Security Notions for Cloud Storage and Deduplication

Colin Boyd[1], Gareth T. Davies[1(✉)], Kristian Gjøsteen[1], Håvard Raddum[2], and Mohsen Toorani[3]

[1] NTNU, Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,gareth.davies,kristian.gjosteen}@ntnu.no
[2] Simula@UiB, Bergen, Norway
haavardr@simula.no
[3] University of Bergen, Bergen, Norway
mohsen.toorani@uib.no

Abstract. Cloud storage is in widespread use by individuals and enterprises but introduces a wide array of attack vectors. A basic step for users is to encrypt their data, yet it is not obvious what security properties are required for such encryption. Furthermore, cloud storage providers often use techniques such as data deduplication for improving efficiency which restricts the application of semantically-secure encryption. Generic security goals and attack models have thus far proved elusive: primitives are considered in isolation and protocols are often proved secure under ad hoc models for restricted classes of adversaries.

We formally model natural security notions for cloud storage and deduplication using a generic syntax for storage systems. We define security notions for confidentiality and integrity in encrypted cloud storage and determine relations between these notions. We show how to build cloud storage systems that satisfy our defined security notions using standard cryptographic components.

1 Introduction

When handing over their data to third parties, it is natural that users regard security and privacy as critical concerns. Some users may be willing to trust a cloud storage provider (CSP) to secure their data, but as the Snowden revelations have shown, even well-meaning providers are not immune from compromise. Users increasingly want to manage confidentiality and integrity of their outsourced data without the need to trust the CSP.

It is perhaps surprising that up to now there seems to be no general model of security for remote storage. What are the essential components of a remote storage system, and how should users protect their data so that they can interact usefully with the system while maintaining security requirements? It may seem obvious that users should simply encrypt their data, but the remote storage scenario is different from that of communication or local storage. Multiple users interact and files are vulnerable to manipulation by the CSP. Moreover,

J. Baek et al. (Eds.): ProvSec 2018, LNCS 11192, pp. 347–365, 2018.
https://doi.org/10.1007/978-3-030-01446-9_20

efficiency factors may conflict with user goals. Specifically, CSPs extensively use *deduplication* for removing redundant copies of data and saving storage.

Security Goals for Cloud Storage. Users who trust their storage provider can send plaintext data for storage: this is today the most common situation. Several commercial storage providers, however, support client-side encryption so that the cloud provider cannot obtain the plaintext. It is not immediately obvious which security properties are most appropriate for client-side encryption. In this paper we create a fine-grained approach to adversarial capabilities in terms of compromise of both users and servers. We consider three different security goals and show how they can be achieved within our model.

IND This is the usual standard for strong confidentiality of encrypted data: *indistinguishability* of ciphertexts. We will show that this can be achieved in our cloud storage model by appropriate choice of encryption scheme.

PRV Deduplication cannot take place if strong encryption is deployed. Privacy under a *chosen distribution attack* (PRV-CDA) [4] is used to identify achievable security in the presence of message-derived keys. We will show how this primitive-level goal can be transferred to the protocol level using our model.

INT In many scenarios the user wishes to remove local copies of outsourced files so has no way checking if a retrieved file has been modified. We introduce a notion of integrity of ciphertexts for cloud storage schemes (INT-SC), with three flavors corresponding to differing levels of server compromise. Furthermore, we consider integrity in deduplicating schemes and link existing definitions of tag consistency to our framework.

Contributions. The literature on secure cloud storage has tended to focus on ad hoc solutions rather than generic models that capture classes of realistic adversaries. We fill this gap by providing a comprehensive definition of cloud storage in terms of the input/output behavior of the entities in the system. For various security properties we use our framework to define game-based notions.

We identify the limits of a number of key security properties in cloud storage and provide generic security models for encrypted storage and deduplication. Our framework covers many natural and practically-deployed cloud storage solutions and this approach enables practitioners to identify which components of a storage scheme need to satisfy certain criteria for a given security goal. Specifically, we:

- create a modular framework for security models in cloud storage;
- cast known and novel attack models and security notions in our framework;
- consider known attacks on schemes.

Previous Work. From the point of view of a single enterprise or an individual, secure outsourced storage seems straightforward: encrypt all files at the client side using strong symmetric encryption, use a message authentication code (MAC) or AE for integrity and keep the key(s) secret. In this mindset, cloud storage appears similar to disk encryption [11], and this is the approach

recently taken by Messmer et al. [21]. Such an approach ignores more complex interactions between different clients and servers lacking mutual trust.

Moreover, the business model that allows CSPs to provide cheap storage relies on individuals not employing encryption so as not to interfere with data deduplication. Since the concept of convergent encryption [8] was formalized by Bellare et al. [4] there have been a number of proposals for secure deduplication [10,15,18,27], with each appearing to provide a new threat model. This has led to uncertainty over what security guarantees these schemes provide. For example, the protocol of Liu et al. (CCS '15) [18], as noted later in a revision to the ePrint version [19] and also in subsequent work by some of the same authors [20], only provides the security claims if one round of the protocol is considered: for more than one round, any user can infer whether or not any file is stored on the cloud – a side channel that can result in serious security issues [2,13].

Specific functionalities designed for the cloud storage scenario have been modelled and analysed extensively. These include, but are not limited to: protocols for proofs of retrievability (PoR) [14], proofs of data possession (PDP) [3], proofs of ownership (PoW) [12], secure auditing [29], and privacy of interactions (queries and results) between a data owner and a malicious server [26].

2 Preliminaries

We use the notation $a \leftarrow f(b)$ to denote assignment of a to the result of computing $f(b)$ if f is either a function or an algorithm. $a \xleftarrow{\$} D$ means that either a has been chosen from set D (uniformly) or according to some distribution D. If $\mathbf{a}$ is a vector then denote the i^{th} component by $\mathbf{a}[i]$, with $|\mathbf{a}|$ denoting the number of components. We denote concatenation of two values, usually bit-strings, by $a||b$. If L is a list then the code $\mathsf{L} \xleftarrow{\cup} \{a\}$ indicates that a is appended to L.

Throughout this work we assume that all security parameters and public values are known to all parties (and algorithms). This means that if ever we need to initialize a primitive or protocol, generation of such values is implicit. In situations where algorithms are run with no inputs given, these public values are still provided. Our security experiments consider an adversary that possibly interacts with some oracles before terminating and providing an output. We use the concrete security framework throughout, thus we do not regard adversaries in terms of security parameters: in particular we avoid use of negligible advantage since in the cloud setting the (possibly adversarial) server can perform huge numbers of operations per second. In pseudocode for security games, **return** $b' \stackrel{?}{=} b$ is shorthand for **if** $b' = b$ **then return** 1 // **else return** 0, with output of 1 indicating successful adversarial behavior. We will use an init procedure to represent initialization of cloud storage systems – this encompasses a number of possible subroutines but for generality and brevity we use a single line. An oracle in a security game that corresponds to the environment simulating some functionality func is denoted by $\mathcal{O}.\mathsf{func}$ (the simulation may invoke some restrictions on func).

