Security notions for cloud storage and deduplication

Colin Boyd, Gareth T. Davies and Kristian Gjøsteen
NTNU, Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,gareth.davies,kristian.gjosteen}@ntnu.no

Mohsen Toorani
University of Bergen, Bergen, Norway
mohsen.toorani@uib.no

Håvard Raddum
Simula@UiB
haavardr@simula.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, but it is not obvious what precise security properties are required for 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 provide a generic syntax for storage systems that allows us to formally model natural security notions for cloud storage and deduplication. 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 generic cryptographic components.

Contents

1 Introduction

In recent years cloud computing has become a dominant paradigm for providing resources and services over the Internet. Cloud storage is an important and extensively used aspect, providing attractive services for individuals and enterprises. Dropbox, one of the most popular cloud storage providers [DMM+12], had over 500 million users by 2016[1].

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 storage provider to secure their data, but lessons from the Snowden revelations show that even well-meaning providers are not immune from compromise. Users increasingly want to manage confidentiality and integrity of their own outsourced data without the need to trust the cloud storage provider. Note that in our context, the term *users* may refer equally to individuals and to businesses accessing cloud storage servers through a gateway.

Today it is widely accepted that the only reliable way to achieve high security of a system is to provide a formal model of security and analyze the system design within that model. Security and cryptography researchers have indeed developed such models for a number of security properties relevant to the cloud environment. Examples include proofs of retrievability for stored data [JJ07, SW13], proofs of ownership for shared user data [HHPS11], message-locked encryption (MLE) [BKR13] and models for securing stored data in such a way that it can be computed with [CS15]. We follow the approaches taken to define security for primitives such as symmetric encryption [BDJR97, KY06], authenticated encryption (AE) [BN00, Rog02] and disk encryption [Gjø05].

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 the security requirements? Perhaps it may be thought obvious that users 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 storage provider. Moreover, efficiency factors may conflict with user goals. Specifically, cloud storage providers (CSPs) extensively use *deduplication* for removing redundant copies of data and saving storage.

The overall goal of this paper is to understand how users can achieve basic security properties in remote storage systems. We define a generic abstract model for cloud storage and a generic security model which can be instantiated in various ways depending on the security goal and the adversary's capabilities. We focus on symmetric (authenticated) encryption as the main mechanism, but we believe that our model is appropriate for extension to other cryptographic primitives, particularly public key encryption.

Security Goals for Cloud Storage. Users who trust their storage provider can send plaintext data for storage. Although this is today the most common situation, there are several commercial storage providers supporting client-side encryption so that the cloud provider cannot obtain the plaintext [WA14]. It is not immediately obvious what is the correct security property that should be met by 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 generic model.

IND This is the usual standard for strong security of encryption, *indistinguishability* of ciphertexts. We will show that this can be achieved in the 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) was considered by Bellare et al. [BKR13] 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 generic model.

[1] `https://blogs.dropbox.com/dropbox/2016/03/500-million/`

INT Integrity of stored data is essential in cloud storage systems: in many scenarios a 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 notions of tag consistency to our framework.

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 generic definition of cloud storage in terms of the input/output behavior of the entities in the system, and for various functionalities we can use our generic framework to define game-based notions.

Previous Work. From the point of view of a single enterprise or an individual, secure outsourced storage is straightforward: simply encrypt all files at the client side using strong symmetric encryption, use a message authentication code (MAC) or AE for integrity and keep your key(s) secret. In this mindset, cloud storage appears similar to disk encryption [Gjø05, KMV17], and this is the approach recently taken by Messmer et al. [MRAM17]. However, 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 [DAB+02] was formalized by Bellare et al. [BKR13] there have been a number of proposals for secure deduplicating storage [KBR13, Dua14, SSAK14, LAP15a], 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) [LAP15a], as noted in a revision to the ePrint version [LAP15b] of their paper and also in subsequent work by some of the same authors [LDLA17], 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 [HPS10, ABD+17].

Specific functionalities such as proofs of retrievability (PoR) [JJ07, SW13], proofs of data possession (PDP) [ABC+07], proofs of ownership (PoW) [HHPS11] and secure auditing [YJ13, LLXC16] protocols have been discussed extensively in the literature. PoR schemes assure clients that the data is available on the cloud storage and can be entirely downloaded if needed. PDP schemes enable a client to verify that its stored data has not undergone any modifications. PoW protocols are executed between a client and a cloud storage server and ensures the server that the user who wants to outsource data really owns it. In secure auditing protocols there is an independent auditing service that interacts with the CSP on behalf of users to check data integrity. A number of works [Goh03, EFG06, HRAM13, CS15, AHMR15] have considered privacy of interactions (queries and results) between a data owner and a malicious server. Work on query privacy for outsourced storage is by now rather extensive [BM17], and it is arguable that the literature concerning formal models has indeed caught up with the security properties claimed in papers detailing protocols and implementations.

Contributions. 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 can help to identify which components of a storage scheme need to satisfy certain criteria for a given security goal. Specifically, in this paper 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.

The rest of this paper is organized as follows. After introducing some preliminaries and definitional choices in Section 2 we give a formal model for cloud storage systems in Section 3. Fine-grained security notions for confidentiality in cloud storage are introduced in Section 4. Cloud storage integrity, and the link between our framework and deduplication tags used in prior literature, is analyzed in Section 5.

2 Preliminaries

2.1 Notation

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 bistrings, by $a||b$.

Throughout this paper 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, the generation of such values is implicit. In situations where algorithms are run with no inputs given, these public values are still provided. In our security experiments, we consider an adversary that possibly interacts with some oracles before terminating and providing some value as output. We use the concrete security framework throughout: we make no definition of what it means for a scheme to be secure under a notion and instead formally define adversarial advantage and the security experiment for that notion. For this reason we do not regard adversaries in terms of some 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. We normalize adversarial advantage so that if some adversary's success probability is worse than guessing then this adversary can be modified to one that wins with probability greater than just by guessing. This removes the need to use absolute values in advantage statements. In our pseudocode for security games we will use **return** $b' \stackrel{?}{=} b$ as shorthand for **if** $b' = b$ **then return** 1 // **else return** 0, with an output of 1 indicating successful adversarial behavior. We will use an init procedure to represent initialization of cloud storage systems – this encompasses a large 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 the functionality). In our proofs, if an experiment's oracle is modified we denote this by $\mathcal{O}.\mathsf{func}'$.

2.2 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 symmetric encryption, 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. We will revisit these considerations – particularly when concerned with data integrity – later on.

We fix the syntax of a symmetric encryption scheme. Our results in Sections 4 and 5 consider authenticated encryption with the ability to handle some *associated data* (AEAD), but for now we consider SKE without associated data as a distinct primitive. A symmetric encryption scheme SKE over some message space and associated data space is defined by a tuple of algorithms $(\mathsf{KG}, \mathsf{E}, \mathsf{D})$. The algorithms operate as follows:

- KG: Key generation outputs a symmetric key k, denoted $k \leftarrow \mathsf{KG}$.
- E: Encryption takes as input a message m and key k and outputs a ciphertext c, we write this as $c \leftarrow \mathsf{E}_k(m)$.
- D: Decryption takes a ciphertext c and key k and outputs either a message m or error symbol $\perp$. We denote this by $m/\perp \leftarrow \mathsf{D}_k(c)$.

Correctness requires that $\mathsf{D}_k(c) = m$ for all messages m, all $c \leftarrow \mathsf{E}_k(m)$ and all $k \leftarrow \mathsf{KG}$. Key generation is a randomized algorithm, the encryption algorithm may be randomized or stateful, and decryption is deterministic.

Note that many authors use the notation $\mathsf{E}(k, m)$ to denote that message m is encrypted under key k; we will use the subscript notation throughout.

2.2.1 Confidentiality for Symmetric Encryption

We now define multi-challenge, single-key, left-or-right indistinguishability for symmetric encryption. In this game, an adversary selects a sequence of pairs of equal-length messages[2]. A left-or-right encryption oracle $\mathcal{O}.\mathsf{LR}_b(\cdot,\cdot)$ encrypts one of the messages in each pair according to the bit b: we write this oracle with a subscript b to emphasize that it has the challenge bit hardwired. We also include non-triviality conditions on messages that the adversary provides to $\mathcal{O}.\mathsf{LR}$: we will often omit these conditions in the rest of the paper for clarity.

Note that an adversary can of course send $m_0 = m_1$ to its $\mathcal{O}.\mathsf{LR}$ oracle to simulate an encryption oracle E: many authors make such an oracle explicit because in many scenarios a tight reduction is desired in terms of the number of $\mathcal{O}.\mathsf{LR}$ queries the adversary makes. We do not require this distinction and so we only consider a single $\mathcal{O}.\mathsf{LR}$ oracle.

We define two variants of left-or-right indistinguishability: one for a chosen plaintext attack (IND-CPA), and one for a chosen ciphertext attack (IND-CCA2) where the adversary is allowed to decrypt any chosen ciphertexts (with the obvious exception of decrypting the *challenge ciphertexts* that have been output by $\mathcal{O}.\mathsf{LR}$).

Definition 1 (IND-CPA and IND-CCA2 Security for Symmetric Encryption)**.** *Let* $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *be a symmetric encryption scheme. Then the* IND-atk *advantage for an adversary* $\mathcal{A}$ *against* SKE*, for* $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA2}\}$*, is defined by*

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

where experiment $\mathbf{Exp}^{\mathsf{IND\text{-}atk}}_{\mathsf{SKE},\mathcal{A}}$ *is given in Fig. 1.*

$\mathbf{Exp}^{\mathsf{IND\text{-}atk}}_{\mathsf{SKE},\mathcal{A}}$:
$\mathsf{CL} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$k \leftarrow \mathsf{KG}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{LR}_b,(\mathcal{O}.\mathsf{D})}$
return $b' \stackrel{?}{=} b$

$\mathcal{O}.\mathsf{LR}_b(m_0, m_1)$:
if $|m_0| \neq |m_1|$ **then**
return $\perp$
$c \leftarrow \mathsf{E}_k(m_b)$
$\mathsf{CL} \xleftarrow{\cup} \{c\}$
return c

$\mathcal{O}.\mathsf{D}(c)$:
if $c \in \mathsf{CL}$ **then**
return $\perp$
$m \leftarrow \mathsf{D}_k(c)$
return m

Figure 1: *The experiments defining* IND-CPA *and* IND-CCA2 *security for symmetric encryption. For* IND-CPA*, the adversary only has access to oracle* $\mathcal{O}.\mathsf{LR}_b$*, while for* IND-CCA2 *the adversary additionally has access to a decryption oracle* $\mathcal{O}.\mathsf{D}$*.*

2.2.2 Confidentiality for MLE-like Symmetric Encryption

One method of enabling data deduplication while allowing encryption is to enforce that users with the same file arrive at the same ciphertext. This was the idea behind the *convergent encryption* (CE)

[2] Equality in message length is not necessarily obvious in the cloud storage scenario. Many authors simply assume that files are bitstrings (of finite length) and thus security experiments and results are straightforward. However, this ignores the processing and segmentation that will be done by the user's device before any upload occurs.

protocol of Douceur et al. [DAB+02], designed for enterprise backup storage. In this scenario, to encrypt file F the user computes $k \leftarrow \mathsf{H}(\mathsf{F})$ for some hash function H and encrypts F under k using some deterministic SKE scheme. This inherently enables perfect data deduplication, since there is only one possible ciphertext for each plaintext file.

