EURECOM
Department of Networking and Security
Campus SophiaTech
CS 50193
06904 Sophia Antipolis cedex
FRANCE

Research Report RR-13-288

[1] EURECOM's research is partially supported by its industrial members: BMW Group Research & Technology, IABG, Monaco Telecom, Orange, SAP, SFR, ST Microelectronics, Swisscom, Symantec.

ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage

Augustth, 2013

Pasquale Puzio, Refik Molva, Melek Önen and Segio Loureiro

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : pasquale@secludit.com, refik.molva@eurecom.fr,
melek.onen@eurecom.fr, sergio@secludit.com

ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage

Pasquale Puzio, Refik Molva, Melek Önen and Sergio Loureiro

Abstract

With the continuous and exponential increase of the number of users and the size of their data, data deduplication becomes more and more a necessity for cloud storage providers. By storing a unique copy of duplicate data, cloud providers greatly reduce their storage and data transfer costs. The advantages of deduplication unfortunately come with a high cost in terms of new security and privacy challenges. We propose ClouDedup, a secure and efficient storage service which assures block-level deduplication and data confidentiality at the same time. Although based on convergent encryption, ClouDedup remains secure thanks to the definition of a component that implements an additional encryption operation and an access control mechanism. Furthermore, as the requirement for deduplication at block-level raises an issue with respect to key management, we suggest to include a new component in order to implement the key management for each block together with the actual deduplication operation. We show that the overhead introduced by these new components is minimal and does not impact the overall storage and computational costs.

1 Introduction

With the potentially infinite storage space offered by cloud providers, users tend to use as much space as they can and vendors constantly look for techniques aimed to minimize redundant data and maximize space savings. A technique which has been widely adopted is cross-user deduplication. The simple idea behind deduplication is to store duplicate data (either files or blocks) only once. Therefore, if a user wants to upload a file (block) which is already stored, the cloud provider will add the user to the owner list of that file (block). Deduplication has proved to achieve high space and cost savings and many cloud storage providers are currently adopting it. Deduplication can reduce storage needs by up to 90-95% for backup applications [11] and up to 68% in standard file systems [23].

Along with low ownership costs and flexibility, users require the protection of their data and confidentiality guarantees through encryption. Unfortunately, deduplication and encryption are two conflicting technologies. While the aim of deduplication is to detect identical data segments and store them only once, the result of encryption is to make two identical data segments indistinguishable after being encrypted. This means that if data are encrypted by users in a standard way, the cloud storage provider cannot apply deduplication since two identical data segments will be different after encryption. On the other hand, if data are not encrypted by users, confidentiality cannot be guaranteed and data are not protected against curious cloud storage providers.

A technique which has been proposed to meet these two conflicting requirements is convergent encryption [18, 25, 26] whereby the encryption key is usually the result of the hash of the data segment. Although convergent encryption seems to be a good candidate to achieve confidentiality and deduplication at the same time, it unfortunately suffers from various well-known weaknesses [15, 24] including dictionary attacks: an attacker who is able to guess or predict a file can easily derive the potential encryption key and verify whether the file is already stored at the cloud storage provider or not.

In this paper, we cope with the inherent security exposures of convergent encryption and propose ClouDedup, which preserves the combined advantages of deduplication and convergent encryption. The security of ClouDedup relies on its new architecture whereby in addition to the basic storage provider, a metadata manager and an additional server are defined: the server adds an additional encryption layer to prevent well-known attacks against convergent encryption and thus protect the confidentiality of the data; on the other hand, the metadata manager is responsible of the key management task since block-level deduplication requires the memorization of a huge number of keys. Therefore, the underlying deduplication is performed at block-level and we define an efficient key management mechanism to avoid users to store one key per block. To summarize our contributions:

- ClouDedup assures **block-level deduplication** and **data confidentiality** while coping with weaknesses raised by convergent encryption. Block-level deduplication renders the system more **flexible and efficient**;

- ClouDedup preserves **confidentiality and privacy even against potentially malicious cloud storage providers** thanks to an additional layer of encryption;
- ClouDedup offers an efficient key management solution through the metadata manager;
- The new architecture defines several different components and a **single component cannot compromise** the whole system without colluding with other components;
- ClouDedup works **transparently** with existing cloud storage providers. As a consequence, ClouDedup is fully compatible with standard storage APIs and any cloud storage provider can be easily integrated in our architecture.

Section 2 explains what deduplication and convergent encryption are and why convergent encryption is not a secure solution for cloud storage. Section 3 provides an overview on the related work. Sections 4, 5 and 6 describe ClouDedup's architecture and the role of each component. Section 7 analyzes the computational and storage overhead introduced by ClouDedup and evaluates its resilience against potential attacks. Finally, Section 8 presents our conclusions and planned future work.