Cloud storage infrastructure includes some always-available *servers* (the CSP) and some *clients* (that act on behalf of *users*) that are sometimes available and interact with the servers. There may additionally be some parties that interact with the clients and servers to provide extra functionality, such as a key-server [15] or an auditing mechanism [29]. We regard *users* as the entities with distinct logins to a system, and *clients* as the devices that interact with the server on behalf of their owner, the user. This allows us to consider two clients that have the same key material, e.g. laptop and phone, of one user.

Symmetric-Key Encryption (SKE). Our results aim to build secure schemes from the most simple and well-understood building block in cryptography: symmetric encryption. In traditional SKE, two parties agree some key in advance and then communicate over some (presumed insecure) channel. In the context of outsourced storage the two parties are often the same user at different points in time. Additionally, the 'channel' is not only the communication lines between the user and the server but also the server's storage when the ciphertext is at rest. Our syntax and definitions of security follow standard practice and are given in the full version [6]; here we highlight some choices we have made to facilitate our results. Our games for IND-CPA, IND-CCA2 and AEAD represent multi-challenge left-or-right indistinguishability. We do not restrict what we regard as *files* for these notions due to the myriad of ways in which a file can be processed before, during and after the store procedure. Instead we insist that trivial wins are disallowed by *some restriction* on the relation between files sent in $(\mathsf{F}_0, \mathsf{F}_1)$ calls to left-or-right oracles: normally this will mean that the segmentation procedure applied to both files must yield the same number of blocks. We also require PRV-CDA and tag consistency as given by Bellare et al. [4].

3 Modelling Cloud Storage

Our goal is to study security of cloud storage in terms of confidentiality and integrity of files. Such analysis is only possible if the model provides sufficient detail about adversarial capabilities. The challenge is to provide a sufficiently detailed model that allows analysis, yet is generic enough to facilitate study of natural schemes. It is desirable that the model can easily be extended to incorporate particular exotic design choices. We present here what is to our knowledge the first such model for (secure) outsourced storage that accommodates both widely-deployed (and conceptually straightforward) solutions as well as much of the literature (in particular schemes facilitating encrypted data deduplication). As for any storage scheme, a user of a cloud storage scheme should be able to store, retrieve and delete files. A user must be able to specify which previously stored files to retrieve or delete, and we shall achieve that by having the user choose a unique file handle (identifier) for each file when storing. Correctness can then be defined in the expected way, stated here for a notational introduction:

Definition 1 (Correctness). *If user* uid *previously stored* F *under handle* id *then when it later retrieves* id *the result will be* F*, unless client has sent* $\mathsf{del}(\mathsf{id})$.

$\underline{\mathsf{init}}$
01. $\mathsf{ST} \leftarrow \emptyset$

$\underline{\mathsf{newu}(\mathsf{uid})}$
02. $\mathsf{uk}_{\mathsf{uid}} \leftarrow \mathsf{kgen}$
03. $\mathsf{KT}_{\mathsf{uid}} \leftarrow \emptyset$
04. **return** $\mathsf{uk}_{\mathsf{uid}}$

$\underline{\mathsf{del}(\mathsf{uid}, \mathsf{id})}$
05. $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\perp, \mathsf{id})\}$
06. **if** $\exists$ $\{\mathsf{uid}, \cdot, \mathsf{id}, 0\}$ in ST **then**
07. $\quad \mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, -, \mathsf{id}, 1)\}$

$\underline{\mathsf{upl}(\mathsf{uid}, c, \mathsf{id})}$
08. $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$

$\underline{\mathsf{store}(\mathsf{uid}, \mathsf{F}, \mathsf{id})}$
09. $\mathsf{fk} \leftarrow \mathsf{fkeyGen}(\mathsf{F}, \mathsf{uk}_{\mathsf{uid}})$
10. $c \leftarrow \mathsf{E}_{\mathsf{fk}}(\mathsf{F}, \mathsf{id})$
11. $\mathsf{upl}(\mathsf{uid}, c, \mathsf{id})$
12. $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathsf{fk}, \mathsf{id})\}$

$\underline{\mathsf{retr}(\mathsf{uid}, \mathsf{id})}$
13. **if** $\exists$ $(\mathsf{uid}, \cdot, \mathsf{id}, 1) \in \mathsf{ST}$ **or**
14. $\quad \exists(\perp, \mathsf{id}) \in \mathsf{KT}_{\mathsf{uid}}$ **then**
15. $\quad$ **return** $\perp$
16. **if** $\exists$ $(\mathsf{uid}, c, \mathsf{id}, 0) \in \mathsf{ST}$ **and**
17. $\quad \exists(\mathsf{fk}, \mathsf{id}) \in \mathsf{KT}_{\mathsf{uid}}$ **then**
18. $\quad \mathsf{F} \leftarrow \mathsf{D}_{\mathsf{fk}}(c, \mathsf{id})$
19. $\quad$ **return** F
20. **else**
21. $\quad$ **return** $\perp$

Fig. 1. Definition of a cloud storage scheme CS.

We use storage handles, denoted by id, to indicate the value that the user wishes to use in the future to retrieve that file. We regard the generation of id as outside the scope of the model. It is perhaps easiest to think of id as a random value that is generated by the user's device for each file. In practice all a user sees is a list of filenames (which are certainly not suitable for our purposes due to non-uniqueness): this approach allows us to focus on issues directly related to confidentiality and integrity. This handle is distinct from the deduplication 'tags' used in prior literature on message-locked encryption [1,4,8]. In client-side-deduplicating systems the user first sends some short, message-derived tag (for example in convergent encryption [8] this is $\tau = \mathsf{H}(\mathsf{H}(C))$ for ciphertext C) and if the server already has this tag, informs the user not to send the full ciphertext and updates that ciphertext's metadata to indicate that the user can in future retrieve the ciphertext. Note that this process also occurs in deduplicating schemes that do not use any encryption. In this context, this tag is all that is required to claim ownership of a file. Our handles do not have this feature: they simply ensure that retrieve queries work 'correctly'. In Sect. 5 we will discuss integrity in the context of deduplicating and non-deduplicating cloud storage, and highlight the differences between our handles and these tags in more detail.

3.1 A Model for Cloud Storage

Our model for cloud storage is depicted in Fig. 1. A cloud storage scheme $\mathsf{CS}[\mathsf{SKE}, \mathsf{fkeyGen}] = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ is parameterized by a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ and a file-key generation procedure $\mathsf{fkeyGen}$, and supports natural functionalities: init for initialization, newu for adding a new user, store for storing a file, retr for retrieval and del for deletion.

Each user is associated with a user identification uid, and each file is identified by a storage handle id. We define per-user keys uk and per-file keys fk. Each user has some (preferably small) local storage and the server maintains (what is from an abstract perspective at least) a vast data structure. Generation of per-user key material uk (line 02) may include keys for a number of different purposes. The user stores this material and their own KT ('Key Table') locally and the server(s) maintains a database ST ('Store Table') that it uses to track file ownership and retrieval handles. This means that there is only one ST but there could be many KTs. Our model retains generality: to our knowledge it incorporates almost all intuitive schemes and all protocols from the literature (more details in next subsection). We make no assumption about how files are handled in terms of segmentation, nor do we consider redundancy at the server's backend. The model that follows is, by design, modular and generic enough to cope with straightforward modifications to incorporate such processes.

