The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04751-6

Secure cloud storage with anonymous deduplication using ID-based key management

Mohammed Gharib[1] · **MohammadAmin Fazli**[2]

Accepted: 30 July 2022

Abstract

Cloud storage systems have been turned into the primary services of Internet users nowadays. While the application of such systems is exponentially increasing, deduplication algorithms help face scalability issues. Although source-side deduplication optimizes both storage and bandwidth, the main concern that deduplication algorithms suffer from is still data confidentiality. Message-locked encryption (MLE) is a well-known key management framework for secure deduplication to provide confidentiality. This framework is the basis of almost all the proposed secure deduplication solutions. Even though there are lots of literature works trying to provide secure deduplication algorithms, to the best of our knowledge, none of them provide an effective anonymity service for data owners. In this paper, we propose an *N*-anonymity algorithm to provide an effective anonymity service, capable of prohibiting even the cloud storage provider from knowing which users are storing the same data. The algorithm is analytically studied, and the results are validated by exhaustive implementations using real data. Furthermore, we propose an ID-based key management algorithm as the cornerstone of the secure cloud storage system. The proposed algorithm, which could be considered as an asymmetric extension of MLE, is easy to implement and compatible with the existed cloud architectures as well as the proposed anonymity-based deduplication system.

Keywords Cloud storage · Deduplication · Anonymity

Mohammed Gharib and MohammadAmin Fazli have contributed equally.

✉ MohammadAmin Fazli
fazli@sharif.edu

Mohammed Gharib
gharib@ipm.ir

[1] Institute for Research in Fundamental Sciences, Tehran, Iran

[2] Sharif University of Technology, Tehran, Iran

Published online: 14 August 2022

1 Introduction

Cloud storage systems have been attracting the attention of many users, while the volume of stored data in such systems has been explosively increasing during the last years [1–4]. The ease of use, accessibility, synchronization between different devices, and reliability of cloud storage systems make them more popular insofar as many users are using them as their main storage [2, 5, 6], i.e., not just as a backup storage [7]. Dropbox [8], Google drive [9], and Mozy [10] are some well-known instances. However, cloud storage systems suffer from scalability issues because many duplicated copies of the same data are stored in the cloud. Deduplication techniques are the best-known solutions to face the explosive increment in cloud storage data [11–14]. Such techniques prohibit redundant data from duplicate storage. According to [15, 16], deduplication can reduce the required storage for backup applications up to 95%. In comparison, up to 68% of the storage required for standard file storage is saved by deduplication. It is worth noting that Dropbox, Google Drive, and Mozy are all using deduplication to optimize their systems [17].

In deduplication techniques, data could be considered as one single file or several equal-sized data blocks. Before storing each file or block of data, it is checked whether the data are formerly stored or not. In the case of former storage, the data are discarded. In the simplest form of deduplication, this check is done after uploading the data. In such a naive technique, which is referred to as server-side deduplication, the bandwidth is wasted in case of former storage. Furthermore, the data are stored in its plain format, while it is well known that attacking the stored data on the cloud storage is not avoidable [18, 19]. Hence, the cloud storage provider (CSP) cannot be assumed to be fully trusted. Accordingly, bandwidth inefficiency and the confidentiality of the stored data are considered the two CSP storage issues in server-side deduplication.

The solution to the wasted bandwidth of server-side deduplication is to transfer the deduplication decision process to the source side. The client can calculate a signature for its file and asks the CSP about a file with such a signature, whether previously stored or not. In a positive case, the client does not send the file to the cloud. Hence, the bandwidth is also saved. Such a signature is referred to as a file tag and could be a fixed-size hash value of the data itself. After uploading its file or ensuring that a copy of the data is already stored on the cloud, the client deletes the file from its own storage and only stores its tag along with a proof of ownership (PoW). PoW is used to retrieve the file [14], while it is almost considered the same as the tag. However, according to the deduplication algorithm, PoW could differ from the tag. Accordingly, for retrieving the data, the client sends the PoW to the CSP, and the CSP, in its turn, checks the matched data and returns the data to the client.

The solution for the second issue, i.e., data confidentiality, is to encrypt data before sending it to the cloud. Since in convenient encryption algorithms, each file or block of data is encrypted with a different client key, the hash value of the encrypted data is not equal to the hash value of the same data encrypted by another client. In such a case, the cloud provider cannot detect the duplicated data, whereas the deduplication process could not be applied correctly. For the

encryption problem to be solved in deduplication algorithms, convergent encryption [20] is proposed. In this solution, the encryption key is extracted from the data itself; thus, the key for the same data becomes unique. The encryption key could be the hash value of the plain data. Hence, the copy of the same data generates the same encrypted data. Accordingly, the deduplication process becomes feasible. It is worthy to note that, using such an idea, the encrypted data are still not confidential. The reason is owing to the hash value of the plain data, which is used some times as the ID of the data. Generally, it is not considered a secret value [19], i.e., any adversary node may have access to the hash value of the plain data.

Bellare et al. [21] proposed a framework that generalizes convergent encryption and comprises all its extensions. The proposed framework is referred to as message-locked encryption (MLE). Both convergent encryption and MLE are built based on a symmetric cryptosystem. In this paper, we propose a new and rigid ID-based key management that makes the secure deduplication system much more resistant against different attacks. The proposed key management uses elliptic curve cryptography (ECC) and could be considered as an asymmetric extension of MLE framework. ID-based key management proposed by Shamir [22] suggests extracting the public key of each user from its unique ID. To the best of our knowledge, it is the first time that ID-based key management has been used to deploy secure data deduplication in cloud storage systems.

It is also notable that, even for the encrypted data, which is confidential for CSP, the cloud storage provider could recognize the users who own the exact copy of the data. For example, consider a tax profile of a specific client or an employee's salary form. The CSP can recognize those who have the same salary or tax statements. The problem of side-channel information leakage in deduplication systems has also been previously mentioned by Zhang et al. [23]. In this paper, we propose an anonymous deduplication approach, based on the N-anonymity concept, for the data to be confidential so that the CSP could not realize who are storing the same data. N-anonymity guarantees that when a client stores the same copy of an existing file on the cloud, there exist at least $N-1$ other files that the CSP cannot recognize which one is the stored file. To reach this aim, we provide a client-side algorithm to find the tag of at least $N-1$ previously stored files so that the CSP cannot distinguish which one is the client's file. It is worth noting that the anonymity problem was proposed earlier, along with some solutions. However, to the best of our knowledge, all proposed solutions gain anonymity at the cost of losing client authentication. Our proposed approach guarantees the N-anonymity, while all the clients are still authenticated. The proposed approach is studied analytically, while its complexity is calculated based on the number of queries needed to be sent to CSP. Exhaustive implementation is also done using real datasets to validate the analytical results. Hereupon, this paper's main contribution is to propose a secure data deduplication system providing N-anonymity service for CSP, using ID-based key management. The more detailed contributions are (i) to propose ID-based key management as the cornerstone of the secure deduplication system, (ii) to propose an N-anonymous, secure deduplication algorithm, (iii) to theoretically analyze the complexity of the proposed anonymity algorithm, (iv) to analyze the security of the proposed secure deduplication system,

and (v) to implement the proposed algorithm exhaustively in a real world and validate the theoretical results.

The rest of this paper is organized in the following. The related works are reviewed in Sect. 2, while the problem is formally defined in Sect. 3. The proposed solutions for both the key management and anonymous deduplication are proposed in Sects. 4 and 5, respectively. Security analysis of the proposed deduplication system, simulation details, and the evaluation process of the *N*-anonymity algorithm are altogether presented in Sect. 6. Finally, the paper is concluded in Sect. 7.

2 Related works

Sun et al. [24] proposed an efficient deduplication system for cloud storage, referred to as DeDu. Although DeDu is an efficient solution for deduplication, it is not applicable to encrypted data. In this section, we review the literature on secure deduplication systems for CSPs where the deduplication system can handle encrypted data. In this case, it needs a rigid key management system capable of efficiently managing the keys such that the CSP assigns a specific key for each block or file of data. The general framework for such a key management system is proposed by Bellare et al. [21] and referred to as MLE. MLE is a symmetric-based encryption system, and since it is a general framework, different secure deduplication instances proposed earlier to MLE are comprised of MLE. However, such instances suffered from the absence of precise definition and theoretical treatment. Bellare et al. [21] formalized the MLE framework, which encompasses most of the former works. Some MLE instances include [20, 25–28] where convergent encryption [20] significantly attracted the attention of many researchers. This attracts several algorithms to be proposed based on the convergent encryption, including but not limited to [29–33]. Accordingly, in this section, we first review the MLE framework and some of its extensions. Next, we review some older works that utilized just the raw idea of convergent encryption. Finally, we review the literature on considering anonymity in deduplication systems and what is their shortcomings that lead to our contribution being needed.