2 Background

2.1 Deduplication

According to the data granularity, deduplication strategies can be categorized into two main categories: file-level deduplication [29] and block-level deduplication [17], which is nowadays the most common strategy. In block-based deduplication, the block size can either be fixed or variable [27]. Another categorization criteria is the location at which deduplication is performed: if data are deduplicated at the client, then it is called source-based deduplication, otherwise target-based. In source-based deduplication, the client first hashes each data segment he wishes to upload and sends these results to the storage provider to check whether such data are already stored: thus only ”undeduplicated” data segments will be actually uploaded by the user. While deduplication at the client side can achieve bandwidth savings, it unfortunately can make the system vulnerable to side-channel attacks [19] whereby attackers can immediately discover whether a certain data is stored or not. On the other hand, by deduplicating data at the storage provider, the system is protected against side-channel attacks but such solution does not decrease the communication overhead.

2.2 Convergent Encryption

The basic idea of convergent encryption (CE) is to derive the encryption key from the hash of the plaintext. The simplest implementation of convergent encryption can be defined as follows: Alice derives the encryption key from her message M such that $K = H(M)$, where H is a cryptographic hash function; she can encrypt the message with this key, hence: $C = E(K, M) = E(H(M), M)$, where E is a block cipher. By applying this technique, two users with two identical plaintexts will obtain two identical ciphertexts since the encryption key is the same; hence the cloud storage provider will be able to perform deduplication on such ciphertexts. Furthermore, encryption keys are generated, retained and protected by users. As the encryption key is deterministically generated from the plaintext, users do not have to interact with each other for establishing an agreement on the key to encrypt a given plaintext. Therefore, convergent encryption seems to be a good candidate for the adoption of encryption and deduplication in the cloud storage domain.

2.3 Weaknesses of Convergent Encryption

Convergent encryption suffers from some weaknesses which have been widely discussed in the literature [9, 15, 24]. As the encryption key depends on the value of the plaintext, an attacker who has gained access to the storage can perpetrate the so called "dictionary attacks" by comparing the ciphertexts resulting from the encryption of well-known plaintext values from a dictionary with the stored ciphertexts. Indeed, even if encryption keys are encrypted with users' private keys and stored somewhere else, the potentially malicious cloud provider, who has no access to the encryption key but has access to the encrypted chunks (blocks), can easily perform offline dictionary attacks and discover predictable files. This issue arises in [28] where chunks are stored at the storage provider after being encrypted with convergent encryption.

As shown in [24], the two following attacks are possible against convergent encryption: confirmation of a file (COF) and learn-the-remaining-information (LRI). These attacks exploit the deterministic relationship between the plaintext and the encryption key in order to check if a given plaintext has already been stored or not. In COF, an attacker who already knows the full plaintext of a file, can check if a copy of that file has already been stored. If the attacker is the cloud provider or an insider, he might also learn which users are the owners of that file. Depending on the content of the file, this type of information leakage can be dangerous. For instance, while some users could not be worried about leaking such information, it is worth pointing out that by performing this attack, it is possible to find out if a user has illegally stored a movie or a song.

While COF might be considered as a non-critical problem, LRI can disclose highly sensitive information: in LRI, the attacker already knows a big part of a file and tries to guess the unknown parts by checking if the result of the encryption matches the observed ciphertext. This is the case of those documents that have a predefined template and a small part of variable content. For instance, if users store letters from a bank, which contain bank account numbers and passwords, then an attacker who knows the template might be able to learn the account number and password of selected users. The same mechanism can be used to guess passwords and other sensitive information contained in files such as configuration files, web browser cookies, etc. In general, the more the attacker knows about the victim's data, the more the attack can be effective and dangerous. Hence, a strategy is needed to achieve a higher security degree while preserving combined advantages of both convergent encryption and deduplication.

3 Related Work

Many systems have been developed to provide secure storage but traditional encryption techniques are not suitable for deduplication purposes. Deterministic encryption, in particular convergent encryption, is a good candidate to achieve both confidentiality and deduplication [22, 30] but it suffers from well-known weaknesses which do not ensure protection of predictable files against dictionary attacks [12, 18]. In order to overcome this issue, Warner and Pertula [24] have proposed to add a secret value S to the encryption key. Deduplication will thus be applied only to the files of those users that share the secret. The new definition of the encryption key is $K = H(S|M)$ where $|$ denotes an operation between S and M. However, this solution overcomes the weaknesses of convergent encryption at the cost of dramatically limiting deduplication effectiveness. Most importantly, learning the secret compromises the security of the system. Our approach provides data confidentiality without impacting deduplication effectiveness. Indeed, ClouDedup is totally independent from the underlying deduplication technique.