We now discuss the design choices that require further attention. In line 02 we explicitly regard the per-user key generation procedure as occurring separately from the other procedures, this is to retain generality and to allow us to focus on file-key generation. ST tracks deletion status of each file for each user (lines 13–15), using a bit as the fourth value in each entry. In deployed systems this abstract procedure may not be done as directly as we describe. Line 07 indicates that the server may at this point delete the ciphertext for the deleted file, however we do not enforce this: the '1' flag indicates deletion has occurred[1] for user uid and handle id. Encryption algorithm E takes id as input (line 10): if SKE is an AEAD scheme then id could be the associated data – we model this construction later on. Lines 16–19 specify that the file can only be retrieved if it has not been removed either by the client or the server: in particular line 17 says that if there exists an fk such that $(\mathsf{fk}, \mathsf{id}) \in \mathsf{KT}_{\mathsf{uid}}$ then the retrieve is allowed to continue.

We differentiate between store – the entire process of storing a file on the server and updating the client's local storage – and upl– the specific action that occurs server-side. The definition generalizes to include the simplest and most widely-deployed solution, which is without any client-side encryption at all. Any scheme that distributes files among multiple servers is also included, incurring a rather complicated outsourced state ST, however the results in the remainder of this paper will mainly focus on the single server case. To satisfy correctness we require an implicit assumption that the CSP forwards all requests honestly: this approach reflects cryptographic models for key exchange. If fk used for store (encryption) is not the same as the one used for retr (decryption) then no scheme can be correct. The adversaries that we consider cannot modify KT so key symmetry is implicit in our model and for the rest of the paper.

[1] Many CSPs never actually delete files at the backend, and this is understandable: the cost of finding, accessing and removing a file and all its redundant copies is often considerable, and if the CSP uses client-side deduplication then if the user (or any other) uploads that file in the future this will incur a bandwidth cost.

	Scheme	$\mathsf{fkeyGen}(\mathsf{F}, \mathtt{uk})$
1	No encryption	$\mathsf{fk} \leftarrow \perp$
2	Per-user key	$\mathsf{fk} \leftarrow \mathtt{uk}$
3	Per-file key	$\mathsf{fk} \leftarrow \mathsf{KG}$
4	MLE [4]	$\mathsf{fk} \leftarrow \mathsf{H}(\mathsf{F})$
5	Liu et al. [18]	$\mathsf{fk} \leftarrow \mathbf{PAKE.Out}(\mathsf{F})$
6	DupLESS [15]	$\mathsf{fk} \leftarrow \mathbf{OPRF.Out}(\mathsf{F})$
7	Duan [10]	$\mathsf{fk} \leftarrow \mathbf{DOKG}(\mathsf{F})$
8	Stanek et al. [27]	$\mathsf{fk} \leftarrow \mathbf{Thr.PKE.KG}(\mathsf{F})$
9	CDStore [17]	$\mathsf{fk} \leftarrow \mathbf{SS}(\mathsf{H}(\mathsf{F}))$

Fig. 2. Specification for $\mathsf{fkeyGen}$ procedure for existing cloud storage schemes

3.2 Modelling Existing Schemes and Literature

In Fig. 2 we detail the file-key generation procedure for natural constructions and a number of schemes from the existing literature. The natural scenarios include a CSP that does not support client-side encryption (line 1), a CSP wherein each user holds a per-user key and encrypts all files with that key (line 2), and a CSP wherein a per-file key is randomly chosen at the point of the file being uploaded (line 3). The per-user key scenario (line 2) allows deduplication of a particular user's files (but not cross-user deduplication) which can still allow great savings, particularly in the backup setting. This case also reflects some enterprise scenarios in which an organization has a storage gateway (that may interact with trusted hardware, such as a hardware security module) that deduplicates files and encrypts (under one key) on behalf of all of its employees before sending to some public cloud (CSP). The per-file key scenario (line 3) intuitively provides increased confidentiality, but introduces challenging key management. A gateway can also be used in this case as described in the Omnicloud architecture [16]: this of course requires the gateway to additionally manage the vast number of keys that could be generated in the enterprise scenario.

Schemes in lines 4–9 all aim to provide 'secure cross-user deduplication' to some extent, providing more confidentiality than using no encryption (line 1) but at the risk of opening a side channel that may allow a user to learn if a file is already stored on the server [2]. In many schemes such as those of Keelveedhi et al. (DupLESS) [15] and Liu et al. [18], the $\mathsf{fkeyGen}$ procedure is not a single algorithm but a protocol run between the user and the key server or the other users in the protocol, respectively. Stanek et al. [27] use both convergent encryption and an outer layer threshold encryption scheme to produce ciphertexts, and the $\mathsf{fkeyGen}$ protocol interacts with two trusted third parties. Duan [10] attempts to avoid the single point of failure inherent in having a single (semi-trusted) key server (KS) in DupLESS-like schemes: $\mathsf{fkeyGen}$ generates encryption keys using a distributed oblivious key generation, instantiated using a (deterministic) threshold signature scheme. The CDStore protocol of Li et al. [17] distributes shares of a file to multiple cloud servers using so-called *convergent dispersal*.

The restriction to SKE in line 10 of Fig. 1 is for the purposes of results in Sects. 4 and 5. Note here that schemes 1–7 in Fig. 2 precisely fit our model while schemes 8 and 9 do not simply encrypt using SKE – for these schemes E represents some other encryption mechanism. In the schemes that do precisely fit our model, generation of file key fk could happen as part of the key generation procedure kgen: for example in the per-user key case (line 2 of Fig. 2) $\mathsf{fkeyGen}$ is the identity function. This is one of many potential modular extensions of our framework: we could of course consider a model in which (for example) the $\mathsf{fkeyGen}$ and E algorithms are general functions with arbitrary inputs.

Cloudedup [23] uses block-level convergent encryption to send ciphertexts to a third party that adds further (symmetric) encryption and manages metadata. Dang and Chang [7] similarly assume a trusted entity, in their case hardware. A trusted enclave uses an oblivious PRF (similarly to DupLESS) to get block-derived keys to allow the enclave to perform deduplication: the enclave acts as a deduplication gateway then applies randomized encryption before sending ciphertexts to the CSP. In these schemes encryption is done in two phases and the per-block keys are managed by the third party; this does not quite fit our model but it is straightforward to modify how KT (and SKE) works to analyze such schemes. Recently Shin et al. [25] attempted to distribute the role of the key server in DupLESS-like schemes by additionally using inter-KS deduplication. Again, allowing this type of scheme is a simple extension of our model.

To simplify much of our analysis later on we require that every time a new file is stored by a client, a new id is generated. This leads to the following assumption:

Assumption 1. *In all cloud storage schemes* CS *considered in this paper,* store *is never called on the same* id *twice.*

We emphasize that id is the retrieval handle chosen by the client, and is distinct from the deduplication 'tags' used in prior literature. This assumption (and the existence of the id) emphasizes that our handles are there to distinguish file uploads from one another: each $\{\mathsf{uid}, \mathsf{id}\}$ pair can only ever occur once.

