International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 09
May 2016

Secure Authorized Duplicate Check Scheme for Hybrid Cloud Using Convergent Key

[1] Md Thouheed & [2] M.Eranna
[1]M.Tech Dept of CSE, PVKK College, Affiliated to JNTUA, AP, India .
[2]Assistant Professor, Dept of CSE, PVKK College, Affiliated to JNTUA, AP, India

Abstract— *Recent years have been witnessed the trend of leveraging cloud-based resources and services for large scale content storage space, processing, and distribution. Privacy and security are among top concerns for the public cloud environments. Towards these security challenges, we propose and implement, on OpenStack Swift and a new client-side deduplication method for securely storing and sharing outsourced data passing through the public cloud. The creativity of our proposal is twofold. First, it ensures better privacy towards not permitted users. That is, every client computes a per data key to encrypt the data that he intends to accumulate in the cloud. As such, the data right to use is maintained by the data owner. Second, by Combining access rights in metadata file, an certified user can decode an encrypted file only with his private key.*

Keywords **–Cloud Storage; Data Security; Deduplication; Confidentiality; Proof of Ownership.**

I. INTRODUCTION

With the quickly growing amounts of data shaped worldwide, networked and multi-user storage systems are flattering very popular. However, concerns over data security still prevents many users from migrating data to remote storage. The conventional solution is to encrypt the data before it leaves the owner's premises. While sound from a security standpoint, this approach prevents the storage provider from effectively applying storage effectiveness functions, such as compression and deduplication, which would permit optimal practice of the resources and accordingly lesser service cost. Client-side data deduplication in exacting ensures that multiple uploads of the same content only swig network bandwidth and storage space of a single upload. Deduplication is energetically used by a number of cloud support providers (e.g. Bitcasa) and various cloud services Unfortunately, encrypted data is pseudorandom and thus cannot be deduplicated: as a significance, current approaches have to entirely forgo either security or storage efficiency. In this paper, we present a scheme that permits a more fine-grained trade-off. The intuition is that outsourced data may require different levels of protection, depending on how popular it is: content shared by many users, such as a popular song or video, arguably requires less protection than a personal document, the copy of a payslip or the draft of an unsubmitted scientific paper. Around this intuition we build the following contributions: (i) we present Eμ, a novel threshold cryptosystem (which can be of independent interest), together with a security model and formal security proofs, and (ii) we commence a scheme that uses Eμ as a building block and enable to control popularity to achieve both security and storage efficiency. Finally, (iii) we talk about its overall security. But customers may want their data encrypted, for reasons ranging from personal privacy to corporate policy to legal regulations. A client could encrypt its file, under a user's key, before storing it. But common encryption modes are randomized, making deduplication impossible since the SS (Storage Service) effectively always sees different ciphertexts regardless of the data. If a client's encryption is deterministic (so that the same file will always map to the same ciphertext) deduplication is possible, but only for that user. Cross-user deduplication,which allows more storage savings, is not possible because encryptions of different clients, being under different keys, are usually different. Sharing a single key across a group of users makes the system brittle in the face of client compromise.One approach meant at resolving this anxiety is message-locked encryption (MLE) . Its the majority famous instantiation is convergent encryption (CE), introduced earlier by Douceur et al. [2] and others . CE is used within a wide variety of commercial and research SS systems [1, 2, 5, 6, 8, 12, 15, 32,33, 55, 60, 66, 71, 78, 79]. Letting M be a file's contents, hereafter called the message, the client first computes a key $K \leftarrow H(M)$ by applying a cryptographic hash function H to the message, and then computes the ciphertext $C \leftarrow E(K, M)$ via a deterministic symmetric encryption scheme. The short message-derived key K is stored separately encrypted under a per-client key or password. A second client B encrypting the same file M will produce the same C, enabling deduplication However, CE is subject to an inherent security limitation, namely susceptibility to offline brute-force dictionary attacks. Knowing that the target message M underlying a target ciphertext C is drawn from a dictionary $S = \{M1,..., Mn\}$ of size n, the attacker can recover M in the time for $n = |S|$ off-line encryptions: for each $i = 1,..., n$, it simply CE-encrypts Mi to get a ciphertext denoted Ci and returns the Mi such that $C = Ci$.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 09
May 2016