Basically, MLE proposes extracting the key of each data block or file from the data itself. Douceur et al. [20] proposed a convergent encryption system in which the output of a deterministic and cryptographic hash function could be used as the data key. Since the hash algorithm is deterministic, the key is unique for the same data. Accordingly, the encrypted block of the same data becomes equal for all users with the same data, making the deduplication process feasible. On the other hand, the hash value of the data used as the encryption key is encrypted with the client's public key and stored along with the encrypted file as metadata for the PoW. The same authors proposed Farsite [29] to utilize the raw idea of convergent encryption. Storer et al. [30] proposed a chunk-based secure deduplication system, with high similarity to that of [29]. Authors of [30] separated the storage of metadata from that of chunks into two independent servers. Zooko et al. [31] proposed and implemented a convergent encryption-based storage system using a capability access control model. This system, which referred to as Tahoe, quickly became operational and

many customers relied on it for storage. Rahumed et al. [32] presented Fade version, a secure deduplication system capable of deleting some versions of the files permanently, while other versions remain unaffected. This system is also built based on convergent encryption.

Even after proposing MLE [21], many other works are yet proposed based on the raw convergent encryption idea. Puzio et al. [33], as an instance, proposed a secure block-level deduplication system called ClouDedup. Authors of [33] added a new component to support the key management process of block-level deduplication. They showed that the overhead caused by the new component does not seriously affect the overall computational and storage complexity. Wen et al. [34] proposed another convergent encryption-based deduplication system for image files. The authors proposed to use a verification server beside the storage server to be able to verify the deduplication process. As another instance of a convergent-based deduplication system, Li et al. [35] formally addressed the problem of reliably and efficiently managing a huge number of keys. Furthermore, Li et al. [36] proposed a hybrid approach for secure deduplication combined with the authorization process. According to their claim, it is the first attempt to address the authorization problem in secure deduplication formally.

However, as we mentioned earlier, the convergent encryption-based deduplication systems were not defined formally based on a general framework. Bellare et al. [21] proposed a general framework, referred to as MLE, that encompasses the convergent encryption system and all its extensions. MLE uses symmetric encryption to provide secure deduplication. The same authors of MLE [21] mentioned that MLE is subject to brute-force attacks, and accordingly, they proposed DupLESS [27] to improve MLE by making it resistant against brute-force attacks. The authors proposed to add a key server to the existed storage server. The task of the new server is to authenticate the keys without any knowledge about the data itself. Chen et al. [28] proposed a secure deduplication system based on the MLE framework, referred to as block-level MLE (BL-MLE), capable of providing deduplication service for both files and blocks of data. BL-MLE shows more efficiency for large files than its former works.

The whole presented works were proposed based on the raw idea of convergent encryption or under the general framework of MLE. Hence, all of them use symmetric encryption. This paper proposes a new secure deduplication system with asymmetric key management. To the best of our knowledge, this is the first asymmetric-based key management system for secure deduplication in cloud storage. It is worth noting that Yan et al. [17] used an asymmetric key encryption system for secure deduplication just for user authentication. In this work, we propose an ID-based key management scheme to manage the data encryption using an elliptic curve cryptosystem.

Furthermore, this paper proposes a solution to provide N-anonymity for data owners. The anonymity problem was also mentioned earlier in [23, 30, 37]. Storer et al. [30] proposed two secure deduplication systems in which one authenticates the users, while the second, which is an anonymous system, does not do authentication. The solution of [30] is to gain anonymity at the cost of losing authentication. In the other work, Jung et al. [37] proposed an alternative attribute-based solution in which

each data file has its attributes, and the data key is extracted from the concatenation of the attributes. Since such a system is built based on the attribute of the data, users are kept anonymous. Basically, in [37] the storage process is handled by the data attributes, and it is not user-aware system at all. Accordingly, the client is authenticated neither in [30] nor in [37]. Zhang et al. proposed a k-anonymity solution to prevent information leakage by side-channel attacks. In a side-channel attack, the attacker tries to get access to some confidential information, such as patient information, from the uploaded files. In this case, it makes the same file with the patient information and changes the disease name. If the server tells the attacker that the same file already exists, the attacker knows that the patient has the disease. The idea of [23] is not to let the user know the same file is already uploaded except if there are at least k other copies of the same file stored on the server. In this case, even if the file for a patient with the same information is uploaded, the server tells the attacker to upload its own file. This clever solution, however, has the disadvantage of storage overhead. Furthermore, if the attacker knows the value of k, it can upload $k-1$ the same file and check whether it gets the duplication message from the server or not. In the N-anonymity solution proposed in this work, the clients are authenticated, while the CSP cannot recognize between N different files which one is owned by the specific client.

3 System model

In this section, the detailed system model is described, and further, the problem of secure deduplication in CSP is formally presented. We consider a typical secure CSP system with deduplication. Such a standard system has a general architecture shown in Fig. 1 which is matched with many existing secure cloud storage systems such as [17, 33, 36, 37]. This paper proposes an asymmetric ID-based key

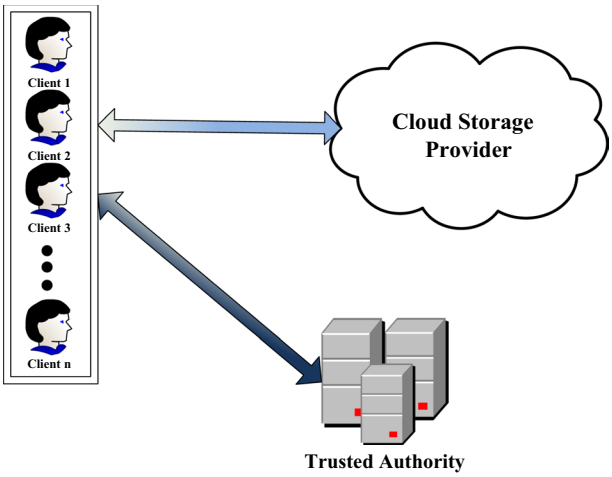

Fig. 1 A general architecture of a typical secure cloud storage with deduplication

management as the cornerstone of the typical secure CSP with deduplication and adds N-anonymity service to this system without any severe change in the overall architecture. This architecture has a set of n clients, each with its own data. There is also a cloud storage provider, responsible for all storage management problems and issues. In addition to the CSP, there is a trusted third authority responsible for issuing the keys or certificates. This component is required to handle security issues, precisely the key management problems.

Basically, a secure deduplication process requires five algorithm sets for parameter generation, key generation, encryption, decryption, and tag generation [21]. According to the general framework of MLE, the mentioned algorithm sets are represented with a five-tuple $(\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{T})$, respectively. Hereupon, the general scenario of data storage is as the following. First of all, each system component that is responsible for generating system parameters has to do its corresponding tasks, defined in the algorithm set $\mathcal{P}$. Then, the algorithm set $\mathcal{K}$ is used cooperatively between clients and the trusted third party to generate the required keys. It is worth noting that both data and clients have their own keys in which the data key is generated from the data itself, while in almost all algorithms, each client has a pair of private–public keys used for authentication reasons. In the basic idea of convergent encryption, the hash value of the data is used as the data encryption key. As we mentioned earlier, the data hash value may be known by others [19], and hence, it is not a suitable choice to be used as the encryption key.

The client then encrypts its data using the data key and algorithm set $\mathcal{E}$. The client further applies the algorithm set $\mathcal{T}$ on the encrypted data to generate a data tag. Obviously, the tag has to be consistent for the same data. It means that the tag of the same data generated by the different users has to be the same as the previous one. The data tag is sent to the CSP, and the CSP, in its turn, responds to the client whether the data are previously stored or not. In both cases, the client encrypts the data's tag by its public key to generate PoW. PoW is stored along with the data in CSP to guarantee the access of the data owner. Whenever the client needs to access its data, the client sends the PoW to the CSP, and CSP then checks the PoW and returns the data to the client. The retrieved data are encrypted; thus, the client uses the data key and algorithm set $\mathcal{D}$ to decrypt the data and access the plain one. For more information about the MLE framework details, we refer the readers to the original MLE work [21].

Since the stored data are encrypted in this system, the CSP cannot access the data. It is worth noting that whenever we talk about the untrustworthy of the CSP, it does not always mean that we do not trust the CSP itself. However, the data are stored somewhere, maybe accessible to adversaries or untrusted agents. On the other hand, the CSP is always under different attacks, and such attacks are unavoidable [18]. The following assumptions are also considered to define secure cloud storage with deduplication.

Assumption 1 The system includes n clients, whereas the value of n is constantly changing. Hence, the system has to be dynamically scalable.