4 Confidentiality

Now that we have defined a suitable syntax for cloud storage schemes, we can begin to consider the many ways in which security features can be obtained. In this section we turn our attention to confidentiality of files with respect to realistic adversaries. Defining confidentiality notions of security is a two-step process: We first define what we want to prevent the adversary from learning (the *goal*), and then we specify the adversary's capabilities.

There are several possible goals. The classical cryptographic goal is indistinguishability, where one of two adversary-chosen files was stored and the adversary is unable to decide which file was stored. This is similar to semantic security, where a file sampled from one of two adversary-chosen probability spaces was stored, and the adversary is unable to decide which distribution the file was sampled from. A weaker notion is to sample a file from one of two pre-chosen

high-entropy probability spaces. The adversary has two distinct capabilities when attacking a cloud storage system. The first is the ability to influence the actions of the honest users. The second is the ability to influence the CSP.

When considering corruption of users, it is important to note that an adversary can usually create genuine logins to a system, and thus receive a valid uid and $\mathtt{uk}$ for an arbitrary number of users. We model this by distinguishing between two types of newu query: $\mathcal{O}.\mathsf{newuC}$ creates a valid (Corrupt) user and outputs its $\mathtt{uk}$ to the adversary, and $\mathcal{O}.\mathsf{newuH}$ that only creates a valid (Honest) user[2]. For its corrupted users the adversary may not necessarily use $\mathtt{uk}$ and $\mathsf{fkeyGen}$ correctly (which $\mathcal{O}.\mathsf{store}$ cannot handle): we model this capability by giving the adversary access to an $\mathcal{O}.\mathsf{upl}$ oracle that pushes some $\{(\mathsf{uid}, c, \mathsf{id}, 0)\}$ tuple to the server's storage table ST. We regard the minimum adversarial capability as being able to have full control over a number of corrupted users and to make honest users store files, we refer to this notion as a *chosen store attack* (CSA). The adversary may even be able to get honest users to retrieve files from the cloud storage system, a *chosen retrieve attack* (CRA). Analogously to encryption, CSA and CRA somewhat correspond to CPA and CCA, respectively.[3]

The adversary's control of the CSP can be usefully divided into three levels: the adversary may have no influence at all on the CSP then we have an honest CSP giving the adversary *zero access* (Z). The adversary may also be able to look at the CSP's storage and key material, but not tamper with anything, a *passively corrupt* (P) CSP. This models both honest-but-curious CSPs and snapshot hackers (of the cloud's storage servers or the communication channel). And finally, the adversary may have full control over the CSP, an *actively corrupt* (A) adversary. When the CSP is honest, it may seem that our model always guarantees confidentiality because the adversary would never have access to ciphertexts. However, this is not the case, since the file key generation procedure is regarded as a protocol and may leak information (as mentioned earlier with the protocol of Liu et al. [18]). Roughly speaking, we can say that when the CSP is honest, we consider only the security of the file key generation protocol. When the CSP is passively corrupt, we must additionally consider the confidentiality of the encryption used. When the CSP is actively corrupt, we must also consider integrity in the encryption mechanism. This separation of concerns is by design.

4.1 Defining Confidentiality for Cloud Storage

In combination we define a generic $\mathsf{IND\text{-}atk\text{-}csp}$ experiment with six distinct cases: $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ and this $\mathsf{IND\text{-}atk\text{-}csp}$ experiment is detailed in Fig. 3. Just as in our general definition for storage protocols (Fig. 1) we keep track of the retrieval capability by using a table ST, initially set to empty.

[2] It is certainly possible to extend this model to adaptive corruptions, however this would add considerable extra complexity to any scheme.

[3] It is possible to define an equivalent of a passive adversary, however since our definitions are multi-challenge, the adversary can always call $\mathcal{O}.\mathsf{LR}_b$ on $\mathsf{F}_0 = \mathsf{F}_1$ to mimic a store query (though it cannot query $\mathcal{O}.\mathsf{retr}$ on these ciphertexts).

$\mathbf{Exp}^{\mathsf{IND\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}}$:
  init
  $b \xleftarrow{\$} \{0,1\}$
  $\mathsf{CL}, \mathsf{users_C}, \mathsf{users_H} \leftarrow \emptyset$
  $b' \leftarrow \mathcal{A}^{\text{oracles}}$
  return $b' \stackrel{?}{=} b$

$\mathcal{O}$.newuC(uid) :
  $\mathsf{users_C} \xleftarrow{\cup} \mathsf{uid}$
  $\mathsf{uk_{uid}} \leftarrow \mathsf{kgen}$
  $\mathsf{KT_{uid}} \leftarrow \emptyset$
  return $\mathsf{uk_{uid}}$

$\mathcal{O}$.newuH(uid) :
  $\mathsf{users_H} \xleftarrow{\cup} \mathsf{uid}$
  $\mathsf{uk_{uid}} \leftarrow \mathsf{kgen}$
  $\mathsf{KT_{uid}} \leftarrow \emptyset$
  return $\perp$

$\mathcal{O}$.store(uid, F, id) :
  do store(uid, F, id)

$\mathcal{O}$.upl(uid, c, id) :
  if uid $\in \mathsf{users_C}$ **then**
    do upl(uid, c, id)

$\mathcal{O}$.del(uid, id) :
  do del(uid, id)

$\mathcal{O}$.LR$_b$(uid, F_0, F_1, id) :
  if uid $\notin \mathsf{users_H}$ **then**
    return $\perp$
  else
    $\mathcal{O}$.store(uid, F_b, id)
    $\mathsf{CL} \xleftarrow{\cup} \{(\mathsf{uid}, \mathsf{id})\}$

$\mathcal{O}$.retr(uid, id) : // CRA only
  if uid $\notin \mathsf{users_H}$ **or** (uid, id) $\in \mathsf{CL}$ **then**
    return $\perp$
  else
    F $\leftarrow$ retr(uid, id)
    return F

$\mathcal{O}$.peek(uid, id) : // P, A only
  return $\{(\mathsf{uid}, c, \mathsf{id}, 0/1) \in \mathsf{ST}\}$

$\mathcal{O}$.erase(uid, id) : // A only
  $\mathsf{ST} \leftarrow \mathsf{ST} \setminus \{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{ST}\}$

$\mathcal{O}$.insert(uid, c, id, d) : // A only
  $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, d)\}$

Fig. 3. The experiment defining IND-atk-csp security for cloud storage, for atk $\in \{\mathsf{CSA}, \mathsf{CRA}\}$, csp $\in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$. All adversaries have access to $\mathcal{O}$.newuC, $\mathcal{O}$.newuH, $\mathcal{O}$.store, $\mathcal{O}$.upl, $\mathcal{O}$.del and $\mathcal{O}$.LR. CRA additionally has access to $\mathcal{O}$.retr, P additionally has the $\mathcal{O}$.peek oracle and finally A additionally has $\mathcal{O}$.erase and $\mathcal{O}$.insert.