In schemes where the encryption key depends solely on the plaintext[3], it is not possible to achieve the standard notions of indistinguishability mentioned previously. To see this for CE: adversary chooses distinct m_0, m_1, computes the key and thus the ciphertext for each value, sends just a single query to $\mathcal{O}.\mathsf{LR}$ and compares. In Fig. 2 we detail the PRV-CDA security notion of Bellare et al. [BKR13], which captures the intuition that such a scheme should provide confidentiality if the plaintext values are unpredicatble. The definition is parameterized by a *message source* $\mathtt{MS}$, an efficient algorithm that outputs two (equal-sized) vectors of messages from some message space along with some auxiliary information: $(\mathbf{m}_0, \mathbf{m}_1, Z) \leftarrow \mathtt{MS}$. In the challenge phase the challenger runs $\mathtt{MS}$, then encrypts the output plaintext messages and returns the resulting ciphertexts and Z to the adversary. Bellare et al. also considered a real-or-random version of this definition, denoted PRV$-CDA, where the source only outputs one message vector, and the adversary must distinguish an encryption of that from a vector of random strings, but we will not need this notion. Note that unlike the other definitions in this section, we do not include associated data. For ordinary encryption schemes we would use associated data to achieve integrity, but for MLE-like schemes a specific integrity mechanism is often provided by means of a tag, which for example could be the hash of the ciphertext.

Definition 2 (PRV-CDA Security for Symmetric Encryption)**.** *Let* $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *be a symmetric encryption scheme. Then the* PRV-CDA *advantage for an adversary* $\mathcal{A}$ *against* SKE *is defined by*

$$\mathbf{Adv}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\,\mathtt{MS},\,\mathcal{A}} = 2 \cdot \left[\mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\,\mathtt{MS},\,\mathcal{A}} = 1\right] - \frac{1}{2}\right]$$

where experiment $\mathbf{Exp}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\,\mathtt{MS},\,\mathcal{A}}$ *is given in Fig. 2.*

$\underline{\mathbf{Exp}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\,\mathtt{MS},\,\mathcal{A}}:}$
$b \xleftarrow{\$} \{0,1\}$
$(\mathbf{m}_0, \mathbf{m}_1, Z) \leftarrow \mathtt{MS}$
for $i = 1, \ldots, |\mathbf{m}_b|$ **do**
  $k \leftarrow \mathsf{KG}(\mathbf{m}_b[i])$
  $\mathbf{c}[i] \leftarrow \mathsf{E}_k(\mathbf{m}_b[i])$
$b' \leftarrow \mathcal{A}(\mathbf{c}, Z)$
return $b' \stackrel{?}{=} b$

Figure 2: *The experiment defining* PRV-CDA *security for symmetric encryption.*

2.2.3 Authenticated Encryption with Associated Data (AEAD)

We define authenticated encryption (AE) that can handle associated data as a distinct primitive from SKE. A secure AE scheme will simultaneously provide confidentiality and integrity. The algorithms of $\mathsf{AE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ are specified as follows for completeness:

- KG: Key generation outputs a symmetric key k, denoted $k \leftarrow \mathsf{KG}$.
- E: Encryption takes as input a message m, some associated data $\mathtt{AD}$ and key k and outputs a ciphertext c, written $c \leftarrow \mathsf{E}_k(m, \mathtt{AD})$.

[3] Note that in the literature referenced in this subsection, authors use the term messages rather than files, since files are subject to segmentation and other processes: their messages refer to the inputs to their storage algorithms and protocols. We attempt to avoid ambiguity by referring to files where possible in this paper.

- D: Decryption takes a ciphertext c, some associated data $\mathtt{AD}$ and key k and outputs either a message m or error symbol $\perp$. We write this $m/\perp \leftarrow \mathsf{D}_k(c, \mathtt{AD})$.

Correctness requires that $\mathsf{D}_k(c) = m$ for all messages m, associated data $\mathtt{AD}$, all $c \leftarrow \mathsf{E}_k(m)$ and all $k \leftarrow \mathsf{KG}$. Key generation is a randomized algorithm, the encryption algorithm may be randomized or stateful, and decryption is deterministic.

In many definitions of AEAD security, including the original formulation by Rogaway [Rog02], the encryption oracle either works correctly (in the real game) or returns a random string. We consider generic ciphertext spaces in the following definition, and follow e.g. Paterson et al. [PRS11, BPS15, JSSW17] in giving a left-or-right variant of the all-in-one notion of Rogaway and Shrimpton [RS06]. Any scheme secure under Rogaway and Shrimpton's notion will be secure in the definition below for message space $\mathcal{M} = \{0,1\}^*$.

Definition 3. *Let* $\mathsf{AE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *be a symmetric encryption scheme with some message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*. Then the* AEAD *advantage for an adversary* $\mathcal{A}$ *against* AE *is*

$$\mathbf{Adv}^{\mathsf{AEAD}}_{\mathsf{AE},\mathcal{A}} = 2 \cdot \left[\mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{AEAD}}_{\mathsf{AE},\mathcal{A}} = 1\right] - \frac{1}{2} \right]$$

where experiment $\mathbf{Exp}^{\mathsf{AEAD}}_{\mathsf{AE},\mathcal{A}}$ *is given in Fig. 3.*

$\underline{\mathbf{Exp}^{\mathsf{AEAD}}_{\mathsf{AE},\mathcal{A}}}$:
$\mathsf{CL} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$k \leftarrow \mathsf{KG}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{LR}_b,\ \mathcal{O}.\mathsf{D}_b}$
return $b' \stackrel{?}{=} b$

$\mathcal{O}.\mathsf{LR}_b(m_1, m_0, \mathtt{AD})$:
$c_b \leftarrow \mathsf{E}_k(m_b, \mathtt{AD})$
$\mathsf{CL} \xleftarrow{\cup} \{(c_b, \mathtt{AD})\}$
return c_b

$\mathcal{O}.\mathsf{D}_b(c, \mathtt{AD})$:
if $(c, \mathtt{AD}) \in \mathsf{CL}$ **then**
return $\perp$
$m \leftarrow \mathsf{D}_k(c, \mathtt{AD})$
if $b = 1$ **then**
return m
if $b = 0$ **then**
return $\perp$

Figure 3: *The experiment defining* AEAD *security for symmetric encryption.*

Note that the pair $\{(c, \mathtt{AD})\}$ is added to the experiment's forbidden list CL rather than just c, so the adversary is allowed to ask its decryption oracle $\mathcal{O}.\mathsf{D}$ for some $\{(c, \mathtt{AD}')\}$ if $\mathtt{AD} \neq \mathtt{AD}'$.

2.2.4 Integrity for Symmetric Encryption

While AE schemes provide integrity by design, it is often useful to consider integrity for symmetric encryption in isolation. To this end we will define integrity in terms schemes that can handle associated data, but of course this input can be ignored to get notions for the setting in which AD is not used. There are three main notions of integrity for symmetric encryption: *non-malleability*, *plaintext integrity* and *ciphertext integrity* [BN00]. Non-malleability formalizes an adversary's inability to transform a given ciphertext into a different ciphertext so that their corresponding plaintexts are "meaningfully related". Plaintext integrity formalizes an adversary's inability to create a new ciphertext decrypting to a message that the sender had never encrypted. Ciphertext integrity formalizes an adversary's inability to create a ciphertext that decrypts correctly.

Ciphertext integrity implies plaintext integrity, but the two notions are not equivalent. The two integrity notions are usually independent of non-malleability: non-malleability usually implies confidentiality, while the two integrity notions do not. Plaintext integrity is often the intuitively correct notion

for an application, but usually ciphertext integrity is needed for proofs. This is why we only consider ciphertext integrity in this paper.

In the following definitions we need an SKE that can handle associated data, $\mathtt{AD}$. Formally, this simply means that $\mathtt{AD}$ is an input to both $\mathsf{E}_k()$ and $\mathsf{D}_k()$.

Definition 4. *Let* $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *be a symmetric encryption scheme. Then the* $\mathsf{INT\text{-}CTXT}$ *advantage for an adversary* $\mathcal{A}$ *against* SKE *is*

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

where experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SKE},\mathcal{A}}$ *is given in Fig. 4.*

$\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SKE},\mathcal{A}}$:
$b \leftarrow 0$
$\mathsf{CL} \leftarrow \emptyset$
$k \leftarrow \mathsf{KG}$
$\mathcal{A}^{\mathcal{O}.\mathsf{E},\mathcal{O}.\mathsf{D}}$
return b

$\mathcal{O}.\mathsf{D}(c, \mathtt{AD})$:
$m \leftarrow \mathsf{D}_k(c, \mathtt{AD})$
if $m \neq \perp \wedge (c, \mathtt{AD}) \notin \mathsf{CL}$ **then**
$b \leftarrow 1$
return m

$\mathcal{O}.\mathsf{E}(m, \mathtt{AD})$:
$c \leftarrow \mathsf{E}_k(m, \mathtt{AD})$
$\mathsf{CL} \xleftarrow{\cup} \{(c, \mathtt{AD})\}$
return c

Figure 4: *The experiment defining ciphertext integrity (*$\mathsf{INT\text{-}CTXT}$*) for symmetric encryption.*

Just like in the AEAD experiment, if $\{(c, \mathtt{AD})\} \in \mathsf{CL}$ then the adversary is allowed to ask its decryption oracle $\mathcal{O}.\mathsf{D}$ for some $\{(c, \mathtt{AD}')\}$ if $\mathtt{AD}' \neq \mathtt{AD}$.

3 Modelling Cloud Storage

While our main goal is to study security of cloud storage in terms of confidentiality and integrity of files, such analysis is only possible in the presence of a generic and meaningful underlying model. In this section we present 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 academic literature (in particular schemes facilitating encrypted data deduplication).

The 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 [KBR13] or an auditing mechanism [YJ13, LLXC16]. 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. a laptop and a phone, of one user.

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 machine for each file. In practice all a user sees is a list of filenames – which are certainly not suitable for our purposes – so this approach will allow 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 [DAB+02, BKR13, ABM+13, BK15]. In client-side-deduplicating systems the user first sends some short, message-derived tag (for example in convergent encryption [DAB+02] 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[4]. In this context, this tag is all that is required to claim ownership of a file. Our handles do not have this feature: they are to ensure that retrieve queries work 'correctly': see Def. 5 later on in this section. In Section 5 we will discuss integrity in the context of deduplicating and non-deduplicating cloud storage, and there we will discuss the differences between our handles and these tags in more detail.

[4] This process also occurs in deduplicating schemes that do not use any encryption.

