An implementation for secure data deduplication on end-to-end encrypted documents

1st Toraș-Mihnea Jipianu
Faculty of Information Systems and Cyber Security
Military Technical Academy "Ferdinand I"
Bucharest, Romania
mihnea.jipianu@mta.ro

2nd Iulian Aciobăniței
Faculty of Information Systems and Cyber Security
Military Technical Academy "Ferdinand I"
Bucharest, Romania
iulian.aciobanitei@mta.ro

***Abstract*—In the realm of data storage and management, secure data deduplication represents a cornerstone technology for optimizing storage space and reducing redundancy. Traditional client-side deduplication approaches, while efficient regarding storage and network traffic, expose vulnerabilities that allow malicious users to infer the existence of specific files through traffic analysis. Even using a Proof of ownership scheme does not guarantee protection from all attack scenarios, specific to data deduplication.**

This paper introduces a novel secure data deduplication framework employing a deduplication proxy that operates on-premise, effectively mitigating the risk of such inference attacks.

By leveraging convergent encryption, and Merkle tree challenges for proof of ownership, our solution ensures that data deduplication does not compromise data privacy or security. The deduplication proxy acts as an intermediary, performing deduplication processes on-premise. This approach not only preserves the efficiency benefits of deduplication but also enhances security by preventing external visibility into data traffic patterns.

Our implementation, publicly available on Github, demonstrates the efficacy of the method for enforcing end-to-end encryption while maintaining data deduplication's storage-saving advantages. The proposed framework is suitable for organizations aiming to safeguard their data while optimizing storage resources.

***Index Terms*—Data Deduplication, Merkle Hash Tree, Merkle Tree, Proof of Ownership**

I. Introduction

In the digital age, the exponential growth of data has become a double-edged sword for organizations worldwide. While data is a valuable asset for business intelligence, its management poses significant challenges, particularly in terms of storage efficiency and data security. Data deduplication, the process of eliminating duplicate copies of repeating data, has emerged as a vital solution for optimizing storage space and reducing costs. However, as more organizations migrate to cloud-based storage solutions, the need for secure data deduplication strategies becomes paramount to protect data from unauthorized access and breaches. Data deduplication can lead to a storage reduction from 50% for usual files, up to 95% for backups [1].

Over time, many attack scenarios were developed regarding data deduplication:

1) **File Content Checking**. If using a weak PoW (Proof of Ownership) protocol, the following scenario could be possible: Alice uploads a document in the cloud. Eve obtains the hash of the document and then uploads the hash to prove the ownership of the file. After that, Eve can download the file. This attack was actually deployed against Dropbox.
2) **Content learning**. The exact same principle as attack 1, but it is done on documents with low entropy. Malicious users brute-force the document content and not the encryption key.
3) **Data Poisoning**. Somewhat based on the same principle. *Eve* would generate a *fileA* and prove it's ownership to the server. Then, instead of *fileA*, would upload a malware file. If another user (*Alice*) would try to upload the same *fileA* afterwards, the server would consider it to be already uploaded by *Eve*. Later on, *Alice* tries to download *FileA* and actually receives the malware.
4) **Data exfiltration**. A malicious client would generate as many proofs of ownership as possible. Those would actually be random-generated hashes. If the generated hash matches to any file server-side, malicious user will have access over the entire document content afterwards.

The main goal of this paper is to present, in detail, an implementation of a secure framework for document storage with data deduplication. Documents are actually stored by a public cloud storage provider, therefore we used end-to-end encryption to ensure data privacy. Solution design had in mind all four attacks presented above. Especially to mitigate the content learning attack, we designed a deduplication proxy (gateway) so that one cannot learn about what files are deduplicated or not. Our proposal is publicly available on Github [2].

The rest of the paper is organized as follows. Section II presents the main work done in the field of secure data deduplication. Section III describes in detail how our proof of concept solution is designed and implemented. Section IV lays down a discussion on the security and efficiency of our proposal and how it ensures efficiency. Finally, paper's conclusions are presented in section V.

II. State of the art

To obtain more efficient solutions for data transfer and storage, data deduplication techniques were designed and

implemented. Still, it was proved that those first schemes, implemented by large companies, such as Dropbox, were insecure and allowed for unauthorized data access [3]. Since then, many schemes for secure data deduplication have been proposed.