The security experiment keeps track of the $\mathcal{O}$.LR queries using a forbidden list CL to prevent trivial wins. In order to model the attacker's influence on the CSP, we introduce three new oracles: $\mathcal{O}$.peek, $\mathcal{O}$.erase and $\mathcal{O}$.insert. These are not functionalities of storage systems so they are not included in Fig. 1. $\mathcal{O}$.peek allows the adversary to see the ciphertext (and deletion status) for some user uid and some handle id, and this is available to a passively corrupt adversary (P). $\mathcal{O}$.insert and $\mathcal{O}$.erase model actively malicious (or completely compromised) CSPs, granting the ability to store or delete arbitrary items in the CSP's database: these two oracles are only available to an actively corrupt (A) attacker.

Definition 2 (IND-atk-csp *Security for Cloud Storage). Consider any cloud storage scheme* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$. *The* IND-atk-csp *advantage for an adversary* $\mathcal{A}$ *and* atk $\in \{\mathsf{CSA}, \mathsf{CRA}\}$, csp $\in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ *against* CS *is defined by*

$$\mathbf{Adv}^{\mathsf{IND\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}} = 2 \cdot \left[\mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{IND\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}} = 1 \right] - \frac{1}{2} \right]$$

where experiment $\mathbf{Exp}^{\mathsf{IND\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}}$ *is given in Fig. 3.*

On Our Model. Our weakest notion of server compromise, IND-atk-Z, refers to a very limited adversary, with no access to the server's database and only capable of making 'challenge' store queries (modelled by $\mathcal{O}.\mathsf{LR}_b$) with users that it does not have key material for, resulting in ciphertexts that it cannot access. Thus even a scheme with no encryption can be secure under this notion. This is by design: the only schemes that do not meet this requirement are those that leak information about a file *to other users* during the store procedure.

It is possible to imagine adversaries that may wish to act without being noticed by the users they have infiltrated. This CRA adversary would thus retrieve but not store or delete – and yet seems to be more 'limited' than a CSA adversary that does perform store/delete operations and does not mind if the user notices its behavior. This is the nature of adversaries in cloud storage: the clear hierarchy that exists for encryption does not easily translate.

The concept of length equality for files in cloud storage is not as clear cut as it is for bitstrings in an IND-based game for encryption. If the encryption scheme is not length hiding and the adversary submits one $\mathcal{O}.\mathsf{LR}$ query and one $\mathcal{O}.\mathsf{peek}$ query: if the ciphertext lengths differ then the adversary trivially wins the game. As mentioned earlier, this means that an inherent restriction exists on $\mathcal{O}.\mathsf{LR}$ queries: if the length of (the segmentation of) F_0 and F_1 differs then the experiment does not go ahead with the store procedure[4].

Relations Between Notions. While we have just defined six adversarial capabilities, in fact only three are distinct. Figure 4 summarizes how the notions relate to each other, and we detail these relations fully in the full version [6]. We give a brief intuition here. If notion A has strictly more oracles than notion B then any CS secure under A will also be secure under notion B. This means that IND-atk-A $\Rightarrow$ IND-atk-P $\Rightarrow$ IND-atk-Z for $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, and also IND-CRA-csp $\Rightarrow$ IND-CSA-csp for $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$. This leaves three equivalences and two separation results. IND-CSA-Z and IND-CRA-Z are equivalent since it is always possible to simulate the $\mathcal{O}.\mathsf{upl}$ queries of an IND-CRA-Z adversary: this adversary can only use $\mathcal{O}.\mathsf{store}$ to place items in ST that it can later retrieve (since $\mathcal{O}.\mathsf{LR}$ and $\mathcal{O}.\mathsf{upl}$ are forbidden), and by correctness this means a simulator can just keep track of these queries in a table. A similar approach can be used to show that IND-CSA-P and IND-CRA-P are equivalent. To show that IND-CSA-P and IND-CSA-A are equivalent, the simulator needs to successfully simulate $\mathcal{O}.\mathsf{insert}$ and $\mathcal{O}.\mathsf{erase}$ queries. This is indeed possible: the simulator keeps track of such queries in a table. As we have mentioned, the CS built using no encryption is IND-atk-Z. It is however not IND-CSA-P: the adversary simply performs one $\mathcal{O}.\mathsf{LR}$ query with distinct files and then queries $\mathcal{O}.\mathsf{peek}$ on that entry. We henceforth refer to these three distinct notions using IND-atk-Z, IND-CSA-P and IND-CRA-A.

[4] In deduplicating schemes segmentation can be a side channel in itself [24]. If the adversary can observe a distinguishable error symbol as part of its $\mathcal{O}.\mathsf{LR}$ queries then this may cause issues. We strictly disallow this by not returning anything to the adversary and assuming a stringent restriction on allowed file pairs for $\mathcal{O}.\mathsf{LR}$.

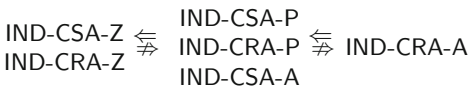

Fig. 4. Relations between IND notions for confidentiality of cloud storage systems.

4.2 Achieving Confidentiality in Cloud Storage

In the full version we show four straightforward reductions, showing that the intuitive protocols that we expect to meet all security goals – strong encryption with random file identifiers – do in fact provide confidentiality. Formal statements are omitted due to space constraints, but we summarize the results in Fig. 5. We show that if users encrypt using an IND-CPA-secure SKE scheme using their own fixed key then the overall system is IND-CSA-P secure (and thus also IND-CRA-P and IND-CSA-A secure). Specifically, key generation for CS outputs a random key to each user (or, kgen runs the SKE's KG algorithm) and fkeyGen(F, uk) outputs uk for all F. We go on to show that this same construction, when implemented with an AEAD scheme, yields an IND-CRA-A-secure cloud storage system, our strongest notion. The scenario in which a random symmetric key is created for each file, perhaps surprisingly meets the strongest IND-CRA-A notion of security even with IND-CPA-secure encryption. Finally we consider the secure deduplication setting with (a reduction to) PRV-CDA security of the underlying encryption: for this we need a modified security experiment (see full version). These theorems emphasize the simplicity and versatility of the model.

Theorem	Key Usage	Encryption	Conf of CS
1	Per-user	+ IND-CPA	⇒ IND-CSA-P
2	Per-user	+ AEAD	⇒ IND-CRA-A
3	Per-file	+ IND-CPA	⇒ IND-CRA-A
4	File-derived	+ PRV-CDA	⇒ PRV-CSA-P

Fig. 5. Summary of the composition results.

Deduplicating Systems Using File-Derived Keys. A natural way for using SKE in deduplicating systems is to derive encryption keys from the files themselves [4,8]. Cloud storage schemes with this property cannot achieve the usual indistinguishability notion because the adversary knows the possible files and therefore the possible encryption keys used. For such schemes, PRV-CDA [4] asks an adversary to distinguish ciphertexts when files are sampled from some pre-chosen high-entropy probability space and then encrypted. The probability space must be independent of the encryption scheme to avoid pathological situations, hence pre-chosen. This security notion can be achieved by both deterministic [4] and randomised schemes [1,4]. Based on such an encryption scheme, we define

a natural cloud storage scheme by having $\mathsf{fkeyGen}$ simply run the encryption scheme's key derivation algorithm. We define a notion of security for such cloud storage similar to $\mathsf{PRV\text{-}CDA}$, where we sample two vectors of files and store every file from one of those vectors. The adversary's task is to determine which vector was stored. In the full version we define this notion and prove the natural theorem, stated as the last line of Fig. 5.