3.1 A Model for Cloud Storage

To model cloud storage it is necessary to make some concrete assumptions about our schemes. Each user has some (preferably small) local storage and the server maintains (what is from an abstract perspective at least) a vast data structure. 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. Define a cloud storage scheme $\mathsf{CS}[\mathsf{SKE}, \mathsf{fkeyGen}] = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$, parameterized by a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ and a file-key generation procedure fkeyGen, as depicted in Fig. 5. Generation of per-user key material uk (line 2) 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 (see below) intuitive schemes and all protocols from the literature.

```
init
1. ST ← ∅

newu(uid)
2. uk_uid ← kgen
3. KT_uid ← ∅
4. return uk_uid

del(uid, id)
5. KT_uid ←∪ {(⊥, id)}
6. if ∃ {uid, ·, id, 0} in ST then
7.     ST ←∪ {(uid, −, id, 1)}

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

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

retr(uid, id)
13. if ∃ (uid, ·, id, 1) ∈ ST  or
14.         ∃(⊥, id) ∈ KT_uid then
15.     return ⊥
16. if ∃ (uid, c, id, 0) ∈ ST  and
17.         ∃(fk, id) ∈ KT_uid then
18.     F ← D_fk(c, id)
19.     return F
20. else
21.     return ⊥
```

Figure 5: *Definition of a cloud storage scheme.*

There are choices here that require further attention. In line 2 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 7 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[5] for that user uid and that handle id. The encryption algorithm E takes id as input (line 10): if SKE is an AEAD scheme then id could take the role of the associated data, and in fact we model this construction in Theorem 2. Lines 16-19 specify that the file can only be retrieved if it has not been removed either by the client or the server, and 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.

[5] 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 [BKR13]	$\mathsf{fk} \leftarrow \mathsf{H}(\mathsf{F})$
5	Liu et al. [LAP15a]	$\mathsf{fk} \leftarrow \mathbf{PAKE.Out}(\mathsf{F})$
6	DupLESS [KBR13]	$\mathsf{fk} \leftarrow \mathbf{OPRF.Out}(\mathsf{F})$
7	Duan [Dua14]	$\mathsf{fk} \leftarrow \mathbf{DOKG}(\mathsf{F})$
8	Stanek et al. [SSAK14]	$\mathsf{fk} \leftarrow \mathbf{Thr.PKE.KG}(\mathsf{F})$
9	CDStore [LQLL16]	$\mathsf{fk} \leftarrow \mathbf{SS}(\mathsf{H}(\mathsf{F}))$

Figure 6: *Specification for* fkeyGen *procedure for existing cloud storage protocols*

3.2 Modelling Existing Schemes and Literature

In Fig. 6 we detail the file-key generation procedure for natural constructions and a number of schemes from the existing literature. The CSP could of course not support client-side encryption (line 1): this is by far the most widely used of the protocols listed. Each user could hold their own encryption key (line 2) and encrypt all files with that key: this 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 per-user 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 outsourced storage provider. If a key is randomly chosen at the point of the file being uploaded (line 3) then this intuitively provides increased confidentiality, but introduces challenging key management. A gateway can also be employed in the per-file key case as described in the Omnicloud architecture [KW14]: this of course requires the gateway to additionally manage the vast number of keys that could be generated in the enterprise scenario.

The 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 [HPS10, ABD+17]. In many schemes such as those of Keelveedhi et al. (DupLESS) [KBR13] and Liu et al. [LAP15a], the 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. [SSAK14] use both convergent encryption and an outer layer threshold encryption scheme to produce ciphertexts, and the fkeyGen protocol interacts with two trusted third parties. The protocol of Duan [Dua14] attempts to avoid the single point of failure inherent in having a single (semi-trusted) key server (KS) in DupLESS-like schemes: fkeyGen generates encryption keys using a distributed oblivious key generation (DOKG), instantiated using a (deterministic) threshold signature scheme. The CDStore protocol of Li et al. [LQLL16] distributes shares of a file to multiple cloud servers using so-called *convergent dispersal* where the randomness in the process is replaced by a file-dependent key.

The restriction to SKE in line 10 of Fig. 5 is for the purposes of results in Sections 4 and 5. Note here that entries 1-7 in Fig. 6 precisely fit our model while 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 fk could happen as part of the key generation procedure kgen– i.e. fkeyGen is the identity function – this is the per-user key case in line 2 of Fig. 6.

Cloudedup [PMÖL13] employs block-level convergent encryption to send ciphertexts to a third party (or trusted server) that adds further (symmetric) encryption and manages the metadata. Dang and Chang[DC17] similarly assume some trusted entity, in their case hardware. A trusted enclave uses an oblivious PRF in a similar manner to the key-server in DupLESS to get block-derived keys to allow the enclave to perform deduplication: the enclave acts as a deduplication gateway then uses randomized encryption before sending the ciphertexts to the CSP, in order to hide from the server when two or more

users own the same file. Since in these schemes the encryption is essentially done in two phases and the per-block keys are managed by the third party, their approach does not quite fit our model but it is certainly straightforward to modify how KT and SKE work to incorporate such schemes. Recently Shin et al. [SKYH17] presented a scheme that attempts to distribute the role of the key server in DupLESS-like schemes by incorporating an additional inter-KS deduplication procedure. Again it is straightforward to modify our model to allow this type of scheme.

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 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 regarding id use.

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

As mentioned earlier and discussed in more detail later, 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. Our definition of correctness is unsurprising: once used, the retrieval handles are explicitly linked to the uploaded file.

Definition 5 (Correctness)**.** *If user* uid *previously stored* F *under handle* id *then when it later sends* id *to* retr *the reply will be* F*, unless client has sent* del(id).

Here we require an implicit assumption that the server forwards all requests honestly: this approach reflects cryptographic models for key exchange. Also note that 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 later on cannot modify KT so key symmetry is implicit in our model and for the rest of the paper.

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 chapter we turn our attention to confidentiality of files with respect to realistic adversaries. Defining confidentiality notions of security is a two-step process. We must first define what we want to prevent the adversary from learning, the *goal*. Next, we must 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 cloud storage provider.

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 uk for an arbitrary number of users. We model this by distinguishing between two types of newu query: $\mathcal{O}$.newuC creates a valid (Corrupt) user and outputs its uk to the adversary, and $\mathcal{O}$.newuH that only creates a valid (Honest) user[6]. For its

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

corrupted users the adversary may not necessarily use uk and fkeyGen correctly (which $\mathcal{O}$.store cannot handle): we model this capability by giving the adversary access to an $\mathcal{O}$.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*[7] *(*CRA*)*. Analagously to encryption, CSA and CRA correspond to CPA and CCA, respectively[8].

The adversary's control of the cloud storage provider can be usefully divided into three levels. Clearly, the adversary may have no influence at all with the cloud storage provider, in which case we have an honest storage provider giving the adversary *zero access (*Z*)*. The adversary may also be able to look at the cloud storage provider's storage and key material, but not tamper with anything, a *passively corrupt (*P*)* storage provider. 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 storage provider, an *actively corrupt (*A*)* adversary. When the storage provider 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. Roughly speaking, we can say that when the storage provider is honest, we consider only the security of the file key generation protocol. When the storage provider is passively corrupt, we must additionally consider the confidentiality of the encryption used. And when the storage provider is actively corrupt, we shall see that we must also consider the integrity of the encryption mechanism.

4.1 Indistinguishability for Cloud Storage Schemes

In combination we have a generic IND-atk-csp experiment with six distinct cases: $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ and this IND-atk-csp experiment is detailed in Fig. 7. Just as in our general definition for storage protocols (Fig. 5) we keep track of the retrieval capability by using a table ST, initially set to empty. 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 over 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 are not included in Fig. 5. $\mathcal{O}$.peek allows the adversary to see the ciphertext (and deletion status) for some user uid and some storage handle id, and this is available to an adversary that is passively corrupt (P). $\mathcal{O}$.insert and $\mathcal{O}$.erase model actively malicious (or completely compromised) CSPs, granting the adversary 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 6 (IND-atk-csp Security for Cloud Storage)**.** *Let* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *be a cloud storage scheme. Then the* IND-atk-csp *advantage for an adversary* $\mathcal{A}$ *and* $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ *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. 7.*

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 the key material for, resulting in ciphertexts that it

[7] Note that other authors including Kamara and Katz [KK08] use CRA as an acronym for other adversarial capabilities. [8] 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 $\mathcal{O}$.retr on these ciphertexts due to CL).

$\mathbf{Exp}^{\text{IND-atk-csp}}_{\text{CS},\,\mathcal{A}}$:

init
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{CL}, \mathsf{users_C}, \mathsf{users_H} \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}^{\text{oracles}}$
return $b' \stackrel{?}{=} b$

$\mathcal{O}.\mathsf{newuC}(\mathsf{uid})$:
$\mathsf{users_C} \xleftarrow{\cup} \mathsf{uid}$
$\mathtt{uk}_{\mathsf{uid}} \leftarrow \mathsf{kgen}$
$\mathsf{KT}_{\mathsf{uid}} \leftarrow \emptyset$
return $\mathtt{uk}_{\mathsf{uid}}$

$\mathcal{O}.\mathsf{newuH}(\mathsf{uid})$:
$\mathsf{users_H} \xleftarrow{\cup} \mathsf{uid}$
$\mathtt{uk}_{\mathsf{uid}} \leftarrow \mathsf{kgen}$
$\mathsf{KT}_{\mathsf{uid}} \leftarrow \emptyset$
return $\perp$

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

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

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

$\mathcal{O}.\mathsf{LR}_b(\mathsf{uid}, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$:
if $\mathsf{uid} \notin \mathsf{users_H}$ **then**
return $\perp$
else
$\mathcal{O}.\mathsf{store}(\mathsf{uid}, \mathsf{F}_b, \mathsf{id})$
$\mathsf{CL} \xleftarrow{\cup} \{(\mathsf{uid}, \mathsf{id})\}$

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

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

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

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

Figure 7: *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 the oracles on the left-hand side and* $\mathcal{O}.\mathsf{LR}_b$. CRA *additionally has access to* $\mathcal{O}.\mathsf{retr}$, P *additionally has the* $\mathcal{O}.\mathsf{peek}$ *oracle and finally* A *additionally has the* $\mathcal{O}.\mathsf{erase}$ *and* $\mathcal{O}.\mathsf{insert}$ *oracles.*

cannot access. Consequently, even a scheme with no encryption can be secure under this notion. This is by design: the only schemes that do not meet this minimum requirement are those that leak information about the file *to other users* during the storage 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[9].

[9] In deduplicating schemes the segmentation procedure can be a side channel in itself, see Ritzdorf et al. [RKSC16], so 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 $\mathcal{O}.\mathsf{LR}$.

Relations Between Notions. While we have just defined six adversarial capabilities, in fact only three are distinct. Fig. 8 summarizes how the notions relate to each other, and we detail these relations fully in Appendix A. 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 atk $\in \{\mathsf{CSA}, \mathsf{CRA}\}$, and also IND-CRA-csp $\Rightarrow$ IND-CSA-csp for 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 always possible to simulate the $\mathcal{O}$.upl queries of an IND-CRA-Z adversary: this adversary can only use $\mathcal{O}$.store to place items in ST that it can later retrieve (since $\mathcal{O}$.LR and $\mathcal{O}$.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 $\mathcal{O}$.insert and $\mathcal{O}$.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 $\mathcal{O}$.LR query with distinct files and then queries $\mathcal{O}$.peek on that entry. We give the final separation result after Theorem 1. We will henceforth refer to these three distinct notions using IND-atk-Z, IND-CSA-P and IND-CRA-A.

$$\begin{matrix}\text{IND-CSA-Z}\\ \text{IND-CRA-Z}\end{matrix} \;\not\Leftarrow\Rightarrow\; \begin{matrix}\text{IND-CSA-P}\\ \text{IND-CRA-P}\\ \text{IND-CSA-A}\end{matrix} \;\not\Leftarrow\Rightarrow\; \text{IND-CRA-A}$$

Figure 8: *Relations between* IND *notions for confidentiality of cloud storage systems.*

4.2 Building Secure Cloud Storage Schemes

We now 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. In Theorem 1 we show that if users encrypt using an IND-CPA-secure encryption scheme using their own key then the overall system is IND-CSA-P secure (and thus also secure in the sense of IND-CRA-P and IND-CSA-A). By this we mean that the key generation procedure for CS outputs a random key to each user (or in other words that kgen runs the SKE's KG algorithm) and fkeyGen(F, uk) just outputs uk for all F. In Theorem 2 we show that this same construction, when implemented with an AEAD scheme, yields an IND-CRA-A-secure cloud storage system, our strongest notion. These theorems provide justification for the choices made in defining our adversarial capabilities. Theorem 3 considers the scenario in which a random symmetric key is created for each file, and perhaps surprisingly this construction meets the strongest IND-CRA-A notion of security even with IND-CPA-secure encryption. Finally Theorem 4 considers the secure deduplication setting with a reduction to PRV-CDA security of the underlying encryption: for this we need a modified security experiment (Fig. 13). For the results here and in the next section, if we are describing key usage for a specific symmetric-key encryption scheme (i.e. for a defined fkeyGen) then we use a subscript SKE_i, and for generic results (i.e. result holds for any scheme) we omit the suffix and use SKE. We summarize the results from this section in Fig. 9.