(This works because CE is deterministic and keyless.) Security is thus only possible when the target message is drawn from a space too large to exhaust. We say that such a message is unpredictable. The unpredictability assumption. The above-mentioned work puts security on a firm footing in the case messages are unpredictable. In practice, however, security only for unpredictable data may be a limitation for, and threat to, user privacy. We suggest two main reasons for this. The first is simply that data is often predictable. Parts of a file's contents may be known, for example because they contain a header of known format, or because the adversary has sufficient contextual information. Some data, such as very short files, are inherently low entropy. This has long been recognized by cryptographers [43], who typically aim to achieve security regardless of the distribution of the data.The other and perhaps more subtle fear with regard to
the unpredictability assumption is the difficulty of validating it or testing the extent to which it holds for "real" data. When we do not know how predictable our data is to an adversary, we do not know what, if any, security we are getting from an encryption mechanism that is safe only for unpredictable data. These concerns are not merely theoretical, for offline dictionary attacks are recognized as a significant threat to CE in real systems [77] and are currently hindering deduplication of outsourced storage for security-critical data.This work. We design and implement a new system called DupLESS (Duplicateless Encryption for Simple Storage) that provides a more secure, easily-deployed solution for encryption that supports deduplication. In DupLESS, a group of affiliated clients (e.g., company employees) encrypt their data with the aid of a key server (KS) that is separate from the SS. Clients authenticate themselves to the KS, but do not leak any information about their data to it. As long as the KS remains inaccessible to attackers, we ensure high security. (Effectively, semantic security , except that ciphertexts
leak equality of the underlying plaintexts. The latter is necessary for deduplication.) If both the KS and SS are compromised, we retain the current MLE guarantee of security for unpredictable messages.

II. EXISTING SYSTEM

DupLESS starts with the observation that brute-force ciphertext recovery in a CE-type scheme can be dealt with by using a key server (KS) to derive keys, instead of setting keys to be hashes of messages. Access to the KS is preceded by authentication, which stops external attackers. The increased cost slows down brute-force attacks from compromised clients, and now the KS can function as a (logically) single point of control for implementing rate-limiting measures. We can expect that by scrupulous choice of rate-limiting policies and parameters, brute-force attacks originating from compromised clients will be rendered less effective, while normal usage will remain unaffected.

We start by looking at secret-parameter MLE, an extension to MLE which endows all clients with a systemwide secret parameter sk (see Section 4). The rationale here is that if sk is unknown to the attacker, a high level of security can be achieved (semantic security, except for equality), but even if sk is leaked, security falls to that of regular MLE. A server-aided MLE scheme then is a transformation where the secret key is restricted to the KS instead of being available to all clients. One simple approach to get server-aided MLE is to use a PRF F, with a secret key K that never leaves the KS. A client would send a hash H of a file to the KS and receive back a message-derived key K← F(K, H). The other steps are as in CE. However, this approach proves unsatisfying 3 from a security perspective. The KS here becomes a single point of failure, violating our goal of compromise resilience:n attacker can obtain hashes of files after gaining access to the KS, and can recover files with bruteforce attacks. Instead, DupLESS employs an oblivious PRF (OPRF) protocol [64] between the KS and clients, which ensures that the KS learns nothing about the client inputs or the resulting PRF outputs, and that clients learn nothing about the key. In Section 4, we propose a new server-aided MLE scheme DupLESSMLE which combines a CE-type base with the OPRF protocol based on RSA blind-signatures [20, 29, 30]. Thus, a client, to store a file M, will engage in the RSA OPRF protocol with the KS to compute a messagederived key K, then encrypt M with K to produce a ciphertext Cdata. The client's secret key will be used to encrypt K to produce a key encapsulation ciphertext Ckey.Both Ckey and Cdata are stored on the SS. Should two
clients encrypt the same file, then the message-derived keys and, in turn, Cdata will be the same (the key encapsulation Ckey will differ, but this ciphertext is small). Building a system around DupLESSMLE requires careful design in order to achieve high performance. DupLESS uses at most one or two SS API calls per operation. (As we shall see, SS API calls can be slow.) Because interacting with the KS is on the critical path for storing files, DupLESS incorporates a fast client-toKS protocol that supports various rate-limiting strategies. When the KS is overloaded or subjected to denial-ofservice attacks, DupLESS clients fall back to symmetric encryption, ensuring availability. On the client side, DupLESS introduces dedup heuristics to determine whether the file about to be stored on the SS should be selected for deduplication, or processed with randomized encryption. For example, very small files or files considered particularly sensitive can be prevented from deduplication. We use deterministic authenticated encryption (DAE) to protect, in a structurepreserving way,