4.3 Deduplicating Schemes with Non-trivial fkeyGen Procedures

The results so far in this section have only considered schemes for which $\mathsf{fkeyGen}$ is an operation that can be run locally, without the need for communicating with other users, the server or third parties (i.e. lines 1–4 of Fig. 2)[5]. While our model (Fig. 1) can handle deduplicating schemes with complex $\mathsf{fkeyGen}$ protocols such as that of Liu et al. [18], Duan [10] and Keelveedhi et al. [15], our security definitions do not fully capture them due to the 'unnatural' inputs to $\mathsf{fkeyGen}$ when 'called' by store. A simple extension to our framework allows analysis of such schemes: an $\mathcal{O}.\mathsf{fkeyGen}$ oracle that can be called on arbitrary inputs.

Our model is also easily extensible to the distributed storage context: the $\mathcal{O}.\mathsf{peek}$ oracle, instead of returning the tuple $\{(\mathsf{uid}, c, \mathsf{id}, 0/1)\}$, could take as input some indices that correspond to different servers and return the information stored on that subset of the servers, if any, under uid and id. This would enable a rigorous analysis of schemes such as CDStore [17].

It is straightforward to create a variant, $\mathsf{D\text{-}IND}$, of our generic IND experiment for deterministic encryption: the adversary is not allowed to send the same file to $\mathcal{O}.\mathsf{LR}$ or $\mathcal{O}.\mathsf{store}$. In particular, the experiment initializes an empty list, and on each F or $(\mathsf{F}_0, \mathsf{F}_1)$ query to store, resp. $\mathcal{O}.\mathsf{LR}$, that value is added to the list. If the adversary later attempts to perform $\mathcal{O}.\mathsf{store}$ or $\mathcal{O}.\mathsf{LR}$ with a file already on that list, return $\perp$. Certainly any scheme that is $\mathsf{IND\text{-}atk\text{-}csp}$ is also $\mathsf{D\text{-}IND\text{-}atk\text{-}csp}$ for some $\{\mathsf{atk}, \mathsf{csp}\}$, and furthermore $\mathsf{D\text{-}IND\text{-}atk\text{-}csp} \Rightarrow \mathsf{PRV\text{-}atk\text{-}csp}$.

Since Duan [10] showed that the DupLESS system achieves $\mathsf{D\text{-}IND\$}$ (in the random oracle model), we would expect that DupLESS would meet strong security in our model. However given that the adversary has a $\mathsf{fkeyGen}$ oracle as described above, DupLESS does not even meet $\mathsf{D\text{-}IND\text{-}CSA\text{-}P}$. The attack is straightforward: The adversary calls its $\mathsf{fkeyGen}$ oracle on F_0 to get $\mathsf{fk}_{\mathsf{F}_0}$; then again for some distinct F_1 to get $\mathsf{fk}_{\mathsf{F}_1}$; calls $\mathcal{O}.\mathsf{LR}_b(\mathsf{F}_0, \mathsf{F}_1)$ for some $(\mathsf{uid}, \mathsf{id})$ and does $\mathcal{O}.\mathsf{peek}(\mathsf{uid}, \mathsf{id})$ to receive the c that F_b is stored under. All that is left to do is to attempt to decrypt c using the two keys it got from $\mathsf{fkeyGen}$ earlier to get F_b, then output b. This indicates how weak a $\mathsf{D\text{-}IND\$}$ notion is: in a realistic attack setting, it is trivial for an adversary that has (even only snapshot) access to the cloud's storage to be able to distinguish ciphertexts.

[5] The threshold scheme of Stanek et al. [27] is a special case since $\mathsf{fkeyGen}$ is run locally but the encryption algorithm is not a symmetric encryption scheme.

5 Integrity

Once a user of a cloud storage system has decided to use encryption to ensure confidentiality of files, the user will also wish that integrity is retained for ciphertexts sent to the CSP. One approach to this requirement is proofs of retrievability (PoR) [14], where users embed some data in their files (ciphertexts) and periodically engage in a protocol with the CSP to check that the files have not been deleted or modified. We consider the simpler problem of ensuring that retrieved files are correct. Our approach is inspired by ciphertext integrity notions from the cryptography literature. As before, we focus on generic results rather than concrete instantiations. We formally define a notion of integrity of ciphertexts for cloud storage schemes, denoted INT-SC (INTegrity of Stored Ciphertexts). The experiment is given in Fig. 6. An adversary, in control of a number of users of the cloud storage scheme CS, wins the game by making a user retrieve a file that either the user had *previously deleted*, or that the user *did not store in the first place*. This rules out schemes for which possessing a file hash alone indicates ownership (Dropbox pre-2011 [9,22], content distribution networks, etc.): in Sect. 5.3 we discuss ciphertext integrity in such deduplicating systems.

5.1 Defining Integrity for Cloud Storage

What follows is a definition of integrity for cloud storage with three flavours corresponding to the different levels of server compromise detailed in Sect. 4.1. We call this notion INT-SC-csp for $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$.

We use a second storage table TrueST to track all activities that the adversary makes the (notional) *users* do: store, retr and del. The other ST tracks all of these activities in addition to the oracles modelling active server compromise: $\mathcal{O}$.erase and $\mathcal{O}$.insert. In Sect. 5.2 we focus on actively corrupted servers manipulating the storage database: the adversary will always have access to $\mathcal{O}$.peek, $\mathcal{O}$.erase and $\mathcal{O}$.insert and this corresponds to INT-SC-A. We will later consider integrity in client-side deduplicating systems: there an adversarial client (INT-SC-Z) is (inherently) given more power by the mechanism that saves communication bandwidth. Note that in the description of $\mathcal{O}$.retr′, the code **if** $\{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{ST}\} \neq \{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{TrueST}\}$ means that for fixed uid and id, if there exists an entry in ST and an entry in TrueST such that the tuples are not exactly equal then this condition is met. Thus if the ciphertext component or the deletion bit (or both) being different means that this condition is achieved.

