RESEARCH ARTICLE OPEN ACCESS

Privacy Aware Data Deduplication for Side Chanal in Cloud Storage

Mrs. B.Vijaya MCA M.Tech [1] , S Vishnu Vardhan [2]

[1] Associate Professor, Department of Computer Applications

[2] Student, Department of Computer Applications

[1],[2] Chadalawada Ramanamma Engineering College(Autonomous)

ABSTRACT

With the help of cloud storage services, businesses and individuals may send their data to distant servers instead of keeping it on-premises. In order to reduce storage and bandwidth needs, most cloud storage services use data deduplication, which involves retaining just one duplicate of a file. On the other hand, deduplication techniques may be used by an attacker to steal data. To intrude on a user's privacy, an attacker may use the duplication check to see whether a certain file (say, a pay stub with a certain name and salary amount) has previously been saved (by someone else). The ZEUS (zero-knowledge deduplication response) framework is proposed in this study. We create two privacy-aware deduplication protocols, ZEUS and ZEUS; ZEUS offers less assurance of privacy but is more efficient in terms of communication cost, while ZEUS gives better privacy assurances but at a higher communication cost. The cost and complexity of cloud storage are reduced because to ZEUS, which, to the best of our knowledge, is the first solution that handles two-side privacy without relying on any additional hardware or on heuristically determined parameters utilised by the previous systems. Finally, our proposed framework is shown to be effective in removing data deduplication-based side channels while maintaining the advantages of deduplication via testing on real-world datasets and comparison to current methods.

Keywords: - Cloud Computing, Data Security, Privacy, Protection.

I. INTRODUCTION

Outsourcing data has become more common in recent years, and as a result, the quantity of information kept in cloud storage (like Dropbox [6]) has exploded. To save money and bandwidth, cloud storages perform cross-user client-side data deduplication [25, 31], which means that only one copy of the data is stored in the cloud. To be more explicit, when a user wishes to upload a file, (s)he sends a duplication check request (dc request) to the cloud storage. The cloud service checks its archives to see whether it already has the requested file. Unless a duplicate is identified, the user must upload the complete file to the cloud storage; otherwise, a specific duplication check response (dc response) is sent to identify the presence of the file and a reference is added to the existing file. The foregoing signalling behaviour, in which the cloud delivers a dc response indicating the file existence status to the user before to the explicit file uploading, establishes a side channel for privacy leakage despite the advantages of conserving storage and bandwidth. Specifically, an attacker may detect the existence of a file by partially replicating the uploading processes and looking for evidence of deduplication.

To test which pay stub gets deduplicated, an attacker may, for instance, submit many iterations of the same pay stub from the same company, each with a different name and salary amount. Security and privacy vulnerabilities, such as the confirmation-of-a-file [15], learn-the-remaining [15], related-files attack [26], and hidden channel [15], are directly attributable to such innocuous file status spying. The inextricable link between the dc request and the dc answer lies at the heart of the deduplication-based side channel. When a cloud service determines that it already has the dc-requested file in its storage, it will always respond with a positive dc response, thus turning off the explicit file uploading. As a result of what has been said, randomising the duplication check methods is a simple technique for the side channel defence. Sadly, only a small number of countermeasures [14, 15], [20], [26], [30] have been implemented in the cloud storage system or suggested in the literature. Based on the concept of zero-knowledge response for cross-user client-side deduplication, we propose zero-knowledge deduplication response (ZEUS) as a side channel defence, which, with a weak assumption on user behaviour, provides the two-side privacy with little additional communications. As an additional step, we suggest the state-of-the-art countermeasure ZEUS, which combines ZEUS with the random threshold solution [15] to provide a more robust privacy guarantee at the expense of a little increase of communications.

