PASCOINFOG/PASFOG: privacy-preserving data deduplication algorithms for fog storage systems

Article

Accepted Version

PASCOINFOG/PASFOG: Privacy-preserving Data Deduplication Algorithms for Fog Storage Systems

Zahra Pooranian
Department of Computer Science, University of Reading, UK

Mohammad Shojafar
5G/6GIC, University of Surrey, UK

Rahim Taheri
University of Portsmouth, UK

Rahim Tafazolli
5G/6GIC, University of Surrey, UK

***Abstract*—The forthcoming Fog storage system should provide end users with secured and faster access to cloud services and minimise storage capacity using data deduplication. This method stores a single copy of data and provides a link to the cloud/fog owners. In client-side data deduplication, the system can reduce network bandwidth levels by *duplicate check*. This solution fails to cover user privacy and optimise the latency of real-time communications.**

Motivated by this, this magazine paper develops PrivAcy-preServing data deduplication in Fog stOraGe system (PASFOG) as a data deduplication protocol implemented between cloud storage and users to mitigate brute-force and poison attacks. PASFOG is implemented in fog computing to reduce real-time delay and communication when performing duplicate checks. Also, we propose PrivAcy-preServing data dedupliCatiOn in blockchaIN-based Fog stOraGe system (PASCOINFOG) utilised blockchain techniques to realise a reliable system. In PASCOINFOG, when users want to send chunks to the cloud/fog nodes, process the duplicate check and create a new block for the blockchain to reduce real-time latency/communication and protect the cloud from attackers. The proposed protocols can enhance user privacy and reduce real-time communication delay, crucial for consumer electronics applications such as cloud storage and IoT devices.

THE POPULARITY OF CLOUD STORAGE is increasing due to its cross-platform, high availability, high scalability, and nearly unlimited capacity to outsource data [1]. However, the growth of data generated by the Internet of Things (IoT) shows that centralized cloud storage can not provide service satisfaction to users. Fog computing acts as an intermediate layer between cloud computing and IoT devices to provide

Digital Object Identifier 10.1109/MCE.2023.Doi Number

Date of publication 00 xxxx 0000; date of current version 00 xxxx 0000

end users with faster access to cloud services such as storage, computing, and networking capabilities by using Fog devices near IoT devices [2]. On the one hand, centralized cloud storage cannot quickly address large volumes of data due to limited network bandwidth. In contrast, distributed storage, i.e., fog machines, cannot provide users with universal and permanent computing services due to limited resources. Hence, commercial online storage services aim for efficient resource management between cloud and fog devices (especially storage space and network bandwidth). Some problems, such as insufficient storage space and bandwidth, are due to explosive data growth. The data deduplication technique solves these problems to save bandwidth and minimise storage capacity [3] by storing a single copy of data and providing a link to the owners. Client-side data deduplication, a type performed on the client side, can reduce network bandwidth levels by *duplicate check* [4]. The *duplicate check* process is as follows: the user first calculates the hash of the chunk $h(ch)$ (where $h(\cdot)$ refers to the cryptographic hash function), then sends it to the could. Then, if the file does not exist, the cloud sends a negative response, and if the file exists, a positive response. Upon receiving a positive response, the data deduplication will be done.

However, applying deduplication in fog storage scenarios can cause the risk of attack from an external attacker and an insider (e.g., the Fog storage provider). Another critical point is that the standard deduplication method cannot guarantee data confidentiality because the application scenario needs different requirements. Privacy-conscious clients prefer to encrypt their data on the client side utilizing a semantically secure cryptosystem. Conventional encryption is the first way to encrypt data to protect data confidentiality, but it disables deduplication because the same plaintext generates different ciphertexts under different keys. To enable data deduplication, other solutions, like convergent encryption (CE) [5] and message-locked encryption (MLE) [6], use plaintext to generate keys to generate the ciphertext. However, the external attacker can perform a brute force attack to obtain the user's private data because the file content has low min-entropy. Some more solutions like DupLESS [7], Encrypt-with-Signature (EwS) [8], and ClouDedup [9] use additional independent servers to generate the keys which are difficult to meet in commercial contexts. Moreover, these solutions have the potential risk of a single point of failure. The authors in [10] claim to eliminate the need for an independent server. They use password authenticated key exchange (PAKE) and lightweight symmetric additive homomorphic encryption protocols in key sharing, leading to computation inefficiency. In addition, due to the significant computational resources required for model training, it is not possible to apply and implement methods based on machine learning or deep learning models on massive collected data to preserve privacy [11]. Another type of attack called *poison attack* [6] tries to create a mismatch between the tag of chunk $h(ch)$ and the stored chunk ch'. Consequently, the subsequent users with ch will have access to a fake copy of the chunk in the cloud after the duplicate check. Specifically, to poison ch, the external attacker tries to send $h(ch)$ but uploads ch'. In this case, all subsequent users with the tag $h(ch)$ will be referred to the fake chunk ch' in the cloud due to tag matching. Consequently, when users download the chunk ch from the cloud, they will receive the fake chunk ch'.