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 09
May 2016

the path and filename associated to stored files. Here we have several choices along an efficiency/security continuum. Our approach of preserving folder structure leaks some information to the SS, but on the other hand, enables direct use of the SS-provided API for file search and moving folders.DupLESS is designed for a simple SS API, but can be adapted to settings in which block-oriented deduplication is used, and to complex network storage and backup solutions that use NFS , CIFS and the like, but we do not consider these further.Several deduplication schemes have been anticipated by the research community showing how deduplication allows very appealing reductions in the usage of storage resources . Most works do not consider security as a concern for deduplicating systems; recently however, Harnik et al. [7] have presented a number of attacks that can lead to data leakage in storage systems in which client-side deduplication is in place. To thwart such attacks, the concept of proof of ownership has been introduced [8, 9]. None of these works, however, can provide real end-user confi- dentiality in presence of a malicious or honest-but-curious cloud provider. Convergent encryption is a cryptographic primitive introduced by Douceur et al. [1, 2], attempting to combine data confidentiality with the possibility of data deduplication. Convergent encryption of a message consists of encrypting the plaintext using a deterministic (symmetric) encryption scheme with a key which is deterministically derived solely from the plaintext. Clearly, when two users independently attempt to encrypt the same file, they will generate the same ciphertext which can be easily deduplicated. Unfortunately, convergent encryption does not provide semantic security as it is vulnerable to content-guessing attacks. Later, Bellare et al. formalized convergent encryption under the name message-locked encryption. As expected, the security analysis presented in highlights that message-locked encryption offers confidentiality for unpredictable messages only, clearly failing to achieve semantic security. Xu et al. [3] present a PoW scheme allowing client-side deduplication in a bounded leakage setting. They provide a security proof in a random oracle model for their solution, but do not address the problem of low min-entropy files. Recently, Bellare et al. presented DupLESS [4], a server-aided encryption for deduplicated storage. Similarly to ours, their solution uses a modified convergent encryption scheme with the aid of a secure component for key generation. While DupLESS offers the possibility to securely use server-side deduplication, our scheme targets secure client-side deduplication.

III. PROPOSED SYSTEM

We implemented a fully functional DupLESS client. The client was written in Python and supports both Dropbox [3] and Google Drive [7]. It will be straightforward to extend the client to work with other services which export an API s. The client uses two threads during store operations in order to parallelize the two SS API requests. The client takes user credentials as inputs during startup and provides a command line interface for the user to type in commands and arguments. When using Google Drive, a user changing directory prompts the client to fetch the file list ID map asynchronously. We used Python's SSL and Crypto libraries for the client-side crypto operations and used the OPRFv2 KS protocol. We now describe the experiments we ran to measure the performance and overheads of DupLESS.We will compare both to direct use of the underlying SS API (no encryption) as well as when using a version of DupLESS modified to implement just MLE, in particular the convergent encryption (CE) scheme, instead of DupLESSMLE. This variant computes the messagederived key K by hashing the file contents, thereby avoiding use of the KS. Otherwise the operations are the same. Test setting and methodology. We used the same machine as for the KS tests.Measurements involving the network were repeated 100 times and other measurements were repeated 1,000 times. We measured running times using the timeit Python module. Operations involving files were repeated using files with random contents of size 2 2i KB for i $\in$ {0, 1,..., 8}, giving us a file size range of 1 KB to 64 MB. Storage and retrieval latency. We now compare the time to store and retrieve files using DupLESS, CE, and the plain SS. Figure 7 (top left chart) reports the median time for storage using Dropbox. The latency overhead when storing files with DupLESS starts at about 22% for 1 KB files and reduces to about 11% for 64 MB files. As we mentioned earlier, Dropbox and Google Drive exhibited significant variation in overall upload and download times. To reduce the effect of these variations on the observed relative performance between DupLESS over the SS, CE over the SS and plain SS, we ran the tests by cycling between the three settings to store the same file, in quick succession, as opposed to, say, running all plain Dropbox tests first. We adopted a similar
approach with Google Drive.

