Secure Generalized Deduplication via Multi-Key Revealing Encryption

Daniel E. Lucani[1], Lars Nielsen[1], Claudio Orlandi[1], Elena Pagnin[2], and Rasmus Vestergaard[1]

[1] Aarhus University, Aarhus, Denmark
{daniel.lucani, lani, rv}@eng.au.dk, orlandi@cs.au.dk
[2] Lund University, Lund, Sweden
elena.pagnin@eit.lth.se

Abstract. Cloud Storage Providers (CSPs) offer solutions to relieve users from locally storing vast amounts of data, including personal and sensitive ones. While users may desire to retain some privacy on the data they outsource, CSPs are interested in reducing the total storage space by employing compression techniques such as deduplication. We propose a new cryptographic primitive that simultaneously realizes both requirements: Multi-Key Revealing Encryption (MKRE). The goal of MKRE is to disclose the result of a pre-defined function over multiple ciphertexts, even if the ciphertexts were generated using different keys, while revealing *nothing else* about the data. We present a formal model and a security definition for MKRE and provide a construction of MKRE for generalized deduplication that only uses symmetric key primitives in a black-box way. Our construction allows *(a)* cloud providers to reduce the storage space by using generalized deduplication to compress encrypted data across users, and *(b)* each user to maintain a certain privacy level for the outsourced information. Our scheme can be proven secure in the random oracle model (and we argue that this is a necessary evil). We develop a proof-of-concept implementation of our solution. For a test data set, our MKRE construction achieves secure generalized deduplication with a compression ratio of 87% for 1KB file chunks and 82.2% for 8KB chunks. Finally, our experiments show that, compared to generalized deduplication setup with un-encrypted files, adding privacy via MKRE introduces a compression overhead of less than 3% and reduces the storage throughput by at most 6.9%.

Keywords: Private Cloud Storage · Secure Deduplication · Revealing Encryption

1 Introduction

Cloud Storage Providers (CSPs) are offering vast amounts of storage at a low cost to users who desire to outsource their data storage. In order to provide this service at a low cost, CSPs employ compression techniques to reduce their storage cost. In particular, *data deduplication* has become a popular technique for compression of data across files generated by users of the system. For example, if two users upload the same file to the server, only one copy is stored, which both users are allowed to retrieve. Deduplication is often carried out across file chunks (parts), which increases the potential to reduce the storage footprint of the system, as two files that are different as a whole may have significant portions that are equal, *e.g.,* when different versions of the same file are stored.

Generalized deduplication [31] is a recent generalization of this principle. By enabling deduplication of data chunks that are similar, rather than identical, this method can achieve a better compression than classic deduplication techniques. As a conceptual and oversimplified example of generalized deduplication, consider two users that hold identical pictures of the Eiffel tower, except for a different person in the foreground. If the CSP could see the two plaintext files, it could easily identify the two pictures as "almost identical", and store the background only once, reducing the overall storage space required. As a great number of people take pictures of the Eiffel tower, we could imagine that many users might upload similar images to the CSP, which would allow the data to be compressed more than is possible when only exact copies of files are deduplicated.

In order to compress the data it stores, the CSP will need to access it to identify where (generalized) deduplication can be used. A naïve implementation would allow CSPs to directly access all uploaded files in cleartext to determine how the data can be compressed. Such a solution will unfortunately undermine the privacy of the data, reducing the types of files that the CSP should be trusted with. A privacy-conscious user could patch this weakness by encrypting their data under a private key before uploading it, but using semantically secure encryption schemes would prevent the CSP from performing any meaningful deduplication. Therefore, previous work has established encryption schemes with relaxed security guarantees that can enable the server to identify when two files are identical and thus enable secure deduplication over encrypted data (see, *e.g.,* convergent [13] or message-locked [6] encryption).

In this paper, we propose an encryption scheme that allows the server to identify when encrypted data corresponds to similar, rather than identical, data. In particular, using such an encryption scheme, CSPs can apply generalized deduplication directly on the encrypted data. Albeit the server learns whether two ciphertexts correspond to similar plaintexts or not, nothing else is revealed about the original plaintext data. This provides another opportunity for protecting sensitive information while enabling storage compression techniques.

Overview of contributions. Our first contribution is a definitional model that generalizes the notion of Revealing Encryption[1] (RE) [15,23] to the multi-user setting, thus the name Multi-Key Revealing Encryption (MKRE). In a nutshell, a RE scheme is parametrized by some function f, and, given two ciphertexts c_1, c_2 obtained by encrypting x_1, x_2 respectively (under the same key), there

[1] The term "Revealing Encryption" was first introduced in an oral presentation by Adam O'Neill.

exist some public evaluation function g such that $g(c_1, c_2) = f(x_1, x_2)$.[2] MKRE extends this to the case where the data is encrypted with different, potentially independent keys.[3]

In this paper, we are mostly interested with the case where the function f computes whether x_1, x_2 are "similar" or not, as this is the function required for secure generalized deduplication. Note that MKRE for this f can also be seen as a natural generalization of message-locked encryption (and indeed message-locked encryption is a crucial component of our solution) and therefore we share the same issues in providing a meaningful and intellegible security definition. In a nutshell, in message-locked encryption a message is encrypted using the message itself as the key. Thus, any user encrypting the same message will produce the same ciphertext, allowing to check equality (and therefore perform deduplication) across encryptions performed by different users. Defining security for message-locked encryption is however quite tricky: The problem is that we cannot prevent the adversary from encrypting messages locally and compare them with the output of the challenge oracle, and we encountered similar challenges when attempting to define security for MKRE. Previous work in message-locked encryption solved this issue by parameterizing the security definition via a message distribution. We find those definitions to be quite complex to parse and therefore quite unfriendly to the practioneers who should decide whether such a primitive provides the right level of security for their applications. As our work is motivated by real-world interest in secure deduplication, we decided to provide what we see as a simpler alternative, which instead relies on idealized primitives: In our definition the adversary can only perform encryptions interacting with the challenger or the random oracle. Looking ahead, this will allow the reduction to "know everything that the adversary knows" and therefore we do not need to worry about trivial attacks. Intuitively, our definition guarantees that the adversary learns nothing about the content of a ciphertext, unless it ever requests an encryption of a "close" message, in which case it is allowed to learn this fact and nothing else. As a downside, our definition implies that our construction must use idealized primitives (e.g., random oracles).

We provide a generic construction of MKRE for our use case from a black-box combination of message-locked, deterministic, and randomized encryption schemes. Furthermore, we show a concrete instantiation based solely on hash functions and prove its security in the random oracle model, which we have argued is a necessary evil given our security definition. In a nutshell, our technique splits files into two parts: one that we consider less sensitive and deduplication-friendly (*e.g.,* the background with the Eiffel tower) and one more sensitive, on which no public clustering is desired (*e.g.,* the person in the foreground). Then a form of deterministic encryption is applied to the former part of the data (*e.g.,* the image backgrounds) while semantically secure encryption is applied

[2] Revealing encryption can be seen as a special case of functional encryption where a single decryption key is published together with the public parameters when the system is initialized.

[3] Similarly, MKRE can be seen as a special case of multi-input functional encryption.

to "fully" protect the other, more sensitive part (*e.g.,* the image foregrounds), which thus cannot be deduplicated. While, in retrospect, our construction is really simple, we find it very fascinating that it is at all possible to perform any kind of meaningful computation across ciphertexts encrypted independently by different users using independent keys, and we leave as a major open problem whether there exist "simple" MKRE schemes for other natural, useful functions.

Our findings are supported by a software implementation that allows us to characterize the trade-off among added security, processing speed of the system, and overall compression performance. Our experiments show that our construction is only negligibly less efficient than generalized deduplication applied to the plaintexts directly, while achieving a greater privacy level: Adding MKRE to generalized deduplication reduces the overall system storage throughput by no more than 6.9%, and the compression capabilities by at most 3%.[4]

Limitations. As any revealing encryption scheme, MKRE for generalized deduplication reveals some specific information about the uploaded data to the CSP. This leakage is unavoidable: if one wishes to deduplicate *similar* data, then deduplicated chunks must come from similar files. This means that a malicious CSP can see how many ciphertexts of the same user are deduplicated and learn the statistical distribution of the deduplications. Frequency analysis on deduplication reveals the distribution of messages which obviously impacts the security of the system. This cannot be mitigated while targeting high-compression deduplication and thus falls outside the scope of this work.

2 Related work

The conflicting interests of CSPs employing deduplication and privacy-concerned users have been studied from many angles. A generic framework that explains the various constraints and allows for comparison of secure deduplication strategies is presented in [8].

To enable privacy-aware server-side deduplication, the most common approach is to force users with the same plaintext to arrive at the same ciphertext. The first work of this nature was convergent encryption [13], followed by the more general Message-Locked Encryption (MLE) scheme in [6], where a hash of the plaintext is used as the encryption key. Some recent developments seek to optimize MLE, *e.g.,* for deduplication of file chunks rather than the entire files and enable efficient updates to the stored data [34]. Other proposals use an oblivious PRF against a key-server to determine the file-derived keys [5], or have clients determine the keys in a distributed manner using a PAKE-based protocol [21,22]. All of these schemes reveal that deduplicated ciphertexts correspond to *identical* data. In contrast, our scheme only reveals that deduplicated ciphertexts correspond to *similar* data. Deduplication of similar data is also dealt with

[4] In this work, we perform generalized deduplication based on Hamming codes, however, the principles we develop are general and can be easily transferred to any transformation function.

in [19], albeit quite differently. They have a highly specific use case in mind and remove perceptually similar images by comparing a 'perceptual hash' calculated over the plaintext images, which can lead to duplicate detection under modifications such as resizing, compression, or flipping. Thus, their compression is lossy, not every image can be retrieved as it was uploaded, but something "similar" might be returned. On the contrary, our method is lossless and everything can be retrieved exactly as it was uploaded. Further, what is considered similar in our method is more general, as the notion of similarity is tied to the chosen transformation function for generalized deduplication.