An alternative approach [21], which makes use of proxy re-encryption, has been proposed but information on performance and overhead were not provided. To the best of our knowledge, the most recent work on this topic is [14], which provides an algorithm to deterministically generate a key without disclosing any information on the plaintext. Keys are generated through a key server which retains a secret. If an attacker learns the secret, the whole system is compromised and the confidentiality of unpredictable files is no longer guaranteed. Also, this technique is limited to file-level deduplication and is not scalable in the case of block-level deduplication, which achieves higher space savings [23]. Moreover, it does not address either side-channel attacks [19] or attacks based on the observation of access patterns, which can leak confidential information and compromise users' privacy. We propose ClouDedup, which does not rely on the security of one single component and manages block-level deduplication in an efficient manner. Furthermore, thanks to its architecture, ClouDedup can address side-channel attacks and preserve users' privacy.

4 ClouDedup

The scheme proposed in this paper aims at deduplication at the level of blocks of encrypted files while coping with the inherent security exposures of convergent encryption. The scheme consists of two basic components: a server that is in charge of access control and that achieves the main protection against COF and LRI attacks; another component, named as metadata manager (MM), is in charge of the actual deduplication and key management operations.

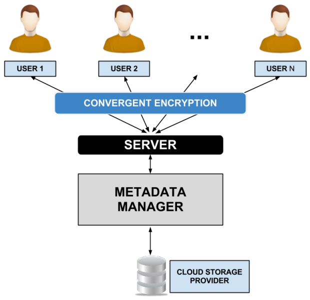

Figure 1: High-level view of ClouDedup

4.1 The Server

A simple solution to prevent the attacks against convergent encryption (CE) consists of encrypting the ciphertexts resulting from CE with another encryption algorithm using the same keying material for all input. This solution is compatible with the deduplication requirement since identical ciphertexts resulting from CE would yield identical outputs even after the additional encryption operation. Yet, this solution will not suffer anymore from the attacks targeting CE such as COF and LRI.

We suggest to combine the access control function with the mechanism that achieves the protection against CE through an additional encryption operation. Indeed, access control is an inherent function of any storage system with reasonable security assurance. Enhancing the trusted component of the storage system, that implements access control, with the new mechanism against COF and LRI attacks, seems to be the most straightforward approach. The core component of ClouDedup is thus a server that implements the additional encryption operation to cope with the weaknesses of CE, together with a user authentication and an access control mechanism embedded in the data protection mechanism. Each data segment is thus encrypted by the server in addition to the convergent encryption operation performed by the user. As to the data access control, each encrypted data segment is linked with a signature generated by its owner and verified upon data retrieval requests. The server relies on the signature of each segment to properly identify the recipient.

4.2 Block-level Deduplication and Key Management

Even though the mechanisms of the server cope with the security weaknesses of CE, the requirement for deduplication at block-level further raises an issue with respect to key management. As an inherent feature of CE, the fact that encryption keys are derived from the data itself does not eliminate the need for the user to memorize the value of the key for each encrypted data segment. Unlike file-level deduplication, in case of block-level deduplication, the requirement to memorize and retrieve CE keys for each block in a secure way, calls for a fully-fledged key management solution. We thus suggest to include a new component, the metadata manager (MM), in the new ClouDedup system in order to implement the key management for each block together with the actual deduplication operation.

4.3 Threat Model

The goal of the system is to guarantee data confidentiality without losing the advantage of deduplication. Confidentiality must be guaranteed for all files, including the predictable ones. The security of the whole system should not rely on the security of a single component (single point of failure), and the security level should not collapse when a single component is compromised. We consider the server as a trusted component with respect to user authentication, access control and additional encryption. The server is not trusted with respect to the confidentiality of data stored at the cloud storage provider. Therefore, the server is not able to perform offline dictionary attacks. Anyone who has access to the storage is considered as a potential attacker, including employees at the cloud storage provider and the cloud storage provider itself. In our threat model, the cloud storage provider is honest but curious, meaning that it carries out its tasks but might attempt to decrypt data stored by users. We do not take into account cloud storage providers that can choose to delete or modify files. Our scheme might be extended with additional features such as data integrity [16] and proofs of retrievability [20]. Among the potential threats, we identify also external attackers. An external attacker does not have access to the storage and operates outside the system. This type of attacker attempts to compromise the system by intercepting messages between different components or compromising a user's account. External attackers have a limited access to the system and can be effectively neutralized by putting in place strong authentication mechanisms and secure communication channels.