Assumption 2 Each client is recognized with a unique ID. This ID could be an IP address, phone number, or any other unique ID.

Assumption 3 We have a CSP that stores encrypted data from a universe U. The defined system could be a block-level or file-level CSP. It is worth noting that the block-level data storage is considered to have better performance than file-level storage in the context of deduplication [38–40]. However, our proposed system is able to handle both block-level and file-level deduplication. Accordingly, to propose the system, we use the term data file through the paper, while it could be substituted by data block.

Assumption 4 There is a reliable symmetric cryptosystem, represented by *Encrypt(Data,Key)*. This system symmetrically encrypts the "Data" with the "Key", whereas the data encrypted remain confidential unless their key is compromised. This encryption system is not the subject of this paper and can be any well-known symmetric encryption system, such as AES [41] and RC6 [42].

Assumption 5 The CSP is considered a black-box storage service with a vast storage capacity. From our point of view, we do not see the internal CSP tasks such as job scheduling, resource management between different cloud entities, and backup operations. Hence, from the vantage point of this work, there is a vast integrated, reliable, and available storage system. Accordingly, just one copy of each data is required to be stored in such a black-box storage system, i.e., the backup tasks, including the storage of several copies of the file for higher reliability, are not subject to this work.

Assumption 6 $H : U \rightarrow \{0,1\}^k$ is a publicly known, deterministic, and cryptographic hash function that converts each file of data into a fixed size, i.e., k bit, hash value. This hash function is considered a collision-free and irreversible hash function. The mentioned hash function is deterministic and thus returns the same hash value for the same data.

Assumption 7 We assume that the currently stored encrypted files on the server are defined by the set $M \subseteq U$ with $|M| = m$.

In this paper, we propose a secure deduplication system with ID-based key management, an asymmetric encryption-based solution that could be considered an asymmetric extension of MLE. On the other hand, in a typical secure deduplication system, the CSP is able to recognize the clients that store the same data. Such a fact is an obvious violation of client privacy. As we mentioned earlier, a tax statement or a salary form could be considered data that could be confidential. In such cases, the CSP can recognize, for instance, that the salary of this client is as equal to that one, which is some identity privacy violation. In this paper, we propose an N-anonymity solution for secure deduplication capable of anonymously doing the cross-user deduplication, while the CSP cannot recognize which clients have the same data.

4 ID-based secure deduplication system

This section proposes a new secure deduplication system using an ID-based key management algorithm. ID-based key management is an asymmetric algorithm proposed for the first time by Shamir [22]. The main idea of ID-based key management is to extract the public key of each network client from its unique ID using a deterministic algorithm. Hence, there will be no need to certify public keys. Since, in the context of secure data deduplication, we need to encrypt the same data with the same key, this idea seems very interesting. However, Shamir just proposed a signature-based algorithm and left the problem of ID-based encryption as an open problem for other researchers. Hereupon, Boneh and Franklin [43] proposed the first functional ID-based encryption system using highly complicated pairing operations. Such a system requires a central trusted authority called a private key generator (PKG), responsible for initializing secure network parameters and generating private keys. It is worthy to note that, although PKG is considered a central authority, it could be implemented in distributed manner [44]. However, we propose our algorithm using a central PKG for simplicity reasons.

In ID-based key management, public keys are extracted easily from unique IDs. We use the basic idea of ID-based encryption, in which the complex pairing operations used in [43] are replaced with much more efficient elliptic curve point multiplications. Hence, the proposed algorithm is ID-based key management built on an elliptic curve encryption system. This algorithm is designed to be suitable for secure deduplication in CSP and compatible with the N-anonymity-based algorithms proposed in Sect. 5.

Nevertheless, the ID-based key management algorithm to be used as the cornerstone of a secure deduplication system has to generate unique private-public key pair for each file and client. The public key could be extracted from the user or data unique ID. Hence, even though we know from Assumption 2 that all clients have their unique IDs, we need further to define IDs for the data such that the defined IDs satisfy two main conditions. First, the data ID should be unique for the same data, even when another client generates the ID. Second, data IDs have to be non-overlapping with client IDs, i.e., all IDs have to be unique. Hence, we propose an algorithm set to generate a unique ID for both data and clients that satisfies the mentioned conditions. This algorithm set is referred to as the ID generation algorithm set and is represented by $\mathcal{I}$.On the other hand, as we mentioned earlier, MLE is considered the most typical framework for secure deduplication. Henceforward, we propose our secure deduplication system as an asymmetric extension of MLE. Accordingly, we define the new system with the same five-tuple algorithm sets of MLE, i.e., $(\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{T})$. Each of the mentioned algorithm sets, i.e., the five-tuple algorithm sets along with the ID generation, is described in detail in the rest of this section. Figure 2 represents the schematic view of the algorithm sets, and Table 1 collects the notations used in this paper along with their brief descriptions.

ID generation algorithm set ($\mathcal{I}$): We suggest using the hash value of the data as the symmetric key to symmetrically encrypt the data using the symmetric

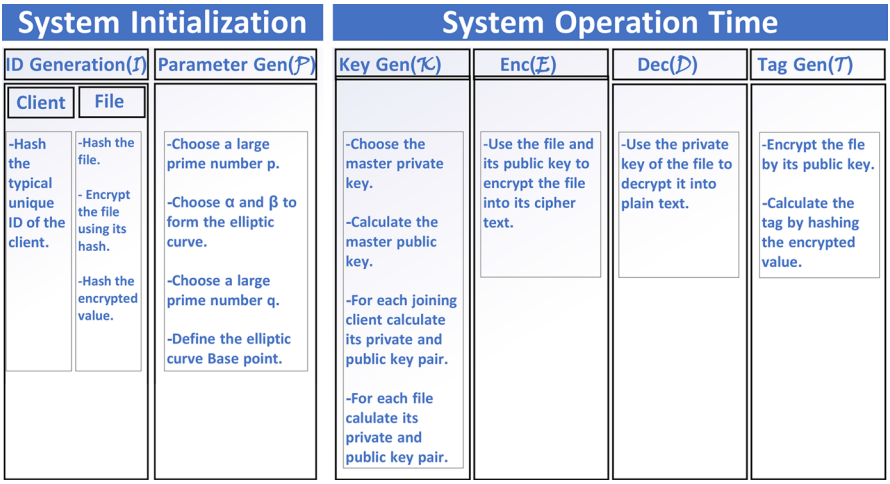

Fig. 2 A schematic view of the ID-based secure deduplication system

cryptosystem mentioned in Assumption 4. Then, the data ID will be the hash value of the symmetrically encrypted data. For the IDs to be unique for both data and clients, we suggest using the hash value of the client ID instead of the client ID itself. Hereupon, to distinguish between the original ID of client i from its new ID defined to be unique for both data and clients, we use the notations ID_{c_i} and $\mathbb{ID}_{c_i}$, respectively. Since the hash function is collision-free, and both data $\mathbb{ID}$ and client $\mathbb{ID}$ are the output of the hash function, all $\mathbb{ID}$s are unique. The formulas of calculating new unique $\mathbb{ID}$ for data and clients are represented in Eqs. (1) and (2), respectively. The new data $\mathbb{ID}$ is unknown to others and is considered a secret value. We will use this value to exchange the private key of the data with PKG securely. It could also be used as the PoW to guarantee upcoming data accesses. The detail of key exchange and data access is proposed in the rest of this section.

$$\mathbb{ID}_{f_i} = H(Encrypt(f_i, H(f_i))) \tag{1}$$

$$\mathbb{ID}_{c_i} = H(\mathrm{ID}_{c_i}) \tag{2}$$

Parameter generation algorithm set ($\mathcal{P}$): As we mentioned earlier, PKG is responsible for parameter generation algorithm, i.e., $\mathcal{P}$. The first parameter is a large prime number p, which is used to generate a non-singular elliptic curve over the field $\mathbb{F}_p$. Such an elliptic curve has the form of Eq. (3) and represented by $E(\mathbb{F}_p)$. Parameters α and β are two constants, chosen form $\mathbb{F}_p$. The elliptic curve base point $\mathbb{B}$ is also defined based on a large prime number q such that $q|\#E(\mathbb{F}_p)$. This algorithm also needs a collision-free hash function, just like the function used for deduplication tasks, with one extra condition connecting it with parameter q. The condition is represented in Eq. (4), while it states that the hash function's output has to be a number smaller than q. Since parameter q is a large prime number, the hash function output could be considered to be with a bit length less than q. Accordingly, any well-known