We view our privacy-aware server-side deduplication scheme as a generalization to the multi-user setting of the notion of Revealing Encryption (RE) [15]. Our approach also shares similarities with Predicate Encryption (PE) [17], Searchable Encryption (SE) [12], Multi-Input Functional Encryption (MIFE) [14], ad-hoc MIFE [1] and Multi-Client FE (MCFE) [11,20]. At a very abstract level, all these primitives share a common goal, *i.e.,* to allow to encrypt data in such a way that it is possible to perform *computation on a set of ciphertexts*. However, they do so by exploiting different mechanisms and assumptions, and allow for different degrees of freedom in choosing which function f should be computed and which users should be able to compute the function. In PE, SE, and (MK)RE the function f is determined a priori; the different flavors of functional encryption allow f to be chosen adaptively. In contrast to other primitives, the function computed by (MK)RE is *public* and can be evaluated on set of ciphertexts *without* the need of a secret key (or secret state). Notably, in functional encryption f is linked to the decryption algorithm, and thus it is intrinsically a secret-key operation. This is a core difference to the RE setting. Moreover, in (MK)RE data are encrypted using a secret key, which is different from what happens in all other settings.[5]

As the names suggest MIFE, MCFE and MKRE are the only primitives that consider multi-user settings. Unfortunately, it is known that MIFE and MCFE for generic functionalities can only be realized using "heavy tools", such as obfuscation and multi linear maps [14], and is therefore not currently of practical interest. Recent work has thus favored concrete and efficient realizations for more restricted functionalities that rely on less demanding assumptions, *e.g.,* MCFE for inner product in the random oracle model from DDH [11] and MCFE for linear functions in the standard model from LWE [20]. Following this philosophy, our aim is to build MKRE from "minimalist tools", *i.e.,* hash functions, and instantiate a scheme with immediate practical applications.

3 Preliminaries

Notation. For $n, n_1, n_2 \in \mathbb{N}$, let $[n_1 : n_2]$ be the set $\{n_1, n_1 + 1, \ldots, n_2\}$ and $[n]$ be the set $[1 : n]$. For $x \in \mathbb{Z}$, let $|x|$ denote the absolute value of x. Let $x \xleftarrow{\$} S$

[5] With the exception of SE for which there exist realization both in the asymmetric [3] and in the symmetric settings [28].

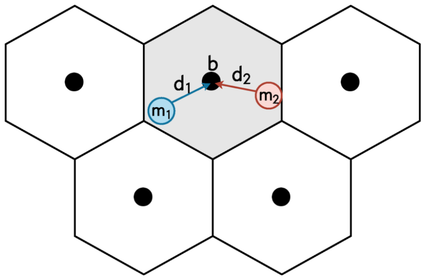

Fig. 1. Visualizing generalized deduplication. Similar chunks map to the same base, but different deviations

denote that x is sampled uniform random from the set S. We denote the security parameter of cryptographic primitives as λ.

3.1 Generalized deduplication

A typical deduplication strategy is to segment files into chunks and look for chunks that appear more than once in the file or that match a chunk from previously stored files [33]. We will use the terms “file” and “chunk” interchangeably, since they both denote the object to be deduplicated, albeit at different granularities. In many settings, this works well. However, deduplication is traditionally limited by the fact that it requires chunks to be identical. Thus, even if just one bit differs between two chunks, they are stored independently.

Generalized deduplication can alleviate this issue [25,31] by allowing a systematic deduplication of near-identical chunks. The idea is to employ a transformation function ϕ which takes as input an n-bit chunk m and decomposes it into a k-bit *base*, b, and an l-bit *deviation*, d. Concretely, $\phi : \{0,1\}^n \to \{0,1\}^k \times \{0,1\}^l$ with $\phi(\mathsf{m}) = (\mathsf{b}, \mathsf{d})$, and there exists a function $\phi^{-1} : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^n$ that reconstructs a data chunk from a given base and a deviation, *i.e.,* for any chunk $\mathsf{m} \in \{0,1\}^n$ it holds that $\phi^{-1}(\phi(\mathsf{m})) = \mathsf{m}$. Fig. 1 visualizes the partition induced by ϕ on the chunk space: two similar chunks, m_1 and m_2, are mapped to the same base b, with different deviations ($\mathsf{d}_1 \neq \mathsf{d}_2$). For usability, we require $k > l$, meaning that the base contains most of the chunk's information and the deviation is small. In our example of portrait pictures with the Eiffel tower, the background image corresponds to the (common) base, while the person's figure is the (file-dependent) deviation. The bases can then be deduplicated, so that each base is stored only once. Finally, the deviation is stored alongside a reference to the base, which ensures that each file can be reconstructed without any loss of information. Classic deduplication can be obtained by letting the transformation function be the identity function, so bases are the original chunks and the deviation is empty.

We follow the approach of [25,31] and set ϕ to be a Hamming code [16]. In this case, the transformation function “decodes” the chunk m to obtain its base b, and then the deviation d is derived by finding the difference between the original chunk and the encoding of its base. As an example, for 4 KB chunks the one-bit error-correction capability of the Hamming means that 32768 different, but similar, chunks will have the same base. In general, generalized deduplication can thus match more bases than classic deduplication, as shown theoretically in [31] and achieve a better compression, as evaluated experimentally in [25,30,32].

3.2 Revealing encryption (RE)

The first ingredient in RE schemes is an *authorized function* f from a set of inputs $\mathcal{M}^n$ to a set of output values $\mathcal{V}$. Formally, $f : \mathcal{M}^n \to \mathcal{V}$. Examples of authorized functions are $f = \max\{\cdots\}$ and $f = \texttt{deduplicate}(\cdot, \cdot)$ (see Definition 7). Revealing encryption schemes are built around the chosen *authorized function* and are defined as follows.[6]

Definition 1 (Revealing encryption (RE) [15]). *Let $f : \mathcal{M}^n \to \mathcal{V}$ be an n-ary authorized function. An RE scheme for the function f is a tuple of algorithms $\mathsf{RE}_f = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Reveal})$ defined as follows:*

$\mathsf{Setup}(1^\lambda)$**:** *on input security parameter λ (in unary), this randomized algorithm outputs a secret key sk and some public parameters pp. pp are input to all the following algorithms even when not explicitly written.*

$\mathsf{Enc}(\mathsf{sk}, \mathsf{m})$**:** *on input the secret key sk and a message $\mathsf{m} \in \mathcal{M}$, this randomized algorithm outputs a ciphertext c.*

$\mathsf{Reveal}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$**:** *on input the public parameters pp and n ciphertexts this deterministic algorithm outputs a value $v \in \mathcal{V}$.*

Revealing correctness essentially states that the output of Reveal evaluated on a set of ciphertexts should equal the output of the authorized function f evaluated on the corresponding set of plaintexts. Security is defined using a *leakage function* that sizes what the adversary learns from evaluating Reveal on any collection of n ciphertexts.

Definition 2 (Leakage function and optimal RE_f). *A function $L : \mathcal{M}^* \to \{0,1\}^*$ is a leakage function for a revealing encryption scheme RE_f if, given any tuple of input values, L outputs the information leaked by RE_f when running Reveal on any possible size-n subset of the corresponding ciphertexts. A RE_f is said to be optimal if it leaks precisely what is required by the functionality, that is:*

$$L(\mathsf{m}_1, \ldots, \mathsf{m}_q) = \{f(\mathsf{m}_i\,;\ i \in S),\ S \subseteq [q], |S| = n\}.$$

Further details on RE and its security notion can be found in [10,15].

[6] To improve readability and have an homogeneous language when extending RE to multiple users (MKRE), we use Reveal instead of Eval in [15]. Also, in Section 4 we will split the algorithm Setup from [15] into a global set up procedure, called Setup, and a user-dependent KeyGen.

4 Multi-key revealing encryption

We now introduce Multi-Key Revealing Encryption (MKRE), a cryptographic primitive that extends revealing encryption to handle functions evaluated on data encrypted using distinct secret keys. In addition to the standard algorithms of RE, MKRE includes KeyGen, needed to extract user specific key material from the common parameters, and Dec, that allows users to efficiently retrieve their data. After presenting the MKRE framework, we provide detailed discussions of our security notions in Section 4.1 and of the need for random oracles to tolerate user corruption in Section 4.2.