II. RELATEDWORKS

When the cloud storage service gets several copies of the same file, it performs a process called data deduplication to create a logical reference to the original, unique copy of the file. Data deduplication may be implemented in a variety of ways, with choices based on factors such as where it takes place, how broadly it is applied, and how finely at-granularly the data is split. Client-side deduplication involves the user proactively doing the duplicate check through the interaction with the cloud storage, while server-side deduplication relies on the cloud storage to assess the need for an extra copy only after receiving the complete file. Fig. 1 depicts the usage of dc request and dc response for client-side data deduplication, where the dc request (e.g., "Is file f in cloud?") and dc response (e.g., "Yes/No") are used to determine whether the user needs to upload the complete data set. Please take into account that the dc check only applies to the processes involved in the passing of dc requests and dc responses. In most situations, the cryptographic hash (like SHA-256) of the data is used to implement the dc request. Because the cryptographic hash function avoids collisions, the user may determine whether or not f exists simply by checking whether or not the corresponding hash exists in the cloud. However, in cross-user (or inter-user) deduplication, just a single copy of the data will be retained, regardless of data ownership, whereas in single-user (or intra-user) deduplication, the deduplication takes place only among the data supplied by the same user. That is to say, with cross-user deduplication, almost all of the users share a single disc in the cloud storage. In addition, the deduplication's granularity determines whether it operates on files or chunks. For instance, Dropbox [6] uses a 4 MB chunk size for their chunk-level deduplication, which means that each file is divided into equal-sized chunks and the deduplication is performed on the pieces. Rolling hash (e.g., Rabin fingerprint [23]) shows effective in locating the common components of two identical items, and the chunk size may be adjusted as well [32]. A side channel is produced by the deduplication signal (i.e., dc response) that lets the user know whether a certain chunk has previously been stored in the cloud. The following privacy leaks and abuses are possible due to this kind of side channel, which is first officially stated in [15]. Despite being initially introduced in the context of convergent encryption [8], the confirmation-of-a-file [11] is easily adaptable to ours. Specifically, an attacker who suspects the presence of a certain chunk does the duplicate check to determine whether the deduplication really takes place, hence validating his or her suspicion. For example, the confirmation-of-a-file may be thought of as the simplest privacy breach caused through a side channel. The learn-the-remaining-information [15] technique is similar to brute force in that the attacker produces all possible unknown pieces and then checks for duplicates. If the dc answer is true (false), then the relevant chunk exists; else, the chunk does not exist. Here, learn-the-remaining-information approach may be seen as a series of confirmation-of-a-file invocations to learn the victim's sensitive information [17], owing to the low min-entropy nature (i.e., high predictability) of the user content. An alternative interpretation of the related-chunks attack [26] is that it is an improved form of the confirmation-of-a-file attack. In particular, because to the interdependence of all file pieces, proof of file existence may be established by verifying the presence of some subset of file chunks. This allows linked chunks to more efficiently and effectively assert ownership of the file. When two parties are not legally permitted to transmit information with one another, they may use a steganographic channel known as a "covert channel." Here, attackers may use the side channel to establish a secret channel for communicating with one another in order to avoid detection and censorship [13], [15]. One side may choose to upload or remove a preset chunk c, for instance.

The existence or absence of c, encoded as bits 0 and 1, is determined by another process that runs the duplication check on c and examines the dc response. After discovering the potential for a side channel in deduplicated cloud storage, Harnik et al. [15] advocated shuffling the deduplication threshold across different sized chunks. The deduplication threshold is the minimum number of duplicates that must be identified in order for deduplication to take place; the value of one is often used for all chunks, which implies that future uploads of the same chunk will be deduplicated if a copy is found. Since the deduplication threshold is well-documented and stable, Harnik et al. observed that it provides a backdoor into the system. It is more exact to say that an attacker may tell deduplication is triggered and a chunk is in storage if the dc response1 is positive (negative). Harnik et al. [15] advocated using random threshold (RT) on a per-chunk basis to hide the dc response.

Unlike in MT, the RT deduplication criteria per chunk are not publicly available. Therefore, it is possible that a handful of copies have been stored even if the total number of copies does not yet above the threshold and the dc response is still negative. Lee and Choi [20] claimed to have higher privacy

