Postprint: A Secure Deduplication Scheme for Cloud Storage Based on Dynamic Bloom Filter

Authors: Wang Pingyan, Liu Yi

Date: 2018-10-11T00:00:00+00:00

Abstract

Existing proof-of-ownership deduplication schemes are vulnerable to threats from honest-but-curious servers, and resorting to a trusted third party to address this issue incurs excessive overhead. This paper proposes an improved, trusted-third-party-free proof-of-ownership secure deduplication scheme based on dynamic Bloom filters. The scheme employs convergent encryption algorithms to resist honest-but-curious servers and prevents data pollution attacks by enabling the server to verify the consistency between data block ciphertexts and tags. Additionally, a key chain mechanism is adopted for convergent key management, which resolves the problem of excessive storage space occupied by convergent keys in existing schemes. Analysis and comparisons demonstrate that the proposed scheme achieves lower key storage overhead and transmission overhead.

Full Text

Secure Deduplication Scheme Based on Dynamic Bloom Filter in Cloud Storage

Wang Pingyan, Liu Yi
(School of Computers, Guangdong University of Technology, Guangzhou 510006, China)

Abstract

Existing Proof of Ownership (PoW) deduplication schemes are vulnerable to threats from honest-but-curious servers, and addressing this issue with trusted third parties leads to excessive overhead. This paper proposes an improved, trusted-third-party-free secure deduplication scheme based on Dynamic Bloom Filter that employs convergent encryption to resist honest-but-curious servers

and prevents data pollution attacks by having the server verify consistency between ciphertext blocks and their tags. Additionally, a key chaining mechanism manages convergent keys, solving the problem of excessive storage space consumption for convergent keys in existing schemes. Analysis and comparison demonstrate that the proposed scheme achieves lower key storage overhead and transmission overhead.

Key words: cloud storage; data deduplication; Bloom filter; convergent encryption; proof of ownership

0 Introduction

As cloud computing technology continues to evolve, an increasing number of users choose to outsource data storage and management to the cloud. According to the Cisco Global Cloud Index (2015-2020) [1], global data center traffic reached 4.7 ZB in 2015, with cloud data accounting for 3.8 ZB (81%). By 2020, total data center traffic is projected to triple to 15.3 ZB, with cloud data comprising 14.1 ZB (92%). Faced with this unprecedented data volume, achieving economical, efficient, and secure storage has become a critical challenge for cloud service providers.

Data deduplication technology, also known as duplicate data elimination, removes redundant files or data blocks within files. By retaining only a single copy of duplicate data, this technique can significantly reduce cloud storage space requirements [2,3]. However, typical storage systems often use file digests as ownership credentials, enabling attackers to obtain entire files using only the digest [4]. To address this vulnerability, Halevi et al. [5] first introduced the concept of Proof of Ownership (PoW), requiring users to prove file ownership before gaining access rights. Unfortunately, this approach incurs substantial computational and I/O overhead on the client side.

Di Pietro et al. [6] proposed an improved scheme called s-PoW that verifies users by selecting random bits from files, enhancing client efficiency and reducing bandwidth consumption but suffering from low server-side efficiency. Blasco et al. [7] introduced the bf-PoW scheme based on Bloom filters, enabling cloud servers to quickly verify user responses. While more efficient on the server side than s-PoW, this scheme suffers from increasing false positive rates as more elements are added. Yu et al. [8] presented a PoW deduplication scheme for mobile cloud computing that improves client efficiency. However, these schemes neglect data confidentiality and remain vulnerable to honest-but-curious server threats [9].

Some approaches [10-14] address confidentiality by employing trusted third parties for data management, but this introduces prohibitive overhead that makes practical implementation difficult. González-Manzano et al. [15] proposed the ce-PoW scheme, which combines convergent encryption with a trusted-third-party-free design to effectively prevent honest-but-curious servers from viewing files while also considering data pollution attacks. However, this scheme con-

sumes significant client storage space for convergent keys. Liu et al. 16 presented an efficient and secure deduplication scheme based on Bloom filters and convergent encryption, but it fails to address the increasing false positive rate problem and cannot resist pollution attacks. Zhang et al. [17] proposed a trusted-third-party-free adaptive deduplication scheme for encrypted data that first checks data popularity and uses PAKE protocols for key transfer when data is unpopular, but PAKE requires both parties to be online simultaneously, limiting practical utility.