Table 1 Table of notations

N	The degree of anonymity
n	The number of nodes
$H(.)$	A collision-free hash function
$\mathcal{I}$	The ID generation algorithm
$\mathcal{P}$	The parameter generation algorithm
$\mathcal{K}$	The key generation algorithm
$\mathcal{E}$	The encryption algorithm
$\mathcal{D}$	The decryption algorithm
$\mathcal{T}$	The tag generation algorithm
ID_{c_i}	The typical unique ID of client i
$\mathbb{ID}_{c_i}$	The new unique ID for the client i
$\mathbb{ID}_{f_i}$	The new unique ID for the data file i
p and q	Two large prime numbers.
α and β	Two constants chosen from the field $\mathbb{F}_p$
$\mathbb{B}$	The elliptic curve base point
(x_m, y_m)	The master private–public key pair
(x_{c_i}, y_{c_i})	The private–public key pair of client i
(x_{f_i}, y_{f_i})	The private–public key pair of data file i
f_i	The data file i
$F_i : (\mathbb{F}_1, \mathbb{F}_2)$	The encrypted data file i
T_{f_i}	The tag of data file i
U	The universe of files
M	The set of previously stored files on CSP
m	The number of files in M
k	The bit length of hash value generated by H
$C(F_i)$	The first client stored the encrypted file F_i
H_j	The set of CSP candidates for the tag of F_j
S_j	The set of tag prefixes as the CSP candidates of F_j
$\mathcal{Q}$	The set of possible queries expressed by query type
σ	A random bit string, to be checked whether it is a tag of previously stored file or not

collision-free hash function could be used for both key generation and deduplication tasks.

$$E(\mathbb{F}_p): \quad f^2(x) = x^3 + \alpha x + \beta \pmod{p} \tag{3}$$

$$H: \{0,1\}^* \rightarrow \mathbb{Z}_q \tag{4}$$

Key generation algorithm set ($\mathcal{K}$): The key generation algorithm requires two separate phases. In the first phase, master private–public key pair (x_m, y_m) is generated. PKG uses this key pair to generate all other keys. The second phase is to generate private–public key pairs of data files and clients. Client keys are required to guarantee secure communication between the clients and the PKG, while data keys are

used for data encryption, i.e., to ensure the confidentiality of stored data. The master private key, i.e., x_m, is chosen randomly from $\mathbb{Z}_p^*$ by PKG, while the master public key is calculated using Eq. (5).

$$y_m = x_m.\mathbb{B} \tag{5}$$

Each client must get its private key from PKG once it joins the cloud system. The PKG generates the private key according to Eq. (6). The corresponding public key is also extractable form the client $\mathbb{ID}$ using Eq. (7). The private and public keys of data files are generated using similar formulas presented in Eqs. (8) and (9), respectively. After receiving the private–public key pair, each client can easily check its validity using Eq. (10).

$$x_{c_i} = x_m.H(\mathbb{ID}_{c_i}) \pmod{q} \tag{6}$$

$$y_{c_i} = y_m.H(\mathbb{ID}_{c_i}) \pmod{q} \tag{7}$$

$$x_{f_i} = x_m.H(\mathbb{ID}_{f_i}) \pmod{q} \tag{8}$$

$$y_{f_i} = y_m.H(\mathbb{ID}_{f_i}) \pmod{q} \tag{9}$$

$$\begin{aligned} y_{c_i} &= y_m.H(\mathbb{ID}_{c_i}) \\ &= x_m.\mathbb{B}.H(\mathbb{ID}_{c_i}) \\ &= x_{c_i}.\mathbb{B} \end{aligned} \tag{10}$$

Encryption and decryption algorithm sets ($\mathcal{E}$ **and** $\mathcal{D}$): Encryption and decryption algorithm sets for the plain data file f_i with the key pair (x_{f_i}, y_{f_i}) are defined in the following. First, a random number $\tau \in \mathbb{Z}_q^*$ is chosen. The cipher text is then represented by F_i and consists of a pair $(\mathbb{F}_1, \mathbb{F}_2)$ in which $\mathbb{F}_1$ is a mapping to a point on the elliptic curve. Encryption and decryption processes are, respectively, shown by Eqs. (11) and (12). Accordingly, it is easy to observe that the decryption process of Eq. (12) returns the exact lain data f_i. The validation of such an observation is shown in Eq. (13). It is worthy to note that the encryption process proposed here is asymmetric encryption for the data before storing it on the CSP. It obviously differs from the symmetric encryption used for $\mathbb{ID}$ generation algorithm which could be any symmetric encryption algorithm.

$$F_i = \mathcal{E}(f_i, y_{f_i}) = \begin{cases} \mathbb{F}_1 = \tau.\mathbb{B} \\ \mathbb{F}_2 = f_i + \tau.y_{f_i} \end{cases} \tag{11}$$

$$\mathcal{D}(F_i) = \mathcal{D}(\mathbb{F}_1, \mathbb{F}_2) = \mathbb{F}_2 - x_{f_i}.\mathbb{F}_1 \tag{12}$$

$$\begin{aligned}&\mathbb{F}_2 - x_{f_i}.\mathbb{F}_1\\&= f_i + \tau.y_{f_i} - x_{f_i}.\tau.\mathbb{B}\\&= f_i + \tau.x_{f_i}.\mathbb{B} - x_{f_i}.\tau.\mathbb{B}\\&= f_i\end{aligned} \tag{13}$$

It is further worthy to note that each client to get the private key of its own data file sends its request to PKG as shown in Eq. (14). The request consists of the asymmetrically encrypted data ID, i.e., $\mathbb{ID}_{f_i}$, with the master public key. PKG, in its turn, decrypts the request by the master private key and generates the corresponding private key of the data using Eq. (8). The private key of the data is then encrypted with the client's public key using Eq. (11) and sent back to the client. This algorithm will have a secure channel between PKG and clients. Hence, data keys are exchanged securely between them. Obviously, data public key is also extractable from the data ID, i.e., $\mathbb{ID}_{f_i}$, using Eq. (9).

$$DataPrvKeyReq = \mathcal{E}(\mathbb{ID}_{f_i}, y_m). \tag{14}$$

Tag generation algorithm set ($\mathcal{T}$): The last algorithm set, i.e., $\mathcal{T}$, returns the tag of data which is used to check whether the data are previously stored or not. The data tag is the hash value of the asymmetrically encrypted data as shown in Eq. (15). It is evident that such a tag is consistent, i.e., the tag generated for the same data by different clients has the same value. As we mentioned earlier, the public key of data files f_i is extractable from its ID, i.e., $\mathbb{ID}_{f_i}$; hence, the data could be encrypted easily, and the output is the same for all data owners. Recall that the data ID is the hash value of the symmetrically encrypted data, using the hash value of the plain data as the symmetric key. Hence, data ID and data tag differ from each other.

$$T_{f_i} = H(\mathcal{E}(f_i, y_i)). \tag{15}$$

It is worthy to note that the proposed tag generation mechanism cleverly prevents the duplicate fake attack. We formally mention the definition of this attack here, as it is defined in [21]. The proof that the proposed system is preventing such an attack is then stated in Sect. (6.1).

Definition 1 Duplicate faking attack: Consider an adversary and an honest client with the same file. The adversary generates the file's honest tag and sends it along with a fake file into the CSP. When the honest client sends the tag of his file to check whether the file is previously stored or not, the CSP replies positively. Hence, the file is discarded. When the honest client retrieves his file, the received file is not true, while the right file is missed. This attack is referred to as a duplicate faking attack.

5 *N*-anonymity-based algorithm

Recall that the scenario of data deduplication, using the ID-based key management proposed in Sect. (4), is as follows. A publicly known hash function is used to hash encrypted files, i.e., generate file tags. To store an encrypted file F_i on the CSP, the client must check whether the file with the same tag is previously stored or not using its hash value $T_i = H(F_i)$. For accessing any encrypted file $F_i \in M$ on the CSP, it is also enough to have $T_i = H(F_i)$. It is worth noting that, in this scheme, the accessed data are encrypted. Hence, getting access to the data tag does not violate the data confidentiality.

For each stored encrypted file $F_i \in M$, the first client who stored the file on the CSP is known by the CSP, shown by $C(F_i)$. A client j wants to store an encrypted file F_j with the tag $T_j = H(F_j)$. Our idea for N-anonymous secure data deduplication is to check whether $F_j \in M$ or not by asking some queries among a set of predefined query types. The queries are asked from the CSP about the tags of the files stored in M. If $F_j \in M$, client j does not want the CSP to know that his file is the same as the file stored by $C(F_i)$. We further assume that the CSP always tries to guess $H(F_j)$. Thus, we have to do our best to increase the number of candidates that can potentially have the right value for $H(F_j)$. This concept is formally defined by N-anonymity.

