A Summary:

Message-Locked Proofs of Retrievability with Secure Deduplication by Vasilopulos et al.

Pasquale Convertini, Alessandro Colucci, Sebastian Sanislav, Helen Möllering

December 2018

1 Introduction

Cloud computing is widely used nowadays by industries and private persons. More than 60% of the companies in Germany have already relied on cloud computing services in 2016 and it is forecasted that more than 2 billion people will use personal cloud storage in the next year.[1, 2] One reason for the increasing use of clouds is that they are a cost-effective solution of storing vast amounts of data. Nevertheless, cloud storage does not come without disadvantages and risks. Data is a sensitive good and can contain companies' business secrets or very personal information of individuals. By outsourcing it to a cloud the data owner looses the control over his data. Much research has been done to tackle security and privacy issues arising by the use of cloud storage.[4, 5, 6] One aspect is to verify data integrity as for example if the cloud is storing uploaded data correctly. Such checks are called **Proofs of Retrievability**. They enable users of cloud storage to verify that data they have outsourced to the cloud is correctly stored. On the other hand, the cloud providers also have to optimize the usage of their resources. Deduplication denotes the elimination of storing more than one copy of the same data. At the state of art in 2016 the proposed solutions for PoR contradict with deduplicating data in the cloud. The reason was that these solutions relied on a secret input by the data owner. Therefore, the same copy of data resulted in different versions when the data owner prepared it for PoR with this secret input before uploading it into a cloud. There were two main approaches at that time: watchdog-based and tag-based PoR. Data owners that use watchdog-based PoR include randomly generated blocks that are called "watchdogs" or "sentinels" in the original data and encrypt the data so that the cloud cannot distinguish between data and random blocks.[7, 8] These random data blocks are requested by the data owner for a proof of retrievability. For tag-based PoR data owners compute an authentication tag for every data block that is transmitted together with the data. When the user asks for a PoR the cloud has to return the requested data blocks and its tags to verify that it still holds the data. The work by Vasilopulos et al. is tackling the problem of PoR with deduplication by proposing a solution that allows to generate PoRs based on the message instead of secret inputs by the data owner. Therefore, preparing the same copy of data will result in the same encoding of the data for a PoR although it is done by different data owners. It follows that deduplication on the server-side is possible. This means that the server is able to detect if he already stores a copy of the data that is uploaded by a data owner and does not have to store is again. Vasilopulos et al. are introducing a protocol that allows to transform watchdog-based and tag-based PoR into message-lock PoR.[3]
In the following section 2 we will formally define PoR and introduce the threat model Vasilopulos et al. are using. Section 3 describes the message-locked key generation protocol introduced by Vasilopoulos et al. In section 4 we will shortly describe our implementation of the protocol. Section 5 is a short overview about the work about PoR that has been done since the publication of Vasilopoulos et al. In section 6, we give a short summary.

2 Preliminaries

In this section we are presenting the five components of a Proof Retrievability scheme and we will introduce the thread model used by Vasilopulos et al.

2.1 Definition of PoR

A PoR scheme consists out of five algorithms that run in polynomial time. It involves the data owner DO and the cloud storage CS that is queried for proving that it is correctly storing the data.[3]

- **KeyGen**: The DO generates key K material with KeyGen and an input with the security parameter τ.
- **Encode**: Taking the key K and the file as input the DO uses Encode to create an encoded version of the file and an unique identifier fid for the file. Both are outsourced to the cloud. The encoded version of the file contains additional data that is later used in the verification process.
- **Challenge**: To request the PoR from CS the DO invokes Challenge with the key K and fid. It outputs the PoR challenge $chal$ that is sent to CS.
- **ProofGen**: CS executes ProofGen with the inputs fid and $chal$ to generate the retrievability P that is returned to DO to verify that CS correctly stores the DO's data.
- **ProofVerif**: Upon receiving P from CS DO executes ProofVerif with the inputs K, fid, $chal$, and P. It outputs 1 if the retrievability P is valid and 0 otherwise.

2.2 Threat Model

Vasilopulos et al. assume that the cloud server is not colluding with users or with the key server that is used in their protocol to generate the message-locked keys for watchdog-based and tag-based PoR schemes. This ensures that the cloud can only access the outsourced data and the messages sent to it. In addition, they also assume that cloud users do not collude with the key server.[3]

3 Message-Locked Key Generation Protocol

In order to generate the same key for the Encode-algorithm for all data owners that want to store the same data in the cloud this key needs to be based on the data instead of randomly generated by each data owner. Such keys are called **message-locked**. It follows that the KeyGen-algorithm needs to output the same key for all the data owners that want to encode the same file. Vasilopulos et al. have proposed such a KeyGen-protocol that takes advantage of a central key server KS that interacts with the user and the cloud server CS to generate the key:

