Open Access

Open Journal of Cloud Computing (OJCC)
Volume 1, Issue 1, 2014
http://www.ronpub.com/ojcc
ISSN 2199-1987

Block-level De-duplication with Encrypted Data

Pasquale Puzio [A] [B], Refik Molva [B], Melek Önen [B], Sergio Loureiro [A]

[A] SecludIT, Sophia Antipolis, France, {pasquale, sergio}@secludit.com
[B] EURECOM, Sophia Antipolis, France, {puzio, molva, onen}@eurecom.fr

Abstract

Deduplication is a storage saving technique which has been adopted by many cloud storage providers such as Dropbox. The simple principle of deduplication is that duplicate data uploaded by different users are stored only once. Unfortunately, deduplication is not compatible with encryption. As a scheme that allows deduplication of encrypted data segments, we propose ClouDedup, a secure and efficient storage service which guarantees block-level deduplication and data confidentiality at the same time. ClouDedup strengthens convergent encryption by employing a component that implements an additional encryption operation and an access control mechanism. We also propose to introduce an additional component which is in charge of providing a key management system for data blocks 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.

Type of Paper and Keywords

Regular research paper: *cloud computing, cloud storage, confidentiality, cryptography, convergent encryption, deduplication, privacy*

1 Introduction

Cloud storage providers offer to their customers a potentially infinite storage space. In this scenario, users use large amount of storage space and vendors constantly look for techniques aimed to minimize redundant data and optimize performance and costs. A common technique that gained popularity lately is cross-user deduplication, for its ability to remove duplicates and store duplicate data only once. If different users upload the same file, instead of storing multiple copies of it, the cloud provider adds the user to access control list of the unique copy of the file. Costs of storing and transferring data can be greatly reduced. As an example, deduplication can reduce 90 to 95% of storage based on the experiments in [5].

Along with low ownership costs and flexibility, users also 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.

Convergent encryption (CE) [13,20,21] is a technique that can meet the requirements of two conflicting solutions between deduplication and encryption. In CE, the encryption key is derived and computed based on the data provided. For instance, the key can be the result of the hash of the data segment. However, convergent en-

cryption has various well-known weaknesses [10,19] despite of its suitability. One common vulnerability is the dictionary attack, in which an attacker manages to generate a potential encryption key and, by comparing the two ciphertexts, check whether a file has already been stored or not.

We propose a solution, ClouDedup, that provides both deduplication and encryption and retains benefits offered by each technique. ClouDedup makes use of convergent encryption but prevents the above-mentioned dictionary attacks. The components involved in ClouDedup are: the basic cloud storage provider, a metadata manager and an additional server. The server guarantees data confidentiality even for predictable files. The metadata manager provides a system for key-management and block-level deduplication.
To summarize our contributions:

- ClouDedup achieves **block-level deduplication** and **data confidentiality** against honest-but-curious cloud storage providers while providing a secure solution against weaknesses raised by convergent encryption;
- ClouDedup offers an efficient key management solution through the metadata manager;
- We propose a robust and secure architecture, in which a **single component cannot compromise** the whole system without colluding with other components;
- ClouDedup is **fully compatible** with existing cloud storage providers.

2 BACKGROUND

2.1 Deduplication

It is worth mentioning that deduplication can either be file-level [23] or block-level [12]. The latter corresponds to the most common strategy and is also the one to which we refer in this paper. The block size in block-based deduplication can either be fixed or variable [22].

2.2 Convergent Encryption

Convergent encryption (CE) derives the encryption key from the plaintext. The most common implementations compute key as the hash of the plaintext. Here is a simple example which illustrates how it works: Adam 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 different users with two identical plaintexts will obtain two identical ciphertexts since the encryption key is the same. This allows the cloud storage provider to perform deduplication on such ciphertexts without having any knowledge on the original plain-texts.

2.3 Weaknesses of Convergent Encryption

Discussions of vulnerabilities affecting convergent encryption have been presented [3, 10, 19]. As discussed in Section 1, potential malicious cloud providers can perform offline dictionary attacks and discover predictable files. This explains why a strategy is needed to enforce security while retaining benefits offered by deduplication and convergent encryption.