To overcome these limitations, this paper proposes DBF-Dedup, an improved Proof of Ownership scheme based on Dynamic Bloom Filter that eliminates the need for trusted third parties. Our approach dynamically manages Bloom Filter states to effectively mitigate the rapid false positive rate growth problem, employs convergent encryption to ensure data confidentiality, and utilizes a key chaining mechanism to manage convergent keys, thereby solving key management issues.

1 System and Threat Models

1.1 System Model

As shown in [Figure 1: see original paper], the system model in our scheme comprises two entities:

a) **Cloud Service Provider (CSP):** An entity providing outsourced data services, consisting of a master server and storage servers. The master server handles file deduplication detection, challenge generation, and tag verification, while storage servers store ciphertext blocks of files and transmit requested files to authorized users.

b) **User:** An entity that outsources data storage to the cloud system. For data not previously stored in the system, users upload it once; for existing data, users need only prove ownership to gain access rights without re-uploading, saving bandwidth consumption.

1.2 Threat Model

In our threat model, we categorize attackers into two types:

a) **Internal attackers:** Typically malicious employees within the CSP who aim to extract valuable information from user data and may collude with external attackers. Additionally, internal attackers may compromise data integrity for specific purposes.

b) **External attackers:** Typically attackers outside the CSP who attempt to impersonate legitimate users to access cloud storage data. They may possess partial information, such as file digests, hoping to obtain complete files. External attackers may collaborate with legitimate users or collude with internal attackers.

2 Preliminaries

2.1 Bloom Filter

A Bloom filter [18] is an efficient probabilistic data structure for testing whether an element belongs to a specific set, typically consisting of a binary vector of m bits and k independent hash functions. Initially, all bits are set to 0. For a set containing n elements, each element is mapped to $\{1, 2, \ldots, m\}$ by computing k hash functions $\{H_1, H_2, \ldots, H_k\}$. When inserting element x, the bits at positions $H_i(x)$ are set to 1. As shown in [Figure 2: see original paper], with $n = 2$ and $k = 3$, the arrows indicate the bit positions mapped by hash functions, which are set to 1.

To query for data object s, we compute $\{H_1(s), H_2(s), \ldots, H_k(s)\}$ and check whether all mapped positions are 1. If not all are 1, s is definitely not in the set; if all are 1, s is likely in the set, though false positives may occur with some probability. A false positive happens when a non-member element' s k hash positions happen to be all 1, incorrectly indicating set membership. As the number of elements increases, the false positive rate grows. The false positive probability p_f is:

$$p_f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

Another limitation of standard Bloom filters is the inability to delete existing elements. Dynamic Bloom Filter (DBF) [19] addresses these drawbacks by controlling the increase in false positive rates. A DBF consists of multiple Standard Bloom Filters (SBFs). Initially, a DBF contains one active SBF (indicating the false positive rate is below the threshold). As new elements are inserted, when the SBF becomes full (reaching the false positive rate threshold), a new SBF is added to maintain an active state.

By monitoring SBF states and maintaining dynamic updates, DBF effectively controls false positive rates. Besides insertion, DBF supports query, deletion, and merge operations. DBF requires initialization of several parameters: maximum false positive rate for DBF, maximum number s of SBFs, maximum false positive rate for each SBF, number of elements n in the set, size m of each SBF, capacity c of each SBF, and number of hash functions k per SBF. Note that $s = \lceil n/c \rceil$.

2.2 Convergent Encryption

Convergent Encryption (CE) [20] is a deterministic encryption algorithm where the encryption key is derived from the data content itself, ensuring identical data produces identical keys. A convergent encryption scheme consists of the following cryptographic primitives:

- **KeyGen(F) ⇒ K:** A convergent key generation algorithm that takes plaintext data F as input and outputs convergent key K, typically generated using a cryptographic hash function H, i.e., $K = H(F)$.
- **Encrypt(K, F) ⇒ C:** An encryption algorithm that takes convergent key K and plaintext F as input and outputs ciphertext C.
- **Decrypt(K, C) ⇒ F:** A decryption algorithm that takes convergent key K and ciphertext C as input and outputs plaintext F.
- **TagGen(F) ⇒ T_F:** A tag generation algorithm that takes plaintext data F as input and outputs corresponding tag T_F.