Secure deduplication methods have been used extensively in cloud storage to reduce network and storage bandwidth usage. Nevertheless, many of these methods consider cloud servers alone and neglect the trade-off between data reliability and confidentiality. These techniques have a single point of failure problem due to using third parties to provide audit or key management services. These schemes are expected to adversely affect many users storing the duplicate chunk when the only chunk stored in the cloud is destroyed (e.g., Fog nodes in our case).

The proposed methods have potential applications in various consumer electronics devices and systems where users must store and access data frequently and securely. The PASFOG and PASCOINFOG protocols can significantly reduce network bandwidth, minimise storage capacity, and protect user privacy by preventing brute-force and poison attacks. In addition, these protocols can optimise the latency of real-time communications, which is crucial for applications such as video conferencing, online gaming, and live streaming.

Some research [12] has focused on the blockchain-based storage system by taking advantage of the blockchain's unique features, including immutability, data element-level security, and distribution (no point of failure). For example, Filecoin [13] is a decentralised storage system built on the blockchain to achieve secure chunk storage via an incentive mechanism and distributed structure. Nevertheless, traditionally they try to achieve data reliability by using tradi-

tional distributed. They provide high fault tolerance by storing multiple complete copies while ignoring the additional waste of system resources. The paper in [14] proposes a system built on a smart contract, which enables cross-user secure deduplication service on encrypted data for decentralised cloud storage. However, they used public blockchain for transparency, which has high latency and low throughput.

As the demand for cloud services grows, faster and more secure data storage and access are becoming increasingly critical. With the proposed methods, the security and privacy of user data are maintained while optimising the real-time communication latency. This makes them ideal for consumer electronics devices and systems such as smart homes, video surveillance systems, and online gaming platforms. The proposed protocols can significantly reduce network bandwidth levels, minimise storage capacity, and protect user privacy, all while optimising the efficiency of fog computing systems. The contributions of this magazine paper are summarised as follows.

- We develop a blockchain-based Fog storage architecture for higher communication performance and elastic expansion of fog demands. The deduplication mechanism is performed in Fog nodes to reduce the delay.
- We designed two data deduplication protocols, PASFOG and PASSCOINFOG, using homomorphic encryption properties to help the cloud find that two distinct encrypted chunks are from the same content.
- We combine Fog and blockchain with the standard cloud storage systems to reduce delay and increase the system's reliability.
- We validate PASFOG and PASCOINFOG in the case of communication and deduplication costs using the Enron [15] dataset.

The rest of this paper is organized as follows. The next section summarises the proposed architecture and security model. Then, we explain the proposed methods. The performance evaluation of both methods comes afterwards. Finally, the work concludes the core at the end of this magazine paper.

PROPOSED ARCHITECTURE

We first present the blockchain-based Fog storage architecture. Then, we present the storage and threat models.

The Considered Architectures. We design our data deduplication model on Fog nodes connected through links, as shown in Fig. 1. To support the efficiency of the fog storage system, we integrate blockchain technology in a way that encrypts the various data coming from different user applications, which are in the IoT layer (lower part of Fig. 1). All the Fog nodes located in the fog node tier (middle part of Fig. 1) are connected to the cloud storage instantiated on the servers in the cloud tier (upper part of Fig. 1). We use blockchain technology to create a distributed deduplication scheme to provide secure, high-reliability Fog storage. There are two types of blockchain data storage: off-chain and on-chain. The on-chain principle means that all user data is stored in the blockchain in each block. The disadvantage of this method is the network overhead because even if the user uploads only a few megabytes of data, the network becomes overloaded. On the other hand, off-chain methods solve the scalability problem by storing metadata in on-chain storage and not storing data on the blockchain. Hence, our protocol combines off-chain and on-chain (middle and pink part Fig. 1). As we can see, users 1 and 2 of the IoT layer upload duplicate files to the fog node. Meanwhile, the adversary seeks to obtain information about the file and access the encrypted data to change its integrity in the fog node. We need to enforce our cloud server to maintain the security and privacy of the data while complying with the data deduplication technique in the system.