3 RELATED WORK

Convergent encryption is a particular type of deterministic encryption, and it is suitable to achieve both confidentiality and deduplication [16, 24]. However, it is known to have several well-known weaknesses which do not ensure protection of predictable files against dictionary attacks [7, 13]. Warner and Perttula [19] have proposed to add a secret S to the encryption key to overcome this issue. Based on their rationale, the new definition of the encryption key is $K = H(S|M)$ where $|$ denotes an operation between S and M. However, there is a limitation to their approach, since they constrain the application of deduplication between a limited set of users. Most importantly, the security of the system can be compromised by knowing the secret S. The most recent work on this topic is [9] in our knowledge, where keys are generated with the aid of a key server which retains a secret. The security of the entire system can be compromised and the confidentiality of unpredictable files is no longer guaranteed while an attacker learns the secret. To address all these ongoing issues, we propose ClouDedup. Our approach provides data confidentiality without impacting deduplication effectiveness, because ClouDedup is entirely independent from the deduplication technique. ClouDedup does not rely on the security of one single component and handles block-level deduplication in an efficient manner. Moreover, ClouDedup protects against side-channel attacks and preserve users's privacy.

4 CLOUDEDUP

ClouDedup proposes a scheme for block-level deduplication of encrypted files. Two components are available. First, a server that implements access control and achieves the main protection against dictionary attacks. Second, the metadata manager (MM), which is responsible for block-level deduplication and key management.

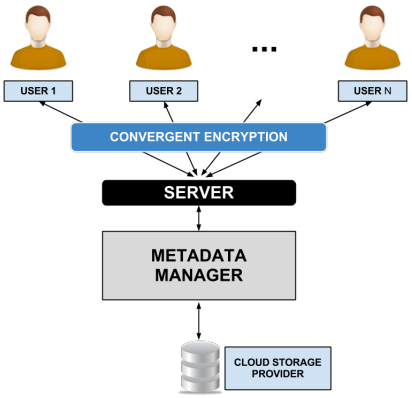

Figure 1: High-level view of ClouDedup

4.1 The Server

In order to prevent attacks against convergent encryption (CE), we propose to encrypt the ciphertexts with another encryption algorithm using the same keying material for all input. The benefit is that encrypted data are confidential while deduplication is still possible. Additionally, our proposed solution is unaffected by weaknesses related to CE. On of the most important tasks of the server is to securely retain the keying material used for the additional encryption. In a real deployment scenario, this goal can be effectively accomplished by using a hardware security module (HSM) [4]. The server is a core component of ClouDedup. User authentication, an access control mechanism and additional encryption operation are all managed by the server. Each encrypted data segment is linked with a signature generated by its owner and verified upon data retrieval requests. The signature of each segment is used by the server to identify the recipient.

4.2 Block-level Deduplication and Key Management

Block-level deduplication raises an issue with respect to key management. Indeed, users need to store the value of the key for each encrypted data segment. Unlike file-level deduplication, in case of block-level deduplication, the requirement to store and retrieve CE keys for each block in a secure way, calls for a key management solution. We thus suggest to include a new component, the metadata manager (MM), in ClouDedup in order to implement the key management for each block together with the actual deduplication operation.

4.3 Threat Model

Achieving data confidentiality without losing the advantage of deduplication is the main goal of ClouDedup, with an emphasis in efficient key management. Even though a single component is compromised, it does not make the security of the entire system collapsed. As explained in Section 4.1, server is trusted for authentication, access control and additional encryption. However, it is not trusted for data confidentiality since it cannot perform offline dictionary attacks. With regard to our model, the cloud storage provider is honest but curious. In other words, cloud providers might attempt to discover the original content of encrypted data stored by users while performing their tasks.

5 Components

5.1 User

Users split files into blocks, encrypt them with the CE, followed by signing the resulting encrypted blocks and creating the storage request. They encrypt each block key and each block with their own secret key. For each file, this key will be used to decrypt and re-build the original file during the retrieval phase. Finally, the user also signs each block with a compact signature scheme. The main architecture is illustrated in 1.

5.2 Server

Three roles are offered by the server. First, it authenticates users during the storage/retrieval phase. Second, it performs access control. Third and most important, it encrypts/decrypts data uploaded by users in order to prevent attacks based on CE weaknesses. The server also verifies the signature of each block with the user's public key during the retrieval phase.

5.3 Metadata Manager (MM)

MM handles deduplication and stores metadata which include block signatures, encrypted keys and pointers to the actual storage. MM has a linked list, which is structured as follows.

The linked list is the main data structure used for the deduplication operation and is structured as follows:

- Each element of the linked list represents a data block. The identifier of each element is computed by hashing the encrypted data block received from the server.
- If there is a link between two elements X and Y, it means that X is the predecessor of Y in a given file. A link between two elements X and Y contains the file identifier and the encryption of the key to decrypt the data block Y.