4.4 Security

In the proposed scheme, only one component, that is the server, is trusted with respect to a limited set of operations, therefore we call it semi-trusted. Once the server has applied the additional encryption, data are no longer vulnerable to CE weaknesses. Indeed, without possessing the keying material used for the additional encryption, no component can perform dictionary attacks on data stored at the cloud storage provider. The server is a simple semi-trusted component that is deployed on the user's premises and is in charge of performing user authentication, access control and additional symmetric encryption. The primary role of the server is to securely retain the secret key used for the additional encryption. In a real scenario, this goal can be effectively accomplished by using a hardware security module (HSM) [10]. When data are retrieved by a user, the server plays another important role. Before sending data to a given recipient, the server must verify if block signatures correspond to the public key of that recipient. The metadata manager (MM) and the cloud storage provider are not trusted with respect to data confidentiality, indeed, they are not able to decrypt data stored at the cloud storage provider. We do not take into account components that can spontaneously misbehave and do not accomplish the tasks they have been assigned.

5 Components

In this section we describe the role of each component.

5.1 User

The role of the user is limited to splitting files into blocks, encrypting them with the convergent encryption technique, signing the resulting encrypted blocks and creating the storage request. In addition, the user also encrypts each key derived from the corresponding block with the previous one and his secret key in order to outsource the keying material as well and thus only store the key derived from the first block and the file identifier. For each file, this key will be used to decrypt and re-build the file when it will be retrieved. Instead, the file identifier is necessary to univocally identify a file over the whole system. Finally, the user also signs each block with a special signature scheme. During the storage phase, the user computes the signature of the hash of the first block: $S_0 = \sigma_{PK_u}(H(B_0))$. In order not to apply costly signature operations for all blocks of the file, for all the following blocks, a hash is computed over the hash of the previous block and the block itself: $S_i = H(B_i|S_{i-1})$. The main architecture is illustrated in Fig. 1.

5.2 Server

The server has three main roles: authenticating users during the storage/retrieval request, performing access control by verifying block signatures embedded in the data, encrypting/decrypting data traveling from users to the cloud and viceversa. The server takes care of adding an additional layer of encryption to the data (blocks, keys and signatures) uploaded by users. Before being forwarded to MM, data are further encrypted in order to prevent MM and any other component from performing dictionary attacks and exploiting the well-known weaknesses of convergent encryption. During file retrieval, blocks are decrypted and the server verifies the signature of each block with the user's public key. If the verification process fails, blocks are not delivered to the requesting user.

5.3 Metadata Manager (MM)

MM is the component responsible for storing metadata, which include encrypted keys and block signatures, and handling deduplication. Indeed, MM maintains a linked list and a small database in order to keep track of file ownerships, file composition and avoid the storage of multiple copies of the same data segments. The tables used for this purpose are file, pointer and signature tables. The linked list is structured as follows:

- Each node in the linked list represents a data block. The identifier of each node is obtained by hashing the encrypted data block received from the server.

- If there is a link between two nodes X and Y, it means that X is the predecessor of Y in a given file. A link between two nodes X and Y corresponds to the file identifier and the encryption of the key to decrypt the data block Y.

The tables used by MM are structured as follows:

- **File table.** The file table contains the file id, file name, user id and the id of the first data block.

- **Pointer table.** The pointer table contains the block id and the id of the block stored at the cloud storage provider.

- **Signature table.** The signature table contains the block id, the file id and the signature.

In addition to the access control mechanism performed by the server, when users ask to retrieve a file, MM further checks if the requesting user is authorized to retrieve that file. This way, MM makes sure that the user is not trying to access someone else's data. This operation can be considered as an additional access control mechanism, since an access control mechanism already takes place at the server. Another important role of MM is to communicate with cloud storage provider (SP) in order to actually store and retrieve the data blocks and get a pointer to the actual location of each data block.

5.4 Cloud Storage Provider (SP)

SP is the most simple component of the system. The only role of SP is to physically store data blocks. SP is not aware of the deduplication and ignores any existing relation between two or more blocks. Indeed, SP does not know which file(s) a block is part of or if two blocks are part of the same file. This means that even if SP is curious, it has no way to infer the original content of a data block to rebuild the files uploaded by the users. It is worth pointing out that any cloud storage provider would be able to operate as SP. Indeed, ClouDedup is completely transparent from SP's perspective, which does not collaborate with MM for deduplication. The only role of SP is to store data blocks coming from MM, which can be considered as files of small size. Therefore, it is possible to make use of well-known cloud storage providers such as Google Drive [7], Amazon S3 [3] and Dropbox [6].