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

Figure 9: *Summary of the results in this section.*

4.2.1 Per-User Keys

Theorem 1. *Let* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *be a cloud storage protocol that uses a symmetric key encryption scheme* $\mathsf{SKE}_1 = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *as in Fig. 5, with per-user keys and at most* n *honest users. For any adversary* $\mathcal{A}_1$ *against* IND-CSA-P *there exists an adversary* $\mathcal{B}_1$ *such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathcal{A}_1} \leq n \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}_1,\,\mathcal{B}_1}$$

Proof. We define $n+1$ hybrid games $\mathsf{G}_0, \mathsf{G}_1, \ldots, \mathsf{G}_n$ corresponding to each of the honest users. The games all work like $\mathbf{Exp}^{\mathsf{IND\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathcal{A}}$, except in the behaviour of their $\mathcal{O}.\mathsf{LR}_b$ and $\mathcal{O}.\mathsf{store}$ oracles. Since the reduction does not know user i's per-user key $\mathtt{uk}$, it needs to call its own $\mathcal{O}_{\mathsf{SKE}}.\mathsf{LR}_b$ oracle when the adversary calls $\mathcal{O}.\mathsf{LR}_b$ or $\mathcal{O}.\mathsf{store}$. In the ith game, the $\mathcal{O}.\mathsf{LR}_b$ oracle will always use F_1 for the first i users, and always use F_0 for the remaining users. In G_0, the $\mathcal{O}.\mathsf{LR}_b$ oracle will always behave as if $b = 0$, while in G_n it will always behave as if $b = 1$. Define $P_i = \mathbf{Pr}\,[b' = 0 \text{ in } \mathsf{G}_i]$. We get that

$$\mathbf{Adv}^{\mathsf{IND\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathcal{A}_1} = |P_0 - P_n| \leq \sum_{i=1}^{n} |P_{i-1} - P_i|.$$

In other words, we have a bound on the adversary's advantage in terms of the difference of behaviour of consecutive games. We must now bound this difference.

$\mathcal{B}_{1,i}$ playing $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}_1}$:
- $\mathsf{ST} \leftarrow \emptyset$
- $b' \leftarrow \mathcal{A}_1^{\text{oracles}}$
- **return** b'

$\mathcal{O}.\mathsf{store}'(\mathsf{uid}, \mathsf{F}, \mathsf{id})$:
- **if** $\mathsf{uid} = \mathsf{uid}_i$ **then**
  - $c \leftarrow \underline{\mathcal{O}_{\mathsf{SKE}}.\mathsf{LR}}(\mathsf{F}, \mathsf{F})$
- **else**
  - $c \leftarrow \mathsf{E}_{\mathtt{uk}_{\mathsf{uid}}}(\mathsf{F})$
- $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$
- $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathtt{uk}_{\mathsf{uid}}, \mathsf{id})\}$

$\mathcal{O}.\mathsf{LR}'_b(\mathsf{uid}, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$:
- **if** $\mathsf{uid} \notin \mathsf{users}_{\mathsf{H}}$ **then**
  - **return** $\perp$
- **if** $\mathsf{uid} = \mathsf{uid}_j, j < i$ **then**
  - **do** $\mathsf{store}(\mathsf{uid}, \mathsf{F}_1, \mathsf{id})$
- **else if** $\mathsf{uid} = \mathsf{uid}_j, j > i$ **then**
  - **do** $\mathsf{store}(\mathsf{uid}, \mathsf{F}_0, \mathsf{id})$
- **else**
  - $c \leftarrow \underline{\mathcal{O}_{\mathsf{SKE}}.\mathsf{LR}}(\mathsf{F}_0, \mathsf{F}_1)$
  - $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$
  - $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathtt{uk}_{\mathsf{uid}}, \mathsf{id})\}$

Figure 10: *Reduction* $\mathcal{B}_{1,i}$ *for proof of Theorem 1, where adversary* $\mathcal{A}_1$*, playing* $\mathbf{Exp}^{\mathsf{IND\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathcal{A}_1}$ *has access to oracles* $= \{\mathcal{O}.\mathsf{newuC}, \mathcal{O}.\mathsf{newuH}, \mathcal{O}.\mathsf{store}', \mathcal{O}.\mathsf{upl}, \mathcal{O}.\mathsf{del}, \mathcal{O}.\mathsf{LR}'_b, \mathcal{O}.\mathsf{peek}\}$*. For queries to the non-specified oracles, the reduction performs the required operations exactly as defined in Fig. 7.*

We use the reduction $\mathcal{B}_{1,i}$ against IND-CPA defined in Fig. 10. Our reduction simulates a left-right oracle for $\mathcal{A}_1$ by always using F_1 for the first $i-1$ users, always using F_0 for the last $n-i-1$ users and using its own left-right oracle to encrypt F_0 or F_1 for the ith user. By inspection, if the reduction's left-right oracle always encrypts F_0, the execution proceeds exactly as in G_{i-1}. If the left-right oracle always encrypts F_1, the execution proceeds exactly as in G_i. It follows that

$$|P_{i-1} - P_i| = \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}_1,\,\mathcal{B}_{1,i}}.$$

We have therefore constructed $n+1$ IND-CPA adversaries. Letting $\mathcal{B}_1$ be one of these adversaries with maximal advantage, the claim follows. □

Remark 1. *The scheme described for this theorem is not* IND-CRA-A *secure, even if* SKE_1 *is* IND-CCA2 *secure. This is due to a* cut-and-paste *attack: an adversary can manipulate* ST *in such a way that it can call* $\mathcal{O}.\mathsf{retr}$ *on something it had previously added via an* $\mathcal{O}.\mathsf{LR}$ *query. See Appendix A for full details of this attack.*

We now show that if the underlying SKE scheme is secure in the sense of AEAD, then a construction using this scheme with id as the associated data yields an IND-CRA-A cloud storage protocol CS.

Theorem 2. *Let* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *be a cloud storage protocol that uses a symmetric key encryption scheme* $\mathsf{AE}_2 = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *as in Fig. 5, with per-user keys (with* $\mathtt{AD} = \mathsf{id}$*) and at most* n *honest users. For any adversary* $\mathcal{A}_2$ *against* IND-CRA-A *there exists an adversary* $\mathcal{B}_2$ *such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_2} \leq n \cdot \mathbf{Adv}^{\mathsf{AEAD}}_{\mathsf{AE}_2,\,\mathcal{B}_2}$$

Proof. The proof follows the same approach as that of Theorem 1 except we now need to deal with the adversary's $\mathcal{O}.\mathsf{retr}$, $\mathcal{O}.\mathsf{erase}$ and $\mathcal{O}.\mathsf{insert}$ queries. The latter two are straightforward: the reduction simply performs the actions according to Fig. 7. Whenever $\mathcal{A}_2$ makes an $\mathcal{O}.\mathsf{store}$ call for uid_i the reduction $\mathcal{B}_2$ must call its own $\mathcal{O}.\mathsf{LR}$ on the same two values: this adds the resulting ciphertext to its forbidden list. To deal with this, $\mathcal{B}_2$ needs to keep track of the $(\mathsf{F}, \mathsf{id}, c)$ tuples that it would be disallowed from decrypting in the event of an $\mathcal{O}.\mathsf{retr}$ query by $\mathcal{A}_2$, it does this using the list AEL. The reduction is detailed in Fig. 11.

$\mathcal{B}_{2,i}$ playing $\mathbf{Exp}^{\mathsf{AEAD}}_{\mathsf{AE}_2}$:
- $\mathsf{ST}, \mathsf{CL}, \mathsf{AEL} \leftarrow \emptyset$
- $b' \leftarrow \mathcal{A}_2^{\text{oracles}}$
- **return** b'

$\mathcal{O}.\mathsf{store}'(\mathsf{uid}, \mathsf{F}, \mathsf{id})$:
- **if** $\mathsf{uid} = \mathsf{uid}_i$ **then**
  - $c \leftarrow \underline{\mathcal{O}_{\mathsf{AE}}.\mathsf{LR}}(\mathsf{F}, \mathsf{F}, \mathsf{id})$
  - $\mathsf{AEL} \xleftarrow{\cup} \{(\mathsf{F}, c, \mathsf{id})\}$
- **else**
  - $c \leftarrow \mathsf{E}_{\mathtt{uk}_{\mathsf{uid}}}(\mathsf{F}, \mathsf{id})$
- $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$
- $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathtt{uk}_{\mathsf{uid}}, \mathsf{id})\}$

$\mathcal{O}.\mathsf{LR}'_b(\mathsf{uid}, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$:
- **if** $\mathsf{uid} \notin \mathsf{users}_{\mathsf{H}}$ **then**
  - **return** $\perp$
- $\mathsf{CL} \xleftarrow{\cup} \{(\mathsf{uid}, \mathsf{id})\}$
- **if** $\mathsf{uid} = \mathsf{uid}_j, j < i$ **then**
  - **do** $\mathsf{store}(\mathsf{uid}, \mathsf{F}_1, \mathsf{id})$
- **else if** $\mathsf{uid} = \mathsf{uid}_j, j > i$ **then**
  - **do** $\mathsf{store}(\mathsf{uid}, \mathsf{F}_0, \mathsf{id})$
- **else**
  - $c \leftarrow \underline{\mathcal{O}_{\mathsf{AE}}.\mathsf{LR}}(\mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$
  - $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$
  - $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathtt{uk}_{\mathsf{uid}}, \mathsf{id})\}$

$\mathcal{O}.\mathsf{retr}'(\mathsf{uid}, \mathsf{id})$:
- **if** $(\mathsf{uid}, \mathsf{id}) \in \mathsf{CL}$ **then**
  - **return** $\perp$
- **if** $\mathsf{uid} = \mathsf{uid}_j, j = i$ **then for** $\{\mathsf{uid}, c, \mathsf{id}, 0\} \in \mathsf{ST}$
  - **if** $\exists x : (x, c, \mathsf{id}) \in \mathsf{AEL}$ **then**
    - $\mathsf{F} \leftarrow x$
  - **else**
    - $\mathsf{F} \leftarrow \underline{\mathcal{O}_{\mathsf{AE}}.\mathsf{D}}(c, \mathsf{id})$
- **else**
  - **do** $\mathsf{F} \leftarrow \mathsf{retr}(\mathsf{uid}, \mathsf{id})$
- **return** F