Definition 3 (Multi-key revealing encryption (MKRE)). *Let $f : \mathcal{M}^n \to \mathcal{V}$ be an n-ary authorized function. A multi-key revealing encryption scheme for the function f is a tuple of algorithms $\mathsf{MKRE}_f = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Reveal}, \mathsf{Dec})$ defined as follows:*

$\mathsf{Setup}(1^\lambda)$**:** *this randomized algorithm outputs a master public key mpk containing at least a description of the function f The mpk serves as public parameters and is implicitly input to all subsequent algorithms.*

$\mathsf{KeyGen}()$**:** *this randomized algorithm outputs a user secret key sk .*

$\mathsf{Enc}(\mathsf{sk}, \mathsf{m})$**:** *on input a secret key sk and a message $\mathsf{m} \in \mathcal{M}$, this randomized algorithm outputs a ciphertext c.*

$\mathsf{Reveal}(\mathsf{c}_1, \dots, \mathsf{c}_n)$**:** *on input the master public key mpk and n ciphertexts this deterministic algorithm outputs $v \in \mathcal{V}$.*

$\mathsf{Dec}(\mathsf{sk}, \mathsf{c})$**:** *on input a secret key sk and a ciphertext c, this deterministic algorithm returns a plaintext m.*

We depart from the convention of omitting the decryption algorithm in RE schemes. This is motivated by the fact that in many practical cases, *e.g.,* cloud storage, it is essential for users to be able to recover the plaintext data at a later point in time. We define two notions of correctness: one for the public revealing method and one for the secret decryption.

Definition 4 (MKRE_f revealing correctness). *A MKRE scheme MKRE_f satisfies* revealing correctness *if, for any n-tuple of messages $\mathsf{m}_1, \dots, \mathsf{m}_n \in \mathcal{M}^n$, for any n-tuple of keys $\mathsf{sk}_1, \dots, \mathsf{sk}_n$ generated by KeyGen, and for any $\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \mathsf{m}_i)$ with $i \in [n]$, it holds that*

$$\mathsf{Reveal}\left(\mathsf{c}_1, \dots, \mathsf{c}_n\right) = f\left(\mathsf{m}_1, \dots, \mathsf{m}_n\right)$$

with all but negligible probability. Note that the n ciphertexts may be encrypted under up to n different secret keys.

Definition 5 (MKRE_f decryption correctness). *A MKRE scheme MKRE_f satisfies* decryption correctness *if for any message $\mathsf{m} \in \mathcal{M}$ and for any secret key sk generated by KeyGen it holds that*

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, \mathsf{m})) = \mathsf{m}.$$

Next, we define security using the real versus ideal world paradigm.

Definition 6 (MKRE_f security). *Let λ be a security parameter, f a revealing function, MKRE_f a multi-key revealing encryption scheme for f with associated leakage function L. Consider the experiments $\mathsf{REAL}_{\mathcal{A}}^{\mathsf{MKRE}_f}(\lambda)$ and $\mathsf{IDEAL}_{\mathcal{A},\mathcal{S},L}^{\mathsf{MKRE}_f}(\lambda)$ depicted below, where $\mathcal{C}$ denotes the corruption oracle, $\mathcal{RO}$ the random oracle, $\mathcal{E}$ the encryption oracle, and prepending an $\mathcal{S}$ denotes a simulated oracle.*

```
$\mathsf{REAL}_{\mathcal{A}}^{\mathsf{MKRE}_f}(\lambda)$
$\mathsf{mpk} \leftarrow \mathsf{Setup}(1^\lambda)$
$b \leftarrow \mathcal{A}^{\mathcal{C}(\cdot),\mathcal{RO}(\cdot),\mathcal{E}(\cdot)}(\mathsf{mpk})$
```

```
$\mathsf{IDEAL}_{\mathcal{A},\mathcal{S},L}^{\mathsf{MKRE}_f}(\lambda)$
$\mathsf{mpk} \leftarrow \mathsf{Setup}(1^\lambda)$
$b \leftarrow \mathcal{A}^{\mathcal{SC}(\cdot),\mathcal{SRO}(\cdot),\mathcal{SE}(\cdot)}(\mathsf{mpk})$
```

We say that MKRE_f is a secure MKRE scheme with respect to L if, for all adversaries $\mathcal{A}$ that make at most $q = \mathsf{poly}(\lambda)$ queries, there exists a simulator $\mathcal{S}$ such that the output distributions of the two experiments described above are computationally indistinguishable, i.e.:

$$\mathsf{REAL}_{\mathcal{A}}^{\mathsf{MKRE}_f}(\lambda) \sim_c \mathsf{IDEAL}_{\mathcal{A},\mathcal{S},L}^{\mathsf{MKRE}_f}(\lambda)$$

We assume that oracles and simulators are stateful. Thus, whenever we query a new message, its leakage can be assumed to be the output of the reveal function evaluated against every previously queried message. To keep track of relevant information contained in the state, we use the following dictionaries:

$\mathcal{D}_{\mathsf{id}}$	: *Set of pairs (identity-identifier, user secret key),*
$\mathcal{D}_{\mathsf{corr}}$	: *Set of identity-identifiers corresponding to corrupted users,*
$\mathcal{D}_{\mathsf{ro}}$	: *Set of (input, output) pairs generated by queries to the random oracle,*
$\mathcal{D}_{\mathsf{enc}}$	: *Set of tuples (identity-identifier, message, ciphertext) generated by encryption queries.*

They are all empty at the beginning of the MKRE_f security game and are populated as shown in Fig. 2.

4.1 Discussion of our definitions

We now compare our security definitions for MKRE with the corresponding notions in RE. We first highlight the similarities, then motivate our changes.

Similarities. In the ideal world, the MKRE_f simulated encryption oracle has to produce ciphertexts having access only to the natural leakage inherent in the revealing encryption scheme and without seeing the actual query. The structure of our simulator for MKRE is similar to the RE simulator [15]: In order to generate a ciphertext for a queried message m, $\mathcal{S}$ is *not* given m; instead it has access to the *leakage* produced by m against all the previously encrypted messages ($L(M)$ in Fig. 2). This exploits the *revealing* feature natural to RE schemes and does not disclose more than what the simulator can learn by applying the public Reveal procedure to the ciphertexts. We remark that giving $\mathcal{S}$ access to $L(M)$ is crucial to ensure that real world ciphertexts only reveal the desired information about the plaintexts and nothing else.

```
$\mathcal{KG}(\mathsf{id})$
1 :  if $(\mathsf{id},\cdot) \notin \mathcal{D}_{\mathsf{id}}$
2 :    $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}()$
3 :    $\mathcal{D}_{\mathsf{id}} \leftarrow \mathcal{D}_{\mathsf{id}} \cup (\mathsf{id}, \mathsf{sk}_{\mathsf{id}})$
4 :  return $\mathsf{sk}_{\mathsf{id}} : (\mathsf{id}, \mathsf{sk}_{\mathsf{id}}) \in \mathcal{D}_{\mathsf{id}}$

$\mathcal{C}(\mathsf{id})$
1 :  $\mathcal{D}_{\mathsf{corr}} \leftarrow \mathcal{D}_{\mathsf{corr}} \cup \mathsf{id}$
2 :  return $\mathcal{KG}(\mathsf{id})$

$\mathcal{RO}(\mathsf{x})$
1 :  if $(\mathsf{x},\cdot) \notin \mathcal{D}_{\mathsf{ro}}$
2 :    $h \xleftarrow{\$} \{0,1\}^N$
3 :    $\mathcal{D}_{\mathsf{ro}} \leftarrow \mathcal{D}_{\mathsf{ro}} \cup (\mathsf{x}, h)$
4 :  return $h : (\mathsf{x}, h) \in \mathcal{D}_{\mathsf{ro}}$

$\mathcal{E}(\mathsf{id}, \mathsf{m})$
1 :  $\mathsf{c} \leftarrow \mathsf{Enc}^{\mathcal{RO}(\cdot)}(\mathcal{KG}(\mathsf{id}), \mathsf{m})$
2 :  $\mathcal{D}_{\mathsf{enc}} \leftarrow \mathcal{D}_{\mathsf{enc}} \cup (\mathsf{id}, \mathsf{m}, \mathsf{c})$
3 :  return $\mathsf{c}$
```

```
$\mathcal{SC}(\mathsf{id})$
1 :  $\mathcal{D}_{\mathsf{corr}} \leftarrow \mathcal{D}_{\mathsf{corr}} \cup \mathsf{id}$
2 :  $\mathcal{D}^{(\mathsf{id})}_{\mathsf{enc}} \leftarrow \{(\mathsf{id}', \mathsf{m}, \mathsf{c}) \in \mathcal{D}_{\mathsf{enc}} : \mathsf{id}' = \mathsf{id}\}$
3 :  $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathcal{S}(\mathsf{mpk}, \mathsf{id}, \mathcal{D}^{(\mathsf{id})}_{\mathsf{enc}})$
4 :  return $\mathsf{sk}_{\mathsf{id}}$

$\mathcal{SRO}(\mathsf{x})$
1 :  $\tilde{\mathsf{m}} \leftarrow \mathcal{S}(\mathsf{mpk}, \mathsf{x})$
2 :  $M \leftarrow M \cup \{\tilde{\mathsf{m}}\}$
3 :  $h \leftarrow \mathcal{S}(\mathsf{mpk}, L(M))$
4 :  return $h$

$\mathcal{SE}(\mathsf{id}, \mathsf{m})$
1 :  if $\mathsf{id} \in \mathcal{D}_{\mathsf{corr}}$
2 :    return $\perp$
3 :  $M \leftarrow M \cup \{\mathsf{m}\}$
4 :  $\mathsf{c} \leftarrow \mathcal{S}(\mathsf{mpk}, \mathsf{id}, L(M))$
5 :  $\mathcal{D}_{\mathsf{enc}} \leftarrow \mathcal{D}_{\mathsf{enc}} \cup (\mathsf{id}, \mathsf{m}, \mathsf{c})$
6 :  return $\mathsf{c}$
```

Fig. 2. Real vs ideal world: $\mathcal{KG}$ (key generation subroutine, $\mathcal{A}$ cannot query directly), $\mathcal{C}$ (corruption oracle), $\mathcal{SC}$ (simulated corruption oracle), $\mathcal{RO}$ (random oracle), $\mathcal{SRO}$ (simulated random oracle), $\mathcal{E}$ (encryption oracle), $\mathcal{SE}$ (simulated encryption oracle). In line 3 of $\mathcal{SRO}$ and in line 4 of $\mathcal{SE}$, $L(M) = L(\mathsf{m}_1, \mathsf{m}_2, \ldots, \mathsf{m}_n)$ denotes the leakage produced from all messages encrypted so far, including the new message ($\tilde{\mathsf{m}}$ or m).