3 Protocol Description

3.1 Symbol Definitions

- k: Data block partitioning parameter
- PRF: Pseudorandom function
- seed: Initialization seed
- B_i: The i-th data block
- C_i: The i-th ciphertext block
- K_i: Convergent key for the i-th data block
- CK_i: Encrypted convergent key for the i-th data block
- token_i: Tag for the i-th ciphertext block
- h_c: Digest of $\{\text{token}_i\}$
- J: Number of challenged data blocks

3.2 System Initialization

a) **User:** Initializes the four algorithms of the convergent encryption scheme (KeyGen, Encrypt, Decrypt, TagGen) and the PoW algorithm for ownership proof.

b) **CSP:** Initializes a storage system for metadata such as file tags and Dynamic Bloom Filters, including k hash functions $\{H_1, H_2, \ldots, H_k\}$.

3.3 File Upload

The file upload process is illustrated in [Figure 3: see original paper] and consists of the following steps:

a) User sends file size to server.

b) Server returns block partitioning parameter k.

c) User partitions file into n blocks $\{B_i\}$ $(1 \le i \le n)$, computes convergent key $K_i = \text{KeyGen}(B_i)$, encrypts each block as $C_i = \text{Encrypt}(K_i, B_i)$, computes tag $\text{token}_i = H(B_i)$ for each ciphertext block, and calculates digest $h_c = H(\text{token})$. User uploads h_c to server.

d) Server checks whether h_c already exists. If duplicate exists, proceed to Step 5 to initiate verification challenge; otherwise, proceed to Step 6 and request file upload.

e) **(Duplicate case):** (a) Server initiates challenge by randomly selecting J block indices; (b) User responds with corresponding J tags $\{\text{token}_j\}$; (c) Server uses token values as seed to initialize PRF, generates J outputs, and maps them to bit positions using Bloom Filter hash functions $\{H_1, H_2, \ldots, H_k\}$. If all mapped bits are 1, user passes ownership proof and server records file permission; otherwise, challenge fails. (d) After passing ownership proof, user stores first block' s convergent key K_1 locally for decrypting key ciphertexts during future downloads.

f) **(No duplicate case):** (a) User computes encrypted convergent keys using key chaining: $CK_i = E(K_{i-1}, K_i)$ for $2 \leq i \leq n$ (encrypting each key with the previous one); (b) User stores K_1 locally and uploads $\{C_i\}, \{CK_i\}$ to server; (c) To prevent pollution attacks, server recomputes $\{\text{token}_i\}$ and h'_c from $\{C_i\}$ and compares with user-uploaded h_c. If $h'_c = h_c$, upload succeeds; otherwise, ciphertext blocks and tags are inconsistent, and server returns warning indicating failed upload. (d) After successful upload, server creates a Dynamic Bloom Filter, initializes PRF with token values as seed, inserts outputs into Bloom Filter by setting corresponding bits to 1. (e) Server monitors Bloom Filter state (active or full); if full, creates a new Bloom Filter.

3.4 File Download

When requesting to download file F:

a) User requests file F from server.

b) Server checks user ID: returns warning if no access permission; otherwise, returns ciphertext blocks $\{C_i\}$ and encrypted convergent keys $\{CK_i\}$.

c) User decrypts all convergent keys recursively using locally stored K_1: $K_2 = \text{Decrypt}(K_1, CK_2)$, $K_3 = \text{Decrypt}(K_2, CK_3)$, and so on, then decrypts $\{C_i\}$ with $\{K_i\}$ to recover file F.

4 Security Analysis

4.1 Proof of Ownership

In our scheme, users must pass PoW protocol verification to obtain file permissions, rather than relying solely on file digests. Attackers may obtain file digests or partial content from other sources and attempt to retrieve entire files from the CSP. Clearly, file digests alone cannot pass verification. We now discuss cases where attackers possess partial file content.

Assume an attacker owns m data blocks of a file containing n total blocks, representing percentage $p = m/n$. The probability of correctly responding to a randomly selected block challenge from the CSP is p. Let event A denote the attacker successfully passing PoW verification, which occurs in two scenarios: (1) The attacker correctly responds to the challenge array, denoted as event R; (2) The Bloom Filter produces a false positive, with false positive rate p_f. Thus:

$$P(A) = P(R \cup \text{false positive}) = P(R) + P(\text{false positive}) - P(R) \cdot P(\text{false positive})$$

For analysis convenience, consider a file with $n = 200$ blocks where the attacker possesses $m = 100$ blocks. With challenge size $J = 5$ and $p_f = 0.05$, equation (4) yields $P(A) = 0.078$. This demonstrates that even when an attacker owns half the file, the probability of passing a single PoW verification remains small. If the attacker possesses no file data, $P(A) = p_f$, which is negligible in our scheme.

