\pi(\cdot, key)$ and a pseudo random function PRF: $f(\cdot, key)$.
- Issue the identification certificates for \mathcal{U} s, \mathcal{RS} , \mathcal{SN} s and \mathcal{TPA} . There is an example to illustrate the structure of each entity's certificate as shown in Fig. 3.

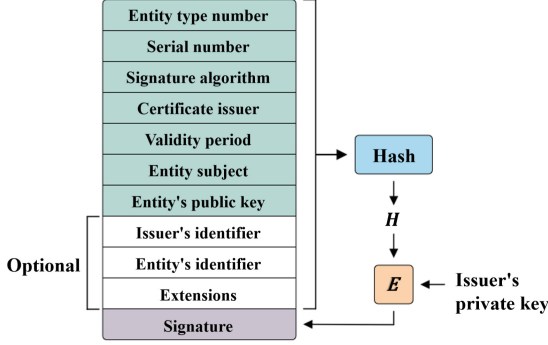


Fig. 3. Certificate issued by \mathcal{FCA} .

- Choose $ssk \in Z_q^*$ randomly, then compute $spk = g^{ssk}$ and send (ssk, spk) to \mathcal{U} .

Data Processing phase.

(2) **AuthGen.** \mathcal{U} generates the encrypted data block set C and the authenticator set ϕ .

- Divide the raw data set F into $f_1||f_2||\dots||f_n$ ² and split the processed file into s data blocks m_{ij} , where $i \in [1, n], j \in [1, s]$.
- Compute the encrypted data block $c_{ij} = Enc(m_{ij}, k_0)$ and obtain the encrypted data block set $C = \{c_{ij}\}_{i \in [1, n], j \in [1, s]}$, where k_0 is the encryption key.
- For each $c_{ij} \in C$, compute the data block authenticator

$$\sigma_{ij} = [H_1(ID_i||j) \cdot g^{c_{ij}}]^{ssk},$$

where ID_i is the unique identifier of f_i . Then, create authenticator set $\phi = \{\sigma_{ij}\}_{i \in [1, n], j \in [1, s]}$.

- Send $\{C, \phi\}$ to certain \mathcal{SN} s randomly and record these \mathcal{SN} s' certificates.

(3) **ExtractKW.** Based on F , \mathcal{U} generates the keyword set W and the index vector set V .

- Extract top- \mathcal{K} ranked keywords $\{\omega_t\}_{t=1,2,\dots,\mathcal{K}}$ from the files using the *time frequency inverse document frequency*³ text keyword extraction method. Then, create the keyword set $W = \{\omega_t\}_{t=1,2,\dots,\mathcal{K}}$.

² The *Reed Solomon* erase-code technique is chosen to redundantly segment F , and its security and efficiency have been widely proven.

³ Time frequency inverse document frequency is a keyword extraction technology that sorts all words according to the frequency of each word in the document, which is used to extract the top- \mathcal{K} words as the keywords of a file in HFKA.

- For ω_t , set up an n -bit binary array as the index vector v_{ω_t} and initialize $v_{\omega_t} = 0$. For each file $f_i \in F$, set $v_{\omega_t}[i] = 1$ if f_i contains the keyword ω_t (e.g., $v_{\omega_t} = \{0, 1, 0, 0, \dots, 0\}$ when only the file f_2 contains ω_t). Create the index vector set $V = \{v_{\omega_1}, v_{\omega_2}, \dots, v_{\omega_K}\}$.

(4) **IndexGen.** Based on (W, V) , \mathcal{U} generates the encrypted index table I_{RS} and challenge generating auxiliary index I_{TPA} .

- For ω_t , compute the index address $f_{\omega_t} = f(\omega_t, k_1)$ via PRF and then update the original index vector v_{ω_t} to $v_{f_{\omega_t}}$, where k_1 is the key of the PRF implementation.
- Compute the encrypted permutation $\pi_{\omega_t} = \pi(v_{f_{\omega_t}}, k_2)$ via PRP, where k_2 is the key of the PRP implementation.
- Compute the encrypted index vector $e_{\omega_t} = v_{f_{\omega_t}} \oplus \pi_{\omega_t}$ to facilitate the restoration of the original index vector.
- Create a set $S_{\omega_t} = \emptyset$ to record the subscripts of files containing ω_t and add i to S_{ω_t} if f_i contains ω_t . Then, compute the index table label

$$\Delta_{\omega_t, j} = [H_2(Z)^{-1} \cdot H_3(f_{\omega_t} || j) \cdot \prod_{i \in S_{\omega_t}} (H_1(ID_i || j)^{-1})]^{ssk},$$

where Z is the number of file updates with an initial value of 1. Let $\Delta_{\omega_t} = \{\Delta_{\omega_t, j}\}_{j=1,2,\dots,S}$ be the ITL set.

- Set the encrypted index table $I_{RS} = \{(f_{\omega_t}, e_{\omega_t}, \Delta_{\omega_t})\}_{\omega_t \in W}$ and challenge generating auxiliary index $I_{TPA} = \{(f_{\omega_t}, \pi_{\omega_t})\}_{\omega_t \in W}$. Send I_{RS} and I_{TPA} to \mathcal{R} and \mathcal{TPA} , respectively.