Minor additions. Our system involves multiple users, and we let the adversary $\mathcal{A}$ corrupt some and learn their secret keys. Our $\mathcal{SE}$ rejects encryption queries for corrupted users. This limitation is done solely to simplify the model and does not restrict the adversary, who can encrypt messages for corrupted parties without interacting with the encryption oracle. (Yet, $\mathcal{A}$ needs to query the random oracle, as we discuss in the next paragraph.)

Major changes. In our security definition for MKRE, both the adversary and the encryption oracle have access to the random oracle. This is a drastic change from current RE models and has been done to enable a formal handling of schemes that adopt deterministic cryptographic primitives (handy in contexts like the one of generalized deduplication we are interested in). Concretely, we propose a new way to deal with the issues faced in several previous works when trying to formally model deterministic encryption and related notions. The thorny point in the model is that MKRE cannot satisfy semantic security, essentially for the same reason as deterministic encryption. Intuitively, the adversary can encrypt all possible messages "locally" (*i.e.,* without interacting with oracles) and then compare the list of ciphertexts with the output of the encryption oracle (resp. computing Reveal in MKRE) to distinguish between the real and the ideal worlds. In all previous work of which we are aware, this problem is circumvented by introducing assumptions on the distribution of the messages. For instance, [6] and subsequent works [5,18] rely on the notion of confidentiality for *unpredictable* messages only (privacy under chosen-distribution attack [4]). There, instead of letting $\mathcal{A}$ query the encryption oracle directly with a chosen message m, the common paradigm is to make $\mathcal{A}$ output a distribution (on the message space) with large enough entropy. The encryption oracle receives the adversary's distribution and samples messages accordingly. We recognize that this is a meaningful way to deal with the problem, however, we find the resulting definitions to be artificial, less intuitive and cumbersome to work with.

This motivates us to put forth a different approach that follows the "intuitive" security guarantees provided by deterministic encryption (and related notions). Concretely, we refer to the guarantee that a ciphertext c discloses no information about the plaintext it encloses, *unless* the adversary has already seen c. In the standard model, since the adversary can perform encryptions locally, without informing any oracle, it is not possible to meaningfully reason about which messages $\mathcal{A}$ has encrypted or not. In contrast, in the random oracle model, the adversary must interact with some oracle (and thus, the simulator) to perform encryptions. Intuitively, the random oracle model makes it possible to "extract" from the adversary the set of messages that $\mathcal{A}$ is encrypting "locally". In MKRE, at every random oracle query we allow the simulator to guess what $\mathcal{A}$ is doing and extract information from it. Concretely, at every random oracle query, $\mathcal{S}$ can use x, the input to $\mathcal{RO}$, to guess a message $\tilde{\mathsf{m}}$, which it adds to the set of simulated messages M and can get the leakage for. We believe that this approach gives the simulator a "more fair" task than what is required by previous models, e.g., [6]. In particular, our model lets the simulator learn the same information the adversary can derive locally by computing f (the authorized function) on messages encrypted both locally and via the encryption oracle.

We stress that without the extra leakage during random oracle queries (line 3 in $\mathcal{SRO}$), the definition could not be instantiated due to the following trivial attack. First, the adversary makes one query, m_1, to the encryption oracle and receives c_1. Then, $\mathcal{A}$ computes "local" encryptions of other message m_i for $i \in [2, n]$ such that $f(\mathsf{m}_1, \ldots, \mathsf{m}_n) = v$ for a random value $v \in \mathcal{V}$ in the range of f. In the real world, the ciphertexts satisfy $\mathsf{Reveal}(\mathsf{c}_1, \ldots, \mathsf{c}_n) = v$. In the ideal world (without $\mathcal{SRO}$ leakage), instead, the simulator gets no leakage, since the encryption oracle is queried only once. When simulating the random oracle, $\mathcal{S}$ has no access to the global leakage $L(\mathsf{m}_1, \ldots, \mathsf{m}_n) = v$. As a result, no simulator can guarantee that the "locally encrypted" ciphertexts will match the random value v, so the adversary distinguishes the two worlds with non-negligible probability $1 - 1/|\mathcal{V}|$.

4.2 On the need for random oracles to tolerate corruptions

Any secure MKRE construction that tolerates corruption of users must rely on *adaptive primitives* such as non-committing encryption [9]. To give an intuition of why this is the case, consider the following trivial attack. First, the adversary asks for encryptions of many random messages m_i for the same identity id^*. Then $\mathcal{A}$ corrupts id^*. In the real world, the secret key for id^* is chosen at the beginning of the game, thus each ciphertext c_i received by $\mathcal{A}$ is indeed an encryption of m_i under the same $\mathsf{sk}_{\mathsf{id}^*}$. On the other hand, in the ideal world the simulator does not have access to the random messages during the encryption queries, and therefore returns ciphertexts that are independent of the m_i. Only upon corruption the simulator learns what message m_i each ciphertext should decrypt to (line 3 in $\mathcal{SC}$). At this moment, $\mathcal{S}$ has to come up with *one* secret key that "explains" all of the previously generated ciphertexts (*i.e.,* such that $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}^*}, \mathsf{c}_i) = \mathsf{m}_i$). Due to the incompressibility of random data, such a secret key must be at least as long as the number of encrypted messages. This means that in the standard model there exists no secure construction of an MKRE scheme that tolerates user corruption *and* has fixed sized key. Thus, efficient MKRE schemes exist only in the random oracle model. This incompressibility issue is known for encryption schemes with adaptive properties, *e.g.,* [2,7,24].

5 MKRE for generalized deduplication of private data

In this section, we show how to instantiate MKRE for generalized deduplication of private data. In Section 5.1 we propose a high-level compiler that combines well-established cryptographic primitives in a black-box way into an MKRE scheme for generalized deduplication. In Section 5.2 we describe how to instantiate our compiler and its building blocks using solely random oracles. This concrete construction is simulatable secure in the model introduced in Section 4. Due to space limitations, we defer to the Appendix B a discussion on how to generically turn our compiler (and thus our explicit construction) from the ad-hoc setting, where each user generates keys independently, to a centralized setting. The latter setting provides an additional layer of security against nosy servers and may be useful in systems that employ a key distribution center.

Setup assumptions. Throughout this section we assume that the function for generalized deduplication ϕ is given and known to all parties involved in the scheme (in our experiments ϕ is a Hamming code, see Section 6 for further details). In detail, we require ϕ to be a function that maps a chunk $\mathsf{m} \in \{0,1\}^n$ into a base and a deviation:

$$\phi(\mathsf{m}) = (\phi_1(\mathsf{m}), \phi_2(\mathsf{m})) = (\mathsf{b}, \mathsf{d}) \in \{0,1\}^k \times \{0,1\}^l.$$

For recovery purposes, ϕ needs to be invertible, *i.e.,* there exists a function $\phi^{-1} : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^n$ such that $\phi^{-1}(\phi(\mathsf{m})) = \mathsf{m}$ for any chunk $\mathsf{m} \in \{0,1\}^n$. Finally, in our use case the size of bases is larger than the security parameter, *i.e.,* $k > \lambda$. We are interested in the following authorized function.

Definition 7 (Authorized function for generalized deduplication). *We define the authorized (revealing) function for generalized deduplication as* $f = \texttt{deduplicate} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ *where*

$$f(\mathsf{m}_1, \mathsf{m}_2) = \texttt{deduplicate}(\mathsf{m}_1, \mathsf{m}_2) = \begin{cases} 1 \text{ if } \phi_1(\mathsf{m}_1) = \phi_1(\mathsf{m}_2) \\ 0 \text{ otherwise} \end{cases} .$$

We note that f returns 1 (deduplication is possible) only on similar chunks, *i.e.,* chunks with the same base b, independently of the deviations.

5.1 Our MKRE compiler

Our MKRE compiler for generalized deduplication of private data uses three building blocks: a message-locked encryption scheme, a deterministic secret key encryption scheme, and a randomized secret key encryption scheme. The authorized function is $f = \texttt{deduplicate}$ (see Definition 7).