than Harnik et altechnique .'s by deciding ti randomly at each upload. However, in terms of privacy assurance, Armknecht et al. [1] claimed that the techniques described in [15] and [20] are equal. Wang et al. [30] used a game-theoretic strategy to calculate the deduplication thresholds rather than uniform sampling across 121;B. Wang et al. specifically characterised the deduplication as an attacker-cloud dynamic non-cooperative game. Over time, the attacker should figure out how to win against the cloud. As long as game-theoretic deduplication thresholds are applied, Wang et al. asserted, efficiency increases while privacy is maintained to the same degree. However, the attacker's strategy options are not taken into account in [30]'s reward matrix since it is static. Deduplication thresholds uniformly selected from 121;B provide the best protection for the natural privacy measure, as recently shown by Armknecht et al. [1]. All of these suggestions are RT, and as such suffer from the same limitations. Using supplementary hardware to mask network traffic is yet another option for side channel protection. The reasoning behind this is that a proxy between the user and the cloud that is able to cache dc requests may hide the network traffic by changing the sequence in which packets are sent.

One group of researchers, Heen et al. [14], made the assumption that every user would have a gateway provided by their cloud storage service. Specifically, the gateway's late forwarding strategy may disrupt the causal chain between dc requests and dc answers. However, Shin and Kim [26] assumed a third-party trusted server capable of carrying out the same function and therefore achieving the differentially private deduplicate check. The additional hardware required by these methods is a drawback. Though Heen et al. asserted feasibility and provided examples of real-world applications like NeufGiga and BT Digital Vault, their gateway configuration is still not a common implementation option, limiting the scope of their potential use cases. Similarly, there is no user-side bandwidth savings when using the technique described in [26]. Mozy [22] used a different strategy; it is based on the idea that only files of a very tiny size need to be protected since their very presence is important. In this way, deduplication is engaged if the size of the incoming file is more than the threshold and deactivated otherwise, provided a threshold for the file size for the deduplication. The selection of the threshold for the file size is, however, the major issue in this approach.

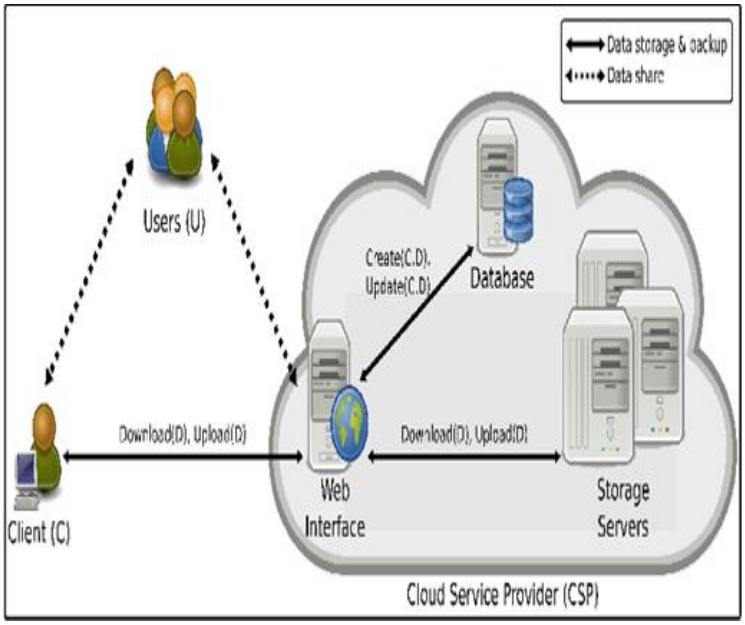

Fig.1 Data Deduplication

III. PROPOSED SYSTEM ARCHITECTURE

We think about a cloud service that does data deduplication on a client-side, shared basis, using chunks of a predetermined size. The check-first-data-later architecture includes deduplication on the client side. In this case, the file to be uploaded is broken up into pieces c, each of which has a bit length f. The user conducts the dc on c, which consists of a dc request and a dc response back and forth. To be more specific, the presence of c may be checked

with the dc request h(c), where h(.) is a cryptographic hash function (such as SHA-256). When the user receives a negative dc answer, they upload the pieces (i.e., a deduplication signal indicating the chunk inexistence). The existence status of many chunks may be determined by the user engaging in several rounds of contacts with the cloud, which is one possible implementation of the duplication check protocol. However, to reduce notational burden, we merely offer the description of a single-chunk, single-round interaction duplication check methodology. A multi-chunk, multi-round interaction duplication check technique may be simply modelled using the following concept. It is also clear that a dc request incorporates the aux component (c) and that a dc answer alludes to the function (f) (c, aux). The existing status of a randomly selected chunk c is unknown to the user (including the attacker), except that the user has uploaded c in the past. To be more precise, it is assumed that the likelihood that any randomly chosen piece of data is in the cloud, denoted by p, is very low.