(5) **TrapdoorGen.** Based on the searched keyword ω' , \mathcal{U} generates the search trapdoor $T_{\omega'}$.

- Compute the search index address $f_{\omega'} = f(\omega', k_1)$ and the encrypted permutation $\pi_{\omega'} = \pi(v_{f_{\omega'}}, k_2)$. Set the search trapdoor $T_{\omega'} = (f_{\omega'}, \pi_{\omega'})$.

Challenge Generating phase.

(6) **InitBF.** \mathcal{TPA} initiates the Bloom filter and sets its parameters tuple BF .

- Compute $Len = \frac{\ln 2 \cdot |T_{\omega'}|}{\mathcal{R}}$. Create a Len -bit array \mathcal{B} and make $\mathcal{B}[i] = 0$, where $i \in [0, Len - 1]$. Build $BF = (\mathcal{B}, \{SHA_i\}_{i=1,2,\dots,\mathcal{R}})$.

(7) **FuzzyTDGen.** Based on $T_{\omega'}$ and I_{TPA} , \mathcal{TPA} updates BF and generates the fuzzy search trapdoor $FT_{\omega'}$. The pseudo-code is shown in **Algorithm 1**.

- When $T_{\omega'}$ is verified to be legitimate with I_{TPA} , extract the index address $f_{\omega'}$ from $T_{\omega'}$ and then update BF : $\mathcal{B}[SHA_i(f_{\omega'}) \bmod Len]_{i=1,2,\dots,\mathcal{R}} = 1$.
- Create a set $F_{\omega'}$ and then take the all files' index address as the input of BF . Write $f_{\omega''}$ into $F_{\omega'}$ if $\mathcal{B}[SHA_i(f_{\omega''}) \bmod Len] = 1$.
- Search $\pi_{\omega''}$ with $f_{\omega''}$ from I_{TPA} , and set $\Pi_{\omega'} = \{\pi_{\omega''}\}$. Let $FT_{\omega'} = (F_{\omega'}, \Pi_{\omega'})$ be the fuzzy search trapdoor.

Algorithm 1: FuzzyTDGen

Input: Bloom filter data tuple BF , search trapdoor $T_{\omega'}$, challenge generating auxiliary index $I_{\mathcal{TPA}}$;
Output: Fuzzy search trapdoor FT_{ω} ;

```

1 for  $\exists f_{\omega'} \in \{T_{\omega'}\}$  is not checked do
2   search  $(f_{\omega}, \pi_{\omega})$  according to  $f_{\omega'}$ ;
3   if  $\pi_{\omega} == \pi_{\omega'}$  then
4     continue;
5   else
6     return "The search trapdoor
7     from  $\mathcal{U}$ 
8     exits problem!";
9 for each  $i \in (0, \mathcal{R})$  do
10   for each  $f_{\omega'} \in \{T_{\omega'}\}$  do
11      $data = SHA_i(f_{\omega'})$ ;
12      $\mathcal{B}[data \bmod Len] = 1$ ;
13 for each  $f_{\omega_i} \in I_{\mathcal{TPA}}$  do
14   for  $j \in (0, \mathcal{R})$  do
15      $data' = SHA_j(f_{\omega_i})$ ;
16     if  $\mathcal{B}[data' \bmod Len] == 1$ 
17       then
18         continue;
19       else
20         break;
21   add  $f_{\omega_i}$  into  $F_{\omega'}$ ,  $\pi_{\omega_i}$  into  $\Pi_{\omega'}$ ;
22 return  $FT_{\omega} = (F_{\omega'}, \Pi_{\omega'})$ ;

```

(8) **ChalGen.** Based on FT_{ω} , \mathcal{TPA} generates the challenge information $Chal$.

- Randomly choose a c -element subset $Q = \{q_1, q_2, \dots, q_c\} \subseteq [1, s]$ and $v_j \in Z_q^*$ for each element of Q .
- Generate $Chal = (FT_{\omega}, Q, \{v_1, v_2, \dots, v_j\})$ and send it to \mathcal{RS} and \mathcal{SN} .

Integrity Auditing phase.

(9) **Retrieval.** Based on $Chal$ and $I_{\mathcal{RS}}$, \mathcal{RS} selects the challenged file set S_{ω_t} and corresponding index table label Δ_{ω_t} .

- Take $f_{\omega''}$ from FT_{ω} and retrieve $f_{\omega_t} = f_{\omega''}$ in $I_{\mathcal{RS}}$. Then, remove the e_{ω_t} and Δ_{ω_t} corresponding to f_{ω_t} from $I_{\mathcal{RS}}$ and remove the $\pi_{\omega''}$ corresponding to $f_{\omega''}$ from FT_{ω} .
- Compute $v_{\omega_t} = e_{\omega_t} \oplus \pi_{\omega''}$.
- Initiate the challenged file set $S_{\omega_t} = \emptyset$. When $v_{\omega_t}[i] = 1, i \in [1, n]$, write i into S_{ω_t} and send $(S_{\omega_t}, \Delta_{\omega_t})$ to \mathcal{SN} .