5.5 A realistic example of ClouDedup

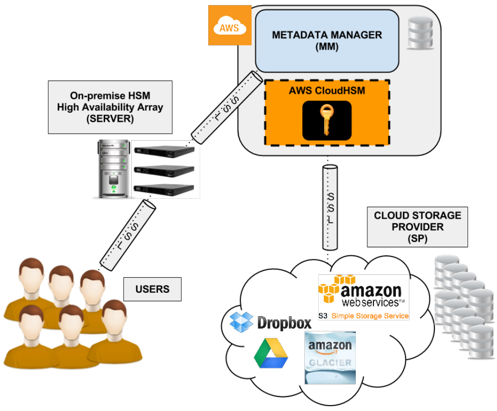

Figure 2: A realistic example of ClouDedup

In this section we show that our proposed solution can be easily deployed with existing and widespread technologies. In the scenario we analyze, a group of users belonging to the same organization want to store their data, save as much storage space as possible and keep their data confidential. As shown in Fig. 2, the Server can be implemented using a Luna SA HSM [10] deployed on the users' premises. As documented in [8], in order to make the system resilient against single-point-of-failure issues, it is possible to build a high availability array by using multiple Luna SA HSMs. This way, in the case the main HSM crashes, it can be immediately replaced by an equivalent HSM without losing the secret key or getting worse performance.

In order to guarantee data confidentiality even in the case the server is compromised, an additional HSM can be deployed between MM and SP. Deploying MM and the additional HSM in the same location, such as AWS [4], helps to minimize network latency and increase performance. This solution achieves higher security (it is very unlikely to compromise both HSMs at the same time) without significantly increasing the costs. MM can be hosted in a virtual machine on Amazon EC2 [1] and make use of a database to store metadata and encrypted keys. The additional HSM can be implemented by taking advantage of Amazon CloudHSM [5] which provides secure, durable, reliable, replicable and tamper-resistant key storage. Finally, very popular cloud storage solutions such as Dropbox [6], Amazon S3 [3], Amazon Glacier [2] and Google Drive [7] could be used as storage providers.

6 Protocol

In this section we describe the two main operations of ClouDedup: storage and retrieval. The description of other operations such as removal, modification and search are out of the scope of this paper.

Notation	
E_K	encryption function with key K
H	hash function
B_i	i^{th} block of a file
B'_i	i^{th} block of a file after convergent encryption
B''_i	i^{th} block of a file after encryption at the server
K_i	key generated from the i^{th} block of a file
K'_i	K_i after encryption at the server
K_A	secret key of server
K_{U_j}	secret key of user j
PK_{U_j}	private key of the certificate of user j
S_i	signature of i^{th} block of a file with PK_{U_j}

6.1 Storage

During the storage procedure, a user uploads a file to the system. As an example, we describe a scenario in which a user U_j wants to upload the file F1.

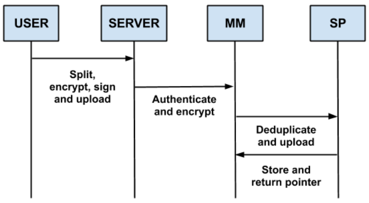

Figure 3: Storage Protocol

USER User U_j splits F1 into several blocks. For each block B_i, U_j generates a key by hashing the block and uses this key to encrypt the block itself. Therefore $B'_i = E_{K_i}(B_i)$ where $K_i = H(B_i)$. U_j stores K_1 and encrypts each following key with the key corresponding to the previous block: $E_{K_{i-1}}(K_i)$. U_j further encrypts each key (except K_1) with his own secret key K_{U_j}: $E_{K_{U_j}}(E_{K_{i-1}}(K_i))$. U_j computes the block signatures as described in 5.1. U_j sends a request to the server in order to upload file F1. The request is composed by:

- U_j's id : ID_{U_j};
- the encrypted file name;

- file identifier : F_{id1};
- first data block : $E_{K_1}(B_1)$;
- for each following data block B_i ($i \geq 2$): key to decrypt block B_i, that is $E_{K_{U_j}}(E_{K_{i-1}}(K_i))$; signature of block B_i, that is S_i; data block B'_i : $E_{K_i}(B_i)$;

In order to improve the level of privacy and reveal as little information as possible, U_j encrypts the file name with his own secret key. File identifiers are generated by hashing the concatenation of user ID and file name $H(user\ ID \mid file\ name)$.

SERVER The server receives a request from user U_j and runs SSL in order to authenticate U_j and securely communicate. Each key, signature and block are encrypted under K_A (server's secret key): $B''_i = E_{K_A}(E_{K_i}(B_i))$, $K'_i = E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))$, $S'_i = E_{K_A}(S_i)$. The only parts of the request which are not encrypted are user's id, the file name and the file identifier. The server forwards the new encrypted request to MM.

MM MM receives the request from the server and for each block B''_i contained in the request, MM checks if that block has already been stored by computing its hash value and comparing it to the ones already stored. If the block has not been stored in the past, MM creates a new node in the linked list, the identifier of the node is equal to $H(B''_i)$. MM updates the data structure by linking each node (block) of file F1 to its successor. A link from block B''_{i-1} to block B''_i contains the following information: $\{F_{id1}, E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))\}$. It is worth pointing out that each key is encrypted with the key of the previous block and users retain the key of the first block, which is required to start the decryption process. This way, a chaining mechanism is put in place and the key retained by the user is the starting point to decrypt all the keys. Moreover, MM stores the signature of each block in the signature table, which associates each block of each user to one signature. For each block B''_i not already stored, MM sends a storage request to SP which will store the block and return a pointer. Pointers are stored in the pointer table, which associates one pointer to each block identifier.