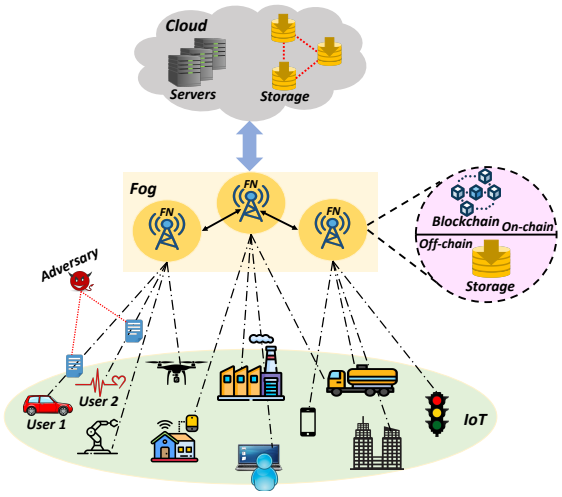


Figure 1. The fog storage system architecture; FN := Fog Node.

Storage Model. Implementing client-side data dedu-

plication in the cloud $\mathcal{C}$ helps reduce storage space and bandwidth. This paper uses the cryptographic hash function $h(\cdot)$ (e.g., SHA256), symmetric encryption $E_k(\cdot)$ (e.g., AES-CBC), and message authentication code (MAC) $h_k(\cdot)$ (e.g., HMAC-SHA256). This paper assumes that the key k generated by user u creates an individual and is not shared with anyone else.

In our setting, the user is assumed to upload a low-entropy chunk ch to the cloud, representing all chunks as CH. Given basic knowledge, such as the format of the document, the minimum entropy of the files is, in fact, usually low and has a high potential for predictability. It is also assumed that the user creates the full-length *chunk hash* $h(ch)$ and *short hash* $sh(ch)$. Short hash can be created by storing part of the bits of $h(ch)$. Using a short hash can reduce the possibility of a brute-force attack because it has a high collision rate, making it difficult for an attacker to reveal the chunk. In total, *chunk key* k_{ch} is calculated by the user to encrypt the chunk and to upload $\mathsf{E}_{k_{ch}}(ch)$.

Threat Model. In the threat model, it is assumed that the cloud is honest-but-curious and user $\hat{u}$ (external attacker) is legitimate but malicious. They aim to reveal a chunk or chunk key belonging to the legitimate user u. If the chunk has low min-entropy, malicious users and the cloud may try to find them by the attack of *internal brute-force* and *external brute-force*.

In an external (internal) brute-force attack, the attacker constructs deterministic representations of all candidate chunks with low min-entropy content to determine which matches the eavesdropping (received). Since chunk is predictable, the degree of randomness in k determines whether brute force can be efficient. The external attacker aims to find out the current status of the chunk and make the tag inconsistency on the chunk. In detail, the attacker tries to discover the existence status of the chunk, then tries to upload poisoned chunk ch' after a duplicate check. It is necessary to ensure that the chunk does not exist in the cloud to perform a poison attack on the chunk, so the attacker must confirm that the chunk does not exist and then upload the poisoned chunk.

The internal attacker will also behave independently of the $\hat{u}$ to perform the side channel and poison attack. In particular, the collision between $\hat{u}$ and c makes side channel and poison attack trivial. c just returns the existence status of the chunk to $\hat{u}$ to reveal privacy and replaces ch with ch' to spoof the content.

PROPOSED METHODS: PASFOG AND PASCOINFOG

PASFOG Algorithm. We can have the following cases to upload a specific piece to the cloud. In the first case, the cloud neither has a copy of a chunk nor a short hash in the index table. The second happens when the cloud finds a short hash in the index table but does not have a file copy. The third one indicates when the cloud has a saved file copy. Algorithm 1 shows the description of the PASFOG protocol.

The algorithm starts by calculating the double hash $\mathcal{D}(ch)$ and the salted hash $\mathcal{S}(ch)$. Double hash refers to hashing $h(ch)$, and salt hash refers to hashing $h(ch)$ plus a salt value (for example, $sh(ch)$). In line 3, the user calculates the short hash and uploads it to the cloud. Because the short hash has a high collision rate and cannot reveal sensitive file content, it resists brute force attacks. Then, the cloud searches the index table $\mathcal{T}$ (**case 1** to find a match in the Algorithm 1). The cloud sends back a negative response, meaning the short hash is not in the table (Line 4). Then, the user receives a negative response and learns he is the first to upload a chunk. After that, the user generates a random key with a high min-entropy (line 5). In lines 6 and 7, the user calculates and uploads $E_{k_{ch}}(ch)$ and the required indexes, $\mathcal{S}(f)$, $E_{\mathcal{D}(ch)}(k_{ch})$, and $E_{k_{ch}}(ch)$, into the cloud. The user stores the key in local memory and removes all intermediate elements. In Line 8, while there is no information that the user has uploaded the chunk once, the user still tries to upload a chunk to the cloud storage. The user calculates and uploads a short hash. The cloud returns a positive response, meaning it could find the short hash in the index table. Therefore, when the user receives a positive response, he can assume that the chunk has already been uploaded by somebody else. In this case, the user uploads the salted hash (line 9) to the cloud to help the cloud find the exact record m (where m includes the record $[E_{\mathcal{D}(ch)}(k_{ch}), E_{k_{ch}}(ch)]$ such that $sh(ch)$= $sh(ch')$ and $\mathcal{S}(ch)$= $\mathcal{S}(ch')$ in the index table. The combination of the short hash and salted hash is unique. Hence, the cloud can find the exact record that matches them and return the matching record to the user (line 9). In addition, this combination reduces record-finding time, so we have a faster algorithm than LEVER [4]. Moreover, the PASFOG scheme provides secure protection against a poison attack because the combination is identical, and an attacker cannot manipulate it to create tag inconsistency. Then,

Algorithm 1 PASFOG.