Definition 8 (An MKRE compiler for generalized deduplication). *Let* $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ *be the authorized function for generalized deduplication from Definition 7. Our MKRE scheme for generalized deduplication is defined by the following algorithms:*

$\mathsf{Setup}(1^\lambda)$: *Set up a message-locked encryption scheme* $ML = (\texttt{mlSetup}, \texttt{mlKD}, \texttt{mlE}, \texttt{mlD})$*, a secret-key deterministic encryption scheme* $DetE = (\texttt{detKG}, \texttt{detE}, \texttt{detD})$*, and a randomized (secret key) encryption scheme* $RandE = (\texttt{randKG}, \texttt{randE}, \texttt{randD})$*. Publish as* mpk *all public parameters of the schemes.*

$\mathsf{KeyGen}()$: *Run* $\texttt{detk}_{\mathsf{id}} \leftarrow \texttt{detKG}()$ *and* $\texttt{randkey}_{\mathsf{id}} \leftarrow \texttt{randKG}()$*. Let* $\mathsf{sk}_{\mathsf{id}} \leftarrow (\texttt{detk}_{\mathsf{id}}, \texttt{randkey}_{\mathsf{id}})$ *and return* $\mathsf{sk}_{\mathsf{id}}$*.*

$\mathsf{Enc}(\mathsf{sk}_{\mathsf{id}}, \mathsf{m})$: *Parse the secret key as* $(\texttt{detk}_{\mathsf{id}}, \texttt{randkey}_{\mathsf{id}})$ *and apply the generalized deduplication transformation to the data record* m *to obtain its base and deviation:* $\phi(\mathsf{m}) = (b, d)$*. Perform the following steps:*
1. $\mathsf{mk} \leftarrow \texttt{mlKD}(b)$*, (generate a base-derived key using ML),*
2. $\beta \leftarrow \texttt{mlE}(\mathsf{mk}, b)$*, (ML encrypt the base using the base-derived key),*
3. $\gamma \leftarrow \texttt{detE}_{\texttt{detk}_{\mathsf{id}}}(\mathsf{mk})$*, (encrypt the base-derived key using DetE),*
4. $\delta \leftarrow \texttt{randE}_{\texttt{randkey}_{\mathsf{id}}}(d)$*, (encrypt deviation using RandE).*

Return the ciphertext $\mathsf{c} = (\beta, \gamma, \delta)$*.*

$\mathsf{Reveal}(\mathsf{c}_1, \mathsf{c}_2)$: *Parse* $\mathsf{c}_i = (\beta_i, \gamma_i, \delta_i)$ *for* $i \in \{1,2\}$*. If the first ciphertext components are equal, i.e.,* $\beta_1 = \beta_2$*, output 1, otherwise, output 0.*

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{c})$: *Parse the secret key* $\mathsf{sk}_{\mathsf{id}}$ *as* $(\texttt{detk}_{\mathsf{id}}, \texttt{randkey}_{\mathsf{id}})$ *and the ciphertext* c *as* (β, γ, δ)*. Perform the following steps:*
1. $\mathsf{mk} \leftarrow \texttt{detD}_{\texttt{detk}_{\mathsf{id}}}(\gamma)$*, (DetE decrypt the base-derived key);*
2. $b \leftarrow \texttt{mlD}(\mathsf{mk}; \beta)$*, (Recover the base using ML);*
3. $d \leftarrow \texttt{randD}_{\texttt{randkey}_{\mathsf{id}}}(\delta)$*, (RandE decrypt the deviation)*

Return the plaintext record $\mathsf{m} = \phi^{-1}(b, d)$*.*

Fig. 3 visualizes the workflow of the algorithms in our high-level MKRE scheme of Definition 8. Next, we show the correctness of the Reveal and the Dec procedures. Security is proven after we describe a concrete instantiation of the building blocks using a random oracle in Section 5.2.

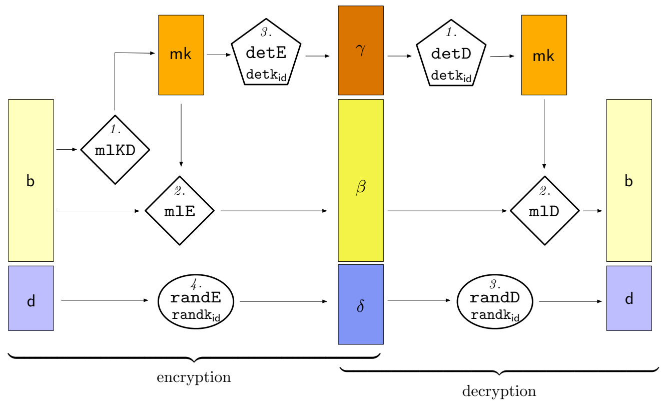

Fig. 3. Visual representation of the workflow of our MKRE construction for generalized deduplication (notation according to Definition 8).

Revealing correctness. Following Definition 4 for the revealing function f of Definition 7, we now prove that, for any pair of messages m_1, $\mathsf{m}_2 \in \{0,1\}^n$ and for any pair of keys $\mathsf{sk}_1, \mathsf{sk}_2$ (potentially belonging to different users), the output of Reveal evaluated on the ciphertexts $\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \mathsf{m}_i)$, $i \in [2]$ equals the output of the authorized function from Definition 7. In detail, we show that $\Pr[\mathsf{Reveal}(\mathsf{c}_1, \mathsf{c}_2) = f(\mathsf{m}_1, \mathsf{m}_2)] \geq 1 - \mathsf{negl}(\lambda)$, where the probability is taken over the choice of keys and the random coins of the algorithms. The functions are defined as:

$$\mathsf{Reveal}(\mathsf{c}_1, \mathsf{c}_2) = \begin{cases} 1 \text{ if } \beta_1 = \beta_2, \text{ where } \mathsf{c}_i = (\beta_i, \gamma_i, \delta_i) \\ 0 \text{ otherwise} \end{cases}$$

$$f(\mathsf{m}_1, \mathsf{m}_2) = \begin{cases} 1 \text{ if } \phi_1(\mathsf{m}_1) = \phi_1(\mathsf{m}_2), \text{ i.e. } \mathsf{b}_1 = \mathsf{b}_2 \\ 0 \text{ otherwise} \end{cases}$$

We distinguish two cases according to the output of f.

$\underline{f(\mathsf{m}_1, \mathsf{m}_2) = 0}$. By definition of f, the event $f(\mathsf{m}_1, \mathsf{m}_2) = 0$ corresponds to the event $\mathsf{b}_1 \neq \mathsf{b}_2$ (recall that $\phi_1(\mathsf{m}_i) = \mathsf{b}_i$). We prove that Reveal outputs 0 as well (with all but negligible probability). This holds since the β component of a ciphertext is generated deterministically from b_i, and by assumption $\mathsf{b}_1 \neq \mathsf{b}_2$. In detail, $\beta_i = \mathtt{mlE}(\mathsf{mk}_i, \mathsf{b}_i)$. Now since $\mathsf{b}_1 \neq \mathsf{b}_2$, it holds that $\mathtt{mlKD}(\mathsf{b}_1) \rightarrow \mathsf{mk}_1 \neq \mathsf{mk}_2 \leftarrow \mathtt{mlKD}(\mathsf{b}_2)$ with all but negligible probability. Likewise, $\mathtt{mlE}(\mathsf{mk}_1; \mathsf{b}_1) \rightarrow \beta_1 \neq \beta_2 \leftarrow \mathtt{mlE}(\mathsf{mk}_2; \mathsf{b}_2)$ also holds with overwhelming probability.

$\underline{f(\mathsf{m}_1, \mathsf{m}_2) = 1}$. By definition of f, the event $f(\mathsf{m}_1, \mathsf{m}_2) = 1$ corresponds to the event $\mathsf{b}_1 = \mathsf{b}_2$. We prove that in this case Reveal always outputs 1 as well. From the assumption that $\mathsf{b}_1 = \mathsf{b}_2$, it follows that $\mathtt{mlKD}(\mathsf{b}_1) = \mathtt{mlKD}(\mathsf{b}_2) =: \mathsf{mk}$, *i.e.,* the two chunks lead to the same message-locked key. In addition, given that message-locked encryption is deterministic, it holds that $\mathtt{mlE}(\mathsf{mk}; \mathsf{b}_1) = \beta_1 = \mathtt{mlE}(\mathsf{mk}; \mathsf{b}_2) = \beta_2 =: \beta$.

Decryption correctness. Decryption correctness naturally follows from the correctness of the primitives employed as building blocks.

5.2 Our MKRE instantiation with random oracles

We now describe how to implement the components of our MKRE construction of Definition 8 in the random oracle model.

Setup & Key Generation. The Setup procedure outputs a description of four hash functions H_1, H_2, H_3, and H_4 in mpk. Essentially, we use one hash function per encryption step, each modeled as a random oracle: $H_1 : \{0,1\}^k \to \{0,1\}^\lambda$; $H_2 : \{0,1\}^\lambda \to \{0,1\}^k$; $H_3 : \{0,1\}^\lambda \times \{0,1\}^k \to \{0,1\}^k$; $H_4 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^l$. All four hash functions can be instantiated with a single random oracle by prepending the index to the input. A user's secret key is *one* random string $\mathsf{sk_{id}} = \mathtt{detk}_\mathsf{id} = \mathtt{randkey}_\mathsf{id} \xleftarrow{\$} \{0,1\}^\lambda$.