As presented in [4]–[6], from 2010 to the present day, security properties for data deduplication were tackled mainly using the following approaches:

- Proof of ownership [7]–[10].
- Message-dependent encryption [11], [12].
- Traffic Obfuscation [13]
- Deterministic information dispersal [14]

Alongside the above mentioned approaches, semantic secure data deduplication schemes were proposed [10], but were proven to be unusable from a user-experience perspective. Most solutions to approach deduplication problem via Proof of ownership are based on Merkle Hash Tree [18], which are a good method to generate and check challenges regarding data content. Although MHT (Merkle Hash Tree) seem to be a sensible approach for this solution, MHT are I/O intensive, therefore are not as efficient as one might wish. In our approach, the server stores a set of precomputed challenges that are to be used when another user would claim the ownership of the same document.

Message-dependant encryption started using a simple approach to encrypt documents using some information generated from the file content itself. This way, one may assure a better deduplication factor, since two identical cleartext have the same cyphertext. This proposals evolved over time by using key servers and they also could generate a randomized tag, that is used to identify the document.

To enhance data deduplication efficiency and security, we propose using a proxy server as an intermediary between the client and server. This approach expedites Proof of Ownership (PoW) computation, reducing client-side workload and mitigating security risks from network monitoring attacks. By acting as a protective barrier, the proxy server enhances overall performance and security in the deduplication process.

III. Proposed Solution

In this section, we present our proposed solution to address the challenges outlined in the preceding discussions. The architecture and methods employed aim to enable secure data deduplication. Additionally, the solution is intended to be resistant to various types of attacks, such as File Content Checking, Data Poisoning, Data Exfiltration, and Identifying Files, Learning the Contents of Files [3].

A. Proposed architecture

The proposed architecture, shown in Figure 1, consists of the following elements:

- The client-side application, which is the user interface.
- The proxy server, which plays a crucial role in the deduplication process.
- The server responsible for file management.
- Cloud Storage, where files are stored in encrypted form.

Additionally, SQL Server is used for data storage at the server level, and MySQL is employed for storage at the proxy-server level.

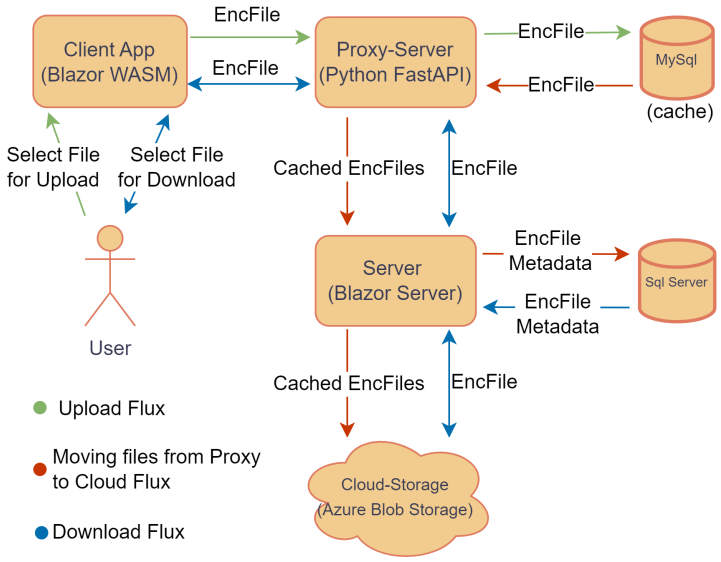

Fig. 1. Proposed Architecture

B. Main Cryptographic Mechanisms

The main objective of this proposal is data confidentiality therefore, the files processed within the system are encrypted client-side. We need an encryption mechanism that produces the same output for the same input. Hence, we utilize Convergent Encryption, which has been previously proposed in other data deduplication solutions [15]. On the client-side application, the following steps are followed:

- SHA3-512 hash is calculated over the plaintext file.
- The obtained hash is split into two parts: a 32-bit encryption key and a 32-bit IV (initialization vector).
- The file is encrypted using AES GCM (Galois Counter Mode).

This way, convergent encryption ensures that identical files will always be encrypted with the same key and IV, and the resulting ciphertexts will be the same. By using AES GCM with a 32-bit key and IV, the encrypted data is resistant to brute-force attacks. Over the encrypted file, a SHA3-256 hash is calculated. The result will be denoted as TAG. Its role is to identify a file in the database, and based on TAG, we will verify if a file is a duplicate or not. The encrypted file, along with the TAG, is transmitted to the proxy. The proxy server assumes the responsibility of determining whether the file is a duplicate or not. Below, we provide an outline of the verification process.

The proxy forwards the TAG to the server for verification. Two situations may arise: either the file is unique, or the file was uploaded previously.