Data Owner		Key Server		Cloud Storage Server
$h \leftarrow H^*(F)$				
$\alpha, \beta \xleftarrow{R} \mathbb{Z}_p^*$				
$\hat{h} \leftarrow h \cdot g_1^{\alpha}$				
	$\xrightarrow{\hat{h}}$			
		$\hat{s} \leftarrow (\hat{h})^{\kappa}$		
	$\xleftarrow{\hat{s}}$			
$s \leftarrow \hat{s} \cdot y_{KS,1}^{-\alpha}$				
$\tilde{s} \leftarrow s \cdot g_1^{\beta}$				
	$\xrightarrow{\tilde{s}}$			
				$\tilde{c} \leftarrow (\tilde{s})^{\gamma}$
	$\xleftarrow{\tilde{c}}$			
$c \leftarrow \tilde{c} \cdot y_{CS,1}^{-\beta}$				
ret: $K_F \leftarrow H(c)$				

Explanation: $H^* : \{0,1\}^* \to \mathbb{G}_1$ and $H^* : \mathbb{G}_1 \to \{0,1\}^{\tau}$ are two cryptographic hash functions that are defined upon the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, while g_1 and g_2 represent their generators. Next, the protocol uses the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.[9] Both KS and CS generate a key pair during the setup and publish their public keys: $y_{KS,1} = g_1^{\kappa}$, $y_{KS,2} = g_2^{\kappa}$, $y_{CS,1} = g_1^{\gamma}$, $y_{CS,2} = g_2^{\gamma}$. The private keys are randomly generated and denoted by $\kappa \in \mathbb{Z}_p^*$ and $\gamma \in \mathbb{Z}_p^*$.

First, DO hashes the File F with H^* and further blinds it with a randomly generated α. The result $\hat{h}$ is sent to KS and signed with its private key. DO removes the randomly generated α from this signing result $\hat{s}$ to calculate the value s and verifies that $e(s, g_2)$ is equal to $e(h, y_{KS,2})$ Then, DO blinds s again with another random value β. The resulting $\tilde{s}$ is sent to CS and signed with its private key. After receiving the signed result $\tilde{c}$ from CS, DO removes β and verifies that $e(c, g_2)$ is equal to $e(s, y_{CS,2})$. If this check succeeds DO hashes c with H and the resulting key K_F is equal to $H(h^{\kappa\gamma})$. Relying on the private keys of both KS and CS protects against offline dictionary attacks of both parties. Online dictionary attacks can be avoided by restricting the amount of queries that can be made and therefore limiting the number of $ML - KeyGen$ runs.

4 Implementation

The programming language we have used for the implementation of the protocol is C, since it facilitated the integration with the PBC[1] (Pairing-Based Cryptography) Library by Ben Lynn. This library is built on the GMP[2] (The GNU Multiple Precision Arithmetic) Library which is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers. The PBC library is designed to be the backbone of implementations of pairing-based cryptosystems, such as the ML-KeyGen protocol. It is structured in such a way that its usage is permitted without requiring the in-depth knowledge of the mathematical details (elliptic curves, number theory) used to implement it.
We have implemented two ADTs: the data owner and the server. We created only one data type for the two servers because the behavior of the key server and the cloud server are exactly the same in the protocol, so we used the same data type for both. The most significant part of the protocol is implemented in *data_owner.c* file where the whole protocol, including the interactions with the two servers takes place. The *main.c* file is used to show that two different data owners, when executing the protocol for the same file, obtain the same key.

5 State of the Art

PoRs have been intensively researched. Armknecht et al. have published SPORT, a multi-tenancy PoR framework that also allows to combine PoR with deduplication.[10] Their solution incorporates tag aggregation based on BLS signatures. While the work of Vasilopulos et al. is relying upon the assumption that keyserver, cloud, and users are not colluding, SPORT can tolerate conspiracy between users and cloud. On the other hand, SPORT comes with higher costs in terms of computation and bandwidth. Nevertheless, both solutions do not provide deduplication with replication in a multiple storage scenario. Replication means that the same file is stored at different servers to improve reliability. Leontiadis et al. are proposing ReDup, a scheme that allows to store multiple replicas at different storage servers and each server uses deduplication while still ensuring integrity, reliability and transparency of the deduplication level. Their security model allows collusions.[11]
Vasilopulos et al. are shortly mentioning that their scheme can also be used for client-side deduplication. Client-side deduplication requires PoW. Liu et al. have published One-tag Checker that deduplicates on the client side. It is a message-locked auditing scheme based upon convergent encryption that does not require an additional proxy server. But it only provides a proof of data possession and does not offer retrievability.[16] Chen et al. proposed a message-locked PoOR that incorporates a PoW by the user to ensure the possession of a file and a PoR to verify the correct file storage by the cloud.[12] This solution does not take dynamically updated files into account. Some data never changes after an upload (e.g. movie files), but others as for example Google-Drive documents are constantly updated. The question arises how PoRs can be adapted according to changes in a file. For the scheme proposed by Vasilopulos et al. an updated file would require to re-run the outsourcing process including the ML-KeyGen algorithm. Several papers have been published about PoR on dynamic data, but to our knowledge none of them considers deduplication in combination with PoR.[13, 14]
Two other issues of PoR that have been recently researched are the simultaneous PoR of several files outsourced by one user and if an original file is recoverable from a damaged file when a PoR fails. Gorke et al. have proposed an auditing scheme that investigates the degree of damage of files to measure if a recovery is feasible. Next, they are exploiting homomorphic properties to aggregate all proofs and to enable the conduct the PoR for multiple files of one user at the same time.[15]