Offline Stage

u keeps matching record m, and chunk keys table k

$\mathcal{C}$ keeps an index table $\mathcal{T}$

Online Stage

1: u : compute $\mathcal{D}(ch) = h(h(ch))$
2: u : compute $\mathcal{S}(ch) = h(h(ch) + sh(ch))$
3: $u \rightarrow \mathcal{C} : sh(ch)$
4: **if** $\mathcal{C} : \mathcal{T}(sh(ch)) = \emptyset$ **then** (**case 1**)
5: u : choose a random key k_{ch}
6: $u \rightarrow \mathcal{C} : \mathcal{S}(ch), E_{\mathcal{D}(ch)}(k_{ch}), E_{k_{ch}}(ch)$
7: $\mathcal{C} : \mathcal{T} = \mathcal{T} \cup [sh(ch), \mathcal{S}(ch), E_{\mathcal{D}(ch)}(k_{ch}), E_{k_{ch}}(ch)]$
8: **else**
9: $u \rightarrow \mathcal{C} : \mathcal{S}(ch)$
10: $\mathcal{C} \rightarrow u : m$, where $[E_{\mathcal{D}(ch)}(k_{ch})] \in m$ s.t. $sh(ch) = sh(ch')$ and $\mathcal{S}(ch) = \mathcal{S}(ch')$
11: u : obtain $k_{ch} = D_{\mathcal{D}(ch)}(E_{\mathcal{D}(ch)}(k_{ch}))$
12: $u : k = k \cup k_{ch}$
13: **if** u drive $E_{k_{ch}}(ch)$ with $k_{ch} \in k$ **then** (**case 2a**)
14: $u \rightarrow \mathcal{C} : E_{k_{ch}}(ch)$
15: **else** (**case 2b**)
16: u : choose a random key k_{ch}
17: $u \rightarrow \mathcal{C} : \mathcal{S}(ch), E_{\mathcal{D}(ch)}(k_{ch}), E_{k_{ch}}(ch)$
18: $\mathcal{C}: \mathcal{T} = \mathcal{T} \cup [sh(ch), \mathcal{S}(ch), E_{\mathcal{D}(ch)}(k_{ch}), E_{k_{ch}}(ch)]$
19: **end if**
20: **end if**

by performing decryption on $E_{\mathcal{D}(ch)}(k_{ch})$, the user tries to get a chunk key for the chunk from the received matching record (line 11).

Two things can happen when the user drives the key. This key can be the valid key for the chunk or the opposite case where it is not the correct key for the chunk. Therefore, the user uses the key to encrypt the chunk. The second upload happens if the user drives $E_{k_{ch}}(ch)$ (**case 2a** in Algorithm 1). Then, the user only uploads $E_{k_{ch}}(ch)$ to find a match in the index table (line 14). Otherwise, this is the case (**case 2b** in Algorithm 1) where nobody uploads the file and follows the procedure as the first upload (line 15-18).

PASCOINFOG Algorithm. The PASCOINFOG algorithm is the same as PASFOG. However, we used the features of blockchain technology to create a distributed deduplication scheme to provide secure, high-reliability Fog storage. Based on the explanation in Section , we used an off-chain blockchain to solve the scalability problem in the storage system. Therefore, we only store the metadata (e.g., $sh(f), \mathcal{S}(f)$) in the blockchain and the file in the cloud storage.

PERFORMANCE EVALUATIONS

This section presents the performance evaluation metrics for PASFOG and PASCOINFOG and the corresponding results.

Evaluation Metrics. The performance evaluation of the data deduplication protocols is based on the following metrics [16]. First, u uploads a discrete random ch to $\mathcal{C}$ for performance evaluation, and second, the requests to upload are uniformly distributed.

Communication Cost ($\mathbb{C}$). The proportion between communication cost and deduplication percentage is reversed in client-side deduplication. Nevertheless, there may be a slight difference between different implementations based on the protocol design. This difference may affect the user experience and create a bottleneck from a cloud perspective.