C. New file uploaded

If the file has not been uploaded previously, the proxy caches the file within a MySQL database, storing files in a table with the LONGBLOB data type. Periodically, at regular intervals, the proxy transmits the encrypted files from its cache to the server using asynchronous operations and mechanisms such as Celery (Python). Before uploading the file to Cloud Storage, the system generates a series of Proof of Ownership challenges. The rationale behind this approach is to enhance the verification process beyond solely relying on the TAG. Therefore, when a file is uploaded for the first time, a predefined set of challenges is generated and stored on the server. The challenges are derived from the Merkle Hash Tree, which represents an adaptation of the mechanism described in [16].

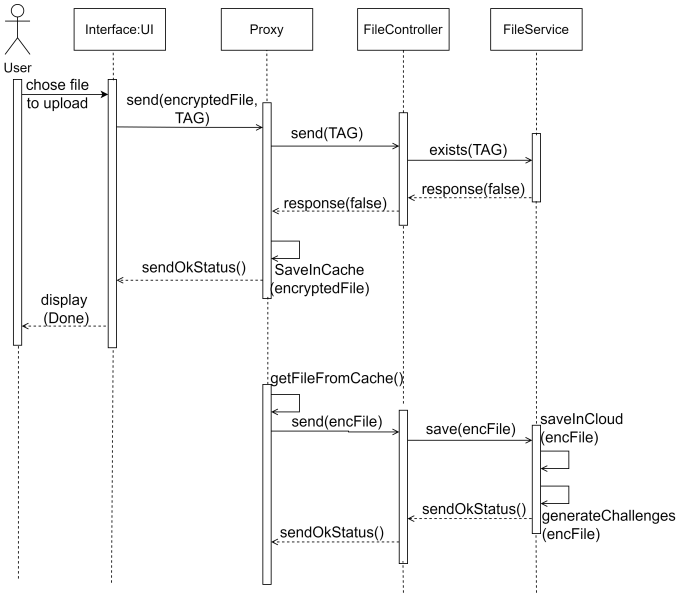

Fig. 2. Flow for uploading an unique file

The construction of the Merkle tree proceeds as follows:

- SHA3-256 hash is calculated on the chunks of the encrypted file. These chunks may vary in length; for instance, they could be 1 KB in length.
- For each subsequent level of the tree, pairs of hashes are concatenated and then hashed again.
- Eventually, a single hash, known as the root hash, is reached.

Due to its computational cost, the Merkle tree is computed only once. Subsequently, a set of challenges is generated using nodes from the tree. A challenge will be used just once. Each challenge involves selecting three random nodes from the tree and performing an XOR operation on them. This yields a string of bytes. In the database, the positions of these three nodes and the corresponding XOR result are stored.

Only the positions and corresponding answers will be stored in the database. Subsequently, the file reference is saved for the user, and the encrypted file is then uploaded to Cloud Storage.

Data: N, NumNodes, MKH[], Pos1, Pos2, Pos3, I
Result: MKA[]
I = 0;
while $I < N$ **do**
  Pos1 = rand(0,NumNodes);
  Pos2 = rand(0,NumNodes);
  Pos3 = rand(0,NumNodes);
  MKA[I] = MKH[Pos1] $\oplus$ MKH[Pos2] $\oplus$ MKH[Pos3];
  I++;
end

- N - Number of challenges
- NumNodes - Number of Merkle Tree Nodes
- MKH[] - Merkle Tree Nodes
- Pos1, Pos2, Pos3 - Node positions
- I - Array index for the answers
- MKA[] - Answers for challenges

D. Previously uploaded file

If the file is a duplicate, it necessitates a Proof of Ownership method to confirm whether the TAG indeed corresponds to the file and whether the user is the rightful owner. Consequently, the proxy requests a challenge from the server specific to the file in question. To perform the verification, the proxy requests the challenge for that file from the server and then computes the Merkle Hash Tree. Subsequently, using the position provided by the challenge, it generates the answer through XOR operations on the three nodes. The resulting answer is then transmitted to the server, which responds affirmatively or negatively.