6 Conclusion

In this paper Vasilopulos et al. have proposed a message-locked Key Generation Protocol that allows to combine recent PoR schemes with secure deduplication. It includes a third entity, the "key server", to protect against

[1]https://crypto.stanford.edu/pbc/
[2]https://gmplib.org/

offline dictionary attacks by the cloud server while the involvement of the cloud server prevents offline dictionary attacks by the key server. To protect against online dictionary attacks rate limiting is possible. Therefore, the protocol securely generates the same key for all users that aim to upload the same file as long as the two servers are not colluding and no user is colluding with one of the server. With this key all users are able to generate the same verification data for the PoR.

References

[1] KMPG, *Usage of cloud computing in companies in Germany in 2016, by company size*, March 2017, `https://www.statista.com/statistics/473307/cloud-computing-usage-by-company-size-germany/`, last access: 09 December 2018.

[2] Cisco Systems, *Forecast number of personal cloud storage consumers/users worldwide from 2014 to 2020 (in millions)*, October 2016, `https://www.statista.com/statistics/499558/worldwide-personal-cloud-storage-users/`, last access: 09 December 2018.

[3] Vasilopoulos, D., Önen, M., Elkhiyaoui, K., Molva, R., *Message-Locked Proofs of Retrievability with Secure Deduplication*, In Proc. of ACM CCSW '16. 73–83, 2016.

[4] Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A., *Proofs of Ownership in Remote Storage Systems*, In Proceedings of the 18th ACM conference on Computer and communications security (CCS '11). ACM, New York, NY, USA, 491-500, 2011.

[5] Douceur, J. R., Adya, A., Bolosky, W. J., Simon, D., Theimer, M., *Reclaiming Space from Duplicate Files in a Serverless Distributed File System*, In Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02), IEEE Computer Society, Washington, DC, USA, 617-, 2002.

[6] Stanek J., Sorniotti A., Androulaki E., Kencl L., *A Secure Data Deduplication Scheme for Cloud Storage*, In N. Christin and R. Safavi-Naini, editors, FC, volume 8437 of LNCS, pages 99–118. Springer, 2014.

[7] Azraoui, M., Elkhiyaoui, K., Molva, R., Önen, M., *StealthGuard: Proofs of Retrievability with Hidden Watchdogs*, In Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS) , pages 239–256, 2014.

[8] Juels, A., Pors, B. S. K. Jr., *Proofs of retrievability for large files*, Proceedings of the ACM Conference on Computer and Communications Security (CCS), pages 584–597, 2007.

[9] Boneh, D., Franklin, M., *Identity-Based Encryption from the Weil Pairing*, Advances in Cryptology — CRYPTO 2001, Springer, Berlin, Heidelberg, pages 213-229, 2011.

[10] Armknecht, F., Bohli, J.-M., Froelicher, D., Karame, G., *SPORT: Sharing Proofs of Retrievability across Tenants*, Cryptology ePrint Archive, Report 2016/724, http://eprint.iacr.org/2016/724, 2016.

[11] Leontiadis, I., Curtmola, R., *Secure Storage with Replication and Transparent Deduplication*, In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18). ACM, New York, NY, USA, pages 13-23, DOI: https://doi.org/10.1145/3176258.3176315, 2018.

[12] Chen, J., Zhang, L., Kun, H., Chen, M., Du, R., Wang, L., *Message-locked proof of ownership and retrievability with remote repairing in cloud*, Security and Communication Networks. 9. 10.1002/sec.1553, 2016.

[13] Fu, A., Li, Y., Yu, S., Yu, Y., Zhang, G., *DIPOR: An IDA-based dynamic proof of retrievability scheme for cloud storage systems*, Journal of Network and Computer Applications, Volume 104, pages 97-106, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2017.12.007, 2018.

[14] Cash, D., Alptekin, K., Wichs, D., *Dynamic Proofs of Retrievability Via Oblivious RAM*, Journal of Cryptology, Vol. 30, pages 22–57 , 2017.

[15] Gorke, A. C., Janson, C., Armknecht, F., Cid, C., *Cloud Storage File Recoverability*, In Proceedings of the Fifth ACM International Workshop on Security in Cloud Computing (SCC '17). ACM, New York, NY, USA, 19-26. DOI: https://doi.org/10.1145/3055259.3055264, 2017.

[16] Liu, X., Wenhai, S., Wenjing, L., Qingqi, P., Zhang, Y., *One-tag Checker: Message-locked Integrity Auditing on Encrypted Cloud Deduplication Storage*, 10.1109/INFOCOM.2017.8056999, 2017.