Definition 2 N-anonymity: Assume that $H(F_j) \in M$ and H_j is the set of CSP candidates which potentially could be $H(F_j)$. We say that N-anonymity is satisfied when $|H_j| = N$.

Assuming that the hash function $H(\cdot)$ is entirely uniform, satisfying N-anonymity is equivalent to finding the hash values of N available files on the CSP, including $T_j = H(F_j)$. These values are used to satisfy anonymity. We refer to this problem as the N-anonymity injection problem.

Definition 3 N-anonymity injection problem (N-AI): Given a set $\mathcal{Q}$ of possible queries, it is expressed by a predefined query type. The problem is to propose a random process that can find a set H_j including N hash values, such that $H_j \subseteq M$ and $H(F_j) \in H_j$ using the queries of $\mathcal{Q}$.

In the following parts, we propose two different query types which could be asked from the CSP. The first one is the naive solution with inefficient complexity. We name it as *complete-equal query*. The second solution is the improved version with much lower complexity regarding the number of queried bits. We name the second solution as *prefix-equal query*. Each query type is proposed in a separate part, while we study the N-anonymity injection problem for each one. In our study, we further consider the optimization version of this problem, whereas the goal is to minimize the number of bits used to query the CSP.

5.1 Complete-equal queries

The complete-equal query is the naive solution for finding a set of N tags in which the CSP previously stores the corresponding files. The definition of such query type is given in the following.

Definition 4 Complete-equal queries: Complete-equal queries are queries in the form of *Complete_Exist*(x), where $x \in \{0,1\}^k$ is a k-bit string for which when asked from a CSP, it should return "True" if there exists an encrypted file F_i with $T_i = x$ or "False" otherwise.

For the complete-equal queries, the best a client can do is to generate random bit strings and query them from the CSP until it can find N tags corresponding to the stored files on the CSP. Among these random bit strings, the query *Complete_Exist*(T_j) is run in random order. If this query returns false and thus the client notices that F_j is not previously stored on the CSP, it quits all the processes and sends F_j to the CSP for storage.

Algorithm (1) states that we query the CSP with random pseudo-hash values until we get N True responses. We inject the tag of our file with a True response between the pseudo-hash values. Accordingly, CSP could not recognize which one of existed files in which the corresponding response was True is the client file. Increasing the hash value bit length increases the algorithm's time complexity. Theorem (1) mathematically analyzes this complexity.

Algorithm 1: Solving N-AI Problem with complete-equal queries

```
H_j = ∅, {The set of found hash values}
Set t a random number from {1, 2, ..., N} {The iteration in which T_j will be queried}
while |H_j| ≤ N do
    flag = False, {The flag becomes True when we find a tag of previously stored file}
    if |H_j| = t − 1 then
        if Complete_Exist(T_j) = True then
            F_j exists on the CSP
        else
            Store F_j on the CSP
            Break
        end if
        H_j = H_j ∪ {T_j}
        flag = True
    end if
    while flag = False do
        Set σ a random k-bit string which is not currently in H_j, {σ is a random bit string for
        checking its existence on the CSP}
        flag = Complete_Exist(σ)
        if flag = True then
            H_j = H_j ∪ {σ}
        end if
    end while
end while
```

Theorem 1 *Algorithm* (1) *sends* $O(2^k(\log m - \log N))$ *complete-equal queries to the CSP.*

Proof The time complexity of Algorithm (1) can be simply computed by the sum of N random variables $X_1, X_2, \ldots, X_N$, for which X_i represents the number of iterations in the ith iteration of the while loop, i.e., lines 3–22 of Algorithm (1). It is trivial that the while loop is run N times, owing to each run adds one member to H_j until we finally find N members of M, i.e., $|H_j| = N$. Clearly, $X_t = O(1)$ and other X_is count the number of times a randomly chosen σ fails to be a member of $M \setminus H_j$ in the while loop of lines 15–21, i.e., to be a tag of a stored file on the CSP. Hence, for $i \in \{1, 2, \ldots, N\} \setminus \{t\}$, we have

$$E[X_i] = 1/Pr\{\sigma \in M \setminus H_j\} = \frac{2^k}{m-i}. \tag{16}$$

Thus, the runtime of Algorithm (1) is

$$\sum_{i=1}^{N} E[X_i] = \sum_{i=1}^{N} \frac{2^k}{m-i} = 2^k\left(\sum_{i=1}^{m} \frac{1}{i}\right) - 2^k\left(\sum_{i=1}^{m-N+1} \frac{1}{i}\right) \tag{17}$$

From the equation $\sum_{i=1}^{n} \frac{1}{i} = O(\log n) + O(1)$, we can deduce that the time taken by the algorithm is equal to

$$\sum_{i=1}^{N} E[X_i] = O(\log m) - O(\log m - N) + O(1). \tag{18}$$

□

Theorem (1) shows that the run time of Algorithm (1) is exponentially increasing. Hence, it cannot be used for large k values, and this algorithm is practically inefficient. Some results representing its complexity in the real world are shown in Sect. (6.2). The results show that even for a small number of users, to achieve N-anonymity, each user needs to perform several billions of queries. Accordingly, one can say that achieving N-anonymity by complete-equal queries is not feasible at all. In the next part of this section, we propose a new query type that can be used for efficient N-anonymity algorithm design.

5.2 Prefix-equal queries

In this part, we show that with a clever improvement in the definition of the query type, we can achieve N-anonymity with much lower complexity. First, we define the new query type, which we name *prefix-equal query*, and then show that the number of such queries required for achieving N-anonymity is with polynomial complexity.

Definition 5 Prefix-equal queries: Prefix-equal queries are in the form of *Prefix_Exist*(x), where $x \in \bigcup_{i=1}^{k} \{0,1\}^i$ is a bit string with length i. Accordingly,

when CSP is asked for x, it has to return "True" if there exists an encrypted file F_i for which $T_i = H(F_i)$ starts with the prefix x and, "False" otherwise.

In the following observation, we show that for a query $Prefix\_Exist(x) = True$, we can find the hash value of a stored encrypted file with a polynomial number of queries in terms of k and $|x|$, where $|x|$ represents the bit length of string x.

Observation 1 For a query $Prefix\_Exist(x) = True$, we can find the hash value of a stored encrypted file with $(k - |x|)$ queries. When $Prefix\_Exist(x) = True$, we can conclude that at least one of the queries $Prefix\_Exist(x||0)$ or $Prefix\_Exist(x||1)$ is True, when $x||b$ defines the concatenation of the string x and the character b. Hence, we can simply query $Prefix\_Exist(x||0)$, while in positive answer we replace x with $x||0$. Otherwise, $Prefix\_Exist(x||1)$ definitely is True, and therefore, x is replaced by $x||1$. Such a process is continued until we reach a k-bit tag of the stored file.

Algorithm (2) is proposed to support N-anonymity with prefix-equal queries. This algorithm needs three phases to be able to find N tags corresponding to N stored files as the anonymity candidates for the encrypted file F_j. In the first phase, we find all $(\log_2 N)$-bit strings that have the same prefix of at least one stored file tag. The number of such strings is at most equal to N. The found strings are inserted into the set S_j. Obviously, to find such strings, we have to query $Prefix\_Exist(\sigma)$ for each $\log_2 N$ -bit string σ. For each σ with True response, we add the σ to S_j.

Algorithm 2: Solving N-AI Problem with prefix-equal queries

{Phase 1: Find a set S_j, containing all tag prefixes of length $\log_2 N$ bit which the file is already stored on the CSP.}
$S_j = \emptyset$, {The set of found prefixes}
for all bit string σ with length $\log_2 N$ **do**
  if Prefix_Exist(σ) **then**
    $S_j = S_j \cup \{\sigma\}$
  else
    if σ is a prefix of T_j **then**
      Store F_j on the CSP and exit
    end if
  end if
end for
{Phase 2: Complete filling the set S_j to contain N prefixes.}
while $|S_j| \leq N$ **do**
  Choose $\sigma \in S_j$ with length less than k bit
  Set b equal to 0 or 1 with equal probabilities
  if Prefix_Exist($\sigma||b$) **then**
    if Prefix_Exist($\sigma||\bar{b}$) **then**
      $S_j = S_j \setminus \sigma \cup \{\sigma||b, \sigma||\bar{b}\}$
    else
      if $\sigma||\bar{b}$ is a prefix of T_j **then**
        Store F_j on the CSP and exit
      end if
      $S_j = S_j \setminus \sigma \cup \{\sigma||b\}$
    end if
  else
    if $\sigma||b$ is a prefix of T_j **then**
      Store F_j on the CSP and exit
    end if
    $S_j = S_j \setminus \sigma \cup \{\sigma||\bar{b}\}$
  end if
end while
{Phase 3: Now S_j contains N tag prefixes including a prefix of T_j. The all members of S_j are expanded to be with the length k, and further added to the set H_j as the final candidates.}
$H_j = \emptyset$, {The set of final candidates}
for all $\sigma \in S_j$ **do**
  while length of σ is less than k **do**
    if σ is a prefix of T_j **then**
      Set b equal to $(|\sigma|+1)^{\text{th}}$ bit of T_j
      if Prefix_Exist($\sigma||b$) **then**
        $\sigma = \sigma||b$
      else
        Store F_j on the CSP and exit
      end if
    else
      Set b equal to 0 or 1 with equal probabilities
      if Prefix_Exist($\sigma||b$) **then**
        $\sigma = \sigma||b$
      else
        $\sigma = \sigma||\bar{b}$
      end if
    end if
  end while
  $H_j = H_j \cup \{\sigma\}$
end for