Figure 11: *Reduction* $\mathcal{B}_{2,i}$ *for proof of Theorem 2, where adversary* $\mathcal{A}_2$*, playing* $\mathbf{Exp}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_2}$ *has access to oracles* $= \{\mathcal{O}.\mathsf{newuC}, \mathcal{O}.\mathsf{newuH}, \mathcal{O}.\mathsf{store}', \mathcal{O}.\mathsf{upl}, \mathcal{O}.\mathsf{del}, \mathcal{O}.\mathsf{LR}_b, \mathcal{O}.\mathsf{retr}', \mathcal{O}.\mathsf{peek}, \mathcal{O}.\mathsf{erase}, \mathcal{O}.\mathsf{insert}\}$*. For queries to the non-specified oracles, the reduction performs the required operations exactly as defined in Fig. 7.*

□

4.2.2 Per-File Keys

When the file encryption keys are generated randomly for each file (fkeyGen ignores uk and selects uniformly at random from the keyspace using KG), the adversary still has the ability to upload related files, which may help in distinguishing target files. Therefore we need to apply a reduction similar

$\mathcal{B}_{3,i}$ playing $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}_3}$:

$\mathsf{ST} \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}_3^{\text{oracles}}$
return b'

$\mathcal{O}.\mathsf{LR}'_b(\mathsf{uid}, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$:
if $\mathsf{uid} \notin \mathsf{users}_\mathsf{H}$ **then**
  return $\perp$
if # of queries $< i$ **then**
  do $\mathsf{store}(\mathsf{uid}, \mathsf{F}_1, \mathsf{id})$
else if # of queries $> i$ **then**
  do $\mathsf{store}(\mathsf{uid}, \mathsf{F}_0, \mathsf{id})$
else
  $c \leftarrow \underline{\mathcal{O}_{\mathsf{SKE}}.\mathsf{LR}}(\mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$
  $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, 0)\}$
  $\mathsf{KT}_{\mathsf{uid}} \xleftarrow{\cup} \{(\mathtt{uk}_{\mathsf{uid}}, \mathsf{id})\}$

Figure 12: *Reduction $\mathcal{B}_{3,i}$ for proof of Theorem 3 where adversary $\mathcal{A}_3$, playing $\mathbf{Exp}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_3}$ has access to oracles $= \{\mathcal{O}.\mathsf{newuC}, \mathcal{O}.\mathsf{newuH}, \mathcal{O}.\mathsf{store}, \mathcal{O}.\mathsf{upl}, \mathcal{O}.\mathsf{del}, \mathcal{O}.\mathsf{LR}_b, \mathcal{O}.\mathsf{retr}, \mathcal{O}.\mathsf{peek}, \mathcal{O}.\mathsf{erase}, \mathcal{O}.\mathsf{insert}\}$. For queries to the non-specified oracles, the reduction performs the required operations exactly as defined in Fig. 7.*

to that for proving multi-user security of encryption, for example using techniques similar to those of Bellare, Boldyreva and Micali [BBM00]. Here, however, the loss in the reduction is the number of adversarial calls to its $\mathcal{O}.\mathsf{LR}$ oracle, rather than the number of users. Perhaps surprisingly, even with an IND-CPA-secure encryption scheme, we get our strongest notion of security for the resulting cloud storage protocol.

Theorem 3. *Let $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ be a cloud storage protocol that uses a symmetric key encryption scheme $\mathsf{SKE}_3 = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ as in Fig. 5, with per-file keys. For an adversary $\mathcal{A}_3$ against* IND-CRA-A *there exists an adversary $\mathcal{B}_3$ such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_3} \leq f \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}_3,\,\mathcal{B}_3}$$

where f is a bound on the number of calls to the $\mathcal{O}.\mathsf{LR}_b$ oracle by the adversary.

Proof. We define $f + 1$ hybrid games. The games all work like $\mathbf{Exp}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}}$, except in the behaviour of their $\mathcal{O}.\mathsf{LR}_b$ oracle. In the ith game, the oracle will always use F_1 for the first i queries, and always use F_0 for the remaining queries. In G_0, the $\mathcal{O}.\mathsf{LR}_b$ oracle will always behave as if $b = 0$, while in G_f it will always behave as if $b = 1$.

The proof now proceeds exactly as the proof of Theorem 1, using the reduction $\mathcal{B}_{3,i}$ as defined in Fig. 12. In particular, since a random key is chosen for $\mathcal{O}.\mathsf{store}$ and $\mathcal{O}.\mathsf{LR}_b$ queries, the reduction can simulate all encryptions except those it needs to send to its own $\mathcal{O}.\mathsf{LR}_b$ oracle. These values are added to CL so cannot be acquired via an $\mathcal{O}.\mathsf{retr}$ query, so the reduction can adequately simulate all oracle queries. The claim follows. □

Remark 2. *Note that the reduction used in the proof of Theorem 3 never uses its $\mathcal{O}_{\mathsf{SKE}}.\mathsf{E}$ oracle, and uses its $\mathcal{O}_{\mathsf{SKE}}.\mathsf{LR}$ oracle exactly once. This means that we could in principle use a one-time secure symmetric cryptosystem, and still get security.*

4.2.3 Deduplicating Systems using File-Derived Keys

Sometimes we want to derive keys from the files themselves, in order to be able to deduplicate. 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 encryption schemes that derive encryption keys from the file itself (as defined in Section 2.2.2), the PRV-CDA notion of security 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

$\underline{\mathbf{Exp}^{\mathsf{PRV\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathtt{MS},\,\mathcal{A}}}$:
init
$\mathtt{flag} \leftarrow 0$
$\mathsf{CL}, \mathsf{users}_{\mathsf{C}}, \mathsf{users}_{\mathsf{H}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\text{oracles}}$
return $b' \stackrel{?}{=} b$

$\mathcal{O}.\mathsf{LR}_b(\mathbf{uid}, \mathbf{id})$:
if $\mathtt{flag} = 1 \ \vee \exists \mathsf{uid} \in \mathbf{uid}$: $\mathsf{uid} \notin \mathsf{users}_{\mathsf{H}}$ **then**
return $\perp$
$(\mathbf{m}_0, \mathbf{m}_1, Z) \leftarrow \mathtt{MS}$
for $i = 1, \ldots, |\mathbf{m}_b|$ **do**
$\mathsf{store}(\mathbf{uid}[i], \mathbf{m}_b[i], \mathbf{id}[i])$
$\mathsf{CL} \xleftarrow{\cup} \{(\mathbf{uid}[i], \mathbf{id}[i])\}$
$\mathtt{flag} \leftarrow 1$

Figure 13: *The experiment defining* PRV-atk-csp *security for cloud storage with file-derived keys, for* $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$. *The algorithms from Fig. 7 are also included. The oracles to which the adversary has access are the same as for our* IND *notions.*

be independent of the encryption scheme to avoid pathological situations, hence pre-chosen.) This security notion can be achieved by both deterministic [BKR13] and randomised schemes [ABM+13].

Based on such an encryption scheme, we define a natural cloud storage scheme by having fkeyGen simply run the encryption scheme's key derivation algorithm. We define a notion of security for such cloud storage similar to PRV-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. We then prove the natural theorem.

Definition 7 (PRV-atk-csp Security for Cloud Storage)**.** *Let* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *be a cloud storage scheme. Then the* PRV-atk-csp *advantage for an adversary* $\mathcal{A}$*, message source* MS *and* $\mathsf{atk} \in \{\mathsf{CSA}, \mathsf{CRA}\}$, $\mathsf{csp} \in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ *against* CS *is defined by*

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

where experiment $\mathbf{Exp}^{\mathsf{PRV\text{-}atk\text{-}csp}}_{\mathsf{CS},\,\mathtt{MS},\,\mathcal{A}}$ *is given in Fig. 13.*

Recall that due to Assumption 1, just as for the IND notions the adversary cannot call $\mathcal{O}.\mathsf{LR}_b$ or $\mathcal{O}.\mathsf{store}$ with any $(\mathsf{uid}, \mathsf{id})$ pair that it has called before. The relations between IND notions from Fig. 8 and Appendix A follow through to the PRV setting.

Theorem 4. *Let* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *be a cloud storage protocol that uses a symmetric key encryption scheme* $\mathsf{SKE} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *as in Fig. 5, with file-derived keys. For an adversary* $\mathcal{A}_4$ *against* PRV-CSA-P *and message source* MS *there exists an adversary* $\mathcal{B}_4$ *such that*

$$\mathbf{Adv}^{\mathsf{PRV\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathtt{MS},\,\mathcal{A}_4} \leq \mathbf{Adv}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\,\mathtt{MS},\,\mathcal{B}_4}.$$

$\mathcal{B}_4$ playing $\mathbf{Exp}^{\mathsf{PRV\text{-}CDA}}_{\mathsf{SKE},\mathtt{MS},\mathcal{B}_4}$:
receive $(\mathbf{c}, Z)$
$\mathsf{ST} \leftarrow \emptyset$
$\mathtt{flag} \leftarrow 0$
$b' \leftarrow \mathcal{A}_4^{\mathcal{O}.\mathsf{LR}_b}$
return b'

$\mathcal{O}.\mathsf{LR}_b(\mathbf{uid}, \mathbf{id})$:
if $\mathtt{flag} = 1$ **then**
return $\perp$
else
for $i = 1, \ldots, |\mathbf{c}|$ **do**
$\mathsf{ST} \xleftarrow{\cup} \{(\mathbf{uid}[i], \mathbf{c}[i], \mathbf{id}[i]), 0\}$
$\mathtt{flag} \leftarrow 1$

Figure 14: *Reduction* $\mathcal{B}_4$ *for proof of Theorem 4. The* $\mathcal{O}$.newuC, $\mathcal{O}$.newuH, $\mathcal{O}$.store, $\mathcal{O}$.upl, $\mathcal{O}$.del *and* $\mathcal{O}$.peek *oracles are not listed, but are as defined in Fig. 7.*

Proof. In the PRV-CDA experiment, parameterized by message source MS, the adversary receives a vector of encryptions of messages chosen from one of the source's output vectors. In the syntax for the

PRV-CDA game (Fig. 2) the adversary is given the ciphertext vector resulting from encrypting one of the message vectors output by MS. This means that when $\mathcal{B}_4$ (playing PRV-CDA) runs $\mathcal{A}_4$ and $\mathcal{A}_4$ makes its $\mathcal{O}$.LR query, $\mathcal{B}_4$'s simulation of the $\mathcal{O}$.store oracle must store the ciphertexts it received from its own challenger (under the user identities and handles provided by $\mathcal{A}_4$).

□

4.3 Deduplicating Schemes with non-trivial fkeyGen Procedures

The results so far in this section have only considered schemes for which 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. 6)[10]. While our model (Fig. 5) can handle deduplicating schemes with complex fkeyGen protocols such as that of Liu et al. [LAP15a], Duan [Dua14] and Keelveedhi et al. (DupLESS) [KBR13], our security definitions do not fully capture them due to the 'unnatural' inputs to fkeyGen when it is 'called' by store. In order to consider the confidentiality of such schemes within our model it is necessary to incorporate a simple extension: an $\mathcal{O}$.fkeyGen oracle that the adversary can call on arbitrary inputs.