Message-locked encryption. This primitive is instantiated using the two hash functions H_1 and H_2. The base-derived key is generated as

$$\mathsf{mk} \leftarrow \mathtt{mlKD}(\mathsf{b}) := H_1(\mathsf{b}).$$

The message-locked encryption of b is then computed as

$$\beta \leftarrow \mathtt{mlE}(\mathsf{mk}, \mathsf{b}) := H_2(\mathsf{mk}) \oplus \mathsf{b}.$$

Finally, the ciphertext β can be decrypted with mk as

$$\mathsf{b} \leftarrow \mathtt{mlD}(\mathsf{mk}; \beta) := H_2(\mathsf{mk}) \oplus \beta.$$

Deterministic encryption. This primitive is instantiated using H_3. Recall that in the random oracle $\mathtt{detk}_\mathsf{id} = \mathsf{sk_{id}} \in \{0,1\}^\lambda$. To implement deterministic encryption of the message-locked key mk in the random oracle model (which cannot be inverted), we must depart slightly from the abstract construction in the previous section and use the encryption of the base β "as IV":

$$\gamma \leftarrow \mathtt{detE}_{\mathtt{detk}_\mathsf{id}}(\mathsf{mk}) := H_3(\mathsf{sk_{id}}, \beta) \oplus \mathsf{mk}.$$

The decryption of a deterministic ciphertext γ (using the "random IV" β) is performed as

$$\mathsf{mk} \leftarrow \mathtt{detD}_{\mathtt{detk}_\mathsf{id}}(\mathsf{mk}) := H_3(\mathsf{sk_{id}}, \beta) \oplus \gamma.$$

Randomized encryption. This primitive is instantiated using H_4. Recall that in the random oracle $\mathtt{randkey}_\mathsf{id} = \mathsf{sk_{id}} \in \{0,1\}^\lambda$. The randomized encryption is peformed as:

$$\delta \leftarrow \mathtt{randE}_{\mathtt{randkey}_\mathsf{id}}(\mathsf{d}) := (\delta_1, \delta_2),$$

where $\delta_1 \xleftarrow{\$} \{0,1\}^\lambda$ and $\delta_2 := (H_4(\mathsf{sk_{id}}, \delta_1) \oplus \mathsf{d})$. The randomized decryption is performed as:

$$\mathsf{d} \leftarrow \mathtt{randD}_{\mathtt{randkey}_\mathsf{id}}(\delta) := H_4(\mathsf{sk_{id}}, \delta_1) \oplus \delta_2.$$

Analysis of construction. Correctness of the above algorithms can be easily verified by inspection. We can now prove the security of our scheme.

Theorem 1. *The MKRE construction of Section 5.2 is simulation secure in our framework (Definition 6) in the random oracle model.*

Due to space constraints, we only provide a brief overview of the core parts of the proof, which is how to deal with random oracle and corruption queries. The full proof is given in Appendix A. In the case of queries to H_1, our model lets the simulator $\mathcal{S}$ learn the adversary's queries to the oracle. If $H_1(\mathsf{x})$ has not yet been initialized, the simulator has the chance to make a guess, $\tilde{\mathsf{m}}$, for the chunk corresponding to the query (step 1 in $\mathcal{SRO}$, see Fig. 2). Concretely, $\mathcal{S}$ asks for $\tilde{\mathsf{m}} = \phi^{-1}(\mathsf{x}, 0)$, and therefore $\mathcal{S}$ learns whether any file with the same base has been queried to $\mathcal{SE}$ previously (thanks to leakage received in step 3, see Fig. 2). The simulator uses the leakage to identify the list of ciphertexts "matching" the queried base x and can set its answer consistently. In case of corruption queries, $\mathcal{S}$ needs to produce a secret key which explains all of the previously produced encryptions for the corrupted identity. To do so, $\mathcal{S}$ receives the list of such ciphertexts and the corresponding plaintexts (line 2 of $\mathcal{SC}$ in Fig. 2). Then $\mathcal{S}$ can pick a random key $\mathsf{sk}_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\lambda$, and for all $(\mathsf{m}, \mathsf{c}) \in \mathcal{D}_{\mathsf{enc}}^{(\mathsf{id})}$, it can program the random oracles to match the expected output. In case H_3 or H_4 were initialized before the corruption query was made the simulator fails in this task and aborts, but this happens with negligible probability.

6 Proof-of-concept Implementation and Evaluation

Having obtained an understanding of the security of our MKRE scheme in the random oracle model, it is interesting to evaluate its practical merits. To do this, we developed a proof-of-concept implementation of the instantiation presented in Section 5.2. Our implementation follows the software architecture of Nielsen *et al.* [25] and adopts the Hamming code also used in [31] as an example of a generalized deduplication transformation. We run our experiments with parameter $l \in \{13, 14, 15, 16\}$ for the Hamming code. A Hamming code has a codeword length of $n = 2^l - 1$ bits and a message length of $k = 2^l - l - 1$ bits. To ensure that the data chunks always align with the data's byte boundaries, we use data chunks of $2^l = n + 1$ bits. The transformation function ϕ then operates as follows. The first n bits are decoded using the Hamming code, providing the k bit base. Then, the last bit and the l-bit syndrome of the Hamming code (representing the location of a single bit difference between the reencoded base and the original chunk) are concatenated to form a deviation of $l + 1$ bits. As a result, our experiments with generalized deduplication transformation based on the Hamming code will ingest chunks with a size between 1 KB and 8 KB.

To instantiate the cryptographic primitives, we use the OpenSSL library [29]. In particular, we chose AES-128-CTR and HMAC-SHA-1 for encryption and hashing, respectively. This choice is somewhat arbitrary, and another choice may be made as desired. We use HMAC-SHA-1 to derive the message-locked keys

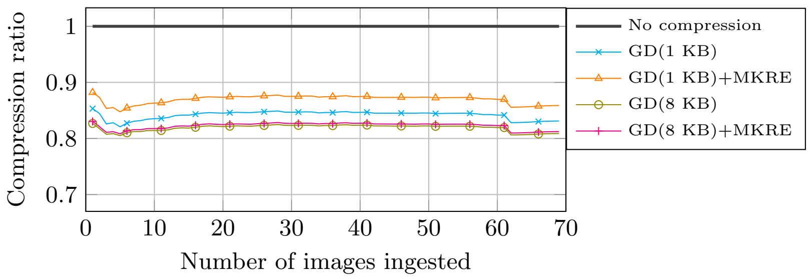

Fig. 4. Compression ratio, as images are stored on the server using generalized deduplication. Each image has a size of approximately 50 MB.

from bases. Deterministic encryption (and thus the message-locked encryption) is implemented by using an all-zero IV in AES. Randomized encryption utilizes a randomly chosen IV for AES, which is concatenated with the ciphertext. While this choice of algorithms is not equivalent to the random oracle model, we assume that this provides a "good enough" approximation to the random oracle behavior. This choice of algorithms allows us to implement an efficient practical solution. Our evaluation is run on a publicly available dataset of images provided by Plant Labs Inc. [27] consisting of 69 high-resolution .tiff images of approximately 50 MB each. As a result, our experiments deal with 3.5 GB of data. We compare the overall performance of applying MKRE for privacy-aware deduplication across users with the unsecured case of storing the unencrypted files directly on the server. The experiments are run on a Backblaze Storage Pod 6.0 with a 3.7 GHz Intel Xeon E5-1620 v2 Quad-Core CPU.

The first point of comparison is compression ratio $= \frac{\text{compressed size}}{\text{original size}}$. A compression ratio of 1 indicates that no compression is achieved, *i.e.,* that the compressed version takes up 100% of the original size. A low compression ratio is clearly desired. Fig. 4 compares the compression ratios achieved with encrypted and unencrypted generalized deduplication, using chunks of 1 KB and 8 KB. We remark that in [25], it was shown that, for the same data set, ZFS, a state-of-the-art file system utilizing classic deduplication [26], achieved negligible compression. Our experiments show that, as expected, the compression ratio of the encrypted version follows the unencrypted ratio closely. The only difference is that some overhead must be stored for the encrypted version, *e.g.,* from padding and IVs. In particular, we see that for 1 KB chunks the overhead is more significant, increasing the average compression ratio from 84% to 87%. On the other hand, the overhead has smaller impact on the ratio when the chunks are longer, as seen for 8 KB chunks, where the average increase is just from 81.8% to 82.2%. This validates that it indeed is possible to achieve similar compression levels when operating on encrypted data, as desired.

A second, equally important, point is the impact on system throughput. This is shown on Fig. 5. Obviously, adding an encryption step can only decrease the system throughput. In our experiments, the drop in throughput is larger for

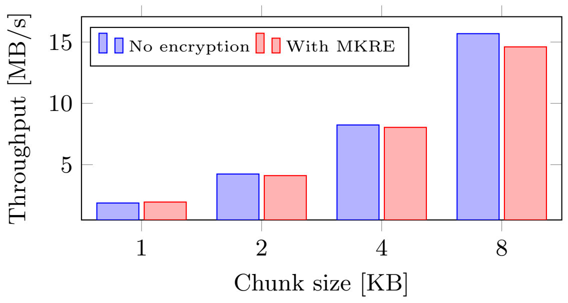

Fig. 5. System-level throughput of our proof-of-concept implementation storing files with generalized deduplication and MKRE.

larger chunks. Indeed, there is actually no significant difference in throughput when chunks are 1 KB. The worst case is seen for 8 KB chunks, where the throughput drops 6.9%. We note that although our system is not thoroughly optimized, throughputs on the order of $2 - 16$ MB/s is observed. This is promising, as with some optimization a throughput on the order of 100s of MB/s should be achievable. Such a throughput will allow our method to be deployed in real-time cloud systems, where the bottleneck then is the read/write speeds of the hard drives or SSDs.