(10) **ProofGen.** Based on $Chal, S_{\omega_t}, \Delta_{\omega_t}, C$ and ϕ , \mathcal{SN} generates the storage proof Prf .

- With $Chal = (FT_{\omega}, Q, \{v_1, v_2, \dots, v_j\})$ from \mathcal{TPA} , $S_{\omega_t}, \Delta_{\omega_t}$ from \mathcal{RS} and $C = \{c_{ij}\}, \phi = \{\sigma_{ij}\}$ from \mathcal{U} , compute

$$T = \prod_{i \in S_{\omega_t}} \prod_{j \in Q} \sigma_{ij}^{v_j} \cdot \prod_{j \in Q} \Delta_{\omega_t, j}^{v_j},$$

$$\mu = \sum_{i \in S_{\omega_t}} \sum_{j \in Q} c_{ij} \cdot v_j,$$

and then set $Prf = (T, \mu)$.

(11) **VerifyProof.** Based on $Chal$ and Prf , TPA verifies the validity of Prf .

– Verify whether Prf is valid via the following equation:

$$e(T, g) \stackrel{?}{=} e(g^\mu \cdot \prod_{j \in Q} [H_2(Z)^{-1} \cdot H_3(f_{\omega_t} || j)]^{v_j}, spk).$$

5 Security Analysis

Theorem 1. *HFKA achieves storage robustness, i.e., semihonest storage nodes are unable to launch successful auditing frequency analysis attacks and auditing proof replay attacks using stored data.*

Proof. We design two games to prove why HFKA can resist the above two attacks.

Game 1. We assume that adversary \mathcal{M}_1 , a malicious \mathcal{SN} that proactively counts the frequency of stored files being audited and attempts to delete files with a minimal auditing frequency to free storage space for other users. M is the number of files stored in \mathcal{M}_1 for a \mathcal{U} , A is the number of files selected by \mathcal{U} in each round, and R is the number of auditing rounds. In the original keyword-based auditing paradigm, as the number of auditing rounds increases, \mathcal{M}_1 can calculate the probability of each file being audited based on the law of large numbers, i.e., knowing \mathcal{U} 's preference of files as shown in the following equation: $P_i = \lim_{R \rightarrow \infty} \frac{\sum_{j=1}^R X_{ij}}{A \cdot R}$, where $X_{ij} = 1$ if f_i is selected in the j -th round, else $X_{ij} = 0$. Given the calculated P_i of each file being audited, \mathcal{M}_1 can delete files with P_i close to 0 and hardly be detected by the \mathcal{U} . In HFKA, as a result of the Bloom filter, additional $A \cdot p$ files are chosen randomly in each round, where p is the false positive rate of the Bloom filter. As a result, the files not selected by \mathcal{U} have a probability of $\frac{A \cdot p}{M - A}$ of being selected by the Bloom filter in each round. The probability of P_i analyzed by \mathcal{M}_1 becomes as follows: $P_i = \lim_{R \rightarrow \infty} \frac{\sum_{j=1}^R X_{ij}}{A(1+p) \cdot R} + \frac{A \cdot p}{M - A} = P'_i + \frac{A \cdot p}{M - A}$, where $X_{ij} = 1$ if f_i is selected in j -th round, else $X_{ij} = 0$. From the above equation, we can see that even if the probability of a file being selected by a \mathcal{U} tends toward 0, the probability of it being audited is still not less than $\frac{A \cdot p}{M - A}$. It is worthwhile for \mathcal{SN} to delete \mathcal{U} 's data with $\frac{A \cdot p \cdot 100}{M - A} \%$ risk.

Game 2. We assume that the adversary \mathcal{M}_2 is a lazy \mathcal{SN} . It does not perform the update operation to save its own computing resources when \mathcal{U} requests a data update. Furthermore, it attempts to use the original data to pass the auditing verification. We analyze whether \mathcal{M}_2 can forge proof with nonupdated data according to the auditing verification formula as follows: $e(T, g) = e(g^\mu \cdot \prod_{j \in Q} [H_2(Z)^{-1} \cdot H_3(f_{\omega_t} || j)]^{v_j}, spk)$. In this formula, except for the T and μ provided by \mathcal{M}_2 , Z , spk is public, and f_{ω_t} and $(j, v_j)_{j \in Q}$ are generated by TPA itself. However, only T and μ are computed under data block c_{ij} , which

suggests that \mathcal{M}_2 can pass the verification with nonupdated data. We provide more details to support the following analysis: $T = \prod_{i \in S_{\omega_t}} \prod_{j \in Q} \sigma_{ij}^{v_j} \cdot \prod_{j \in Q} \Delta_{\omega_t, j}^{v_j}$, $\sigma_{ij} = [H_1(ID_i || j) \cdot g^{c_{ij}}]^{ssk}$, $\Delta_{\omega_t, j} = [H_2(Z)^{-1} \cdot H_3(f_{\omega_t} || j) \cdot \prod_{i \in S_{\omega_t}} (H_1(ID_i || j)^{-1})]^{ssk}$, $\mu = \sum_{i \in S_{\omega_t}} \sum_{j \in Q} c_{ij} \cdot v_j$. Ignoring challenge information $(j, v_j)_{j \in Q}$ and challenged file set $S_{\omega_t} = \{i\}$, μ is related to only c_{ij} , but T also implies Z . We make Z' be \mathcal{M}_2 's update times; then, the system updates data z times after \mathcal{M}_2 does not perform an update operation. That is, the global number of updates is $Z = Z' + z$. The probability of $H(Z) = H(Z')$ can be ignored due to the anticollision property of the hash function. ■

Theorem 2. *HFKA achieves privacy protection, i.e., no entities other than users can obtain specific content about outsourced data or users through the Bloom filter and the index table while executing auditing tasks.*

Proof. We analyze how HFKA achieves privacy protection to resist a semihonest \mathcal{SN} or a curious \mathcal{TPA} .