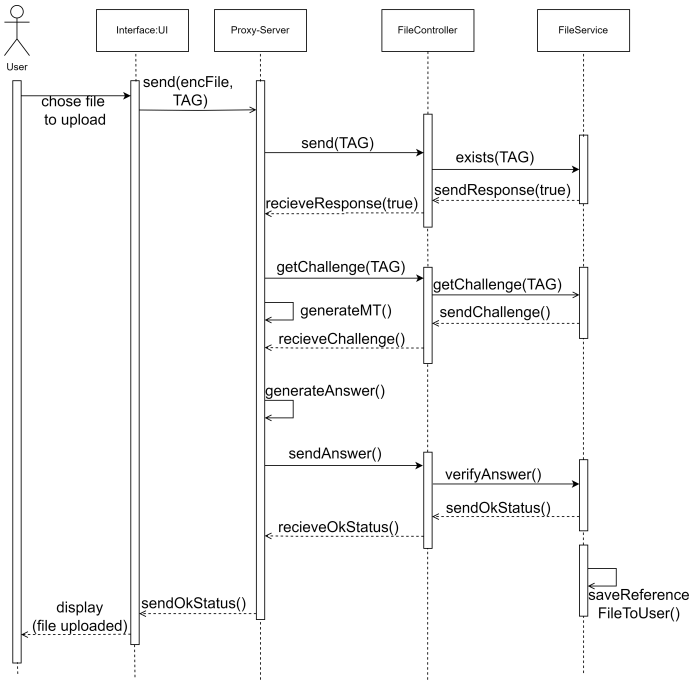

Fig. 3. Flow for uploading a duplicate file

IV. Discussion

This discussion delves into the critical aspects of security and efficiency that underscore our framework's contribution to the field, highlighting its strengths, limitations and the value it brings to secure data management practices.

A. Security

The main concern that is at the base of our designed solution is data privacy while using a cloud provider for storage. This is the main reason we chose to have end-to-end encryption. From the perspective of data deduplication of encrypted document, we considered convergent encryption to be the most straight forward choice.

As mentioned in the introductory section, when using client-side deduplication one may have the ability to monitor the traffic between the client application and server, therefore understanding which files are deduplicated or not. We aimed to mitigate this risk, but introducing the deduplication proxy (or gateway) that receives all the documents from the client itself. Afterwards, it does the PoW with the storage server and decides if server actually needs the file or not. Deduplication proxy does periodically sync actions with the storage server. By doing this, not every individual file is transferred from the proxy to the storage. Using this method, one could not infer any information about the deduplication status of any file.

Through these mechanisms, our framework assures that the data remains encrypted and inaccessible to unauthorized parties, both during transmission to the cloud and within the cloud storage environment.

As a short analysis over the cryptographic algorithms chosen for this implementation we would like to detail SHA-512 and AES GCM. First of all, we chose SHA-512 since it is a secure hashing function. Additionally, it conveniently produces 64 bytes, which are used for the next step of our security flow. The encryption algorithm (AES 256) needs 32 bytes as an encryption key and 32 bytes as an IV. Therefore, SHA-512 matches our convergent encryption needs. GCM was chosen especially for its efficiency properties: ability to accelerate using dedicated hardware, parallel processing and a low computational overhead.

B. Efficiency

This section delves into the efficiency of the proposed system. In this regard, we prioritized user experience, therefore we considered that upload and download operations are the most important to consider.

In the following analysis, we present a comparative analysis between uploading a file directly to the cloud without a proxy and uploading a file from the client to the proxy server. We do not measure encryption time, since either by using a proxy or not, client-side encryption still need to be used. On the other hand, we considered the presence of Proof of Ownership (Merkle tree) in both scenarios. The client application has been developed utilizing Blazor WebAssembly (WASM), while the proxy component has been implemented using Python FastAPI. Azure Blob Storage [17] has been employed as the cloud storage solution.

Unique files comparison:

TABLE I
Client App - Cloud

File Size	Upload Time
1 MB	2.71 sec
10 MB	22.99 sec
50 MB	140.66 sec
100 MB	298.75 sec

TABLE II
Client App - Proxy

File Size	Upload Time
1 MB	1.28 sec
10 MB	11.37 sec
50 MB	56.25 sec
100 MB	124.65 sec

Duplicate files comparison:

TABLE III
Client App - Cloud

File Size	Upload Time
1 MB	2.17 sec
10 MB	21.20 sec
50 MB	97.25 sec
100 MB	212.32 sec

TABLE IV
Client App - Proxy

File Size	Upload Time
1 MB	2.338 sec
10 MB	10.077 sec
50 MB	61.673 sec
100 MB	138.319 sec

The upload process from the client to the proxy demonstrates faster speeds due to the proxy-server's more efficient handling of Merkle tree node computations for the challenge compared to the client-side browser. The other factor is of course the convenient positioning of the proxy server closer to the user than the cloud provider. For unique files, uploading via the proxy is around 52.78% faster compared to direct cloud uploads and for duplicate files the percentage is 39.95 %.

C. Limitations

Although solving a set of challenges, proposed solution has its own limitations. For example, deduplication occurs across files from various users, which increases deduplication rate, but files uploaded cannot be shared within a group; they remain individual. Secondly, incorporating encryption and other cryptographic operations within the client-side application imposes additional overhead and demands a significant allocation of resources. Particularly for files surpassing tens of megabytes, the encryption process consumes considerable time. Still, this is the case for all solutions using ensuring end-to-end encryption.