7 Conclusions

In this work we tackled the challenge of providing private and space-efficient storage solutions for data outsourced by different users. Our solutions combine generalized deduplication techniques with a new cryptographic primitive called Multi-Key Revealing Encryption. As a result, our ciphertexts can be publicly clustered, *i.e.,* any third party can determine whether *any* two ciphertexts (potentially generated by different users) are "close" or not. This allows us to deduplicate similar files, thus compressing the encrypted data across users. We tested a practical implementation of our proposal on a real world dataset. These experiments show that, for a range of common deduplication chunk sizes, our privacy-aware solution achieves a compression ratio that is only 3% worse (in the worst case) than the one provided by generalized deduplication on unencrypted data, and the maximum loss in throughput is 6.9%.

We leave the investigation of other applications which benefit from MKRE as an interesting direction for future research.

Acknowledgements This work was partially financed by: the SCALE-IoT project (Grant No. DFF-7026-00042B) and FoCC (Grant No. DFF-6108-00169) granted by the Danish Council for Independent Research; the AUFF Starting Grant AUFF-2017-FLS-7-1; Aarhus University's DIGIT Centre; the strategic research area ELLIIT; the Concordium Blockhain Research Center, Aarhus University, Denmark; the European Research Council (ERC) under the European

Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC).

References

1. Agrawal, S., Clear, M., Frieder, O., Garg, S., O'Neill, A., Thaler, J.: Ad hoc multi-input functional encryption (2019), https://eprint.iacr.org/2019/356
2. Agrawal, S., Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption: New perspectives and lower bounds. In: CRYPTO 2013. pp. 500–518 (2013)
3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Annual International Cryptology Conference. pp. 535–552. Springer (2007)
4. Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: How to protect against bad randomness. In: ASIACRYPT 2009. pp. 232–249. Springer (2009)
5. Bellare, M., Keelveedhi, S., Ristenpart, T.: DupLESS: Server-Aided Encryption for Deduplicated Storage. USENIX Security Symposium pp. 179–194 (2013)
6. Bellare, Mihir; S, Keelveedhi; T, R.: Message-Locked Encryption and Secure Deduplication. EUROCRYPT (2013)
7. Bendlin, R., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: Lower and upper bounds for deniable public-key encryption. In: ASIACRYPT 2011. pp. 125–142 (2011)
8. Boyd, C., Davies, G.T., Gjøsteen, K., Raddum, H., Toorani, M.: Security Notions for Cloud Storage and Deduplication. In: ProvSec. pp. 347–365. Springer (2018)
9. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: ACM STOC. pp. 639–648 (1996)
10. Chenette, N., Lewi, K., Weis, S.A., Wu, D.J.: Practical order-revealing encryption with limited leakage. In: Fast Software Encryption 2016. pp. 474–493 (2016)
11. Chotard, J., Sans, E.D., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 703–732. Springer (2018)
12. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
13. Douceur, J.R., Adya, A., Bolosky, W.J., Simon, D., Theimer, M., Simon, P.: Reclaiming space from duplicate files in a serverless distributed file system. ICDCS 2002 pp. 617–624 (2002)
14. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F., Sahai, A., Shi, E., Zhou, H.: Multi-input functional encryption. In: EUROCRYPT 2014. pp. 578–602 (2014)
15. Haagh, H., Ji, Y., Li, C., Orlandi, C., Song, Y.: Revealing encryption for partial ordering. In: IMA International Conference on Cryptography and Coding. pp. 3–22. Springer (2017)
16. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal **29**(2), 147–160 (apr 1950)
17. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. vol. 26, pp. 191–224. Springer (2013)
18. Li, J., Chen, X., Li, M., Li, J., Lee, P.P., Lou, W.: Secure deduplication with efficient and reliable convergent key management. IEEE Transactions on Parallel and Distributed Systems **25**(6), 1615–1625 (2013)

19. Li, X., Li, J., Huang, F.: A secure cloud storage system supporting privacy-preserving fuzzy deduplication. Soft Computing (2016)
20. Libert, B., Ţiţiu, R.: Multi-client functional encryption for linear functions in the standard model from lwe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 520–551. Springer (2019)
21. Liu, J., Asokan, N., Pinkas, B.: Secure Deduplication of Encrypted Data without Additional Independent Servers. In: ACM CCS. pp. 874–885 (2015)
22. Liu, J., Duan, L., Li, Y., Asokan, N.: Secure deduplication of encrypted data: Refined model and new constructions. In: CT-RSA 2018. Springer (2018)
23. Michalevsky, Y., Joye, M.: Decentralized policy-hiding ABE with receiver privacy. In: López, J., Zhou, J., Soriano, M. (eds.) Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11099, pp. 548–567. Springer (2018)
24. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: CRYPTO 2002. pp. 111–126 (2002)
25. Nielsen, L., Vestergaard, R., Yazdani, N., Talasila, P., Lucani, D.E., Sipos, M.: Alexandria: A Proof-of-concept Implementation and Evaluation of Generalised Data Deduplication. In: IEEE GLOBECOM Workshop on Advances in Edge Computing (2019)
26. Oracle: What Is ZFS? https://docs.oracle.com/cd/E23823_01/html/819-5461/zfsover-2.html (2019), visited: 23/10-2019
27. Planet Labs Inc: Download samples of our, high resolution imagery, for monitoring, tasking and large area mapping. https://info.planet.com/download-free-high-resolution-skysat-image-samples/ (2019), visited: 17/06-2019
28. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS. vol. 71, pp. 72–75 (2014)
29. The OpenSSL Project: OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, visited: 23/09-2019
30. Vestergaard, R., Lucani, D.E., Zhang, Q.: A Randomly Accessible Lossless Compression Scheme for Time-Series Data. In: IEEE INFOCOM (2020)
31. Vestergaard, R., Zhang, Q., Lucani, D.E.: Generalized Deduplication: Bounds, Convergence, and Asymptotic Properties. In: IEEE GLOBECOM (2019)
32. Vestergaard, R., Zhang, Q., Lucani, D.E.: Lossless Compression of Time Series Data with Generalized Deduplication. In: IEEE GLOBECOM (2019)
33. Xia, W., Jiang, H., Feng, D., Douglis, F., Shilane, P., Hua, Y., Fu, M., Zhang, Y., Zhou, Y.: A Comprehensive Study of the Past, Present, and Future of Data Deduplication. Proceedings of the IEEE **104**(9), 1681–1710 (2016)
34. Zhao, Y., Chow, S.S.M.: Updatable Block-Level Message-Locked Encryption. IEEE Transactions on Dependable and Secure Computing (2019)

Appendix

A Proof of Theorem 1

Proof. We begin by observing that since our simulator $\mathcal{S}$ is stateful, there is no reason for $L(M)$ to return all the pairwise leakage every time. Therefore, *wlog*, we make the simplifying assumption that $L(M)$ returns the leakage of *the last message inserted in* M against every other message already present in M. The simulator deals with queries to the oracles in the following ways.

Random Oracle Queries $\mathcal{SRO}(\mathsf{x})$. Dealing with random oracle queries is perhaps the most important part of our simulation strategy, since this is where the simulator learns the messages that the adversary "encrypts locally" and has to produce consistent replies to the different oracle queries. Note that we only use the power of adding messages to the leakage set when replying to queries to H_1. In particular, the simulator answers to the different queries in the following way:

Queries to H_1 (Used in our construction to generate base-dependent key, the input x refers to a general base b): If $H_1(\mathsf{x})$ was already defined the simulator returns a value that is consistent with the previous answers. Otherwise, $H_1(\mathsf{x})$ has not yet been initalizated. In this case, the simulator first generates a guess message $\tilde{\mathsf{m}}$ using the input x as the base $\mathsf{b} = \mathsf{x}$, e.g., $\tilde{\mathsf{m}} = \phi^{-1}(\mathsf{b}, 0)$ (step 1 in $\mathcal{SRO}$, Fig. 2). Then, $\mathcal{S}$ learns whether any message with the same base b has been queried before to $\mathcal{SE}$ (thanks to leakage received in step 3, Fig. 2). If $L(M)$ is empty, the simulator has no additional information to initialize $H_1(\mathsf{x})$ and thus returns a random value $h \xleftarrow{\$} \{0,1\}^\lambda$. Otherwise, $L(M)$ is not empty, the simulator uses the leakage to identify the list of ciphertexts "matching" b, and randomly selects one $\mathsf{c} = (\beta, \gamma, \delta)$ from this list. The simulator computes the value $y = \beta \oplus \mathsf{b}$ and checks whether the oracle H_2 ever output y. If so, $\mathcal{S}$ programs H_1 to return the preimage of y under H_2, i.e., it sets $H_1(\mathsf{x}) = h$ for a $h \in \{0,1\}^\lambda$ satisfying $H_2(h) = y$. In case the oracle H_2 has not yet output the value y, $\mathcal{S}$ picks a random h and programs $H_2(h) = y$.

Queries to H_2 (Used in construction to encrypt base, the input x is a base-dependent key mk): If $H_2(\mathsf{x})$ was already defined the simulator just provides a consistent answer, otherwise: if $\mathsf{x} \neq H_1(\mathsf{b})$ for all previous queries to the random oracle H_1, pick a random value h, program $H_2(\mathsf{x}) = h$ and return h. If x is a previous output of H_1 e.g, if $\mathsf{x} = \mathsf{mk}$ for some base b, lookup the encryption β corresponding b and program the output of $H_2(\mathsf{x}) = \beta \oplus \mathsf{b}$.

Queries to H_3 (Used in construction to encrypt base-dependent key): If $H_3(\mathsf{x})$ was already defined the simulator just provides a consistent answer, otherwise it returns a random h and programs $H_3(\mathsf{x}) = h$.