The second phase works after the first phase if and only if $|S_j| < N$. This phase is responsible for finding $N \setminus |S_j|$ other tag prefixes in which the corresponding files are already stored on the CSP. Accordingly, this phase is finished whenever

$|S_j|$ becomes equal to N. In each iteration of this phase, we choose a random prefix σ from S_j and consider both of its one-bit expansions, i.e., $\sigma||0$ and $\sigma||1$. It is worthy to note that the selection order between these two choices is done randomly. If both prefixes exist at the CSP, we inject both of them into S_j and remove the old σ. Hence, $|S_j|$ is increased by one member. Otherwise, we inject the prefix of the already existing tag into the S_j and remove the previous σ. In Theorem (2), we prove that this process takes at most polynomially bounded steps in terms of k and N. Among these steps, when we are noticed that some prefix σ does not exist on the CSP, we have to check whether it is a prefix of our file tag, i.e., T_j, or not. In the positive case, the process is terminated, and the file F_j has to be stored on the CSP.

If the second phase is finished with $|S_j| = N$, we reach the third phase. At this phase, $|S_j|$ has N prefixes which exist on the CSP, and at least one of them is a prefix of T_j. In this phase, we expand all the prefixes in S_j according to the process described in Observation (1). We do that in a random order, while for each member of S_j, we find the tag of one stored file. Meanwhile, in some of the random steps, we have to check the prefixes of T_j whether stored on CSP or not.

Theorem 2 *Algorithm* (2) *sends* $O(N(k - \log_2 N))$ *prefix-equal queries to the CSP.*

Proof At the first phase of Algorithm (2), the client sends N queries to the CSP, each one with a $\log_2 N$-bit string. To analyze the second phase, consider the potential function shown in Equation (19). In each iteration of phase 2 main while loop, if the process does not exit, the value of $\Phi(S_j)$ is increased at least by one. Since $\Phi(S_j)$ is equal to $N \log_2(N)$ at the beginning of this phase and is always less than Nk, this phase lasts at most $Nk - N \log_2 N$ iterations. In each of its iterations, two queries, at most, are sent to the CSP. Thus, the number of queries sent to the CSP is less than or equal $2(Nk - N \log_2 N) \in O(N(k - \log_2 N)$. With a similar justification, we can conclude that the number of queries at the third phase is $O(N(k - \log_2 N)$. Hence, the theorem is proved.

$$\Phi(S_j) = \sum_{\sigma \in S_j} |\sigma|. \tag{19}$$

□

6 Security analysis and performance evaluation

It is worth mentioning that the proposed algorithm's implementation could be done at no additional hardware cost compared with the traditional secure storage systems. The trusted third party will be responsible for PKG tasks where the private key shares are generated. In this section, the security of the proposed secure deduplication system is analyzed and presented in Part (6.1). The performance of the N-anonymity algorithm is further evaluated through simulation, using real dataset. The simulation details, along with the evaluation results, are presented in Part (6.2).

6.1 Security analysis

The proposed secure cloud storage system simultaneously provides deduplication and *N*-anonymity. In this part, we analyze the security of the proposed system and show that even if the CSP becomes compromised by adversaries, the confidentiality of the stored data is still preserved. Basically, it is because of ECC-based encryption with rigid key management. We further show that the proposed system cleverly prevents the duplicate faking attack. In the rest of this part, we formally analyze the security of the proposed system.

Lemma 1 *Data* $\mathbb{ID}$, *which is a secret value*, *is known only by the data owners*, *and no one else can access the data* $\mathbb{ID}$. *Equivalently*, *we can say that in the proposed secure deduplication system*, *the PoW is unforgeable*.

Proof Recall that data $\mathbb{ID}$ is the output of the hash function, in which the input is the symmetrically encrypted plain data. The plain data are encrypted with its hash value as a key, i.e., Eq. (1). Since the hash function is collision-free, its output is unique. Furthermore, no one owns the input of the hash function except the data owners. Recall that the hash function is irreversible; hence, only the data owner can generate the data $\mathbb{ID}$. It is notable that, however, the hash value of the plain data is considered a plain parameter, the input of the hash function is the plain data encrypted with its hash value, and it is calculable only by the data owners. On the other hand, data $\mathbb{ID}$ are exchanged only between the data owner and PKG. PKG is a trusted authority, and the exchange process is done securely by encrypting the data $\mathbb{ID}$ by the master public key. Since just the PKG itself has access to the master private key, only the PKG can get access to the $\mathbb{ID}$. □

Lemma 2 *The private key of data is known only by the PKG*, *which is a trusted authority*, *and the data owner*.

Proof We know that the PKG is responsible for the private key generation process. According to Eq. (8), for generating the private key of the data, the PKG requires the data $\mathbb{ID}$ and the master private key. The master private key is known only by the PKG, while according to Lemma (1), the data $\mathbb{ID}$ are known only by the data owners, and they are sent to PKG through a secure channel. The data's private key is also sent to the data owner in an encrypted format. The client's public key encrypts the data's private key; hence, only the client can decrypt it using its private key. □

Lemma 3 *The stored data are confidential and remain confidential even if the adversary gets access to the encrypted data*.

Proof The data are encrypted asymmetrically with its public key using ECC cryptosystem. Hence, the confidentiality of the data is guaranteed as long as the elliptic curve discrete logarithm problem remains intractable in polynomial time or the adversary gets access to the private key of the data along with the encrypted data

itself. Using N-anonymity algorithms proposed in Sect. (5), each client or adversary could easily get access to the tag of many stored files. Having the tag of each file, the client or adversary can retrieve the file from the CSP. However, the adversary has to compromise the PKG or the data owner to get access to the data's private key. It is also worth noting that, although each client or adversary can access the stored encrypted files, the access process is random such that the adversary cannot distinguish the file of which data are accessed. □

By proving that the encrypted data are secure and just the data owners have access to the data's private key, we can conclude that even if the CSP is compromised by adversaries, the data confidentiality remains preserved. Now we show that the proposed secure deduplication system prevents duplicate faking attacks.

Lemma 4 *The proposed secure deduplication system is resistant to duplicate faking attacks*.

Proof The tag of the stored encrypted file is calculated by Eq. (15). Since the tag is the hash value of the encrypted file, the CSP itself calculates the tag of each stored file. On the other hand, we know from Assumption (6) that the hash function is collision-free; thus, there is no additional encrypted file with the same hash value. Hence, the adversary can not send a fake file whose hash value is equal to the honest file tag. □

6.2 Simulation detail and performance evaluation

To validate the results proved analytically in Sect. (5), a python-based simulation is done on a Microsoft Windows machine with 8 GB of main memory and a core $i3$, 3.4 GHz processor. In this simulation, a population of more than 500,000 YouTube video is considered [45]. To make the results more dependable, we used a well-known dataset provided by Simon Fraser University [46]. In this dataset, YouTube video links are stored, each with an 11-digit hash value. Since movies and videos are considered massive data which occupy a high storage ratio, implementing deduplication techniques on such files seems very effective. That is why, we designed such a scenario. In this scenario, each user intends to upload its file and has to start the N-anonymity algorithm using the hash value corresponding to the video file. According to the server responses, the user identity remains N-anonymous, or the file does not exist on the server, and the user needs to upload it.

The goal of this simulation is to show the accuracy of the complexity analysis for proposed N-anonymity algorithms of Theorems (1) and (2) for both complete-equal queries and prefix-equal queries, respectively. For the N-anonymity algorithm to be feasible, at least N files must be stored in the cloud. For this reason, in the initialization phase of the simulation, N files are randomly chosen and stored in the cloud storage. The rest of the simulation scenario is as the following. In each simulation scenario, n users, one by one, randomly choose one file from the data set and start their queries to CSP. According to CSP responses, the user may know that the file is

not uploaded previously and hence uploads the file. Elsewhere, the queries are continued until the N-anonymity is achieved.