V. Conclusions

In addressing the dual challenges of data security and storage efficiency in cloud environments, this study introduces a new approach to secure data deduplication. Central to our framework is the implementation of an on-premise deduplication proxy, which operates in concert with end-to-end encryption and cloud storage solutions. This architecture not only mitigates the security vulnerabilities inherent in traditional deduplication techniques but also leverages the scalability and accessibility of cloud storage.

We presented the architecture, security protocols and the implementation details of the proposed solution, which is

publicly available. Proposed solution combines multiple secure data deduplication approaches, as follows: proof of ownership, client-side encryption and gateway-based traffic obfuscation.

The deduplication proxy, situated within the user's infrastructure, acts as a critical intermediary, ensuring that data is efficiently deduplicated before being securely transmitted to cloud storage. This approach significantly reduces the potential for inference attacks, offering a robust solution to organizations aiming to optimize their storage resources while maintaining stringent data security standards.

This contribution is particularly timely, as it addresses the reconciliation of the efficiency benefits of cloud storage with the need of data privacy and security. Our proposal is suitable especially for companies that use a cloud storage and need to efficiently leverage secure data deduplication.

REFERENCES

[1] Understanding Data Deduplication Ratios, https://www.techtarget.com/searchdatabackup/tip/Understanding-data-deduplication-ratios-in-backup-systems, accessed 01.03.2024

[2] Project Github Repository, https://github.com/jipi2/Secure_Data_Deduplication, accessed 01.04.2024

[3] D. Harnik, B. Pinkas and A. Shulman-Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage," in IEEE Security & Privacy, vol. 8, no. 6, pp. 40-47, Nov.-Dec. 2010, doi: 10.1109/MSP.2010.187.

[4] Shin, Youngjoo, Dongyoung Koo, and Junbeom Hur. "A Survey of Secure Data Deduplication Schemes for Cloud Storage Systems." ACM Computing Surveys (CSUR) 49.4 (2017): 74

[5] Meyer, Dutch T., and William J. Bolosky. "A study of practical deduplication." ACM Transactions on Storage (TOS) 7.4 (2012): 14.

[6] Paulo, João, and José Pereira. "A survey and classification of storage deduplication systems." ACM Computing Surveys (CSUR) 47.1 (2014): 11.

[7] Halevi, Shai, et al. "Proofs of ownership in remote storage systems." Proceedings of the 18th ACM conference on Computer and communications security. Acm, 2011.

[8] Blasco, Jorge, et al. "A tunable proof of ownership scheme for deduplication using bloom filters." Communications and Network Security (CNS), 2014 IEEE Conference on. IEEE, 2014.

[9] Ng, Wee Keong, Yonggang Wen, and Huafei Zhu. "Private data deduplication protocols in cloud storage." Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012.

[10] Jia Xu, Ee-Chien Chang, and Jianying' Zhou. 2011. Leakage-Resilient Client-Side Deduplication of Encrypted Data in Cloud Storage. Retrieved December 7, 2016, from https://eprint.iacr.org/2011/538

[11] J. Liu, N. Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. In Proceedings of the 22th ACM Conference on Computer and Communications Security (CCS'15).

[12] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In Proceedings of USENIX Security Symposium 2013. 179–194.

[13] Ling, Chih Wei, and Anwitaman Datta. "InterCloud RAIDer: A Do-It-Yourself Multi-cloud Private Data Backup System." ICDCN. 2014. Kaczmarczyk, Michal, et al. "Reducing impact of data fragmentation caused by in-line deduplication." Proceedings of the 5th Annual International Systems and Storage Conference. ACM, 2012.

[14] Li, Mingqiang, et al. "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal." IEEE Internet Computing 20.3 (2016): 45-53.

[15] Y. Zhou et al., "SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management," 2015 31st Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 2015, pp. 1-14, doi: 10.1109/MSST.2015.7208297.

[16] Jay Dave, Dutta, A., Faruki, P. et al. Secure Proof of Ownership Using Merkle Tree for Deduplicated Storage. Aut. Control Comp. Sci. 54, 358–370 (2020). https://doi.org/10.3103/S0146411620040033

[17] Mazumdar, P., Agarwal, S., Banerjee, A. (2016). Microsoft Azure Storage. In: Pro SQL Server on Microsoft Azure . Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2083-2_3

[18] Merkle, R.C. (1990). A Certified Digital Signature. In: Brassard, G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_21