MM also checks whether a user is authorized to retrieve a file that he/she has requested. This provides an additional access control. Additionally, MM communicates with cloud service provider (SP) to store and retrieve data blocks.

5.4 Cloud Storage Provider (SP)

The only role of SP is to physically store data blocks and allow MM to access those blocks. Data blocks from MM can be considered as files (or objects) of small size. For this reason, ClouDedup to make use of well-known cloud storage providers such as Amazon S3 [1].

5.5 ClouDedup's architecture in practice

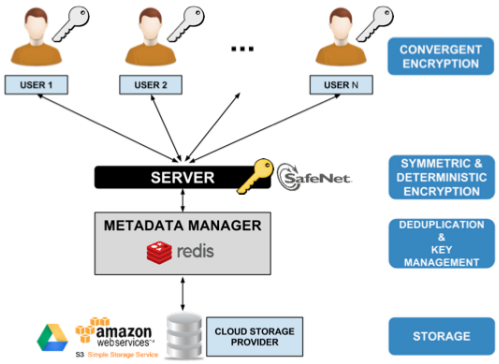

Figure 2: ClouDedup's Architecture

We demonstrate how our proposed architecture can be deployed with existing and widespread technologies in this section. We take into account the following scenario: a group of users store their data in the cloud and want to keep their data confidential while trying to save as much storage space as possible. As shown in Fig. 2, the server can be implemented using a Luna SA HSM [4] deployed on the users' premises or as a virtual appliance in the Cloud. As documented in [2], 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. MM can be hosted on the users' premises or in a virtual appliance. In order to minimize communication delays and improve performance, we suggest to place MM as close as possible to cloud storage provider. In our current implementation, in order to store metadata and encrypted keys, we employ REDIS [6], an advanced, lightweight and high-performance key-value store which can contain lists, hash tables, strings, sets and ordered sets. Finally, very popular cloud storage solutions such as Amazon S3 [1] might 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. We describe a scenario in which a user U_j wants to upload the file F1.

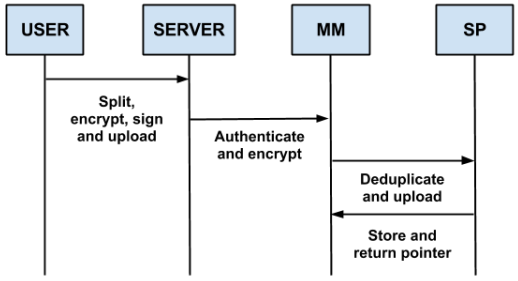

Figure 3: Storage Protocol

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 downloads a file from the system. 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

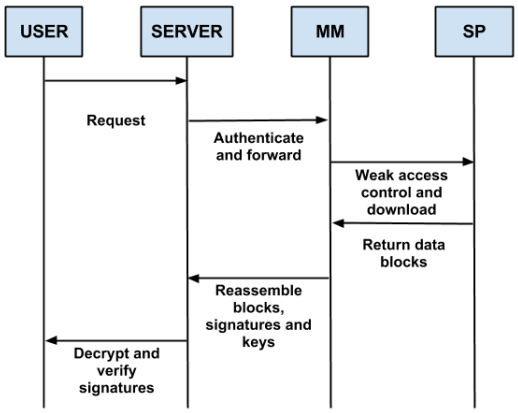

Figure 4: Retrieval Protocol

the user's id ID_{U_j}, the file identifier F_{id1} and his certificate.

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

This section describes our evaluation to test storage space and computational complexity, including the ClouDedup's resilience against potential attacks. We use the same parameters of [17] in order to evaluate a real scenario.

7.1 Storage Space

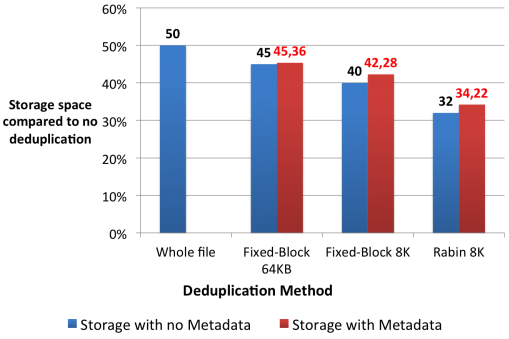

Figure 5: Overhead in terms of storage space of our solution compared to solutions with no metadata