There are two strategies to undermine \mathcal{U} 's privacy for a semihonest \mathcal{SN} : (i) It deciphers ω_t and then infers the file's type and even the content from ω_t . (ii) It does not decipher ω_t but speculates the importance of files based on the frequency of selected f_{ω_t} and the audited files in each round. For the first strategy, HFKA adopts PRF to protect ω_t . Only \mathcal{U} knows its keyword ω_t ; other entities know only the keyword index address: $f_{\omega_t} = f(\omega_t, k_1)$. Because of the backward unpredictability of PRF, \mathcal{SN} cannot invert ω_t by any subsequence of f_{ω_t} . For the second strategy, HFKA resists \mathcal{SN} 's privacy attacks in two aspects. Before formally analyzing the measures, we show how \mathcal{SN} violates \mathcal{U} 's privacy. By asking the \mathcal{TPA} or \mathcal{RS} , the \mathcal{SN} can easily obtain f_{ω_t} in the auditing trapdoor of each round. After accumulating enough rounds of data, \mathcal{SN} can easily guess the documents corresponding to certain high-frequency keywords, as well as the connections between keywords. Two measures are taken to cope with such a situation, firstly using the Bloom filter to make the mapping relationships more ambiguous for each round, and secondly adopting distributed storage so that the audit information received for a certain \mathcal{SN} is not complete and the relationships between keywords and documents and between keywords and keywords are further weakened. Hence, HFKA blocks the adversary attack from the frequency distribution. \mathcal{TPA} faces the same difficulty regarding $f_{\omega_t} \rightarrow \omega_t$. When users are anonymous and PRF is backward unpredictable, a curious \mathcal{TPA} can do nothing. ■

Theorem 3. *HFKA achieves data security, which means that no external adversary can obtain any details about the data in the event that outsourced data are intercepted in transit.*

Proof. In the whole process of information interaction, the data involved can be classified as follows: keyword searchable data (index address f_{ω_t} , search vector π_{ω_t} , encrypted index vector e_{ω_t} , ITL $\Delta_{\omega_t, j}$), auditing information (encrypted

data c_{ij} and data authenticator σ_{ij}). The security of f_{ω_t} is based on PRF, and the security of π_{ω_t} is based on PRP. PRF is indistinguishable and its proof is detailed in [36]; that is, in the face of an attack, the attacker cannot determine whether the same PRF function is selected, i.e., the attacker cannot further guess the function input. PRP also achieves indiscernibility using computationally difficult mathematical problems. Since e_{ω_t} is designed to recover the original index vector v_{ω_t} , its security is off the table. The security of c_{ij} is based on AES-128. AES can resist differential cryptanalysis, linear cryptanalysis and other basic attacks; more details are provided in [38]. Moreover, AES with a 128-bit key has the same safety strength as RSA-3072 and ECC-256, which is estimated to be available until 2040. The security of σ_{ij} and $\Delta_{\omega_t,j}$ is based on a discrete logarithm problem (DLP) [37], which is stated as follows: For $a, g \in G$, $\exists b$ such that $g^b = a$; finding such b is computationally complex, where G is a cyclic group. For σ_{ij} , \mathcal{M} must solve the DLP twice to obtain the data block c_{ij} , which is still protected by AES-128. ■

6 Performance Evaluation

In this section, we evaluate the performance of HFKA on a Lenovo desktop computer equipped with an Intel Core i5 CPU and 8 GB of RAM. All cryptographic operations in HFKA, such as PRP, PRF, Bloom filter and bilinear pairing, are performed using the PBC library. We conduct detailed experiments to demonstrate the unique function of HFKA by balancing the auditing distribution to hide the files' audited frequency.

6.1 Auditing Distribution

The Bloom filter plays an important role in HFKA, which hides the frequency distribution properties of keyword-based auditing. We elaborate how it plays in HFKA from experiments in three perspectives. **Experiment 1** investigates the factors influencing the false positive rate of the Bloom filter and finds the most suitable parameters of it to assist the implementation of HFKA. **Experiment 2** shows the results of implementing the frequency hiding function of HFKA. **Experiment 3** further improves the fuzzing capability of the Bloom filter. It is assumed that the distribution of \mathcal{K} keywords in n files is uniform. That is, the distribution of keywords can represent the auditing distribution of files.