Our model is also easily extensible to the distributed storage context: the $\mathcal{O}$.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 [LQLL16].

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

Since Duan [Dua14] showed that the DupLESS system achieves D-IND$ (in the random oracle model), we would expect that DupLESS would meet strong security in our model. However if we assume that the adversary has a fkeyGen oracle as described above, DupLESS does not even meet D-IND-CSA-P. The attack is straightforward: The adversary calls its 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 (uid, 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 fkeyGen earlier to get F_b, then output b. This indicates how weak a D-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 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 the ciphertexts sent to the CSP. One approach to this requirement is to use proofs of retrievability (PoR) [JJ07, SW13], where users embed some data in their files (ciphertexts) and periodically engage in some 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 the 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. 15. 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 of course rules

[10] The threshold scheme of Stanek et al. [SSAK14] is something of a special case since fkeyGen is run locally but the encryption algorithm is not a symmetric encryption scheme.

out schemes for which a file hash is all that is required to indicate ownership of that file (Dropbox pre-2011 [MSL+11, DMM+12, vdL], content distribution networks, etc.): we will discuss later the meaning of ciphertext integrity in these 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 Section 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 Section 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: a scenario where an adversarial client (INT-SC-Z) is (inherently) given more power by the mechanism that saves communication bandwidth.

Note that in the description of $\mathcal{O}.\mathsf{retr}'$, 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. This means that the ciphertext component or the deletion bit (or both) being different means that this condition is achieved.

Definition 8 (INT-SC-csp for Cloud Storage)**.** *Let* CS *be a cloud storage system based on a symmetric cryptosystem as in Fig. 5, 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. 15.*

$\mathbf{Exp}^{\mathsf{INT\text{-}SC\text{-}csp}}_{\mathsf{CS},\,\mathcal{A}}$:
$b \leftarrow 0$
$\mathsf{ST} \leftarrow \emptyset$
$\mathsf{TrueST} \leftarrow \emptyset$
$\mathcal{A}^{\text{oracles}}$
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 $\{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{ST}\} \neq \{(\mathsf{uid}, \cdot, \mathsf{id}, \cdot) \in \mathsf{TrueST}\}$
$\wedge \mathsf{F} \neq \perp$
then $b \leftarrow 1$
return F

Figure 15: *The experiment defining* INT-SC-csp *for cloud storage. The adversary has access to* $\mathcal{O}$.newuC, $\mathcal{O}$.newuH, $\mathcal{O}.\mathsf{store}'$, $\mathcal{O}$.upl, $\mathcal{O}.\mathsf{del}'$ *and* $\mathcal{O}.\mathsf{retr}'$. *If* $\mathsf{csp} = \mathsf{P}$ *then the adversary additionally has access to* $\mathcal{O}$.peek, *and if* $\mathsf{csp} = \mathsf{A}$ *then the adversary additionally has access to* $\mathcal{O}$.erase *and* $\mathcal{O}$.insert. *The oracles that are not explicitly stated are as defined in Fig. 7.*

Our definition of del in Fig. 5 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 entries. If del would only delete the KT entry after checking existence of a ST entry then this would give the adversary a trivial way to de-synchronize TrueST and ST. We acknowledge that this technical issue is awkward but believe that 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

We now 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. 6). For this we require the syntax for an encryption scheme that can handle associated data (see Section 2.2.4). Just as in Thm. 2 the associated data is the handle id.

Theorem 5. *Let* CS *be a cloud storage system that uses a symmetric cryptosystem* SKE_5 *with per-user keys, with at most* n *users, with* $\mathtt{AD} = \mathsf{id}$*, and let* $\mathcal{A}_5$ *be an adversary against ciphertext integrity for* CS*. Then there exists an adversary* $\mathcal{B}_5$ *against ciphertext integrity for* SKE *such that*

$$\mathbf{Adv}^{\mathsf{INT\text{-}SC\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_5} \leq n \cdot \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{SKE}_5,\,\mathcal{B}_5}.$$

Proof. Since each user's key material is independent of every other user's key material, it is sufficient to prove the result for a single user, since a standard hybrid argument will complete the argument, also incurring a factor n. So consider an adversary $\mathcal{A}_5$ in the experiment $\mathbf{Exp}^{\mathsf{INT\text{-}SC\text{-}A}}_{\mathsf{CS},\mathcal{A}}$ with a single user. Based on this, we can construct an adversary $\mathcal{B}_5$ against INT-CTXT by replacing the encryption and decryption in $\mathcal{O}.\mathsf{store}$ and $\mathcal{O}.\mathsf{retr}$ by calls to the encryption and decryption oracles in the INT-CTXT game. Suppose b is set to 1 in the retrieve oracle. Since deletions are recorded in KT and therefore are correctly handled, it must be the case that the retrieve oracle has read and correctly decrypted a ciphertext that is different from the ciphertext stored by the store oracle. Since the identity is included in every ciphertext as associated data, and since (by assumption) the store oracle is never called twice with the same identity, it follows that the ciphertext that was correctly decrypted was not created by the store oracle under that identity. It then follows that the retrieve oracle has submitted a ciphertext to its decryption oracle that decrypted correctly, but was not returned by the encryption oracle with the same associated data. It follows that $\mathcal{B}_5$ has won the INT-CTXT game. □

Remark 3. *The use of* id *as associated data to tie ciphertexts to handles is certainly necessary here. If* SKE *that uses per-user keys is* IND-CCA2 *but does not use* AD *then the adversary can store some file for some corrupt client (*E *will use* uk *that is known to the adversary) under some* uid *and* id*, erase that entry from* ST*, insert a different ciphertext* $c' \leftarrow \mathsf{E}_{\mathtt{uk}}(\mathsf{F}')$ *under the same* uid *and* id*, then do* $\mathcal{O}.\mathsf{retr}(\mathsf{uid}, \mathsf{id})$ *to win the* INT-SC-A *game.*

We now prove an intuitive theorem inspired by Bellare and Namprempre's IND-CPA $\wedge$ INT-CTXT $\Rightarrow$ IND-CCA2 result for symmetric encryption [BN00].

Theorem 6. *Let* CS *be a cloud storage system, and let* $\mathcal{A}_6$ *be an adversary against indistinguishability under a chosen retrieve attack with an actively compromised server (*IND-CRA-A*). Then there exist adversaries* $\mathcal{B}_6$ *and* $\mathcal{C}_6$ *such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CRA\text{-}A}}_{\mathsf{CS},\,\mathcal{A}_6} \leq \mathbf{Adv}^{\mathsf{IND\text{-}CSA\text{-}P}}_{\mathsf{CS},\,\mathcal{B}_6} + \mathbf{Adv}^{\mathsf{INT\text{-}SC\text{-}A}}_{\mathsf{CS},\,\mathcal{C}_6}.$$

Proof. Game G_0 is the original IND-CRA-A game. In Game G_1, we modify $\mathcal{O}.\mathsf{store}$ and $\mathcal{O}.\mathsf{del}$ so that they keep track of what is added to ST. Likewise, we modify $\mathcal{O}.\mathsf{retr}$ so that when doing a retrieve, it first compares the contents of ST with its records of what ST should contain, as recorded by the modified store and delete oracles. If there is a mismatch between the actual contents of ST and the records, the modified retrieve oracle fails instead of retrieving the data. The modifications to the store and delete oracles are unobservable: the only case where the response of the modified retrieve oracle would differ from the response of the original retrieve oracle corresponds exactly to the case where an integrity adversary wins the INT-SC-A game. If Game G_1 can be distinguished from Game G_0 with advantage ϵ_1, then we have an adversary against INT-SC-A whose success probability is equal to ϵ_1.

In Game G_2, we modify $\mathcal{O}.\mathsf{retr}$ so that it no longer decrypts ciphertexts to get its responses, but instead gets its responses from the data recorded by the modified store and delete oracles. In either

game, the retrieve oracle will only respond if the stored data is unchanged, so by correctness, Game G_2 cannot be distinguished from Game G_1.

Finally, we can make an adversary against IND-CSA-A whose probability of guessing b correctly is exactly the same as the original adversary's probability of guessing b correctly in Game G_2. The claim follows, since without a retrieve oracle, the adversary's erase and insert oracles are useless. □

Remark 4. *Note that the proof of Theorem 6 works equally well if we replace* IND-CRA-A *and* IND-CSA-P *with* PRV-CRA-A *and* PRV-CSA-P. *Indeed, it works for any goal that is* IND*-like.*

5.3 Integrity in Deduplicating Schemes

For deterministic schemes such as convergent encryption ($\mathsf{fk} \leftarrow \mathsf{H}(\mathsf{F})$) it is trivial for an adversary with active server compromise to create new ciphertexts that decrypt correctly. For this reason Bellare et al. (BKR) [BKR13] discussed tag consistency: their tags served a different 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 [SGLM08] 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. Our assumption on handles (Assumption 1) rules out this type of attack, so we must consider a modified definition of a cloud storage system to include an additional tagging algorithm. This formalism is given in Fig. 16.

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: $\mathtt{tag} \leftarrow \mathsf{TGen}(c)$. This is without loss of generality: the HCE1, HCE2 and RCE schemes that they describe calculate $\mathtt{tag} \leftarrow \mathsf{H}(\mathsf{fk})$ to give a ciphertext formed as $\mathtt{tag} || \mathsf{E}_{\mathsf{fk}}(\mathsf{F})$, then the TGen algorithm parses this value and outputs $\mathtt{tag}$. 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 $\mathtt{tag}$, gets a response indicating whether it should send the ciphertext or not, and then finally it sends the ciphertext if asked to. In our syntax line 8 initially only requires $(\mathsf{uid}, \mathsf{id}, \mathtt{tag})$ as inputs and checks its own storage for a tag match: **if** $\exists\{\cdot, c, \cdot, \mathtt{tag}, \cdot\} \in \mathsf{ST}$ **then** $\mathsf{ST} \xleftarrow{\cup} \{(\mathsf{uid}, c, \mathsf{id}, \mathtt{tag}, 0)\}$. If a match is not found, the server sends a message, sometimes referred to as a *deduplication signal* [ABD+17], to the user to let them know that it is necessary for the ciphertext to be transmitted.

Lines 19a-19c 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. We will see shortly that using this check or not creates an important distinction in the integrity properties we achieve in our wider storage protocols.

We now reproduce BKR's definitions of Tag Consistency (TC) and Strong Tag Consistency (STC) that are used to capture duplicate-faking attacks. For TC it should be hard to create a pair (F, c') such that the tags are the same but the ciphertexts decrypt to different files. STC asks that it is hard to create such a pair with the same tag but here the adversary wins even if c' decrypts to $\perp$: if this property holds then it is hard to erase files stored by other users.