Deduplication Percentage ($\mathbb{D}$). The effectiveness evaluation of data deduplication calculates by this percentage [17]. It is defined as $1 - \frac{1}{\omega_o/\omega_i}$, where ω_o is input byte numbers and ω_i is output byte numbers from the deduplicated storage space. Those deduplication protocols that detect all duplicates and achieve a deduplication ratio of $1 - 1/b = (b-1)/b$ given b similar uploaded files have a perfect performance.

PASFOG Settings. PASFOG settings are presented below.

$\mathbb{C}_{\text{PASFOG}}$: The number of bits required for exchanging messages in the PASFOG is shown in Fig. 2a. It is obvious from Algorithm 1 and Fig. 2a that in the case of $\mathcal{T}(sh(ch)) = \emptyset$, u must have send $L_{sh} + L_{\mathcal{S}} + L_k + L_{ch}$ bits to upload the required information to $\mathcal{T}$ and $E_{k_{ch}}(ch)$. Conversely, when $\mathcal{T}(sh(ch)) \neq \emptyset$, the exact number of bits is L_{ch}. Moreover, for ease of calculation, we also assume that $pr[sh(ch) \neq \emptyset | ch \notin \mathcal{C}] = 1$. Therefore, the $\mathbb{C}_{\text{PASFOG}}$ equation is $\mathbb{C}_{\text{PASFOG}} = pr(L_{\mathcal{S}} + L_k + L_{ch}) + (1-pr)(L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}} + L_k + L_{ch})$.

$\mathbb{D}_{\text{PASFOG}}$: This metric shows the amount of space occupied in different uploading cases in the PASFOG approach. As a result, $\mathbb{D}_{\text{PASFOG}}$ can be estimated approximately as $\mathbb{D}_{\text{PASFOG}} = 1 - \frac{1}{L_{ch}/((pr(L_{ch}))+(1-pr)(L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}))}$.

PASCOINFOG Settings. PASCOINFOG settings are presented below.

$\mathbb{C}_{\text{PASCOINFOG}}$: The number of bits required for exchanging messages in the PASCOINFOG is shown in Fig. 3a. Thus, the $\mathbb{C}_{\text{PASCOINFOG}}$ equation is as $\mathbb{C}_{\text{PASCOINFOG}} = pr(L_{\mathcal{S}}+L_k+L_{ch})+(1-pr)(L_{\mathcal{S}}+L_k+L_{ch}+L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}+L_{sh}+L_{\mathcal{S}}+L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}+L_{sh}+L_{\mathcal{S}})$.

$\mathbb{D}_{\text{PASCOINFOG}}$: Fig. 3b shows the amount of space occupied in different cases of uploading in the PASCOINFOG approach. The difference between PASCOINFOG and PASFOG communication cost is the number of bits $sh(ch)$ and $\mathcal{S}(ch)$ sent

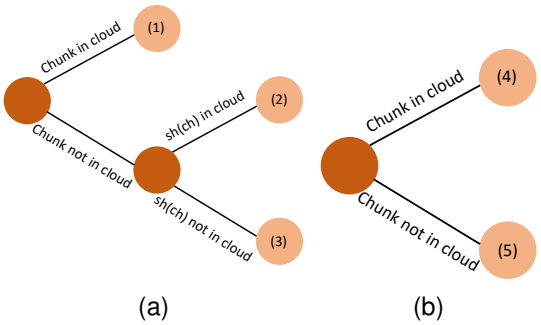


Figure 2. PASFOG metrics. (a) Communication Cost and (b) Deduplication Percentage where (1) $L_{\mathcal{S}} + L_k + L_{ch}$, (2) $L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}} + L_k + L_{ch}$, (3) $L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}$, (4) L_{ch}, and (5) $L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}$.

to the blockchain. As a result, $\mathbb{D}_{\text{PASCOINFOG}}$ can be approximated as $\mathbb{D}_{\text{PASCOINFOG}} = 1 - \frac{1}{(L_{ch})/((pr(L_{ch}))+(1-pr)(L_{sh}+L_{\mathcal{S}}+L_k+L_{ch}+L_{sh}+L_{\mathcal{S}}))}$.

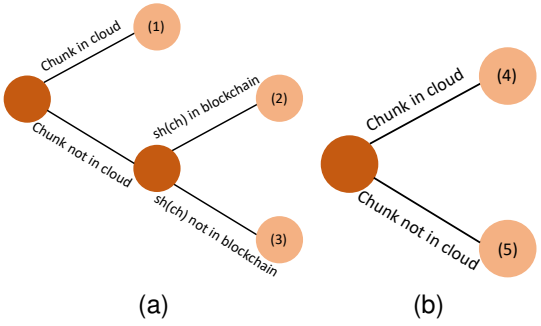


Figure 3. PASCOINFOG metrics. (a) Communication Cost and (b) Deduplication Percentage where (1) $L_{\mathcal{S}} + L_k + L_{ch}$, (2) $L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}}$, (3) $L_{sh} + L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}}$, (4) L_{ch}, and (5) $L_{sh} + L_{\mathcal{S}} + L_k + L_{ch} + L_{sh} + L_{\mathcal{S}}$.