A scenario of 857 file systems was analyzed. The mean number of files per file system is 225K and the mean size of a file is 318K, with the total amount of data as 57T. We use SHA256 as hash function so the key size of each block is 256 bits in our design. Metadata storage space is estimated by taking into account four main data structures:

- **Linked list.** The linked list contains one node (256 bits) and multiple links for each block. A link contains a pointer (64 bits) to a successor block for a given file and additional information such as encrypted block keys (256 bits) and file id (256 bits).
- **Pointer table.** The pointer table stores one record for each block: 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): block id (256 bits), file id (256 bits) and the signature (2048 bits for the first block, 128 bits for the other blocks).
- **File table.** The file table stores one record for each file: file id (256 bits), file name (256 bits), user id (32 bits) and the id of the first data block (256 bits).

Rabin 8K is the best chunking algorithm [17], that results in 68% of space savings. Results in Fig. 5 show that the overhead introduced by the MM does not affect space savings of deduplication. The total storage space required for metadata is equal to 2.22% of the size of non-deduplicated data, while dealing with the best deduplication setup. All these results prove that the overhead for block-level deduplication is affordable even with encryption.

7.2 Computation

This section describes our analysis of the computational cost on storage and retrieval, which are two main operations used in cloud storage. N is the mean number of blocks per file and M the total number of blocks stored 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

There are four types of costs involved. They are encryption, hash, lookup in data structures and other. The computational complexity of encryption and hash is $O(N)$, while the complexity of lookup in data structures is equal to $O(NlogM)$. These results are based on our previous studies [18]. The total cost of the storage operation is linear for the encryption operations.

7.2.2 Retrieval

The computational cost is $O(N)$ for all of encryption, hash, lookup in data structures and other [18]. The total cost of the retrieval operation is linear and the system is scalable for large datasets.

7.3 Security

We described the design, components, functionality and evaluation for ClouDedup. We explained that ClouDedup can retain the advantages of deduplication and convergent encryption. We explained the use-case scenario, and showed that our data can be kept secure in between users, the server and the metadata manager. As we mentioned above, compromising one single component does not make the security of the entire system collapse. For the sake of brevity, in this paper we just analyze the

most significant attack scenario. If the attacker compromises the server, online attacks would be possible since this component directly communicates with users. However, the effect of such a breach is limited since data uploaded by users are encrypted with convergent encryption, which guarantees confidentiality for unpredictable files [10]. A full security analysis of ClouDedup can be found in our previous study [18].

8 Conclusion and Future Work

We explained our motivations to design ClouDedup in a way that confidentiality and block-level deduplication are achieved at the same time. Our proposed solution is built on top of convergent encryption. We showed that the overhead of metadata management is minimal despite the requirements of bock-level deduplication. We proposed three core components: users, the server and the metadata manager. In particular, the server is the core component which adds an additional layer of symmetric encryption. We explained the weaknesses of existing approaches and demonstrated that our proposed approach can fully address CE vulnerabilities and prevent malicious providers from accessing users' data. We performed an evaluation to show that our approach can efficiently reduce costs and storage space in the cloud. We also illustrated a realistic implementation of ClouDedup with existing and widespread technologies. We are currently developing a full prototype of the system and we aim to provide a complete performance analysis very soon. In the performance analysis we will compare the performance of ClouDedup with the most popular cloud storage services and we will analyze performance for both unique files and duplicate files. Furthermore, thanks to the results obtained from the performance analysis, we will work on finding possible optimizations in terms of bandwidth, storage space and computation.

Also, we are currently studying how our system can be integrated with proofs of retrievability (PoR) mechanisms [14] [15], which are based on a challenge-response protocol between users and the cloud provider. Such a functionality would be very valuable from the user point of view since it would guarantee that deduplication is correctly performed and no file has been corrupted or accidentally deleted. Indeeed, without actually downloading the entire file, a user could verify that a given file can be correctly retrieved in the future. Unfortunately, this integration is quite challenging since the existing approaches appear to be incompatible with the underlying deduplication architecture. For instance, the sentinel-based approach [15], in which pseudo-random blocks are injected in the original file before being uploaded, can negatively affect the final deduplication rate since the original file has been modified. On the other hand, the tag-based approach [14] relies on a homomorphic property of the tags in order to aggregate blocks and tags in the response while enabling the user to correctly perform the verification. Unfortunately, in ClouDedup such a property would be lost because of the additional encryption performed by the server.