The goal of a side channel attack is to discover whether or not a certain chunk c really exists. What this means is that the goal of the attacker is to determine whether or not a duplicate of c already exists in the cloud. The attacker who knows c in the traditional deduplication framework must additionally run a duplicate check to see whether c is saved in a cloud server. The presence of chunk c in the storage system may be inferred by the attacker from a positive (negative) dc response. The attacker is under no duty to finish the upload, and may stop the process at any point. In addition, a Sybil attacker is taken into account. In particular, the Sybil attacker may establish many valid accounts (named Sybil accounts) of the cloud storage and regularly execute independent deduplication checks on chunks because of the easy-to-register nature of the existing commercial cloud storage3. Similarly, Sybil accounts are not required to comply with the file uploading method and might choose to do so or not at any moment. We make no presumption on the possible number of Sybil accounts an adversary may generate. This means that our suggested methods may still accomplish their stated privacy even in the extreme scenario when all accounts owned by the attacker are Sybil accounts (i.e., the ratio of Sybil accounts is 100%). The attacker is also looking to use auxiliary gear and software to help in his mission. An intruder may, for instance, insert a network sniffer between the host and the cloud in order to read its contents. Even more so, the attacker is permitted to monitor whether or not the chunk c is accessible. If a chunk is not accessed or communicated after the duplication check, this indicates that the chunk exists. Separate concepts of privacy, "existence privacy" and "inexistence privacy," are laid forth here. Existence privacy occurs when an attacker is only able to verify the existence of the chunks that he or she has uploaded. In a more technical sense, we might say that existence privacy is defined as follows. When the existence privacy condition is met, the dc response of the duplication check protocol does not reveal whether or not c exists. Alternatively, in the inexistence privacy scenario, the attacker is unable to verify the nonexistence of the chunk. Similarly, the following is how we characterise non-existent privacy. In this paper, we claim that the existence privacy is more crucial than the inexistence privacy since the latter leaks less information. Most attacks depend on the fact that a certain portion of data is stored in the cloud, which might reveal sensitive information about the data contained inside. Learn-the-remaining-information is a classic example; after verifying the presence of the chunk, the attacker learns the secret data. One-sided privacy is all that can be accomplished using RT [12] and its derivatives [1, 20, 30]. Assuming a trustworthy gateway, the heuristic presented in [14] lacks a formal privacy guarantee. Similar to two-sided privacy, differential privacy is achieved via the method in [26]. Only for very little files does Mozy [22] disable the side channel. Additional encryption and deduplication compatibility difficulties [18], proof of ownership (POW) [12], key management [19], and poison attack [3] are only some of the security and privacy concerns surrounding cloud storage architecture. These concerns do not relate to the side channel in any way. Only the side channel will be discussed in this study.

One of the simplest ways to remove the correlation between the DC request and DC answer is to randomly generate the DC response [15]. Naive implementation of such a random response method looks like this. Let "chunk existences 0" and "1" stand for the cloud's lack of and presence of the chunks, respectively. Depending on the dc answer (), the user must (may) upload the chunk. Then there's the completely haphazard approach to responding. Please take note that from this point forward, we will refer to the dc table as the table describing the dc requests and dc answers. When the chunk is missing, the cloud must provide a negative dc answer, telling the user to upload it. However, the server has more leeway for the dc response when the chunk is there. Despite the fact that the intention of a random answer is to maintain existence privacy, the fact that a positive response is always returned reveals that the chunk does in fact exist. Sybil accounts may be used to run redundant checks on chunk c, leading the attacker to the conclusion that c does not exist. Specifically, before uploading c, each Sybil account sends up h, receives the dc answer, and then cuts off cloud connectivity. The attacker's actions have no effect on c's continued existence. Attacker has great confidence that c is not in cloud when all Sybil accounts get negative dc answers. Limiting the amount of time that passes between duplicate-checking sessions is one strategy for decreasing the effectiveness of such checks. However, the countermeasure may be readily evaded by the attacker if he or she creates several Sybil