As the result of the simulation, we calculate the number of queries as follows. After the initialization process of storing N files for the first user, we calculate this number. For the next users, we add the number of queries to the previous number and divide the result by the current number of users. Generally, we calculate the average number of queries until the nth user.

Here, two different scenarios are simulated to validate the result of Theorem (1), which proved for the case of complete queries where the second scenario is related to Algorithm (2) to validate the analytical proof of Theorem (2). In the first scenario, after selecting its file, each user runs its complete queries of k bit length. In our simulation, the value of k, which represents the bit length of the data tag, is set equal to 77, i.e., each digit of the hash value is represented with 7 bits. As we mentioned earlier, after some queries, each user will find that its file is not stored formerly, and hence, the file will be sent to the cloud server. Elsewhere, the user continues queries to guarantee the N-anonymity.

Figure 3a represents the number of complete queries for different number of users, i.e., n, and different N values, in logarithmic scale. As it is clear from the figure, after some data storage processes, which we call *warm-up process*, users find that their files are already stored on the cloud storage. The reason is the limitation in the number of files used in the simulation. Accordingly, they keep sending queries to gain the N-anonymity. Such a process leads to the exact amount of query numbers proved in Theorem (1). It is obvious from Fig. 3a that the warm-up process does not need many users to store their files. The obvious fast convergence toward the proved query number prohibits this figure from representing the slope of convergence speed. To better represent the behavior of the warm-up process, Fig. 3b represents the same results for the first 1000 users. Hereupon, the warm-up process is clearly ended after about 200 users, and then, the N-anonymity is achieved.

The second scenario is done according to Algorithm (2) to validate the upper bound of queries which is analytically proved in Theorem (2). Figure 4a represents

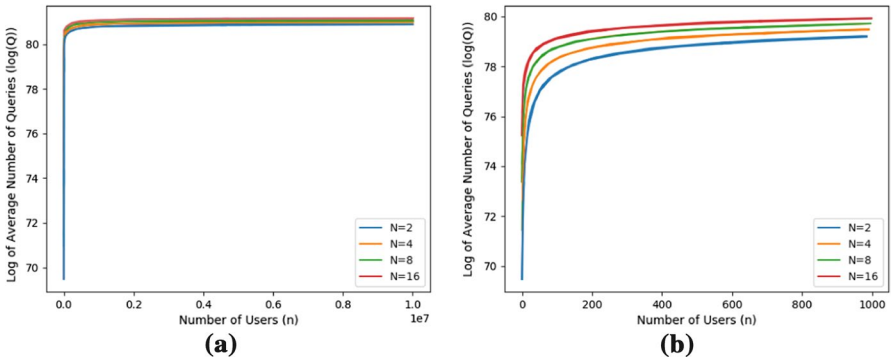

Fig. 3 Complete-equal query: **a** Results for the range of users between one until 10^7. **b** Range of users between one until 10^3

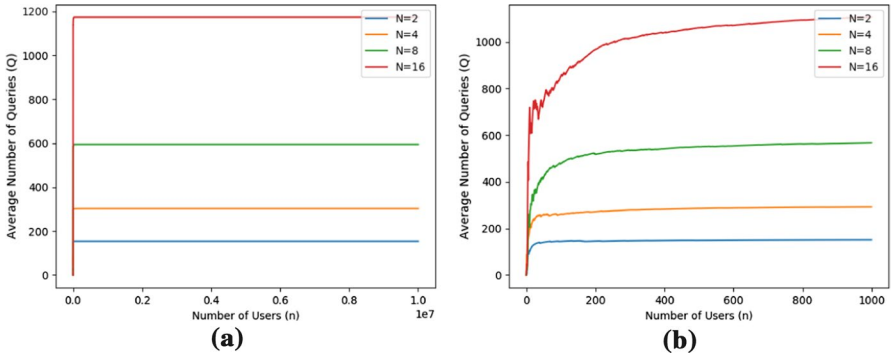

Fig. 4 Prefix-equal query: **a** Results for the range of users between one until 10^7. **b** Range of users between one until 10^3

the results for 10 million users. Since the number of prefix queries is much lower than that of complete queries, the results are not represented on a logarithmic scale. For all N values, the fast convergence is obvious. To show the behavior of the warm-up process, Fig. 4b represents the same results for just 1000 users. Like the complete prefix queries, the warm-up process required about 200 user. For more accurate numerical comparison, we provide Table 2 to represent the numerical results for Algorithms (1) and (2).

Finally, Fig. (5) represents a comparison between the upper bound value for the number of queries proved analytically in Theorem (2) and the simulation results for different N values. The obvious matching between the simulation results and the analytical bound after the warm-up process validates the analytical proofs.

7 Conclusion

Although cloud deduplication techniques could help CSPs efficiently manage their storage, traditional techniques are completely useless against encrypted data. Secure deduplication techniques like convergent encryption and its general

Table 2 Numerical comparison between the average number of required queries for Algorithms (1) and (2)

Average number of queries (Q)								
Algorithm	Complete-equal query				Prefix-equal query			
$n \| \| N$	2	4	8	16	2	4	8	16
100	$4548.0 * 10^{30}$	$7498.4 * 10^{30}$	$12362.7 * 10^{30}$	$20382.8 * 10^{30}$	149.0	225.1	490.2	880.2
200	$8287.0 * 10^{30}$	$12996.7 * 10^{30}$	$18433.1 * 10^{30}$	$22526.5 * 10^{30}$	149.7	270.8	520.4	985.5
300	$10121.8 * 10^{30}$	$17898.1 * 10^{30}$	$22981.6 * 10^{30}$	$27513.9 * 10^{30}$	149.8	279.9	532.9	1020.3
400	$11759.9 * 10^{30}$	$236.81.4 * 10^{30}$	$30407.6 * 10^{30}$	$41045.9 * 10^{30}$	149.9	284.7	541.6	1040.7

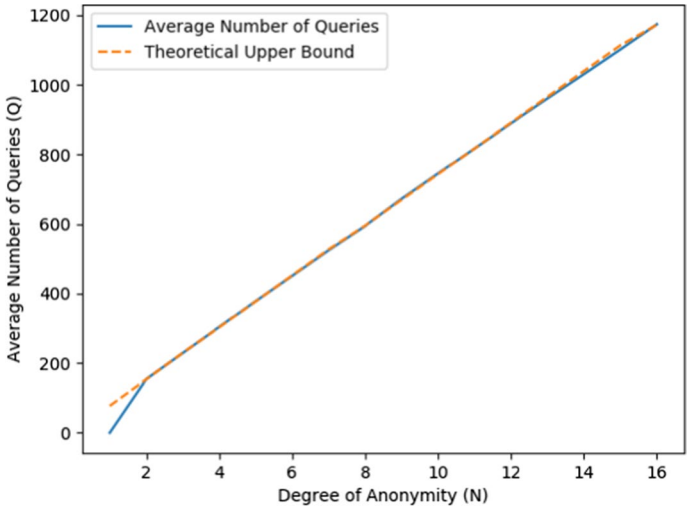

Fig. 5 A comparison between the upper bound proved analytically and simulation results

framework MLE are proposed to face this shortcoming. While convergent encryption, MLE, and all their extensions are built based on symmetric encryption, we proposed a new asymmetric-based secure deduplication system. The proposed system utilizes an ID-based cryptosystem with elliptic curve cryptography to resist duplicate faking attacks as well as compromising the CSP. Furthermore, while in most secure deduplication systems, the CSP can recognize clients with the same data, which is an explicit violation of privacy, we added anonymity to the secure deduplication systems. Some secure deduplication systems paid attention to the anonymity problem; however, they solved this problem at the cost of removing authentication. Our solution provided *N*-anonymity by keeping the authentication process on. The proposed solutions are analytically studied, while comprehensive simulations validate the results of complexity analysis on real-world data. The results of the simulations validated the analytical findings and proved the functionality of the proposed solution.

Author Contributions All authors have participated in conception and design, or analysis and interpretation of this paper.

Funding Not applicable.

Data availibility Not applicable.

Declarations

Conflict of interest The authors declare that they have no competing interests.