Scenario Description. Here, we comprehensively evaluate PASFOG compared to PASCOINFOG, normal data deduplication (N-DD), and without data deduplication (W-DD) protocols. The Enron [15] email dataset is selected to run the experiments, whose traces are shown in Fig. 4. We choose this dataset because we believe regular users want to back up email content to cloud storage. We run tests on a 2.9 GHz Intel Core i5 with 8 GB of RAM on a 64-bit Apple Mac operating system. For the hash function, we used SHA-256 in the OpenSSL library. We follow the file configuration details reported in [18].

Numerical Results. We present numerical results for both proposed algorithms.

Chunk size impact (L_{ch}). We first use variable chunk sizes to evaluate PASFOG and PASCOINFOG by comparing the communication cost and percentage

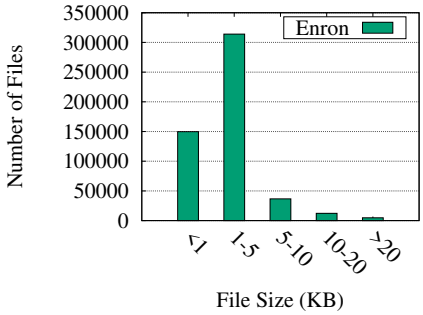


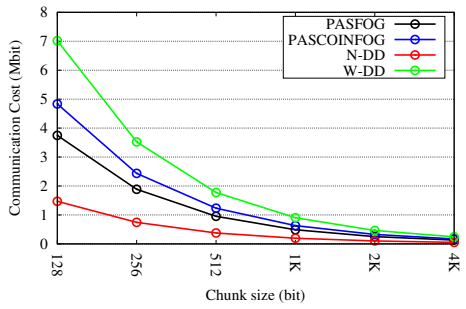
Figure 4. The statistics of Enron email dataset [15].

of deduplication with N-DD and W-DD performance. During the experiment, we fix the length of the short hash to 8 bits (for example, $L_{sh} = 8$) and the probability of each chunk to 0.3 (for example, $pr = 0.3$).

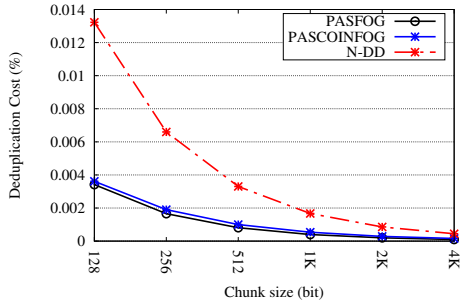
PASFOG and PASCOINFOG can handle data deduplication for storage to help reduce communication and deduplication costs by increasing chunk size. According to the results shown in Fig. 5, both PASFOG and PASCOINFOG could manage the deduplication cost like N-DD but with a minimal higher communication cost. The difference in PASFOG is due to the data encryption we used for each chunk, so we have heavier communications between the user and server for security reasons. This difference is even more significant in PASCOINFOG because we use the blockchain for uploading, so PASCOINFOG spends more resources to transfer files than PASFOG and N-DD. However, PASCOINFOG provides users with a better experience in terms of reliability due to blockchain. Furthermore, since W-DD does not consider deduplication, its percentage is zero.

Persistence probability of a chunk impact (pr). The second part of the simulations is the evaluation of PASFOG and PASCOINFOG with different probability persistence. As shown in Fig. 6, we use chunk size = 128, 4K and $L_{sh} = 8$ for different probabilities, and the results show that when the probability persistence of the chunk increases, the communication cost for PASFOG and PASCOINFOG is decreasing while the percentage of deduplication is increasing. This happens because cloud storage performs data deduplication to save storage and bandwidth. Moreover, the convergence of PASFOG and PASCOINFOG is faster than N-DD because they depend on different P, while N-DD is independent of P.

Short hash length impact (L_{sh}). Third, in Fig. 7, variable short hashes is used to evaluate PASFOG and



(a)



(b)

Figure 5. The impact of chunk size on (a) communication cost and (b) deduplication cost.

PASCOINFOG with communication cost and deduplication percentage against N-DD and W-DD. However, in this case, two values for L_{sh} and pr is considered to be $L_{sh} = 8, 16$ bits and $pr = 0.3, 0.8$, respectively. Comparing Figs. 7a and shows that the communication cost for PASFOG and PASCOINFOG increases when the short hash size is long. Deduplicating with a longer short hash means that each layer has to transfer more bits during duplicate check, so the communication cost increase for longer short hash values. However, these features give the algorithms more adjustability because we can optimise the performance by adjusting the short hash length in real-world applications. It should be noted that the cost for O-DD is fixed b ecause i t i s i ndependent o f t he s hort hash. Conversely, the deduplication cost decreases because the longer short hash has fewer collisions; hence, the cloud can perform deduplication accurately.