The probability of event R (correctly responding to the challenge array) is calculated as follows. The CSP' s challenge contains J block indices. The attacker must respond using only their m blocks. Since challenges are random, the probability that the J challenged blocks are all among the attacker' s m blocks is:

$$P(\text{all } J \text{ blocks in } m) = \frac{\binom{m}{J}}{\binom{n}{J}}$$

When the challenge includes blocks the attacker lacks, they can only guess content or hash values, with success probability P_2 approaching 0. Therefore:

$$P(R) = \frac{\binom{m}{J}}{\binom{n}{J}} + P_2 \approx \frac{\binom{m}{J}}{\binom{n}{J}}$$

Substituting into equation (2):

$$P(A) \approx \frac{\binom{m}{J}}{\binom{n}{J}} + p_f - \frac{\binom{m}{J}}{\binom{n}{J}} \cdot p_f$$

compares the security of our scheme with other Bloom Filter-based PoW deduplication schemes, where √ indicates resistance and × indicates vulnerability.

4.2 Honest-but-Curious Server

Since the CSP is honest-but-curious and may view or leak user data, our scheme employs convergent encryption to resist this threat. Before uploading to the

cloud, user data is encrypted using convergent encryption, ensuring only ciphertext is stored on the server. As the hash function used in convergent encryption is cryptographically secure, attackers cannot decrypt data without convergent keys. Furthermore, convergent keys are themselves encrypted through the key chaining mechanism and stored as ciphertext on the server, preventing key compromise.

4.3 Resisting Pollution Attacks

In typical cloud storage deduplication systems, the initial user uploads file F and its digest $H(F)$. When other users request downloads using $H(F)$, the CSP sends F. However, if an attacker uploads a different file F' with digest $H(F)$, subsequent users requesting downloads with $H(F)$ will receive F' instead of F, causing data pollution. In our scheme, if the initial uploader sends ciphertext blocks $\{C_i\}$ inconsistent with tag digest h_c, subsequent uploaders will fail PoW verification and legitimate users cannot obtain file permissions, resulting in pollution.

To verify block-tag consistency, initial users must upload ciphertext blocks $\{C_i\}$, tags $\{\text{token}_i\}$, and digest h_c. The server recomputes tags and h'_c from $\{C_i\}$ and compares h'_c with h_c. If they match, no pollution exists; otherwise, pollution is detected. Thus, our scheme effectively resists pollution attacks.

5 Performance Analysis

We compare our scheme with the scheme in 16, which also combines convergent encryption and Bloom Filter for PoW deduplication.

5.1 Bloom Filter False Positive Rate

Our scheme primarily considers Bloom Filter false positives. The *active* state indicates the false positive rate is below the threshold, while *full* state indicates it has reached the threshold. Let f_{SBF} denote the false positive rate of Standard Bloom Filter (as in equation (1)), and f_{DBF} denote the false positive rate of Dynamic Bloom Filter.

From Section 3.1, DBF contains multiple SBFs. If the number of elements does not exceed a single SBF' s capacity ($n \leq c$), DBF effectively functions as an SBF with $f_{DBF} = f_{SBF}$. If $n > c$, DBF contains $s-1$ full SBFs and 1 active SBF. The full SBF false positive rate follows equation (1), while the active SBF rate substitutes element count $n-(s-1)c$ into equation (1). The f_{DBF} formula is:

$$f_{DBF} = \begin{cases} \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k & n \leq c \\ \left(1-\left(1-\frac{1}{m}\right)^{k(n-(s-1)c)}\right)^k \cdot \left(1-\left(1-\frac{1}{m}\right)^{kc}\right)^{k(s-1)} & n > c \end{cases}$$

In 16, Standard Bloom Filter is used, yielding a single false positive rate:

$$f_{SBF} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-kn/m}\right)^{k}$$

For illustration, with $k = 4$, $m = 2000$, $c = 200$, [Figure 4: see original paper] plots false positive rates versus element count n. The results show SBF' s false positive rate increases sharply with elements, while DBF' s increases slowly and remains within acceptable bounds. Since 16 uses SBF and our scheme uses DBF, our scheme achieves lower false positive rates, enhancing security as analyzed in Section 5.1.