Definition 9. *Let* $\mathsf{SKE.Dedup} = (\mathsf{fkeyGen}, \mathsf{E}, \mathsf{D}, \mathsf{TGen})$ *be a* deduplicating encryption mechanism. *Then the* TC/STC *advantage for an adversary* $\mathcal{A}$ *against* SKE.Dedup *is*

$$\mathbf{Adv}^{\mathsf{TC/STC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}} = \mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{TC/STC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}} = 1\right]$$

```
init
1. ST ← ∅

newu(uid)
2. uk_uid ← kgen
3. KT_uid ← ∅
4. return uk_uid

del(uid, id)
5. KT_uid ←∪ {(⊥, id, tag)}
6. if ∃ {uid, ·, id, tag, 0} in ST then
7.     ST ←∪ {(uid, −, id, tag, 1)}

upl(uid, c, tag, id)
8. ST ←∪ {(uid, c, id, tag, 0)}

store(uid, F, id)
9. fk ← fkeyGen(F, uk_uid)
10. c ← E_fk(F, id)
11. tag ← TGen(c)
12. upl(uid, c, id, tag)
13. KT_uid ←∪ {(fk, id, tag)}

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

Figure 16: *Definition of a deduplicating cloud storage scheme* DCS[SKE.Dedup].

where experiments $\mathbf{Exp}^{\mathsf{TC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}}$ *and* $\mathbf{Exp}^{\mathsf{STC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}}$ *are given in Fig. 17.*

The attacker considered by (S)TC is a malicious client that does not have any access to the server's database, i.e. Z in our hierarchy. Recall that in our INT-SC-Z game the adversary has access to $\{\mathcal{O}.\mathsf{newuC}, \mathcal{O}.\mathsf{newuH}, \mathcal{O}.\mathsf{store}', \mathcal{O}.\mathsf{upl}, \mathcal{O}.\mathsf{del}', \mathcal{O}.\mathsf{retr}'\}$ only. We note that this model is very similar to the challenger defined in the (S)TC games.

We now consider the relationship between the (strong) tag consistency of the underlying deduplicating encryption mechanism and the resulting DCS when the adversary is corrupted (INT-SC-A).

Remark 5. *If* DCS[SKE.Dedup] *does not implement the tag check (lines 19a-19c in Fig. 16), then* SKE.Dedup *being* STC *does not necessarily imply that* DCS *is* INT-SC-A.

To see this, consider a scheme SKE.Dedup that employs no encryption, and the tagging algorithm just outputs the ciphertext: $\bot \leftarrow \mathsf{fkeyGen}(\mathsf{F})$; $c = \mathsf{F} \leftarrow \mathsf{E}(\mathsf{fk}, \mathsf{F})$; $c \leftarrow \mathsf{D}(\mathsf{fk}, c)$; $c \leftarrow \mathsf{TGen}(c)$. This trivial scheme certainly meets STC since the tagging algorithm is injective. However it is of course not INT-SC-A since an adversary can perform the following queries to win: $\mathcal{O}.\mathsf{store}(\mathsf{uid}, \mathsf{F}, \mathsf{id})$; $\mathcal{O}.\mathsf{erase}(\mathsf{uid}, \mathsf{id})$; $\mathcal{O}.\mathsf{insert}(\mathsf{uid}, c, \mathsf{id}, \mathsf{F}, 0)$ for some $c \neq \mathsf{F}$; $\mathcal{O}.\mathsf{retr}(\mathsf{uid}, \mathsf{id})$.

Remark 6. *A scheme* DCS[SKE.Dedup] *can be* INT-SC-A *yet* SKE.Dedup *is not* TC.

Consider the CS created by using SKE_5 in Thm. 5. We can turn this into a deduplicating cloud storage scheme by including the degenerate tagging procedure $0 \leftarrow \mathsf{TGen}(c)$ for all ciphertexts c. This does not affect INT-SC-A and yet the SKE.Dedup is certainly not TC, since any valid (F, c') will of course give the same tag.

These two remarks emphasize that (S)TC and INT-SC are tangential concerns: deduplication tags serve a different purpose to retrieval handles.

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.

$\underline{\mathbf{Exp}^{\mathsf{TC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}}:}$
$(\mathsf{F}, c') \leftarrow \mathcal{A}$
if $(\mathsf{F} = \bot)$ **or** $(c' = \bot)$ **then**
  return 0
$\mathsf{fk} \leftarrow \mathsf{fkeyGen}(\mathsf{F}, \mathtt{uk})$
$c \leftarrow \mathsf{E}_{\mathsf{fk}}(\mathsf{F})$
$\mathsf{F}' \leftarrow \mathsf{D}_{\mathsf{fk}}(c')$
$\mathtt{tag} \leftarrow \mathsf{TGen}(c)$
$\mathtt{tag}' \leftarrow \mathsf{TGen}(c')$
if $(\mathtt{tag} = \mathtt{tag}') \wedge (\mathsf{F} \neq \mathsf{F}') \wedge (\mathsf{F}' \neq \bot)$ **then**
  return 1
else
  return 0

$\underline{\mathbf{Exp}^{\mathsf{STC}}_{\mathsf{SKE.Dedup},\,\mathcal{A}}:}$
$(\mathsf{F}, c') \leftarrow \mathcal{A}$
if $(\mathsf{F} = \bot)$ **or** $(c' = \bot)$ **then**
  return 0
$\mathsf{fk} \leftarrow \mathsf{fkeyGen}(\mathsf{F}, \mathtt{uk})$
$c \leftarrow \mathsf{E}_{\mathsf{fk}}(\mathsf{F})$
$\mathsf{F}' \leftarrow \mathsf{D}_{\mathsf{fk}}(c')$
$\mathtt{tag} \leftarrow \mathsf{TGen}(c)$
$\mathtt{tag}' \leftarrow \mathsf{TGen}(c')$
if $(\mathtt{tag} = \mathtt{tag}') \wedge (\mathsf{F} \neq \mathsf{F}')$ **then**
  return 1
else
  return 0

Figure 17: *Games defining* TC *and* STC *adapted from BKR [BKR13].*

Theorem 7. *If* DCS[SKE.Dedup] *implements the tag check (lines 19a-19c in Fig. 16) and if* SKE.Dedup *is* STC *then* DCS *is* INT-SC-A.

Proof. We assume an adversary $\mathcal{A}_7$ that wins the INT-SC-A game. If $\mathcal{A}_7$ wins, it must submit a $\mathcal{O}$.retr query that succeeds, even though either the requested file has been deleted before the query or the cloud storage server returns a different ciphertext than was originally stored (and recorded in the game's TrueST table). Because deletion is recorded in KT, the former cannot happen. Since the $\mathcal{O}$.retr query succeeded, we know that the ciphertext returned by the cloud storage service has the same tag as is found in KT. Since the tag found in KT was recorded at the time of storage, we know that the original ciphertext has this tag. In other words, $\mathcal{A}_7$ has produced two distinct ciphertexts with the same tag. From here we can build a reduction $\mathcal{B}_7$ that wins the STC game. □

6 Concluding Remarks

In this paper, we defined notions of security for cloud storage and deduplication. We provided a generic model for cloud storage with a set of functionalities for secure uploading, retrieval and deletion of data on the cloud that covers all the existing cloud storage systems. We defined indistinguishability-based notions of confidentiality for our model and emphasized how to construct secure schemes in each of the three distinct levels of adversarial control. We also defined a notion of integrity of ciphertexts for cloud storage schemes, and showed how to achieve this goal from standard existing notions of integrity for encryption. Furthermore, we showed the relationship between our notion of integrity and the existing notion of tag consistency in deduplicating schemes.

For practitioners our results allow rigorous classification of so-called secure storage schemes in a fine-grained threat model. There are many ways that our model can be further enhanced, some of which have already been mentioned in the text. These include: consideration of multiple storage servers; length-hiding properties to avoid practical attacks based on ciphertext length [RKSC16]; file sharing between different clients for the same user; and additional security properties such as anonymity.

Acknowledgements. We thank Frederik Armknecht for his input to discussions for this project. We would also like to thank Jian Liu and N. Asokan for discussions about their protocol when Gareth T. Davies visited Aalto University in August 2016. This research was funded by the Research Council of Norway under Project No. 248166.

References

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 598–609. ACM, 2007. Cited on page 4.

[ABD+17] Frederik Armknecht, Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Mohsen Toorani. Side channels in deduplication: Trade-offs between leakage and efficiency. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 266–274. ACM, 2017. Cited on pages 4, 11, and 23.

[ABM+13] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-locked encryption for lock-dependent messages. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 2013. Cited on pages 9 and 19.

[AHMR15] Dirk Achenbach, Matthias Huber, Jörn Müller-Quade, and Jochen Rill. Closing the gap: A universal privacy framework for outsourced data. In *Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers*, volume 9540 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2015. Cited on page 4.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000. Cited on page 18.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 394–403. IEEE Computer Society, 1997. Cited on page 3.

[BK15] Mihir Bellare and Sriram Keelveedhi. Interactive message-locked encryption and secure deduplication. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, volume 9020 of *Lecture Notes in Computer Science*, pages 516–538. Springer, 2015. Cited on page 9.

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 296–312. Springer, 2013. Cited on pages 3, 4, 7, 9, 11, 19, 23, and 25.

[BM17] Christian Badertscher and Ueli Maurer. Composable and robust outsourced storage. *IACR Cryptology ePrint Archive*, 2017:133, 2017. Cited on page 4.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology -*

ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000. Cited on pages 3, 8, and 22.

[BPS15] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In *Cryptography and Coding - 15th IMA International Conference, IMACC 2015, Oxford, UK, December 15-17, 2015. Proceedings*, volume 9496 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 2015. Cited on page 8.

[CS15] Melissa Chase and Emily Shen. Substring-searchable symmetric encryption. *PoPETs*, 2015(2):263–281, 2015. Cited on pages 3 and 4.

[DAB+02] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS*, pages 617–624, 2002. Cited on pages 4, 7, and 9.

[DC17] Hung Dang and Ee-Chien Chang. Privacy-preserving data deduplication on trusted processors. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, June 25-30, 2017*, pages 66–73. IEEE Computer Society, 2017. Cited on page 11.

[DMM+12] Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside Dropbox: understanding personal cloud storage services. In *Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference, IMC '12, Boston, MA, USA, November 14-16, 2012*, pages 481–494. ACM, 2012. Cited on pages 3 and 21.

[Dua14] Yitao Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014*, pages 57–68. ACM, 2014. Cited on pages 4, 11, and 20.

[EFG06] Sergei Evdokimov, Matthias Fischmann, and Oliver Günther. Provable security for outsourcing database operations. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 117. IEEE Computer Society, 2006. Cited on page 4.

[Gjø05] Kristian Gjøsteen. Security notions for disk encryption. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 455–474. Springer, 2005. Cited on pages 3 and 4.

[Goh03] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003. Cited on page 4.