CONCLUSION

This paper proposed two encrypted data recovery protocols, PASFOG and PASCOINFOG, to deal with brute force and poison attacks in cloud storage systems. In PASFOG, the deduplication mechanism is performed in Fog nodes to reduce the delay. Then, the combination of the blockchain technique and PASFOG is used to produce the PASCOINFOG protocol to have a reliable and high-performance system. These advancements in fog computing and blockchain techniques can pave the way for the next generation of consumer electronics devices, prioritising user privacy, security, and efficiency. The security analysis of both protocols was proved through the ideal/actual paradigm. In addition, numerical simulations were shown to analyse the performance of PASFOG and PASCOINFOG. We plan to enhance PASCOINFOG's privacy or address side-channel attacks within cloud storage. This will be achieved by establishing a probabilistic connection between the exchanged messages.

ACKNOWLEDGEMENT

This work has been partially funded by the European Space Agency (ESA) as part of the AutoTrust Project, Grant number4000135646/21/UK/ND.

REFERENCES

1. K. Gai *et al.*, "Blockchain meets cloud computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 22, pp. 2009–2030, 2020.
2. T. Wang *et al.*, "Fog-based evaluation approach for trustworthy communication in sensor-cloud system," *IEEE Communications Letters*, 2017.
3. N. Mageshkumar *et al.*, "An improved secure file deduplication avoidance using ckho based deep learning model in a cloud environment," *The Journal of Supercomputing*, vol. 78, no. 13, pp. 14 892–14 918, 2022.
4. Z. Pooranian *et al.*, "Lever: Secure deduplicated cloud storage with encryptedtwo-party interactions in cyber-physical systems," *IEEE Transactions on Industrial Informatics*, 2020.
5. J. R. Douceur *et al.*, "Reclaiming space from duplicate files in a serverless distributed file system," in *IEEE 22nd ICDCS*, 2002.
6. M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Annual Cryptology Conference*. Springer, 2013, pp. 374–391.
7. M. Bellare *et al.*, "Dupless: server-aided encryption for deduplicated storage," in *22nd USENIX Conference on Security*, 2013, pp. 179–194.
8. Y. Duan, "Distributed key generation for encrypted deduplication: Achieving the strongest privacy," in

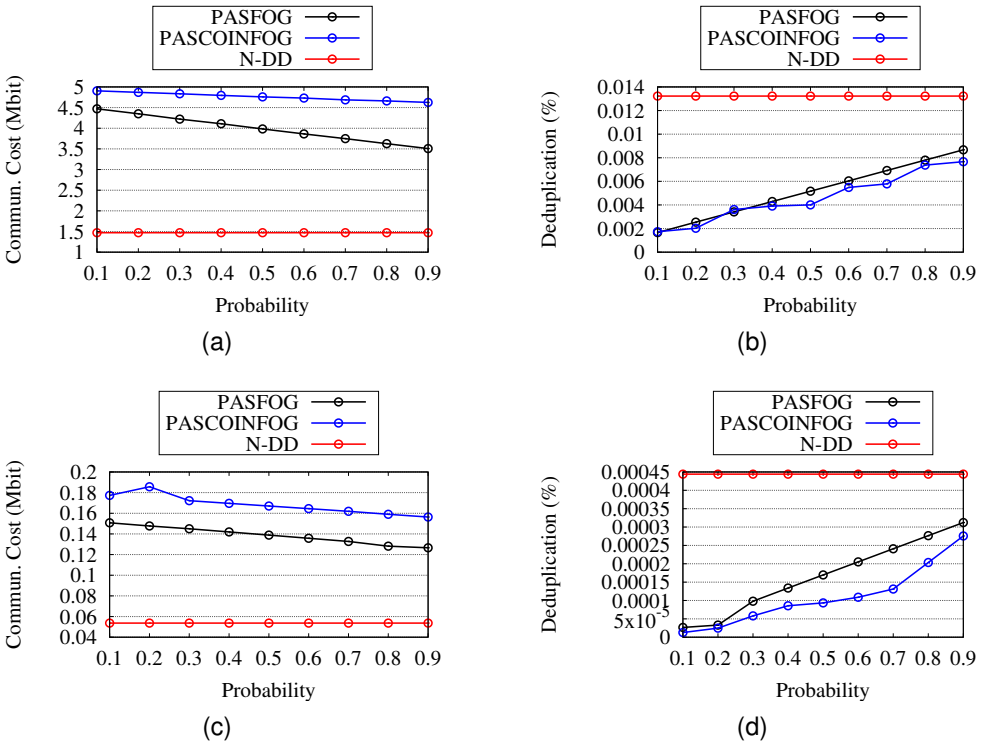


Figure 6. The impact of probability (pr) on the communication cost [(a),(c)] and deduplication cost [(b),(d)].