References

1. Jia G, Han G, Rodrigues J, Lloret J, Li W (2015) Coordinate memory deduplication and partition for improving performance in cloud computing. IEEE Trans Cloud Comput 7(2):357–368. https://doi.org/10.1109/TCC.2015.2511738
2. Fu Y, Xiao N, Jiang H, Hu G, Chen W (2017) Application-aware big data deduplication in cloud environment. IEEE Trans Cloud Comput 7(4):921–934. https://doi.org/10.1109/TCC.2017.2710043
3. Sengupta B, Dixit A, Ruj S (2020) Secure cloud storage with data dynamics using secure network coding techniques. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2020.3000342
4. Yang A, Xu J, Weng J, Zhou J, Wong DS (2021) Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage. IEEE Trans Cloud Comput 9(1):212–225. https://doi.org/10.1109/TCC.2018.2851256
5. Luo S, Zhang G, Wu C, Khan S, Li K (2015) Boafft: distributed deduplication for big data storage in the cloud. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2015.2511752
6. Paulo J, Pereira J (2016) Efficient deduplication in a distributed primary storage infrastructure. Trans Storage 12(4):1–35. https://doi.org/10.1145/2876509
7. Li Y-K, Xu M, Ng C-H, Lee PPC (2014) Efficient hybrid inline and out-of-line deduplication for backup storage. Trans Storage 11(1):2–1221. https://doi.org/10.1145/2641572
8. Dropbox A file-storage and sharing service. http://www.dropbox.com
9. Google drive A file-storage and sharing service. http://drive.google.com
10. Mozy A file-storage and sharing service. http://mozy.com/
11. Mao B, Jiang H, Wu S, Fu Y, Tian L (2014) Read-performance optimization for deduplication-based storage systems in the cloud. Trans Storage 10(2):6–22. https://doi.org/10.1145/2512348
12. Luo S, Zhang G, Wu C, Khan SU, Li K (2020) Boafft: distributed deduplication for big data storage in the cloud. IEEE Trans Cloud Comput 8(4):1199–1211. https://doi.org/10.1109/TCC.2015.2511752
13. Yu C-M, Gochhayat SP, Conti M, Lu C-S (2020) Privacy aware data deduplication for side channel in cloud storage. IEEE Trans Cloud Comput 8(2):597–609. https://doi.org/10.1109/TCC.2018.2794542
14. Huang K, Zhang X-S, Mu Y, Rezaeibagha F, Du X (2021) Bidirectional and malleable proof-of-ownership for large file in cloud storage. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2021.3054751
15. Opendedup Cloud storage gateway and filesystem. http://opendedup.org/
16. Meyer DT, Bolosky WJ (2011) A study of practical deduplication. ACM Trans Storage (ToS) 7(4):1–20
17. Yan Z, Ding W, Yu X, Zhu H, Deng RH (2016) Deduplication on encrypted big data in cloud. IEEE Trans Big Data 2(2):138–150. https://doi.org/10.1109/TBDATA.2016.2587659
18. Wu T, Dou W, Hu C, Chen J (2014) Service mining for trusted service composition in cross-cloud environment. IEEE Syst J 11(1):283–294. https://doi.org/10.1109/JSYST.2014.2361841
19. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A (2011) Proofs of ownership in remote storage systems. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, ACM, New York, pp 491-500
20. Douceur JR, Adya A, Bolosky WJ, Simon P, Theimer M (2002) Reclaiming space from duplicate files in a serverless distributed file system. In: Proceedings 22nd International Conference on Distributed Computing Systems, pp 617–624. https://doi.org/10.1109/ICDCS.2002.1022312
21. Bellare M, Keelveedhi S, Ristenpart T (2013) Message-locked encryption and secure deduplication. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 296-312 Springer, Heidelberg. https://doi.org/10.1007/978-3-642-38348-918
22. Shamir A (1985) Identity-based cryptosystems and signature schemes. In: Proceedings of CRYPTO 84 on Advances in Cryptology. Springer, New York, pp 47–53
23. Zhang Y, Mao Y, Xu M, Xu F, Zhong S (2021) Towards Thwarting Template Side-Channel Attacks in Secure Cloud Deduplications. IEEE Trans Depend Secure Comput 18(3):1008–1018. https://doi.org/10.1109/TDSC.2019.2911502
24. Sun Z, Shen J, Yong J (2011) DeDu: Building a deduplication storage system over cloud computing. In: Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, pp 348–355. https://doi.org/10.1109/CSCWD.2011.5960097

25. Marques L, Costa CJ (2011) Secure deduplication on mobile devices. In: Proceedings of the 2011 Workshop on Open Source and Design of Communication OSDOC '11, ACM, New York, pp 19–26. https://doi.org/10.1145/2016716.2016721
26. Anderson P, Zhang L (2010) Fast and secure laptop backups with encrypted de-duplication. In: Proceedings of the 24th International Conference on Large Installation System Administration LISA'10, USENIX Association, Berkeley, pp 1–8
27. Bellare M, Keelveedhi S, Ristenpart T (2013) Dupless: server-aided encryption for deduplicated storage. In: Proceedings of the 22Nd USENIX Conference on Security SEC'13, pp 179–194
28. Chen R, Mu Y, Yang G, Guo F (2015) Bl-mle: block-level message-locked encryption for secure large file deduplication. IEEE Trans Inf Forensics Secur 10(12):2643–2652. https://doi.org/10.1109/TIFS.2015.2470221
29. Atul A, William JB, Miguel C, Gerald C, Ronnie C, John RD, Jon H, Jacob RL, Marvin T, Roger PW (2002) Farsite: federated available and reliable storage for an incompletely trusted environment. SIGOPS Oper Syst Rev 36:1–4. https://doi.org/10.1145/844128.844130
30. Storer MW, Greenan K, Long DDE, Miller EL (2008) Secure data deduplication. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability StorageSS '08, ACM, New York, pp 1–10. https://doi.org/10.1145/1456469.1456471
31. Wilcox-O'Hearn Z, Warner B (2008) Tahoe: the least-authority filesystem. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability StorageSS '08, ACM, New York, pp 21–26. https://doi.org/10.1145/1456469.1456474
32. Rahumed A, Chen HCH, Tang Y, Lee PPC, Lui JCS (2011) A secure cloud backup system with assured deletion and version control. In: 2011 40th International Conference on Parallel Processing Workshops, pp 160–167. https://doi.org/10.1109/ICPPW.2011.17
33. Puzio P, Molva R, Onen M, Loureiro S (2013) Cloudedup: secure deduplication with encrypted data for cloud storage. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol 1, pp 363–370. https://doi.org/10.1109/CloudCom.2013.54
34. Wen Z, Luo J, Chen H, Meng J, Li X, Li J (2014) A verifiable data deduplication scheme in cloud computing. In: 2014 International Conference on Intelligent Networking and Collaborative Systems, pp 85–90. https://doi.org/10.1109/INCoS.2014.111
35. Li J, Chen X, Li M, Li J, Lee PPC, Lou W (2014) Secure deduplication with efficient and reliable convergent key management. IEEE Trans Parallel Distrib Syst 25(6):1615–1625. https://doi.org/10.1109/TPDS.2013.284
36. Li J, Li YK, Chen X, Lee PPC, Lou W (2015) A hybrid cloud approach for secure authorized deduplication. IEEE Trans Parallel Distrib Syst 26(5):1206–1216. https://doi.org/10.1109/TPDS.2014.2318320
37. Jung T, Li XY, Wan Z, Wan M (2015) Control cloud data access privilege and anonymity with fully anonymous attribute-based encryption. IEEE Trans Inf Forensics Secur 10(1):190–199. https://doi.org/10.1109/TIFS.2014.2368352
38. Harnik D, Pinkas B, Shulman-Peleg A (2010) Side channels in cloud services: deduplication in cloud storage. IEEE Secur Privacy 8(6):40–47
39. Wang B, Lou W, Hou YT (2015) Modeling the side-channel attacks in data deduplication with game theory. In: 2015 IEEE Conference on Communications and Network Security (CNS), pp 200–208. https://doi.org/10.1109/CNS.2015.7346829
40. Meyer DT, Bolosky WJ (2012) A study of practical deduplication. Trans Storage 7(4):14–20. https://doi.org/10.1145/2078861.2078864
41. Daemen J, Rijmen V (2002) The design of Rijndael: AES - the advanced encryption standard, 1st edn. Springer, Heidelberg
42. Ronald L, Rivest RS, Robshaw MJB, Yin YL The RC6 Block Cipher. https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf
43. Boneh D, Franklin M (2001) Identity-based encryption from the weil pairing. In: Kilian J (ed) Advances in Cryptology - CRYPTO 2001, vol 2139, Springer, pp 213–229. https://doi.org/10.1007/3-540-44647-813
44. Gharib M, Moradlou Z, Doostari MA, Movaghar A (2017) Fully distributed ECC-based key management for mobile ad hoc networks. Comput Netw 113:269–283. https://doi.org/10.1016/j.comnet.2016.12.017
45. Youtube. https://www.youtube.com/

46. Cheng X, Dale C, Liu J (2008) Dataset: Statistics and Social Network of YouTube Videos. In: 2008 16th Interntional Workshop on Quality of Service, IEEE, pp 229–238.http://netsg.cs.sfu.ca/youtubedata/