accounts. One alternative is for the cloud to notify users that their accounts will be disabled until they finish uploading, as independent duplication checks rely on them. Full uploading in this context indicates the user uploads the chunk intentionally after obtaining a negative dc answer. However, a Sybil attacker may still circumvent this method by executing the duplication check, disconnecting from the cloud just before chunk uploading, and leaving certain Sybil accounts unblocked. This is because Sybil accounts can be established for little to no cost at all. When putting DBF into action, you have two choices. To begin, the cloud provides the standard Bloom filter with DBF memory of a predetermined size. The adaptive memory use that the dynamic Bloom filter provides compared to the initial implementation option of DBF comes at the expense of somewhat greater programming complexity. It is important to stress that the lack of privacy resulting from the aforementioned consideration is just coincidental, and that this argument is too robust and impracticable to account for the vast majority of common scenarios involving secure data uploading. Therefore, we still believe that ZEUS cannot attain inexistence privacy, notwithstanding the previous rationale. In reality, ZEUS's non-existence privacy is accomplished by counting on the fact that even good users may sometimes cancel uploads owing to issues with the network or their own erratic behaviour. In reality, implementing more trustworthy client-side software may significantly reduce the frequency of such unfavourable scenarios for good users. Furthermore, the number of chunks identified as filthy because of the anomalous communication abortion is very low as compared to the amount of actual nonexistent chunks. Therefore, one may argue that ZEUS's inexistence privacy is predicated on an overly optimistic assumption, rendering it unrealistic in practise.

IV. RESULTS AND DISCUSSION

The side channel protection will have no effect on the space-saving. In particular, the privacy is improved by the schemes in the random threshold category [1], [15], [20], [30] at the expense of bandwidth savings, while the privacy is improved by the methods in the additional hardware category [14], [26] primarily via the delayed forwarding of the request packets. While both ZEUS and ZEUS maintain the ability to save space, they do so by making use of communications that would otherwise be unnecessary. As a result, we limit our analysis to the price of communication. It's important to note that we define the communication cost as the total amount of bits needed for the full chunk uploading procedure, which includes the duplication check (i.e., dc request and dc response) and explicit chunk uploading (i.e., the chunk c, if necessary). We conducted our analysis using the Enron Email Dataset [9], the Oxford Buildings Dataset [29], and the traffic signs- dataset [28]. We opted for these datasets because we anticipate that regular people will want to back up their email and media files to the cloud. We ran the tests on a 3 GHz Intel Core 2 Duo running Fedora 12, with the kernel version 2.6.35.9. The code used for the evaluation was developed in Python 2.7.6. We used OpenSSL's SHA-256 hash algorithm and built it into our system. The data for the three groups are shown in Fig. 2. Our test environment included storing a dataset consisting of one thousand files that were selected at random. After that, we randomly selected 200 files and checked for duplicates, uploading them in explicit chunks if required.

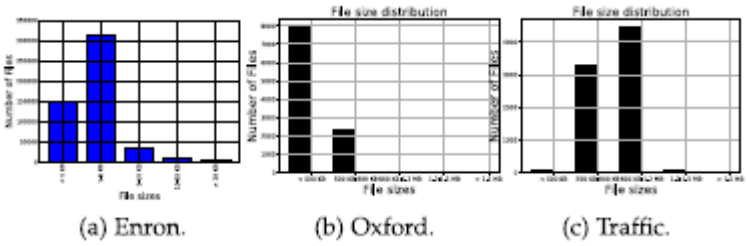

Fig.2 Datasets used

The original data deduplication algorithms are compared, as those in the random threshold category [1], [15], [20], [30] only have inexistence privacy guarantee and those in the extra hardware category [14], [26] assume the aid of additional hardware (no privacy is considered, maximum deduplication opportunity). In this context, "dirty chunks" refer to those chunks on which deduplication does not occur, and "dc requests" refer to those dc requests that do not result in deduplication. As a result, the deduplication effect is diminished when dirty chunks are used. This means the same assessments may be made with zero, ten, or twenty-five percent unclean pieces. The ratio of dirty chunks has an effect on the communication cost, as shown by the assessment findings. The costs of communication at different chunk sizes (without unclean chunks) are shown in Fig. 3. In light of the fact that attackers using ZEUS