Based on the original introduction of the Bloom filter in [34], we know that its false positive rate p is equal to $(1 - e^{-\frac{k \cdot m}{n}})^k$, where k is the number of hash functions, m is the size of the input, and n is the length of the BF array. In **Experiment 1**, we construct the Bloom filter and control its false positive rate by adjusting the parameters. Figure 4 shows the variation in the false positive rate with different parameters. Comparing the four small plots (a), (b), (c), (d) in Fig. 4, it was found that the larger the input size is, the larger the false positive rate is. For any of the small plots, we found that both of the number of hash functions and the size of the BF array are inversely proportional to the false

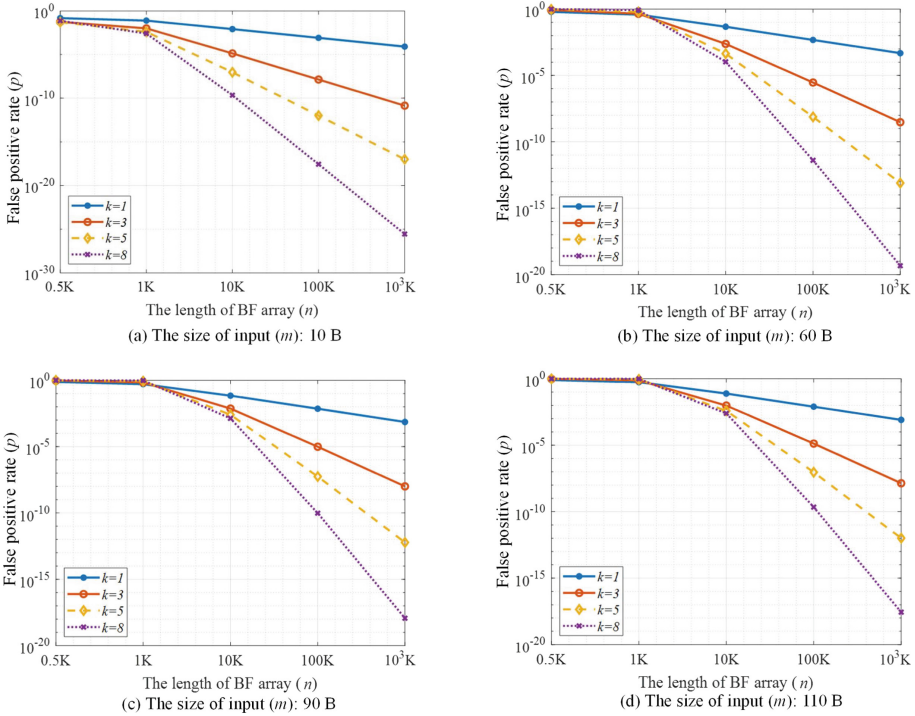


Fig. 4. Effect of array length on false positive rate of *BF*

positive rate. Here, we set the size of each keyword index address to 10 B, so an input size of 60 B means that the user selects 6 keywords in a search trapdoor. According to the fuzzy requirements of HFKA and the specific experimental data, it is most suitable when the input size is between 90 B and 110 B, the array length is 1000 bits, and the hash function number is 3 so that the false positive rate can be approximately 55%.

In **Experiment 2**, we use keywords from the simulated dataset to perform fuzzy matching experiments. We select several keywords to update the Bloom filter in each round, and then input all keywords for matching, and then finally collect these matching results. The total number of keywords is 50, and 9 or 10 keywords are chosen in each round. The false positive rate means that there would be 4 or 5 additional chosen keywords. Figure 5 shows the distribution of the selected keywords after 20 rounds of experiments. Horizontally, the false positive rate of the Bloom filter in each round can be seen, with red ● being the keywords selected by \mathcal{U} and blue ✕ being the fuzzy keywords selected by the Bloom filter. Vertically, the frequency of each keyword that was selected can be seen. The more red ● there are in a column, the higher probability of the keyword would be selected by \mathcal{U} , such as ω_{10} , ω_{24} , while blue ✕ balances the probability of the other keywords being selected, such as ω_{20} , ω_{40} . The frequency of each