Definition 3 (INT-SC-csp for Cloud Storage). *Let* CS *be a cloud storage system based on symmetric encryption as in Fig. 1, and let* $\mathcal{A}$ *be an adversary. Then the* INT-SC-csp *advantage for an adversary* $\mathcal{A}$ *and* $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ *against* CS *is defined by*

$$\mathbf{Adv}^{\mathsf{INT\text{-}SC\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}} = \mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{INT\text{-}SC\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}} = 1\right],$$

where experiments $\mathbf{Exp}^{\mathsf{INT\text{-}SC\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}}$ *are defined in Fig. 6.*

$\mathbf{Exp}^{\mathsf{INT\text{-}SC\text{-}csp}}_{\mathsf{CS},\ \mathcal{A}}$:
$b \leftarrow 0$
$\mathsf{ST} \leftarrow \emptyset$
$\mathsf{TrueST} \leftarrow \emptyset$
$\mathcal{A}^{\text{oracles}}$
return b

$\mathcal{O}.\mathsf{del}'(\mathsf{uid}, \mathsf{id})$:
do $\mathsf{del}(\mathsf{uid}, \mathsf{id})$
if $(\mathsf{uid}, \cdot, \mathsf{id}, 0) \in \mathsf{TrueST}$ **then**
$\mathsf{TrueST} \xleftarrow{\cup} (\mathsf{uid}, \cdot, \mathsf{id}, 1)$

$\mathcal{O}.\mathsf{store}'(\mathsf{uid}, \mathsf{F}, \mathsf{id})$:
do $\mathsf{store}(\mathsf{uid}, \mathsf{F}, \mathsf{id})$
$\mathsf{TrueST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$, where
$(\mathsf{uid}, c, \mathsf{id}, 0) \in \mathsf{ST}$

$\mathcal{O}.\mathsf{retr}'(\mathsf{uid}, \mathsf{id})$:
do $\mathsf{F} \leftarrow \mathsf{retr}(\mathsf{uid}, \mathsf{id})$
if $\{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{ST}\} \neq \{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{TrueST}\}$
and $\mathsf{F} \neq \perp$
then $b \leftarrow 1$
return F

Fig. 6. The experiment defining INT-SC-csp for cloud storage. The adversary has access to $\mathcal{O}.\mathsf{newuC}$, $\mathcal{O}.\mathsf{newuH}$, $\mathcal{O}.\mathsf{store}'$, $\mathcal{O}.\mathsf{upl}$, $\mathcal{O}.\mathsf{del}'$ and $\mathcal{O}.\mathsf{retr}'$. If $\mathsf{csp} = \mathsf{P}$, the adversary additionally has access to $\mathcal{O}.\mathsf{peek}$; if $\mathsf{csp} = \mathsf{A}$, the adversary additionally has access to $\mathcal{O}.\mathsf{erase}$ and $\mathcal{O}.\mathsf{insert}$. Oracles that are not explicitly stated are as defined in Fig. 3.

Our definition of del in Fig. 1 firstly removes the KT entry and then updates ST if an applicable entry exists. This formulation makes it extremely difficult for an adversary to win the INT-SC-csp game by retrieving a file it previously deleted since it has no ability to edit KT. If del would only delete the KT entry after checking existence in ST then this would allow a trivial way to de-synchronize TrueST and ST. We believe this exposition gives the clearest possible definition of ciphertext integrity for cloud storage systems as we have defined them.

5.2 Achieving Integrity in Cloud Storage

In the full version [6] we show how to construct a cloud storage protocol that meets our strongest INT-SC-A notion. The construction is straightforward: each user holds their own symmetric key and uses an encryption scheme that is INT-CTXT secure during the store procedure (line 2 from Fig. 2). For this we require the syntax for an encryption scheme that can handle associated data – the associated data is the handle id.

Theorem 5. Per-user keys + SKE(AD = id) + INT-CTXT ⇒ INT-SC-A.

Further, we prove an intuitive theorem inspired by Bellare and Namprempre's IND-CPA + INT-CTXT ⇒ IND-CCA2 result for symmetric encryption [5].

Theorem 6. IND-CSA-P + INT-SC-A ⇒ IND-CRA-A.

5.3 Integrity in Deduplicating Schemes

For deterministic schemes such as convergent encryption ($\mathsf{fk} \leftarrow \mathsf{H}(\mathsf{F})$) an adversary with active server compromise can trivially create new ciphertexts that decrypt correctly. For this reason Bellare et al. (BKR) [4] discussed tag consistency (see full version for TC and STC exposition). BKR's tags served a different

```
init
01. ST ← ∅

newu(uid)
02. uk_uid ← kgen
03. KT_uid ← ∅
04. return uk_uid

del(uid, id)
05. KT_uid ←∪ {(⊥, id, τ)}
06. if ∃ {uid, ·, id, τ, 0} in ST then
07.     ST ←∪ {(uid, −, id, τ, 1)}

store(uid, F, id)
08. fk ← fkeyGen(F, uk_uid)
09. c ← E_fk(F, id)
10. τ ← TGen(c)
11. upl(uid, c, id, τ)
12. KT_uid ←∪ {(fk, id, τ)}

upl(uid, c, τ, id)
13. ST ←∪ {(uid, c, id, τ, 0)}

retr(uid, id)
14. if ∃ (uid, ·, id, τ, 1) ∈ ST  or
15.        ∃(⊥, id, τ) ∈ KT_uid then
16.     return ⊥
17. if ∃ (uid, c, id, τ, 0) ∈ ST  and
18.        ∃(fk, id, τ) ∈ KT_uid then
19.     F ← D_fk(c, id)
19a.    τ′ ← TGen(c)
19b.    if τ′ ≠ τ then
19c.        return ⊥_tag
20.     return F
21. else
22.     return ⊥
```

Fig. 7. Definition of a deduplicating cloud storage scheme DCS[SKE.Dedup].

purpose to our handles as their syntax assumes the server does not track the set of allowed users for each file (i.e. tag ownership is enough to retrieve). This assumption opens up systems to duplicate-faking attacks [28] in which a malicious client can find a tag collision for a target file, upload an ill-formed ciphertext under that tag, and stop genuine users from retrieving the target file. Assumption 1 rules out this type of attack, so we must consider a modified system model to include an additional tagging algorithm, formalized in Fig. 7.

A *deduplicating cloud storage scheme* is a tuple $\mathsf{DCS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ as before, but in addition to $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ and $\mathsf{fkeyGen}$ we also require a TGen algorithm. We follow BKR and define the TGen algorithm as acting on ciphertexts only: $\tau \leftarrow \mathsf{TGen}(c)$. This is without loss of generality: the HCE1, HCE2 and RCE schemes that they describe calculate $\tau \leftarrow \mathsf{H}(\mathsf{fk})$ to give a ciphertext formed as $\tau || \mathsf{E}_{\mathsf{fk}}(\mathsf{F})$, then the TGen algorithm parses this value and outputs τ. Again following BKR we define a *deduplicating encryption mechanism* $\mathsf{SKE.Dedup} = (\mathsf{fkeyGen}, \mathsf{E}, \mathsf{D}, \mathsf{TGen})$ that combines an SKE's encryption and decryption algorithms with the $\mathsf{fkeyGen}$ and TGen procedures. BKR called this primitive an MLE, and their definition was for generic KG: for our purposes it is sufficient to only consider the $\mathsf{fkeyGen}$ algorithm we have previously described since in the deduplicating scenario $\mathsf{fkeyGen}$ typically does not use any material that is unique to each user. This combined construction defines all the inputs to the wider cloud storage system: we write this as $\mathsf{DCS}[\mathsf{SKE.Dedup}]$.

In a client-side deduplicating cloud storage system, the upl procedure will be a two stage process: first the user sends τ, gets a response indicating whether it should send the ciphertext or not, and finally sends the ciphertext if asked to.

In our syntax line 13 initially only requires $(\mathsf{uid}, \mathsf{id}, \tau)$ as inputs and checks its storage for a tag match: **if** $\exists\{\cdot, c, \cdot, \tau, \cdot\} \in \mathsf{ST}$ **then** $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, \tau, 0)\}$. If a match is not found, the server sends a message, sometimes called a *deduplication signal* [2], to the user indicating that ciphertext transmission is necessary.