[HHPS11] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 491–500. ACM, 2011. Cited on pages 3 and 4.

[HPS10] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, 2010. Cited on pages 4 and 11.

[HRAM13] Rolf Haynberg, Jochen Rill, Dirk Achenbach, and Jörn Müller-Quade. Symmetric searchable encryption for exact pattern matching using directed acyclic word graphs. In *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 403–410. SciTePress, 2013. Cited on page 4.

[JJ07] Ari Juels and Burton S. Kaliski Jr. PORs: proofs of retrievability for large files. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 584–597. ACM, 2007. Cited on pages 3, 4, and 20.

[JSSW17] Tibor Jager, Martijn Stam, Ryan Stanley-Oakes, and Bogdan Warinschi. Multi-key authenticated encryption with corruptions: Reductions are lossy. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2017. Cited on page 8.

[KBR13] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 179–194. USENIX Association, 2013. Cited on pages 4, 9, 11, and 20.

[KK08] Seny Kamara and Jonathan Katz. How to encrypt with a malicious random number generator. In *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 303–315. Springer, 2008. Cited on page 13.

[KMV17] Louiza Khati, Nicky Mouha, and Damien Vergnaud. Full disk encryption: Bridging theory and practice. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2017. Cited on page 4.

[KW14] Thomas Kunz and Ruben Wolf. OmniCloud – the secure and flexible use of cloud storage services. Technical report, Fraunhofer Institute for Secure Information Technology SIT, 2014. Cited on page 11.

[KY06] Jonathan Katz and Moti Yung. Characterization of security notions for probabilistic private-key encryption. *J. Cryptology*, 19(1):67–95, 2006. Cited on page 3.

[LAP15a] Jian Liu, N. Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 874–885. ACM, 2015. Cited on pages 4, 11, and 20.

[LAP15b] Jian Liu, N. Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. *IACR Cryptology ePrint Archive*, 2015:455, 2015. Cited on page 4.

[LDLA17] Jian Liu, Li Duan, Yong Li, and N. Asokan. Secure deduplication of encrypted data: Refined model and new constructions. *IACR Cryptology ePrint Archive*, 2017:1089, 2017. Cited on page 4.

[LLXC16] Jingwei Li, Jin Li, Dongqing Xie, and Zhang Cai. Secure auditing and deduplicating data in cloud. *IEEE Trans. Computers*, 65(8):2386–2396, 2016. Cited on pages 4 and 9.

[LQLL16] Mingqiang Li, Chuan Qin, Jingwei Li, and Patrick P. C. Lee. CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. *IEEE Internet Computing*, 20(3):45–53, 2016. Cited on pages 11 and 20.

[MRAM17] Sebastian Messmer, Jochen Rill, Dirk Achenbach, and Jörn Müller-Quade. A novel cryptographic framework for cloud file systems and CryFS, a provably-secure construction. In *Data and Applications Security and Privacy XXXI - 31st Annual IFIP WG 11.3 Conference, DBSec 2017, Philadelphia, PA, USA, July 19-21, 2017, Proceedings*, volume 10359 of *Lecture Notes in Computer Science*, pages 409–429. Springer, 2017. Cited on page 4.

[MSL+11] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar R. Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011. Cited on page 21.

[PMÖL13] Pasquale Puzio, Refik Molva, Melek Önen, and Sergio Loureiro. Cloudedup: Secure deduplication with encrypted data for cloud storage. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 1*, pages 363–370. IEEE Computer Society, 2013. Cited on page 11.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2011. Cited on page 8.

[RKSC16] Hubert Ritzdorf, Ghassan Karame, Claudio Soriente, and Srdjan Capkun. On information leakage in deduplicated storage systems. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop, CCSW 2016, Vienna, Austria, October 28, 2016*, pages 61–72. ACM, 2016. Cited on pages 14 and 25.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002. Cited on pages 3 and 8.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006. Cited on page 8.

[SGLM08] Mark W. Storer, Kevin M. Greenan, Darrell D. E. Long, and Ethan L. Miller. Secure data deduplication. In *Proceedings of the 2008 ACM Workshop On Storage Security And Survivability, StorageSS 2008, Alexandria, VA, USA, October 31, 2008*, pages 1–10. ACM, 2008. Cited on page 23.

[SKYH17] Youngjoo Shin, Dongyoung Koo, Joobeom Yun, and Junbeon Hur. Decentralized server-aided encryption for secure deduplication in cloud storage. *IEEE Transactions on Services Computing*, PP(99), 2017. Cited on page 12.

[SSAK14] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 99–118. Springer, 2014. Cited on pages 4, 11, and 20.

[SW13] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptology*, 26(3):442–483, 2013. Cited on pages 3, 4, and 20.

[vdL] Wladimir van der Laan. Dropship. `https://github.com/driverdan/dropship` (Date last accessed 13-August-2017). Cited on page 21.

[WA14] Duane C. Wilson and Giuseppe Ateniese. To share or not to share in client-side encrypted clouds. In Sherman S. M. Chow et al., editors, *Information Security - 17th International Conference, ISC 2014*, volume 8783 of *Lecture Notes in Computer Science*, pages 401–412. Springer, 2014. Cited on page 3.

[YJ13] Kan Yang and Xiaohua Jia. An efficient and secure dynamic auditing protocol for data storage in cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 24(9):1717–1726, 2013. Cited on pages 4 and 9.

A Relations between Confidentiality Notions

Here we expand on the relations between our notions for confidentiality introduced in Section 4. Recall that 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 access to $\mathcal{O}$.erase and $\mathcal{O}$.insert. Fig. 18 is a more detailed version of Fig. 8 in Section 4, with labels to indicate the non-trivial remarks below that desribe the relations.

If an adversary in the IND-Y game has access to a subset of the oracles available to an IND-X adversary then trivially IND-X $\Rightarrow$ IND-Y. This gives rise to the trivial relations IND-atk-A $\Rightarrow$ IND-atk-P $\Rightarrow$ IND-atk-Z for atk $\in \{\mathsf{CSA}, \mathsf{CRA}\}$, and also IND-CRA-csp $\Rightarrow$ IND-CSA-csp for csp $\in \{\mathsf{Z}, \mathsf{P}, \mathsf{A}\}$ (these are the right-to-left and bottom-to-top arrows in Fig. 18).

Otherwise, to show that IND-X $\Rightarrow$ IND-Y for some X, Y it is sufficient to show that a reduction playing IND-X can perfectly simulate the oracles from the IND-Y game that it does not have access to, and further that a successful underlying adversary in the IND-Y game will lead to victory in the reduction's IND-X game. This latter requirement is straightforward in our cases because all of these adversaries are distinguishing the actions invoked by their $\mathcal{O}$.LR oracle.

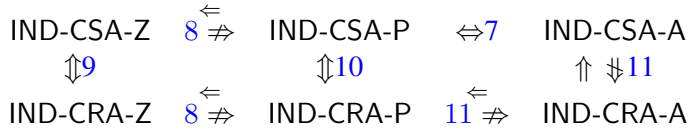

Figure 18: *Relations between* IND *notions for confidentiality of cloud storage systems.*

Remark 7 (IND-CSA-P $\Rightarrow$ IND-CSA-A)**.** *Reduction forwards* $\mathcal{O}$.newuH, $\mathcal{O}$.newuC, $\mathcal{O}$.store, $\mathcal{O}$.upl, $\mathcal{O}$.del, $\mathcal{O}$.LR *and* $\mathcal{O}$.peek *queries to its own oracles. Reduction can simulate the extra* $\mathcal{O}$.insert *and* $\mathcal{O}$.erase *queries since there is no* $\mathcal{O}$.retr *oracle: it simply keeps a table for each of* $\mathcal{O}$.insert *and* $\mathcal{O}$.erase *and whenever adversary does an* $\mathcal{O}$.peek *(or* $\mathcal{O}$.del*) at an entry in either table, reduction responds appropriately.*

Remark 8 (IND-CSA-Z $\not\Rightarrow$ IND-CSA-P and IND-CRA-Z $\not\Rightarrow$ IND-CRA-P)**.** *Let* SKE *be the scheme that employs no encryption:* SKE.E *is identity function,* $c \leftarrow \mathsf{F}$. IND-CSA-P *adversary does one* $\mathcal{O}.\mathsf{LR}_b$ *query with distinct* $\mathsf{F}_0, \mathsf{F}_1$ *for some* uid *and* id*, does* $\mathcal{O}.\mathsf{peek}(\mathsf{uid}, \mathsf{id})$ *to see* F_b*.*

Remark 9 (IND-CSA-Z $\Rightarrow$ IND-CRA-Z)**.** *Reduction needs to simulate adversary's* $\mathcal{O}$.retr *queries. Anything sent via* $\mathcal{O}$.upl *or* $\mathcal{O}$.LR *is forbidden, so needs to deal with* $\mathcal{O}$.store*. If scheme is correct then reduction just keeps a table, which is must update in the event of an* $\mathcal{O}$.del *query.*

Remark 10 (IND-CSA-P $\Rightarrow$ IND-CRA-P)**.** *To simulate* $\mathcal{O}.\mathsf{retr}$ *queries, reduction needs to keep a log of everything the adversary added via its* $\mathcal{O}.\mathsf{store}$ *queries, so it can successfully respond. Correctness of the underlying scheme ensures that this simulation is perfect.*

Remark 11 (IND-CSA-P $\not\Rightarrow$ IND-CRA-A)**.** *Given* $\mathsf{CS} = (\mathsf{init}, \mathsf{newu}, \mathsf{store}, \mathsf{retr}, \mathsf{del})$ *that is* IND-CSA-P*, create* CS' *as follows:*

- $\mathsf{store}'(\mathsf{uid}, \mathsf{F}, \mathsf{id})$*:* $c \leftarrow c'||\mathsf{uid}$ *where* c' *is ciphertext stored by* CS*.*
- $\mathsf{retr}'(\mathsf{uid}, \mathsf{id})$*: parse* c *as* $c'||\mathsf{uid}'$ *and do* $\mathsf{retr}(\mathsf{uid}', \mathsf{id})$*.*

with all other operations unchanged. This scheme is still IND-CSA-P *and correct. Attack:*

- $\mathcal{O}.\mathsf{LR}_b(\mathsf{uid}_a, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id})$
- $\mathcal{O}.\mathsf{peek}(\mathsf{uid}_a, \mathsf{id})$ *to see* $c = c'||\mathsf{uid}_a$
- $\mathcal{O}.\mathsf{insert}(\mathsf{uid}_b, c'||\mathsf{uid}_a, \mathsf{id}, 0)$
- $\mathcal{O}.\mathsf{retr}(\mathsf{uid}_b, \mathsf{id})$

The $\mathcal{O}.\mathsf{retr}$ *call is allowed since* $(\mathsf{uid}_b, \mathsf{id}) \notin \mathsf{CL}$*; it will parse the inserted ciphertext and run* $\mathsf{retr}(\mathsf{uid}_a, \mathsf{id})$ *and output* F_b*.*

Another (less contrived) scheme that demonstrates this separation is per-user keys with IND-CPA (or IND-CCA2 without AD) SKE, as mentioned in Remark 1. The attack uses $\mathcal{O}.\mathsf{insert}$ to overwrite an existing value ($\mathcal{O}.\mathsf{erase}$ does not affect KT), but because of the per-user keys the $\mathcal{O}.\mathsf{retr}$ query will still work correctly:

- $\mathcal{O}.\mathsf{store}(\mathsf{uid}, \mathsf{F}, \mathsf{id}_1)$ for any F
- $\mathcal{O}.\mathsf{LR}_b(\mathsf{uid}, \mathsf{F}_0, \mathsf{F}_1, \mathsf{id}_2)$ for some valid $\mathsf{F}_0 \neq \mathsf{F}_1$
- $\mathcal{O}.\mathsf{peek}(\mathsf{uid}, \mathsf{id}_2)$ to see $c = \mathsf{E}_{\mathtt{uk}_{\mathsf{uid}}}(\mathsf{F}_b)$
- $\mathcal{O}.\mathsf{erase}(\mathsf{uid}, \mathsf{id}_1)$
- $\mathcal{O}.\mathsf{insert}(\mathsf{uid}, c, \mathsf{id}_1, 0)$
- $\mathcal{O}.\mathsf{retr}(\mathsf{uid}, \mathsf{id}_1)$ to get F_b