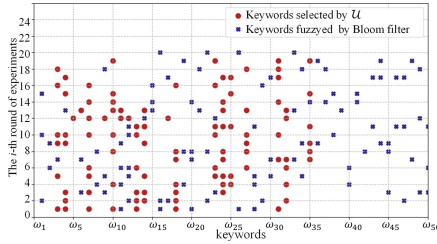


Fig. 5. The keyword distribution of Experiment 2

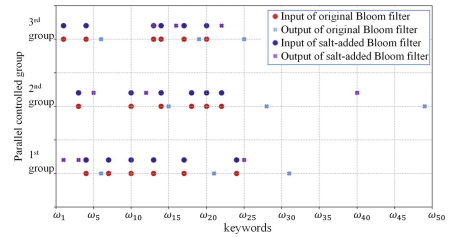


Fig. 6. The use of salt-hashing BF in Experiment 3

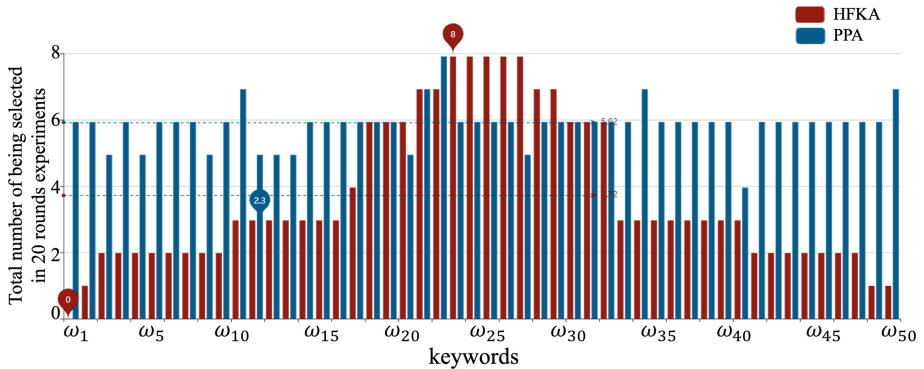


Fig. 7. The auditing frequency comparison between HFKA and PPA

keyword is counted and compared with traditional auditing in Fig. 7. HFKA represents high and low frequencies, which are evenly distributed in itself, while PPA are all evenly distributed. Moreover, we discuss the total auditing overhead. To guarantee detection confidence of 99% probability, TPA needs to select 460 files from 10,000 files randomly in the PPA model. However, in the PKA model [4], the user is only concerned about 2% probability of the total files, i.e., select 200 files from 10,000 files. In HFKA, due to the 55% false positive of the Bloom filter, there are additional 110 files to be selected, i.e., a total of 310 files are selected in each round. Therefore, HFKA audits 150 files less than PPA each time, which saves 32.6% probability of auditing overhead.

In **Experiment 3**, we fix the configuration of the Bloom filter and the search trapdoor, but add different salt to the hash functions in each round. We adopt the control method to verify the efficacy of salted hash functions. To avoid the coincidence of one experimental group, we set three parallel experimental groups, where the 1st, 2nd and 3rd groups have different Bloom filter parameter settings. As shown in the Fig. 6, the red \bullet in each group indicates the input of the original Bloom filter, while the dark-blue \bullet indicates the input of salt-added Bloom filter, which must be consistent with the red \bullet . Correspondingly, the light-blue \times indicates the keywords fuzzied by the original Bloom filter, while

the purple ✖ is the keyword fuzzied by the salt-added Bloom filter. From the results of the above experimental groups, it can be seen that even for the exact same input, different outputs are obtained after using the salting method.

6.2 Computation Overhead

Table 1 shows the symbols used to represent these operations. There are mainly calculations in HFKA: hash operation, exponential operation, multiplication operation and bilinear pairing operation. We ignore the *XOR*, *PRF*, *PRP*, *addition* operations, which have minimal computational cost.

Table 1. Notation description

Notation	Operation
$HASH_{G_1}$	Map a value to G_1
$SHA_{Z_q^*}$	Map a value to Z_q^* with a secure hash algorithm
EXP_{G_1}	Exponentiation operation in G_1
MUL_{G_1}	Multiplication operation in G_1
$PAIR$	Bilinear pairing with G_1, G_2

Table 2 shows the concrete operations of involved data in each phase of HFKA. The data preprocessing phase includes data authenticator generation and index label generation. The challenge generation phase includes Bloom filter updating and fuzzy matching. The integrity auditing phase includes proof generation and verification. Compared to the method of Gao et al. [4], HFKA only increases the Bloom filter updating and the fuzzy matching process, which increase $\mathcal{R} \cdot (|F_{\omega'}| + \mathcal{K})$ HMAC operations in each round of auditing. We assess the run time of SHA operations with 128-bit keys and 1024 bytes input in Python, and the experimental results show that it is only 0.03 ms, which is negligible due to the whole system.

Table 2. Computation overhead

Processing phase	Computation overhead
Authenticator generation	$n \cdot s \cdot (HASH_{G_1} + MUL_{G_1} + EXP_{G_1})$
ITL generation	$\mathcal{K} \cdot s \cdot [(S_{\omega_t} + 2)HASH_{G_1} + (S_{\omega_t} + 1)MUL_{G_1} + EXP_{G_1}]$
<i>BF</i> updating	$\mathcal{R} \cdot F_{\omega'} \cdot SHA_{Z_q^*}$
Fuzzy matching	$\mathcal{K} \cdot \mathcal{R} \cdot SHA_{Z_q^*}$
Proof generation	$c \cdot (2 \cdot S_{\omega_t} + 1) \cdot EXP_{G_1} + c \cdot S_{\omega_t} \cdot MUL_{G_1}$
Proof verification	$2 \cdot PAIR + (c + 1) \cdot MUL_{G_1} + (c + 1) \cdot EXP_{G_1} + 2c \cdot HASH_{G_1}$

6.3 Storage Overhead

Here, we consider the storage overhead in HFKA. Due to the involvement of the keyword-file index, HFKA has some additional storage overhead compared to PPA. We list the additional data structures and the storage space they consume as the number of files sampled increases in Table 3. According to the experimental results, when there are 100 files, \mathcal{RS} requires only 0.37 MB of space to store $I_{\mathcal{RS}}$, and \mathcal{TPA} requires only 0.368 MB of space. When there are 1000 files, these values become 16.64 MB and 16.62 MB. Even when the number of files increases to 10,000, the size of $I_{\mathcal{RS}}$ is only 466.51 MB, and the size of $I_{\mathcal{TPA}}$ is 466.31 MB. Notably, the size of a Bloom filter with a 55% false positive rate is always 1K bits, as will be elaborated in Sect. 6.1. Furthermore, the size of $I_{\mathcal{RS}}$ and $I_{\mathcal{TPA}}$ are related to the number of files. In fact, $I_{\mathcal{RS}}$ and $I_{\mathcal{TPA}}$ are directly related to the number of keywords, and the number of files affects the number of keywords.