Queries to H_4 (Used in construction to encrypt deviation): If $H_4(\mathsf{x})$ was already defined the simulator just provides a consistent answer, otherwise return a random h and program $H_4(\mathsf{x}) = h$.

Corruption queries $\mathcal{SC}(\mathsf{id})$. On this type of queries the simulator gets as input the set $\mathcal{D}_{\mathsf{enc}}^{(\mathsf{id})}$ that contains all of the message-ciphertexts pairs generated so far by the encryption oracle as replies to the adversary's queries for identity id. The simulator needs to produce a secret key $\mathsf{sk_{id}}$ which explains all ciphertexts c (which the simulator produced previously) as encryptions of the corresponding messages m. To do so, $\mathcal{S}$ picks a random key $\mathsf{sk_{id}} \xleftarrow{\$} \{0,1\}^\lambda$. Then, for all $(\mathsf{m},\mathsf{c}) \in \mathcal{D}_{\mathsf{enc}}^{(\mathsf{id})}$, the simulator parses $\mathsf{c} = (\beta,\gamma,\delta)$ and $\phi(\mathsf{m}) = (\mathsf{b},\mathsf{d})$. Finally $\mathcal{S}$ programs the random oracles as follows (repeating the steps for all $(\mathsf{m},\mathsf{c}) \in \mathcal{D}_{\mathsf{enc}}^{(\mathsf{id})}$:

1. If $H_1(\mathsf{b})$ is undefined, $\mathcal{S}$ picks random $\mathsf{mk} \xleftarrow{\$} \{0,1\}^\lambda$ and sets $H_1(\mathsf{b}) = \mathsf{mk}$; otherwise it sets $\mathsf{mk} = H_1(\mathsf{b})$.
2. If $H_2(\mathsf{mk})$ is undefined, the simulator programs $H_2(\mathsf{mk}) = \beta \oplus \mathsf{b}$.
3. If $H_3(\mathsf{sk_{id}},\beta)$ is currently undefined, $\mathcal{S}$ programs it to $H_3(\mathsf{sk_{id}},\beta) = \mathsf{mk} \oplus \gamma$; otherwise the simulator aborts –this happens when $H_3(\mathsf{sk_{id}},\beta)$ had been initializated previous to this corruption query.
4. If $H_4(\mathsf{sk_{id}},\delta_1)$ is currently undefined $\mathcal{S}$ programs $H_4(\mathsf{sk_{id}},\delta_1) = \mathsf{d} \oplus \delta_2$, otherwise –in case $H_4(\mathsf{sk_{id}},\delta_1)$ is already instantiated– the simulator aborts.

Note that the simulator may abort in step 3 and 4 only, and this happens uniquely if H_4 or H_3 were initialized before the corruption query. We argue that this $\mathcal{S}$ aborts only with negligible probability. In detail, in Step 3 the simulator aborts only if $H_3(\mathsf{sk_{id}},\beta)$ was defined prior to this corruption query (the same user might have encrypted multiple messages with the same b which would result in multiple, consistent definitions of the oracle on this point). Since $\mathsf{sk_{id}}$ is chosen at random for each corruption query, the probability that the adversary queries H_3 with precisely the value $(\mathsf{sk_{id}},\beta)$, without knowing the random string $\mathsf{sk_{id}} \in \{0,1\}^\lambda$, is negligible. A similar reasoning applies to H_4 in step 4: the probability that H_3 was already defined on any input of the form $(\mathsf{sk_{id}},\cdot)$ is negligible. Thus, $\mathcal{S}$ only aborts if there are two distinct ciphertexts in $\mathcal{D}_{\mathsf{enc}}^{(\mathsf{id})}$ with the same δ_1, which happens with negligbile probability because (as described below in the simulation of encryption queries), the value δ_1 is chosen uniformly at random and has length equal to the security parameter. Note also that, by construction of the simulator, if H_2 is already defined before the simulator runs step 5 on input mk, then the output of H_2 had already been defined to the exact same value. Therefore, by construction, if the simulation does not abort the ciphertexts now decrypt to the right message under the user key $\mathsf{sk_{id}}$, and the view of the adversary is therefore identical in the real and ideal world.

Encryption queries $\mathcal{SE}(\mathsf{id},\mathsf{m})$. We start by observing the properties of the output of the encryption oracle in the real world. Given any two encryption queries (id,m) and $(\mathsf{id}',\mathsf{m}')$ the outputs $\mathsf{c} = (\beta,\gamma,\delta)$ and $\mathsf{c} = (\beta',\gamma',\delta')$ have the following distributions (unless $\delta_1 = \delta_1'$ which only happens with negligible probability in the security parameter):

1. random and independent if $\phi_1(\mathsf{m}) \neq \phi_1(\mathsf{m}_2)$ (in particular, this implies the bases be different);

2. random and independent under the constraint $\beta = \beta'$ if $\phi_1(\mathsf{m}) = \phi_1(\mathsf{m}_2)$ and $\mathsf{id} \neq \mathsf{id}'$;
3. random and independent under the constraint $\beta = \beta'$ and $\gamma = \gamma'$ if $\phi_1(\mathsf{m}) = \phi_1(\mathsf{m}_2)$ and $\mathsf{id} = \mathsf{id}'$;

Our aim is to construct a simulator that produces ciphertexts satisfying the same distribution, ignoring the case of repeating δ_1's (which happens with negligible probability), thus achieving an indistinguishable view. Recall that $\mathcal{S}$ does not have access to the message m during encryption queries. However, $\mathcal{S}$ is allowed access to the leakage revealed by the ciphertexts themselves, i.e., the output of the leakage function $L(M)$ where M is the set of all messages which have been previously queried as part of encryption oracle queries, *plus* the messages "guessed" by the simulator when the adversary queries the random oracle. Remember that, for simplicity, our model prevents the adversary from querying the encryption oracle on corrupted identities (the oracle returns $\perp$, steps 1 and 2 in Fig. 2). Concretely, the simulator does the following. In any case, it picks a uniformly random $\delta_1 \xleftarrow{\$} \{0,1\}^\lambda$.

Then, if $L(M)$ is empty, $\mathcal{S}$ draws $\beta \xleftarrow{\$} \{0,1\}^k$ and $\gamma \xleftarrow{\$} \{0,1\}^\lambda$ at random. This happens if there were no previous encryptions with the same b. This perfectly simulates the distribution in case 1, since all of the entries of the ciphertexts are distributed at random and independently.

Otherwise, the simulator learns from $L(M)$ one (or more) previous ciphertexts that "match" the current query. Now, let $\mathsf{c}' = (\beta', \gamma', \delta')$ be one of these ciphertexts, and let $\mathsf{id}' \in \mathsf{ID}$ and m' denote the identity and message corresponding to that query (recall that $(\mathsf{id}', \mathsf{m}', \mathsf{c}') \in \mathcal{D}_{\mathsf{enc}}$). This means that the queried message and m' share the same base and therefore the simulator can set $\beta = \beta'$. In addition, if $\mathsf{id} \neq \mathsf{id}'$, $\mathcal{S}$ picks a random $\gamma \xleftarrow{\$} \{0,1\}^\lambda$, perfectly simulating the distribution in case 2.

Otherwise, $\mathsf{id} = \mathsf{id}'$, the messages are equal and so are the base-derived keys (and their ciphertext). Thus, $\mathcal{S}$ sets $\gamma = \gamma'$, fitting the expected distribution in case 3.

To sum up, we have presented a simulator that always replies with outputs distributed "in the same way" as the answers returned by the real world oracles. Thus, we proved the indistinguishability of its output from an execution of the "real" algorithms. This concludes the proof.

B Turning our MKRE constructions into centralized ones

Our instantiations of MKRE for generalized deduplication presented in Section 5 follow an ad-hoc approach were users of the system can generate their secret keys independently one from another. However, since the way we encrypt the base b into β is deterministic and public (no secret keys are involved in this step), our constructions are vulnerable to dictionary attacks. In detail, the server could encrypt data until "hitting" a ciphertext that corresponds to a target file. Once the match is found, the server learns the plaintext corresponding to the base of ciphertexts stored under the target deduplication string. While this attack does not directly imply that the server can efficiently break the confidentiality of the data uploaded by honest users, and it is in fact mitigated by random oracle queries in our construction, it constitutes a possible secrecy threat against the base points used for deduplication. While naïvely re-encrypting β under a user-specific secret key would solve the privacy issue, it destroys the deduplication capabilities (since the same β will be encrypted to different values by different users).

One way of addressing this problem could be to use a centralized approach, where users receive their secret keys from a key distribution authority. The idea is to deliver keys that are made of two parts: a global master secret key msk and user-dependent key material (denoted with the subscript $_{\mathsf{id}}$). The user-dependent key material is the same as in the original, ad-hoc constructions (Definition 8 and Section 5.2). The master secret key msk (shared among users) is then used to wrap an additional (deterministic) encryption layer around the ciphertext β, to obtain β'. In this way, since all users encrypt with the same msk and the encryption scheme is deterministic, equal bases will be sent to the same ciphertext[7] β'. This obviously preserves the deduplication functionality on encrypted data and provides an extra layer of security against an adversary who has no access to msk. Concretely, this centralized approach rules out guessing attacks from nosy servers so long the latter does not collude with a user. If msk is revealed to the adversary, the security of the centralized solution falls back to the security of the ad-hoc scheme.

[7] β' is a ciphertext of the ciphertext β, thus a deterministic encryption of a deterministic encryption of b.