SP SP receives a request to store a block. After storing it, SP returns the pointer to the block.

MM MM receives the pointer from SP and stores it in the pointer table.

6.2 Retrieval

During the retrieval procedure, a user asks to download a file from the system. As an example, we describe a scenario in which a user U_j wants to download the file F1.

USER User U_j sends a retrieval request to the server in order to retrieve file F1. The request is composed by the user's id ID_{U_j}, the file identifier F_{id1} and his certificate.

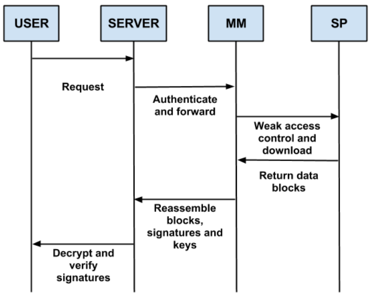
USER
SERVER
MM
SP
Request
Authenticate and forward
Weak access control and download
Return data blocks
Reassemble blocks, signatures and keys
Decrypt and verify signatures

Figure 4: Retrieval Protocol

SERVER The server receives the request, authenticates U_j and if the authentication does not fail, the server forwards the request to MM without performing any encryption.

MM MM receives the request from the server and analyzes it in order to check if U_j is authorized to access F_{id1} (U_j is the owner of the file). If the user is authorized, MM looks up the file identifier in the file table in order to get the pointer to the first block of the file. Then, MM visits the linked list in order to retrieve all the blocks that compose the file. For each of these blocks, MM retrieves the pointer from the pointer table and sends a request to SP.

SP SP returns the content of the encrypted blocks to MM. $B''_i = E_{K_A}(E_{K_i}(B_i))$.

MM MM builds a response which contains all the blocks, keys and signatures of file F1. Signatures are retrieved from the signature table. The response is structured as follows:

- file identifier: F_{id1};
- first data block : $E_{K_A}(E_{K_1}(B_1))$;
- for each following data block $B_i (i \geq 2)$: key to decrypt block B_i, that is $E_{K_A}(E_{K_{U_j}}(E_{K_{i-1}}(K_i)))$; signature of block B_i, that is $E_{K_A}(S_i)$; data block B''_i : $E_{K_A}(E_{K_i}(B_i))$;

MM sends the response to the server.

SERVER The server decrypts blocks, signatures and keys with K_A. If the signature verification does not fail, the server sends a response to U_j. Each key-block pair received by the user, will be structured as follows: $\{E_{K_{U_j}}(E_{K_{i-1}}(K_i)), E_{K_i}(B_i)\}$.

USER U_j can finally decrypt blocks and keys. U_j already knows the key corresponding to the block B_1. For each data block B_i, U_j decrypts block B'_i using K_i and K_{i+1} using K_{U_j} and K_i. U_j can finally rebuild the original file F1.

7 Evaluation

In this section we evaluate the overhead introduced by our system in terms of storage space and computational complexity. We also evaluate ClouDedup's resilience against potential attacks. In order to refer to a real scenario, we use the same parameters of [23], but our calculations hold true for other scenarios.

7.1 Storage Space

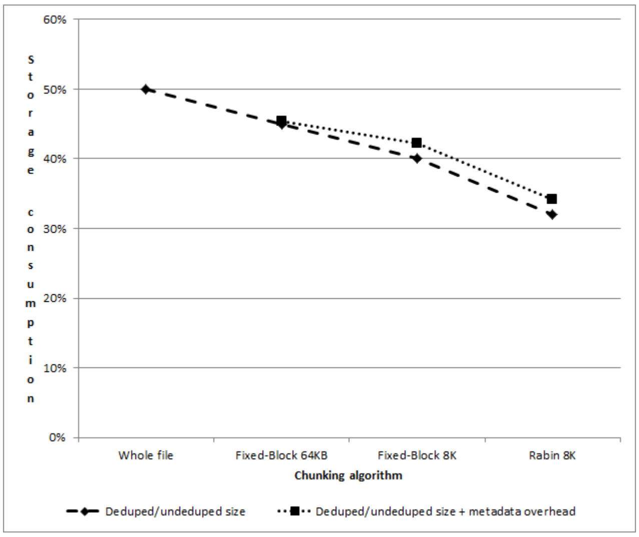

Figure 5: Overhead of metadata management with encryption

We took into account a scenario in which there are 857 file systems. The mean number of files per file system is 225K and the mean size of a file is 318K, resulting in about 57T of data. In our design, we use SHA256 as hash function so the key size of each block is 256 bits. Metadata storage space is estimated by taking into account four main data structures:

- **File table.** The file table stores one record for each file and contains the file id (256 bits), file name (256 bits), user id (32 bits) and the id of the first data block (256 bits).