Table 3. Storage overhead

The number of files	$I_{\mathcal{RS}}$ storage	$I_{\mathcal{TPA}}$ storage	Bloom filter (55%)
100	0.37 MB	0.368 MB	1000 bit
1000	16.64 MB	16.62 MB	1000 bit
10,000	466.51 MB	466.31 MB	1000 bit

7 Conclusion and Future Work

In this paper, we propose a keyword-based auditing scheme, HFKA, to address the auditing frequency leakage problem for a smart government. We utilize the Bloom filter to achieve a specified keyword fuzzy matching. Meanwhile, we design an index table label to resist replay attacks by lazy server nodes and generate storage proofs without exposing any keyword-file privacy. We also separate the retrieval work from the storage work using a special retrieval server to improve the retrieval efficiency, reduce the storage cost and computing cost of the storage node, and further maintain the keyword privacy. Finally, the security of the scheme is proven by rigorous security analysis, and the feasibility of the scheme is proven by performance evaluation.

In future work, we will examine the security protection and economic viability of the keyword-based auditing paradigm in real world applications and strengthen the integration of the auditing model and smart government. First, in order to allow batch auditing and dynamic auditing of various storage nodes, we will enhance the data label construction of outsourced data. Second, we will attempt to store massively outsourced and more finely partitioned data via a directed acyclic graph (DAG). The difficulties of implementing smart government affairs will then be examined, and we will work to overcome security issues and performance bottlenecks in data sharing and integrity auditing.

Acknowledgements. This work was supported in part by the Southwest Petroleum University “Set Sail” Project (Grant No.0202667680-2021QHZ017); in part by the National Natural Science Foundation of China (Grant No.61902327); in part by the China Postdoctoral Science Foundation (Grant No.2020M681316); in part by the Chengdu Key R & D project (Grant No.2021-YF05-00965-SN); and in part by the Southwest Petroleum University Graduate Teaching and Research Reform Project (Grant No.JY20ZD06).

References

1. Mellouli, S., Luna, R., Luis, F.: Smart government, citizen participation and open data. *Inf. Polity* **19**(1–2), 1–4 (2014)
2. Wu, J., Ping, L., Ge, X., Wang, Y., Fu, J.: Cloud storage as the infrastructure of cloud computing. In: International Conference on Intelligent Computing and Cognitive Informatics, pp. 380–383 (2010)
3. Yang, K., Jia, X.: Data storage auditing service in cloud computing: Challenges, methods and opportunities. *World Wide Web* **15**(4), 409–428 (2012)
4. Gao, X., Yu, J., Chang Y., Wang, H., Fan, J.: Checking only when it is necessary: Enabling integrity auditing based on the keyword with sensitive information privacy for encrypted cloud data. *IEEE Trans. Dependable Secure Comput.*, 1 (2021). <https://doi.org/10.1109/TDSC.2021.3106780>
5. Ateniese, G., et al.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 598–609. ACM, Alexandria, VI (2007)
6. Juels, A., Kaliski, B.: Proofs of retrievability for large files. *Proc. Comput. Commun. Secur. Alexandria: AcMPress* **584**, 597 (2007)
7. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Backes, M., Ning, P. (eds.) *ESORICS 2009*. LNCS, vol. 5789, pp. 355–370. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_22
8. Merkle, R.C.: *Secrecy, Authentication, and Public Key Systems*, 2nd edn. Stanford University, Stanford (1979)
9. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: 2010 IEEE INFOCOM, pp. 1–9. IEEE, San Diego, CA (2010)
10. Wang, C., Chow, S.S.M., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. *IEEE Trans. Comput.* **62**(6), 362–375 (2013)
11. Zhou, L., Fu, A., Yang, G., Wang, H., Zhang, Y.: Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics. *IEEE Trans. Dependable Secur. Comput.* **19**(2), 1118–1132 (2022)
12. Zhang, X., Zhao, J., Xu, C., Li, H., Wang, H., Zhang, Y.: CIPPPA: Conditional identity privacy-preserving public auditing for cloud-Based WBANs against malicious auditors. *IEEE Trans. Cloud Comput.* **9**(4), 1362–1375 (2021)
13. Rashmi, K V., Shah, N B., Kumar, P V.: Enabling node repair in any erasure code for distributed storage. In: 2011 IEEE International Symposium on Information Theory, pp. 1235–1239. IEEE, St. Petersburg (2011)
14. Perard, D., Lacan, J., Bachy, Y., Detchart, J.: Erasure code-based low storage blockchain node. In: Proceedings of 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1622–1627. IEEE, Halifax, Canada (2018)