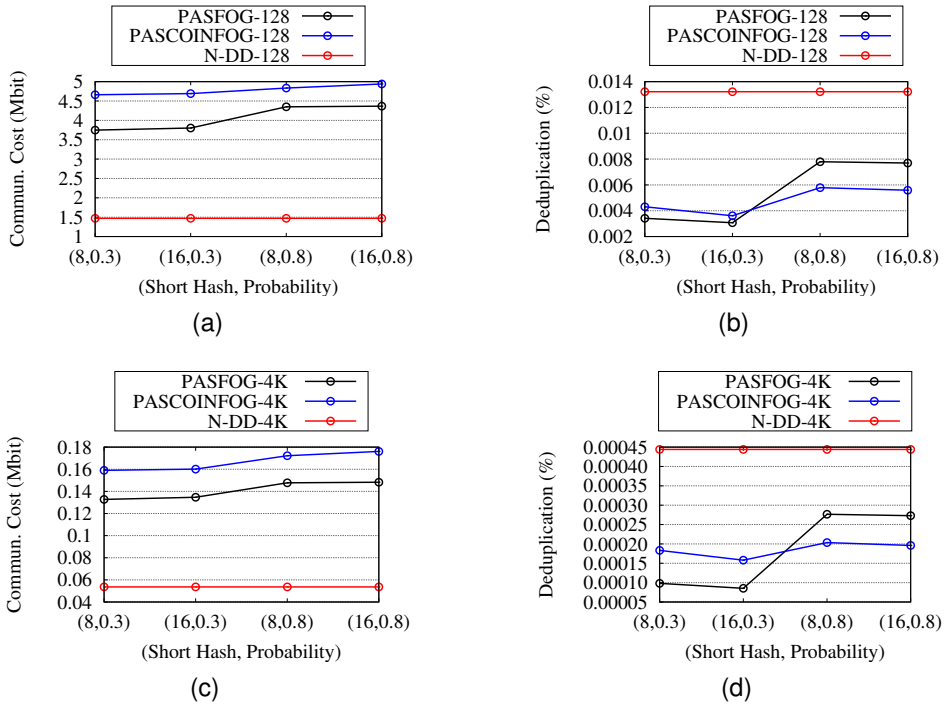


Figure 7. The impact of short hash (Lsh) on the communication cost [(a), (c)] and deduplication cost [(b), (d)].

Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, 2014, pp. 57–68.

9. P. Puzio *et al.*, "Cloudedup: Secure deduplication with encrypted data for cloud storage," in *IEEE 5th*

International Conference on Cloud Computing Technology and Science, vol. 1, 2013, pp. 363–370.
10. H. Shin, D. Koo, and J. Hur, "Secure and efficient hybrid data deduplication in edge computing," *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 3, pp. 1–25, 2022.
11. J. Feng *et al.*, "Tensor recurrent neural network with differential privacy," *IEEE Transactions on Computers*, 2023.
12. J. Li *et al.*, "Block-secure: Blockchain based scheme for secure p2p cloud storage," *Information Sciences*, vol. 465, pp. 219–231, 2018.
13. N. Z. Benisi *et al.*, "Blockchain-based decentralized storage networks: A survey," *JNCA*, vol. 162, p. 102656, 2020.
14. B. Zhang *et al.*, "Enabling secure deduplication in encrypted decentralized storage," in *International Conference on Network and System Security*. Springer, 2022, pp. 459–475.
15. Enron Email Dataset, "Enron," 2020, https://www.cs.cmu.edu/~enron/.
16. C.-M. Yu, "Poster: Efficient cross-user chunk-level client-side data deduplication with symmetrically encrypted two-party interactions," in *ACM SIGSAC*, 2016, pp. 1763–1765.
17. M. Dutch, "Understanding data deduplication ratios," 2008.
18. Z. Pooranian *et al.*, "Rare: Defeating side channels based on data-deduplication in cloud storage," in *IEEE INFOCOM WKSHP*, 2018, pp. 444–449.

Zahra Pooranian is an Assistant Professor at University of Reading, Reading, United Kingdom.

Mohammad Shojafar is an Associate Professor in Network Security at 5G/6GIC, University of Surrey, Guildford, United Kingdom.

Rahim Taheri is an Assistant Professor at University of Portsmouth, Portsmouth, United Kingdom.

Rahim Tafazolli is a Full Professor and head of 5G/6GIC, University of Surrey, Guildford, United Kingdom.