In addition to PoR, we are also investigating how ClouDedup can be further extended with other security features such as search over encrypted data [8] and data integrity checking [11].

References

[1] "Amazon S3," http://aws.amazon.com/s3/.

[2] "High Availability with Luna," http://bit.ly/19dtZLb.

[3] "Is Convergent Encryption really secure?" http://bit.ly/Uf63yH.

[4] "Luna SA HSM," http://bit.ly/17CDPm1.

[5] "Opendedup," http://opendedup.org/.

[6] "Redis," http://redis.io/.

[7] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[8] M. Bellare, A. Boldyreva, and A. ONeill, "Deterministic and efficiently searchable encryption," in *Advances in Cryptology-CRYPTO 2007*. Springer, 2007, pp. 535–552.

[9] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: server-aided encryption for deduplicated storage," in *USENIX Security*, 2013.

[10] ——, "Message-locked encryption and secure deduplication," in *Advances in Cryptology, EUROCRYPT 2013*. Springer, 2013, pp. 296–312.

[11] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653686

[12] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 285–298, 2002.

[13] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 617–624.

[14] A. Giuseppe, B. Randal, C. Reza *et al.*, "Provable data possession at untrusted stores," *Proceedings of CCS*, vol. 10, pp. 598–609, 2007.

[15] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 584–597. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315317

[16] L. Marques and C. J. Costa, "Secure deduplication on mobile devices," in *Proceedings of the 2011 Workshop on Open Source and Design of Communication*. ACM, 2011, pp. 19–26.

[17] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, no. 4, p. 14, 2012.

[18] P. P., M. R., M. Ö., and L. S., "Cloudedup: Secure deduplication with encrypted data," in *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2013)*. IEEE, 2013.

[19] Perttula, "Attacks on convergent encryption," http://bit.ly/yQxyvl.

[20] J. Pettitt, "Hash of plaintext as key?" http://cypherpunks.venona.com/date/1996/02/msg02013.html.

[21] T. F. Project, "Freenet," https://freenetproject.org/.

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

[23] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*. ACM, 2008, pp. 21–26.

[24] J. Xu, E.-C. Chang, and J. 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*. ACM, 2013, pp. 195–206.

AUTHOR BIOGRAPHIES

Pasquale Puzio is a CIFRE PhD Student at SecludIT and EURECOM, under the supervision of Sergio Loureiro and Refik Molva. He got a Master's Degree in Computer Science from University of Bologna and a Master's Degree in Ubiquitous Computing from University of Nice-Sophia Antipolis. The topic of his PhD thesis is Data Storage Security in Cloud Computing but his research interests include also infrastructure security in cloud computing. In 2013, his paper *ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage* was accepted in IEEE CloudCom.

Refik Molva is a full professor and the head of the Networking and Security Department at EURECOM in Sophia Antipolis, France. His current research interests are the design and evaluation of protocols for security and privacy in cloud computing. He previously worked on several research projects dealing with security and privacy in social networks, RFID systems, self-organizing systems, and mobile networks. He was program chair or general chair for security conferences such as ESORICS, RAID, SecureComm, IEEE ICC and various security related workshops. He has been an area editor for various journals such as Computer Networks, Pervasive and Mobile Computing, Computer Communications, and the International Journal of Information Security. Beside security, he worked on distributed multimedia applications over high speed networks and on network interconnection. Prior to joining Eurcom, he worked in the Zurich Research Laboratory of IBM where he was one of the key designers of the KryptoKnight security system. He also worked as a consultant in security for the IBM Consulting Group. Refik Molva has a Ph.D. in Computer Science from the Paul Sabatier University in Toulouse (1986) and a B.Sc. in Computer Science (1981) from Joseph Fourier University, Grenoble, France.

Melek Önen is a senior researcher at EURECOM. Her current research interests are the design of security and privacy solutions for cloud computing, accountability and user-centric networking. She was involved in many European and national French research projects. She holds a PhD in Computer Science from ENST (2005); her thesis was focusing on securing multicast communications in satellite networks.

Sergio Loureiro, CEO and Co-Founder of SecludIT, has worked in network security for more than 15 years. He has occupied top management positions in 2 startups where he was responsible for email security products and services, and security gateways. Before he was the lead architect for a number of security products such as SSL VPNs, log management, web security and SSL crypto accelerators. His career started in several research labs, where he participated in European projects focusing on security. Sergio holds a Ph.D. in computer science from the ENST Paris and MSc and BSc degrees from the University of Porto (Portugal). He is the holder of 3 patents.