and ZEUS are unlikely to be prepared to set up a deduplication-based side channel, and in light of the fact that even benign users seldom experience the aberrant disconnection, this instance represents the true communication cost. As a result, cloud storage would contain negligible amounts of dirty data. The ratio of dirty chunks to clean ones is shown to be a variable in Figs. 3, 4, and 5. Here, we randomly choose a set proportion of chunks as dirty chunks to observe the effect of the amount of dirty chunks on the communication cost, even if the number of dirty chunks will rise with the rising number of dc requests. From Figures 3, 4, and 5, it is clear that more dirty chunks need a higher communication cost due to the deduplication with the side channel defence. The reason for this is self-evident: if the cloud deems either chunk in the dc request unclean, it will abort the dc request's deduplication capabilities. Therefore, more communication cost is expected if there are more dirty chunks in the cloud.

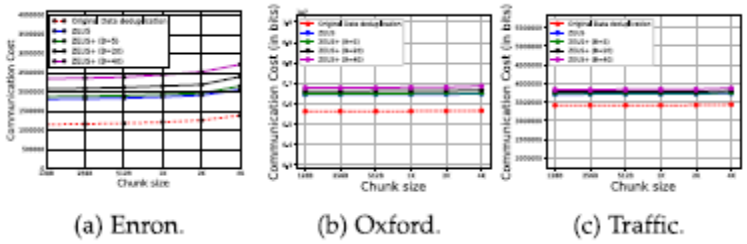

Fig.3 Communication cost for different chunk sizes (no dirty chunk).

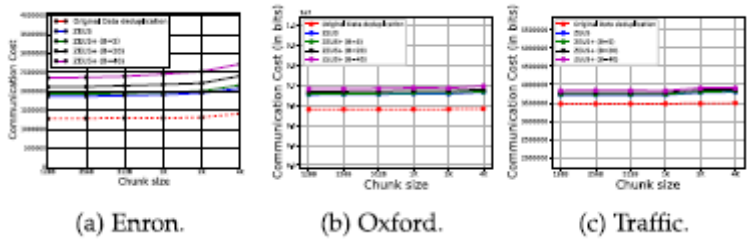

Fig.4 Communication cost for different chunk sizes (10 percent dirty chunks).

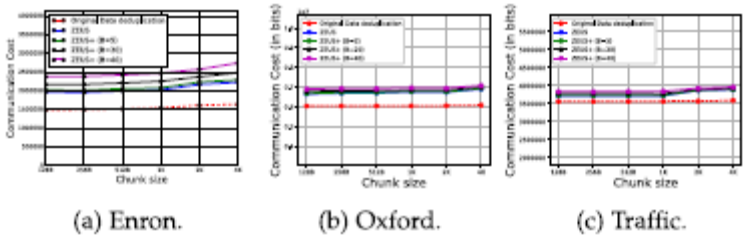

Fig.5 Communication cost for different chunk sizes (25 percent dirty chunks).

V. FUTURE SCOPE AND CONCLUSION

Cloud storage providers have implemented client-side data deduplication to save unnecessary data and communications, however this practise exposes sensitive information about the chunk's existence to potential attackers. Based on the zero-knowledge deduplication response paradigm, this article proposes two methods, ZEUS and ZEUS, that prevent an attacker from learning the existence status of a target via repeated checks. Our real-world dataset analyses reveal that ZEUS and ZEUS incur somewhat additional communications, despite the fact that they are able to provide a better privacy idea, two-side privacy.

REFERENCES

[1] F. Armknecht, C. Boyd, G. T. Davies, and Gjøsteen, "Side channels in deduplication: Trade-offs between leakage and efficiency," in Proc. ACM Conf. Comput. Commun. Security, 2017, pp. 266–274.

[2] Bitcasa. [Online]. Available: http://www.bitcasa.com

[3] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn., 2013, pp. 296–312.

[4] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," Internet Math., vol. 1, no. 4, pp. 485–509, 2004.