We observe that the CE (Convergent Encryption) store
times are close to DupLESS store times, since the KSReq step, which is the main overhead of DupLESS w.r.t CE, has been optimized for low latency. For example, median CE latency overhead for 1 KB files over Dropbox was 15%. Put differently, the overhead of moving to DupLESS from using CE is quite small, compared to that of using CE over the base system. Relative retrieval latencies for DupLESS over Dropbox were lower than the store latencies, starting at about 7% for 1 KB files and reducing to about 6% for 64 MB files. Performance with Google Drive follows a similar trend, with overhead for DupLESS ranging from

33% to 8% for storage, and 40% to 10% for retrieval, when file sizes go from 1 KB to 64 MB.These experiments report data only for files larger than 1 KB, as smaller files are not selected for deduplication by canDedup. Such files are encrypted with non-dedupable, randomized encryption and latency overheads for storage and retrieval in these cases are negligible in most cases. The main intuition behind our scheme is that there are scenarios in which data requires different degrees of protection that depend on how popular a datum is. Let us start with an example: imagine that a storage system is used by multiple users to perform full backups of their hard drives. The files that undergo backup can be divided into those uploaded by many users and those uploaded by one or very few users only. Files falling in the former category will benefit strongly from deduplication because of their popularity and may not be particularly sensitive from a confidentiality standpoint. Files falling in the latter category, may instead contain user-generated content which requires confidentiality, and would by definition not allow reclaiming a lot of space via deduplication. The same can be said about common blocks of shared VM images, mail attachments sent to several recipients, to reused code snippets, etc. This intuition can be implemented cryptographically using a multi-layered cryptosystem. All files are initially declared unpopular and are encrypted with two layers, as illustrated in Figure 1: the inner layer is applied using a convergent cryptosystem, whereas the outer layer is applied using a semantically secure threshold cryptosystem. Uploaders of an unpopular file attach a decryption share to the ciphertext. In this way, when sufficient distinct copies of an unpopular

IV. CONCLUSION

This work deals with the inherent tension between well established storage optimization methods and end-to-end encryption. Differently from the approach of related works, that assume all files to be equally security-sensitive, we vary the security level of a file based on how popular that file is among the users of the system. We present a novel encryption scheme that guarantees semantic security for unpopular data and provides weaker security and better storage and bandwidth benefits for popular data, so that data deduplication can be applied for the (less sensitive) popular data. Files transition from one mode to the other in a seamless way as soon as they become popular. We show that our protocols are secure under the SXDH Assumption. In the future we plan to deploy and test the proposed solution and evaluate the practicality of the notion of popularity and whether the strict popular/unpopular classification can be made more fine-grained. Also, we plan to remove the assumption of a trusted indexing service and explore different means of securing the indexes of unpopular files.

V. REFERENCES

[1] M. Bellare, S. Keelveedhi, and T. Ristenpart. Dupless: Serveraided encryption for deduplicated storage. In USENIX Security Symposium, 2013.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In EUROCRYPT, pages 296–312, 2013.

[3] M. Bellare, C. Namprempre, and G. Neven. Security proofs for identity-based identification and signature schemes. J. Cryptology,22(1):1–61, 2009.

[4] M. Bellare and A. Palacio. Gq and schnorr identification schemes:Proofs of security against impersonation under active and concurrent attacks. In CRYPTO, pages 162–177, 2002.

[5] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider. Twin clouds: An architecture for secure cloud computing. In Workshop on Cryptography and Security in Clouds (WCSC 2011), 2011.

[6] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In ICDCS, pages 617–624, 2002.

[7] D. Ferraiolo and R. Kuhn. Role-based access controls. In 15th NIST-NCSC National Computer Security Conf., 1992.

[8] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management. In IEEE Transactions on Parallel and Distributed Systems, 2013.