5.2 Key Storage Overhead

Assume n users have permission for file F. Our key chaining mechanism requires each user to store only the first block' s convergent key K_1 locally, while remaining keys are stored as ciphertext $\{CK_i\}$ on the server in a single copy shared across users. During download, users recursively decrypt $\{CK_i\}$ using K_1 to obtain all $\{K_i\}$. Since keys are stored as ciphertext on the server, security is maintained.

In 16, each user encrypts convergent keys with their private key *userKey*, producing different ciphertexts that must each be stored on the server. Additionally, each user must store a private key locally. compares key storage overhead, showing our scheme significantly reduces storage costs.

Table 2. Key Storage Overhead Comparison

Scheme	Client Storage	Server Storage
16	User private key *userKey*	n copies of convergent key ciphertext
Our scheme	1 convergent key K_1	1 copy of convergent key ciphertext

Note: n represents the number of users with file permission.

5.3 Computation Overhead

We analyze computation overhead during PoW protocol execution (the deduplication scenario). Let SHA-256 be the convergent key generation algorithm.

Client-side computation: In 16, overhead includes file digest computation (T_f), file segmentation (T_s), CE algorithm (T_{CE}), block tagging (T_{tag}), and encrypting keys with *userKey* (T_K). Our scheme requires block segmentation (T_s), CE algorithm (T_{CE}), block tagging (T_{token}), and digest computation (T_c). Thus:

-
- Our scheme: $T_c + T_s + T_{CE} + T_{token}$

Server-side computation: Bloom Filter verification using k hash functions on J random blocks costs T_J. In 16, computing verification values L_i costs T_L, while our scheme computes PRF outputs at cost T_{PRF}. Thus:

-
- Our scheme: $T_J + T_{PRF}$

[Figure 5: see original paper] shows computation overhead for files of 64 MB, 128 MB, 256 MB, 512 MB, and 1 GB (4 KB blocks, $J = 5\%$ of total blocks, $k = 4$). Results indicate our scheme' s computation overhead is nearly identical to 16. Our scheme computes tags for each block to prevent pollution attacks (see Section 5.3), while 16 computes only a single file digest, making our computation slightly higher but providing stronger security guarantees.

5.4 Transmission Overhead

Similar to computation overhead analysis, we focus on PoW protocol transmission costs. During the challenge phase, both schemes transmit a challenge array with J block indices and receive responses. After each successful PoW verification, 16 requires uploading convergent key ciphertext, while our scheme avoids repeated key uploads through key chaining. Therefore, our scheme achieves lower transmission overhead.

For example, for a 1 GB file with 4 KB blocks, 16' s additional transmission overhead is approximately 8 MB per verification (using SHA-256 for key generation), while our scheme incurs no such overhead.

6 Conclusion

Addressing deficiencies in current cloud storage deduplication technologies, this paper proposes DBF-Dedup, an improved PoW scheme based on Dynamic Bloom Filter. Our approach mitigates the rapid false positive rate growth in standard Bloom Filter schemes, combines convergent encryption to eliminate the need for trusted third parties or complex key management, and effectively resists honest-but-curious servers and pollution attacks. Security and performance analysis demonstrates that our scheme reduces key storage and transmission overhead while achieving a favorable balance between security and efficiency.

Experimental Environment: AMD 1.8GHz quad-core E2-7110, 8GB RAM, Windows 10, Python, SHA-256 for convergent key generation, SHA-1 for other hash functions.

References

[1] Cisco. Cisco global cloud index (2015-2020) [EB/OL]. (2017) [2018-

06-20]. https://www.cisco.com/c/dam/assets/sol/sp/gci/global-cloud-index-infographic.html.

[2] Shin Y, Koo D, Hur J. A survey of secure data deduplication schemes for cloud storage systems [J]. ACM Computing Surveys, 2017, 49(4): 74.

[3] 熊金波, 张媛媛, 李凤华, 等. 云环境中数据安全去重研究进展 [J]. 通信学报, 2016, 37(11): 169-180. (Xiong Jinbo, Zhang Yuanyuan, Li Fenghua, et al. Research progress on secure data deduplication in cloud [J]. Journal on Communications, 2016, 37(11): 169-180.)