15. Jin, H., Luo, R., He, Q., Wu, S., Zeng, Z., Xia, X.: Cost-effective data placement in edge storage systems with erasure code. *IEEE Trans. Serv. Comput.*, 1 (2022). <https://doi.org/10.1109/TSC.2022.3152849>
16. Sun, Y., Liu, Q., Chen, X., Du, X.: An adaptive authenticated data structure with privacy-preserving for big data stream in cloud. *IEEE Trans. Inf. Forensics Secur.* **15**(2020), 3295–3310 (2020)
17. Li, S., Zhang, Y., Xu, C., Chen, K.: Cryptoanalysis of an authenticated data structure scheme with public privacy-preserving auditing. *IEEE Trans. Inf. Forensics Secur.* **16**, 2564–2565 (2021)
18. Xu, S., Cai, X., Zhao, Y., Ren, Z., Du, L., Wang, Q., Zhou, J.: zkrcChain: Towards multi-party privacy-preserving data auditing for consortium blockchains based on zero-knowledge range proofs. *Futur. Gener. Comput. Syst.* **128**, 490–504 (2022)
19. Shen, W., Qin, J., Yu, J., Hao, R., Hu, J., Ma, J.: Data integrity auditing without private key storage for secure cloud storage. *IEEE Trans. Cloud Comput.* **9**(4), 1408–1421 (2019)
20. Zheng, W., Lai, C.F., He, D., Kumar, N., Chen, B.: Secure storage auditing with efficient key updates for cognitive industrial IoT environment. *IEEE Trans. Industr. Inf.* **17**(6), 4238–4247 (2020)
21. Ge, X., Yu, J., Hu, C.: Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *IEEE Access* **6**, 45725–45739 (2018)
22. Miao, Y., Tong, Q., Deng, R., Choo, K.K.R., Liu, X., Li, H.: Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage. *IEEE Trans. Cloud Comput.* **10**, 835–848 (2020)
23. Bringer, J., Chabanne, H., Kindarji, B.: Error-tolerant searchable encryption. In: *IEEE International Conference on Communications*, pp. 1–6 (2009)
24. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: *2010 Proceedings IEEE INFOCOM*, pp. 1–5. IEEE, Cape Town (2010)
25. Wang, Q., et al.: Searchable encryption over feature-rich data. *IEEE Trans. Dependable Secure Comput.* **15**(3), 496–510 (2016)
26. Zhong, H., Li, Z., Cui, J., Sun, Y., Liu, L.: Efficient dynamic multi-keyword fuzzy search over encrypted cloud data. *J. Netw. Comput. Appl.* **149**, 102469 (2020)
27. Liu, Q., Peng, Y., Pei, S., Wu, J., Peng, T., Wang, G.: Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search. *IEEE Trans. Serv. Comput.* **15**, 1799 (2020)
28. Li, X., et al.: VRFMS: verifiable ranked fuzzy multi-keyword search over encrypted data. *IEEE Trans. Serv. Comput.*, 1 (2022). <https://doi.org/10.1109/TSC.2021.3140092>
29. Sahu, I.K., Nene, M.J.: Identity-based integrity verification (IBIV) protocol for cloud data storage. In: *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, pp. 1–6. IEEE, Bhilai (2021)
30. Yoosuf, M.S., Anitha, R.: LDuAP: Lightweight dual auditing protocol to verify data integrity in cloud storage servers. *J. Ambient. Intell. Humaniz. Comput.* **13**(8), 3787–3805 (2022)
31. Tong, Q., Miao, Y., Weng, J., Liu, X., Choo, K.K.R., Deng, R.: Verifiable fuzzy multi-keyword search over encrypted data with adaptive security. *IEEE Trans. Knowl. Data Eng.*, 1 (2022). <https://doi.org/10.1109/TKDE.2022.3152033>
32. Gervais, A., Capkun, S., Karame, G.O., Gruber, D.: On the privacy provisions of bloom filters in lightweight bitcoin clients. In: *Proceedings of the 30th Annual*

- Computer Security Applications Conference, pp. 326–335. ACM, New Orleans, Louisiana (2014)
33. Pagh, A., Pagh, R., Rao, S.: An optimal bloom filter replacement. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 823–829. SODA, Vancouver, BC (2005)
 34. Kiss, S.Z., Hosszu, E., Tapolcai, J., Ronyai, L., Rottenstreich, O.: Bloom filter with a false positive free zone. *IEEE Trans. Netw. Serv. Manage.* **18**(2), 2334–2349 (2021)
 35. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 965–976. ACM, Raleigh, NC (2012)
 36. Bellare, M., Impagliazzo, R.: A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion. *IACR Cryptol. ePrint Arch.* **1999**, 24 (1999)
 37. McCurley, K S.: The discrete logarithm problem. In: Proceedings of Symposium in Applied Mathematics, pp. 49–74 (1990)
 38. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999)
 39. Cachin, C.: Architecture of the hyperledger blockchain fabric. In: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, pp. 1–4. Chicago (2016)
 40. Qu, Y., Nosouhi, M.R., Cui, L., Yu, S.: Existing privacy protection solutions. In: Personalized Privacy Protection in Big Data. DA, pp. 5–13. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-3750-6_2