- **Pointer table.** The pointer table stores one record for each block and contains the block id (256 bits) and the id of the actual block stored at the cloud storage provider (64 bits).

- **Signature table.** The signature table stores one record for each block (non-deduplicated) and contains the block id (256 bits), the file id (256 bits) and the signature (2048 bits for the first block, 128 bits for the remaining blocks).

- **Linked list.** The linked list contains one node (256 bits) and zero or more links for each block. A link contains a pointer (64 bits) to a successor block for a given file and stores additional information such as encrypted block keys (256 bits) and file id (256 bits).

According to the results of [23], Rabin 8K (expected block size of 8K) has proved to be the best chunking algorithm, achieving 68% of space savings. In Fig. 5 we show that the overhead introduced by the MM component is minimal and does not affect space savings of deduplication. In the best deduplication setup (Rabin 8K and deduplication rate of 68%) the total storage space required for metadata is equal to 2.22% of the size of non-deduplicated data. These results prove that the overhead for block-level deduplication is affordable even with encryption.

7.2 Computation

We analyze the computational complexity of the two most important operations: storage and retrieval. N is the mean number of blocks per file and M the total number of blocks in the system.

	Storage	**Retrieval**
Encryption	$O(N)$	$O(N)$
Hash	$O(N)$	$O(N)$
Lookup in data structures	$O(N \log M)$	$O(N)$
Other	$O(N)$	$O(N)$

7.2.1 Storage

The first step of the storage protocol requires the server to encrypt B_i, K_i and S_i. As the encryption is symmetric, the cost of each encryption can be considered constant, so for N blocks the total cost is $O(N)$. The second step of the protocol requires the metadata manager to hash each block in order to compare it with the ones already stored. As for symmetric encryption, the total cost is $O(N)$. In order to perform deduplication, MM has to check if a block has already been stored. In order to do so, he searches (dichotomic search) for a given hash in a pre-ordered table of hash values. The cost of this operation is $O(\log M)$ and it is performed for each block. The cost of the update of the data structures can be considered constant. The last (optional) step of the protocol is the encryption at the additional HSM, which symmetrically encrypts at most N blocks. The total cost of the storage operation is linear for the encryption operations and almost linear for the lookup in data structures, therefore the metadata management is scalable.

7.2.2 Retrieval

The first step of the retrieval protocol requires the metadata manager to compute a hash of the concatenation of user id and file name. The cost of this operation can be considered constant. Even the lookup in the file table, in order to get the pointer to the first block of the file, has a constant cost. Visiting the linked list, searching in the tables and sending a request to the cloud storage provider, have a constant cost and are repeated N times. Once again, the cost of the symmetric decryptions is constant, hence the complexity remains linear. The signature verification process requires the server to verify one signature and compute $N-1$ hashes, hence the cost of this operation is linear. The total cost of the retrieval operation is linear, therefore the system is scalable for very large datasets.

7.3 Deduplication Rate

Our proposed solution aims to provide a robust security layer which provides confidentiality and privacy without impacting the underlying deduplication technique. Each file is split into blocks by the client, who applies the best possible chunking algorithm. When encrypted data blocks are received by MM, a hash of each block is calculated in order to compare them to the ones already stored. This task is completely independent from the chunking technique used by clients. Also, all the encryptions performed in the system do not affect the deduplication effectiveness since the encryption is deterministic. Therefore, ClouDedup provides additional security properties without having an impact on the deduplication rate.

7.4 Security

We explained the main security benefits of our solution in section 4.4. We now focus on potential attack scenarios and possible issues that might arise. As stated in the threat model section, we assume that an attacker, like the malicious storage provider, has full access to the storage. If the attacker has only access to the storage, he cannot get any information. Indeed, files are split into blocks and each block is first encrypted with convergent encryption and then further encrypted with one or more secret keys. Moreover, no metadata are stored at the cloud storage provider. Clearly, thanks to this setup, the attacker cannot perform any dictionary attack on predictable files.

A worse scenario is the one in which the attacker manages to compromise the metadata manager and thus has access to data, metadata and encrypted keys. In this case, confidentiality and privacy would still be guaranteed since block keys are encrypted with users' secret keys and the server's secret key. The only information the attacker can get are data similarity and relationships between files, users and blocks. However, as file names are encrypted by users, these information would be of no use for the attacker, unless he manages to find a correspondence with a predictable file according to its size and popularity.