[4] 陈越, 李超零, 兰巨龙, 等. 基于确定//概率性文件拥有证明的机密数据安全去重方案 [J]. 通信学报, 2015, 36(9): 1-12. (Chen Yue, Li Chaoling, Lan Julong, et al. Secure sensitive data deduplication schemes based on deterministic//probabilistic proof of file ownership [J]. Journal on Communications, 2015, 36(9): 1-12.)

[5] Halevi S, Harnik D, Pinkas B, et al. Proofs of ownership in remote storage systems [C]// Proc of the 18th ACM Conference on Computer and Communications Security. New York: ACM Press, 2011: 491-500.

[6] Pietro R D, Sorniotti A. Boosting efficiency and security in proof of ownership for deduplication [C]// Proc of the 7th ACM Symposium on Information, Computer and Communications Security. New York: ACM Press, 2012: 81-82.

[7] Blasco J, Pietro R D, Orfila A, et al. A tunable proof of ownership scheme for deduplication using Bloom filters [C]// Communications and Network Security. Piscataway, NJ: IEEE Press, 2014: 481-489.

[8] Yu Chiamu, Chen Chiyuan, Chao Hanchieh. Proof of ownership in deduplicated cloud storage with mobile device efficiency [J]. Network IEEE, 2015, 29(2): 51-55.

[9] Xu Jia, Chang Ee-chien, Zhou Jianying. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage [C]// Proc of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. New York: ACM Press, 2013: 195-206.

[10] Bellare M, Keelveedhi S, Ristenpart T. DupLESS: server-aided encryption for deduplicated storage [C]// Proc of the 22nd USENIX Security Symposium. Berkeley, CA: USENIX Association Press, 2013: 179-194.

[11] Puzio P, Molva R, Loureiro S. ClouDedup: secure deduplication with encrypted data for cloud storage [C]// Proc of the 5th IEEE International Conference on Cloud Computing Technology and Science. Piscataway, NJ: IEEE Press, 2013: 363-370.

[12] Stanek J, Sorniotti A, Androulaki E, et al. A secure data deduplication scheme for cloud storage [C]// Proc of International Conference on Financial Cryptography and Data Security. Berlin: Springer, 2014: 99-118.

[13] 毕朝国, 徐小龙. 一种云存储系统中重复数据删除机制 [J]. 计算机应用研究, 2014, 31(10): 3052-3055. (Bi Chaoguo, Xu Xiaolong. Data de-duplication mechanism

for cloud storage systems [J]. Application Research of Computers, 2014, 31(10): 3052-3055.)

[14] Miao Meixia, Wang Jianfeng, Li Hui, et al. Secure multi-server-aided data deduplication in cloud computing [J]. Pervasive & Mobile Computing, 2015, 24: 129-137.

[15] González-Manzano L, Orfila A. An efficient confidentiality-preserving proof of ownership for deduplication [J]. Journal of Network & Computer Applications, 2015, 50: 49-59.

16 刘竹松, 杨张杰. 基于布隆过滤器所有权证明的高效安全可去重云存储方案 [J]. 计算机应用, 2017, 37(3): 766-770. (Liu Zhusong, Yang Zhangjie. Efficient and secure deduplication cloud storage scheme based on proof of ownership by Bloom filter [J]. Journal of Computer Applications, 2017, 37(3): 766-770.)

[17] 张曙光, 咸鹤群, 刘红燕, 等. 云存储中加密数据的自适应重复删除方法 [J]. 计算机应用研究, 2018, 35(9): 2772-2776. (Zhang Shuguang, Xian Hequn, Liu Hongyan, et al. Adaptive deduplication method for encrypted data in cloud storage [J]. Application Research of Computers, 2018, 35(9): 2772-2776.)

[18] Bloom B H. Space//time trade-offs in hash coding with allowable errors [J]. Communications of the ACM, 1970, 13(7): 422-426.

[19] Guo Deke, Wu Jie, Chen Honghui, et al. The Dynamic Bloom Filters [J]. IEEE Trans on Knowledge & Data Engineering, 2009, 22(1): 120-133.

[20] Douceur J R, Adya A, Bolosky W J, et al. Reclaiming space from duplicate files in a serverless distributed file system [C]// Proc of the 22nd International Conference on Distributed Computing Systems. Piscataway, NJ: IEEE Press, 2002: 617-624.

Note: Figure translations are in progress. See original paper for figures.