[5] R. Chen, Y. Mu, G. Yang, and F. Guo, "BL-MLE: Block-level message-locked encryption for secure large file deduplication," IEEE Trans. Inf. Forensics Security, vol. 10, no. 12, pp. 2643–2652, Dec. 2015.

[6] Dropbox, [Online]. Available: https://www.dropbox.com

[7] M. Dutch, "Understanding data deduplication ratios," SNIA Data Manag. Forum, 2008, pp. 1–13, http://www.snia.org/sites/ default/files/Understanding_Data_Deduplication _Ratios- 20080718.pdf

[8] 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 Proc. IEEE Int. Conf. Distrib. Comput. Syst., pp. 617– 624, 2001.

[9] Enron Email Dataset. [Online]. Available: https://www.cs.cmu. edu/_./enron/

[10] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," IEEE Trans. Knowl. Data Eng., vol. 22, no. 1, pp. 120–133, Jan. 2010.

[11] Hack Tahoe-LAFS!, [Online]. Available: https://tahoe-lafs.org/ hacktahoelafs/drew_perttula.html

[12] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in Proc. ACM Conf. Comput. Commun. Security, 2011, pp. 491–500.

[13] H. Hovhannisyan, K. Lu, R. Yang, W. Qi, J. Wang, and M. Wen, "A novel deduplication-based covert channel in cloud storage service," in Proc. IEEE Global Commun. Conf., 2015. pp. 1–6.

[14] O. Heen, C. Neumann, L. Montalvo, and S. Defranc, "Improving the resistance to side-channel attacks on cloud storage services," in Proc. Int. Conf. New Technol., Mobility Security, 2012, pp. 1–5.

[15] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," IEEE Security Privacy, vol. 8, no. 6, pp. 40–47, Nov./Dec. 2010.

[16] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and efficient cloud data deduplication with randomized tag," IEEE Trans. Inf. Forensics Security, vol. 12, no. 3, pp. 532–543, Mar. 2017.

[17] S. Keelveedhi, M. Bellare, and T. Ristenpart, "DupLESS: Server aided encryption for deduplicated storage," USENIX Security Symp., 2013, pp. 179–194.

[18] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," ACM Conf. Comput. Commun. Security, 2015, pp. 874–885.

[19] J. Li, X. Chen, M. Li, J. Li, P. P. C. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[20] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," Int. Conf. ICT Convergence. 2012, pp. 329–330.

[21] Mega. [Online]. Available: https://mega.nz

[22] Mozy. [Online]. Available: https://mozy.com/

[23] M. O. Rabin, "Fingerprinting by random polynomials," Center Res. Comput. Technol., Harvard Univ., Cambridge, MA, Tech. Rep. TR-CSE-03–01, Mar. 22, 2007.

[24] H. Ritzdorf, G. O. Karame, C. Soriente, and S. _Capkun, "On information leakage in deduplicated storage systems," ACM Cloud Comput. Security Workshop, 2016, pp. 61–72.

[25] V. Rabotka and M. Mannan, "An evaluation of recent secure deduplication proposals," J. Inf. Security Appl., vol. 27, pp. 3–18, Apr. 2016.

[26] Y. Shin and K. Kim, "Differentially private client-side data deduplication protocol for cloud storage services," Security Commun. Netw., vol. 8, pp. 2114–2123, 2015.

[27] Tahoe-LAFS. [Online]. Available: https://tahoelafs.org

[28] Traffic signs dataset, [Online]. Available: http://www.cvl.isy.liu. se/en/research/datasets/traffic-signs-dataset/download/"

[29] The Oxford Buildings Dataset. [Online]. Available: http://www. robots.ox.ac.uk/%7Evgg/data/oxbuildings/

[30] B. Wang, W. Lou, and Y. T. Hou, "Modeling the side-channel attacks in data deduplication with game theory," in Proc. IEEE Conf. Commun. Netw. Security, 2015, pp. 200–208.
[31] W. Xia, et al., "A comprehensive study of the past, present, and future of data deduplication," Proc. IEEE, vol. 104, no. 9, pp. 1681– 1710, Sep. 2016.
[32] W. Xia, et al., "FastC DC: A fast and efficient content-defined chunking approach for data deduplication," USENIX Annu. Tech. Conf., 2016, pp. 101–114.