The system must guarantee confidentiality and privacy even in the unlikely event where the server is compromised. The additional HSM proposed in section 5.5 and located between the metadata manager and the storage provider will then enforce data protection since it also offers another encryption layer; therefore confidentiality is still guaranteed and offline dictionary attacks are not possible. On the other

hand, if the attacker compromises the server, only online attacks would be possible since this component directly communicates with users. The effect of such a breach is limited since data uploaded by users are encrypted with convergent encryption, which achieves confidentiality for unpredictable files [15]. Furthermore, a rate limiting strategy put in place by the metadata manager can limit online brute-force attacks performed by the server.
In the worst scenario, the attacker manages to compromise both HSMs. In this case, the attacker will be able to remove the two additional layers of encryption and perform offline dictionary attacks on predictable files. However, confidentiality for unpredictable files is guaranteed.
Finally, we analyze the impact of an attacker who attempts to compromise users and have no access to the storage. If an attacker compromises one or more users, he can attempt to perform online dictionary attacks. As the server is not compromised, the attacker will only retrieve data belonging to the compromised user (access control mechanism). Furthermore, the server can limit such attacks by setting a maximum threshold for the rate with which users can send requests.

8 Conclusion and Future Work

We designed a system which achieves confidentiality and enables block-level deduplication at the same time. Our system is built on top of convergent encryption. We showed that it is worth performing block-level deduplication instead of file-level deduplication since the gains in terms of storage space are not affected by the overhead of metadata management, which is minimal. Additional layers of encryption are added by the server and the optional HSM. Thanks to the features of these components, secret keys can be generated in a hardware-dependent way by the device itself and do not need to be shared with anyone else. As the additional encryption is symmetric, the impact on performance is negligible. We also showed that our design, in which no component is completely trusted, prevents any single component from compromising the security of the whole system. Our solution also prevents curious cloud storage providers from inferring the original content of stored data by observing access patterns or accessing metadata. Furthermore, we showed that our solution can be easily implemented with existing and widespread technologies. Finally, our solution is fully compatible with standard storage APIs and transparent for the cloud storage provider, which does not have to be aware of the running deduplication system. Therefore, any potentially untrusted cloud storage provider such as Amazon, Dropbox and Google Drive, can play the role of storage provider.

As part of future work, ClouDedup may be extended with more security features such as proofs of retrievability [20], data integrity checking [16] and search over encrypted data [13]. In this paper we mainly focused on the definition of the two most important operations in cloud storage, that are storage and retrieval. We plan to define other typical operations such as edit and delete. After implementing a prototype of the system, we aim to provide a full performance analysis. Furthermore, we will work on finding possible optimizations in terms of bandwidth, storage space and computation.

References

[1] Amazon EC2. `http://aws.amazon.com/ec2/`.

[2] Amazon Glacier. `http://aws.amazon.com/glacier/`.

[3] Amazon S3. `http://aws.amazon.com/s3/`.

[4] Amazon Web Services. `http://aws.amazon.com/`.

[5] AWS Cloud HSM. `http://aws.amazon.com/cloudhsm/`.

[6] Dropbox. `http://www.dropbox.com`.

[7] Google Drive. `http://drive.google.com/`.

[8] High Availability with Luna. `http://bit.ly/19dtZLb`.

[9] Is Convergent Encryption really secure? `http://bit.ly/Uf63yH`.

[10] Luna SA HSM. `http://bit.ly/17CDPm1`.

[11] Opendedup. `http://opendedup.org/`.

[12] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[13] Mihir Bellare, Alexandra Boldyreva, and Adam ONeill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology-CRYPTO 2007*, pages 535–552. Springer, 2007.

[14] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. 2013.

[15] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYPT 2013*, pages 296–312. Springer, 2013.

[16] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS ’09, pages 187–198, New York, NY, USA, 2009. ACM.

[17] Landon P Cox, Christopher D Murray, and Brian D Noble. Pastiche: Making backup cheap and easy. *ACM SIGOPS Operating Systems Review*, 36(SI):285–298, 2002.

[18] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002.

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

[20] Ari Juels and Burton S. Kaliski, Jr. Pors: proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[21] Chuanyi Liu, Xiaojian Liu, and Lei Wan. Policy-based de-duplication in secure cloud storage. In *Trustworthy Computing and Services*, pages 250–262. Springer, 2013.

[22] Luis Marques and Carlos J Costa. Secure deduplication on mobile devices. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, pages 19–26. ACM, 2011.

[23] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[24] Perttula. Attacks on convergent encryption. `http://bit.ly/yQxyvl`.

[25] John Pettitt. Hash of plaintext as key? `http://cypherpunks.venona.com/date/1996/02/msg02013.html`.

[26] The Freenet Project. Freenet. `https://freenetproject.org/`.

[27] Michael O Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[28] Mark W Storer, Kevin Greenan, Darrell DE Long, and Ethan L Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008.

[29] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM, 2008.

[30] Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 195–206. ACM, 2013.