Lines 19a–c in Fig. 7 represent an optional tag check that has a (possibly distinguishable) error symbol: this operation is employed by the HCE2 and RCE schemes described by BKR. If the tag check procedure is enforced as part of retr, then using a SKE.Dedup that is STC does in fact yield a DCS that is INT-SC-A. The result is stated here informally; the proof is in the full version.

Theorem 7. *If* DCS[SKE.Dedup] *implements the tag check (lines 19a-19c in Fig. 7) and if* SKE.Dedup *is* STC *then* DCS *is* INT-SC-A.

Acknowledgements. We thank Frederik Armknecht and Yao Jiang for input to discussions. We also thank anonymous reviewers for useful feedback. This research was funded by the Research Council of Norway under Project No. 248166.

References

1. Abadi, M., Boneh, D., Mironov, I., Raghunathan, A., Segev, G.: Message-locked encryption for lock-dependent messages. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 374–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_21
2. Armknecht, F., Boyd, C., Davies, G.T., Gjøsteen, K., Toorani, M.: Side channels in deduplication: trade-offs between leakage and efficiency. In: Karri, R., Sinanoglu, O., Sadeghi, A., Yi, X. (eds.) AsiaCCS 2017, pp. 266–274. ACM (2017). https://doi.org/10.1145/3052973.3053019
3. Ateniese, G., et al.: Provable data possession at untrusted stores. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F., (eds.) CCS 2007, pp. 598–609. ACM (2007). https://doi.org/10.1145/1315245.1315318
4. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure deduplication. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 296–312. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_18
5. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_41
6. Boyd, C., Davies, G.T., Gjøsteen, K., Toorani, M., Raddum, H.: Security notions for cloud storage and deduplication. IACR Cryptology ePrint Archive 2017/1208 (2017). http://eprint.iacr.org/2017/1208
7. Dang, H., Chang, E.: Privacy-preserving data deduplication on trusted processors. In: Fox, G.C. (ed.) 10th International Conference on Cloud Computing, pp. 66–73. IEEE (2017). https://doi.org/10.1109/CLOUD.2017.18
8. Douceur, J.R., Adya, A., Bolosky, W.J., Simon, D., Theimer, M.: Reclaiming space from duplicate files in a serverless distributed file system. In: ICDCS 2002, pp. 617–624 (2002). https://doi.org/10.1109/ICDCS.2002.1022312

9. Drago, I., Mellia, M., Munafò, M.M., Sperotto, A., Sadre, R., Pras, A.: Inside dropbox: understanding personal cloud storage services. In: Byers, J.W., Kurose, J., Mahajan, R., Snoeren, A.C. (eds.) IMC 2012, pp. 481–494. ACM (2012). https://doi.org/10.1145/2398776.2398827
10. Duan, Y.: Distributed key generation for encrypted deduplication: achieving the strongest privacy. In: Ahn, G., Oprea, A., Safavi-Naini, R. (eds.) CCSW 2014, pp. 57–68. ACM (2014). https://doi.org/10.1145/2664168.2664169
11. Gjøsteen, K.: Security notions for disk encryption. In: di Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 455–474. Springer, Heidelberg (2005). https://doi.org/10.1007/11555827_26
12. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) CCS 2011, pp. 491–500. ACM (2011). https://doi.org/10.1145/2046707.2046765
13. Harnik, D., Pinkas, B., Shulman-Peleg, A.: Side channels in cloud services: deduplication in cloud storage. IEEE Secur. Priv. 2010 **8**(6), 40–47 (2010). https://doi.org/10.1145/2046707.2046765
14. Juels, A., Kaliski, Jr., B.S.K.: PORs: proofs of retrievability for large files. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) CCS 2007, pp. 584–597. ACM (2007). https://doi.org/10.1145/1315245.1315317
15. Keelveedhi, S., Bellare, M., Ristenpart, T.: DupLESS: server-aided encryption for deduplicated storage. In: King, S.T. (ed.) USENIX Security 2013, pp. 179–194. USENIX (2013). https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bellare
16. Kunz, T., Wolf, R.: OmniCloud - the secure and flexible use of cloud storage services. Technical report Fraunhofer Institute SIT (2014). https://www.omnicloud.sit.fraunhofer.de/download/omnicloud-whitepaper-en.pdf
17. Li, M., Qin, C., Li, J., Lee, P.P.C.: CDStore: toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. IEEE Internet Comput. **20**(3), 45–53 (2016). https://doi.org/10.1109/MIC.2016.45
18. Liu, J., Asokan, N., Pinkas, B.: Secure deduplication of encrypted data without additional independent servers. In: CCS 2015, pp. 874–885 (2015). https://doi.org/10.1145/2810103.2813623
19. Liu, J., Asokan, N., Pinkas, B.: Secure deduplication of encrypted data without additional independent servers. In: IACR Cryptology ePrint Archive 2015/455 (2015). http://eprint.iacr.org/2015/455
20. Liu, J., Duan, L., Li, Y., Asokan, N.: Secure deduplication of encrypted data: refined model and new constructions. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 374–393. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_20
21. Messmer, S., Rill, J., Achenbach, D., Müller-Quade, J.: A novel cryptographic framework for cloud file systems and CryFS, a provably-secure construction. In: Livraga, G., Zhu, S. (eds.) DBSec 2017. LNCS, vol. 10359, pp. 409–429. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61176-1_23
22. Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M., Weippl, E.R.: Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In: USENIX Security 2011. USENIX (2011)
23. Puzio, P., Molva, R., Önen, M., Loureiro, S.: Cloudedup: secure deduplication with encrypted data for cloud storage. In: CloudCom 2013, pp. 363–370. IEEE (2013). https://doi.org/10.1109/CloudCom.2013.54

24. Ritzdorf, H., Karame, G., Soriente, C., Capkun, S.: On information leakage in deduplicated storage systems. In: Weippl, E.R., Katzenbeisser, S., Payer, M., Mangard, S., Androulaki, E., Reiter, M.K. (eds.) CCSW 2016, pp. 61–72. ACM (2016). https://doi.org/10.1145/2996429.2996432
25. Shin, Y., Koo, D., Yun, J., Hur, J.: Decentralized server-aided encryption for secure deduplication in cloud storage. In: IEEE Transactions on Services Computing (2017). https://doi.org/10.1109/TSC.2017.2748594
26. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Security and Privacy 2000, pp. 44–55. IEEE (2000). https://doi.org/10.1109/SECPRI.2000.848445
27. Stanek, J., Sorniotti, A., Androulaki, E., Kencl, L.: A secure data deduplication scheme for cloud storage. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 99–118. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_8
28. Storer, M.W., Greenan, K.M., Long, D.D.E., Miller, E.L.: Secure data deduplication. In: Kim, Y., Yurcik, W. (eds.) StorageSS 2008, pp. 1–10. ACM (2008). https://doi.org/10.1145/1456469.1456471
29. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Trans. Parallel Distrib. Syst. **24**(9), 1717–1726 (2013). https://doi.org/10.1109/TPDS.2012.278