Poster: Forcing the Cloud to Forget by Attesting Data Deletion

Michael Backes
CISPA, Saarland University
backes@cs.uni-saarland.de

Fabian Bendun
CISPA, Saarland University
benduncs.uni-saarland.de

***Abstract*—Cloud computing requires that data is given to a third party with an unknown IT infrastructure for a specific purpose. Consequently, this raises numerous questions regarding the control over the data: how can be ensured that the data is used for a specific purpose and only for that purpose. There are several solutions for a predefined purpose such as storing data [7], [1] or for more general purposes using multi-party computations [5]. However, it is still an open problem to ensure deletion of data. In this work, we propose a mechanism that enables remote attestation for deletion of data.**

I. Introduction

Outsourcing data and computations is a growing market. A survey by CISCO[1] shows that 25% of the survey's respondents would outsource storage first. However, the importance of data is enormous for most business applications and hence, companies need to keep control over their stored data.

There are several possible operations on data. It can not only be stored, transfered or deleted; it can also be combined in computations with other data. As a consequence, controlling data is increasingly difficult. In particular, how can be ensured data is treated as promised, i.e., is it still available after a request to delete?

Another risk of outsourcing data is that cloud providers become more valuable targets for data breaches. Successfully attacking one cloud provider can result in access to the data of many companies. Thus, the lack of control combined with the aggregation over the data increases the risk and impact of attacks on cloud services.

In this work, we present a mechanism that allows attestation of deletion as basis for a cloud storage that can be leveraged to cloud services involving arbitrary computations. Hereby, we require data encryption that reduces the risk of data breaches to the assumption of trusted hardware combined with cryptographic assumptions. Therefore, the mechanism is a building block to regain control over outsources data and mitigating the risk of data breaches in the cloud while keeping the cloud flexible regarding new tasks.

Since an untrusted operating system can replicate data without being noticed, we consider data to be deleted if the data is *unaccessible* assuming that the trusted hardware works properly and that the cryptographic assumptions hold.

Related work. There are various proposals in the area of managing cloud data. One is using cryptographic approaches such as searchable encryptions [7]. One the one hand, these approaches lead to the strongest guarantees, but on the other hand, they are mostly inefficient for practical purposes. There are other approaches based on data replication and use techniques from the area of secret sharing [11], [12]. Finally, there are also approaches that leverage trusted hardware either by enforcing data expiration such as X-Pire [2], [4] or by creating trusted database storages [9], [6]. In this work, we follow the latter by basing on trusted hardware, however, we consider as instantiation rather cheap commodity hardware whereas the trusted database storage was built on top of hardware such as IBM's secure co-processor.

[1] 2012 Cisco Global Cloud Networking Survey

II. Assumptions & the Adversary

Data breaches occur when – in the worst case – the adversary gets control over the machine storing the data. Consequently, we need to assume that the operating system is malicious. However, we assume the hardware to be trusted and containing at least a TPM v1.2 and the requirements to run Flicker [10]. These requirements are an Intel Trusted eXecution Technology (Intel TXT) or an AMD processor supporting the SKINIT instruction. In addition, we assume an asymmetric and a symmetric encryption scheme supported which is CPA secure and an unforgeable message authentication code.

What is Flicker? Flicker provides a trusted computing base by pausing the operating system and all but one processes, followed by a reset of the TPM's dynamic registers. The TPM's PCR 17 register measures the executed code by constructing a hash chain of executed instructions. Consequently, for a known code, you can predict the register's value and you can bind the TPM's access control policies to that value. Given this mechanism Flicker can attest the triple (input, output, code); so if you trust the code and the input, you know that the output was computed correctly. After the computation, flicker hands the control over to the operating system again.

The TPM's access control can be used to access keys or the limited trusted storage (NVRAM). We will use the access to keys to protect the secrecy of the data and the NVRAM to prevent replay attacks.

III. Key Idea – Attesting deletions

The main challenge – imposed by the operating system adversary – is to prevent deleted data from being used for a replay attack. Thus the trusted code needs to verify that it runs on the most recent data without checking the whole storage. This problem is similar for certificates, i.e., a browser needs

to verify that a certificate has not been revoked when using it. As a consequence we use a data revocation tree similar to a certificate revocation tree in order to store what was deleted.

The aforementioned data revocation tree needs to be protected from tampering. Thus every node of the tree is signed and encrypted using keys protected by the TPM. This does not prevent replay attacks; however, we attach a value from the TPM's NVRAM (which can be made only readable and writable with a specific PCR 17 value) to the root node. This value is rerandomized whenever a deletion is done. In addition, every parent node of the tree contains a hash of its child nodes.

Storing data in the cloud. For data storage, the owner of the data first receives an attested output of (`start`,pk,start code). Here, the receiver knows that the start code outputs a public key that is only accessible for the PCR 17s of the legitimate processing code. The data owner encrypts the data together with an access code using pk and sends it to the cloud. Let D be the corresponding ciphertext. The cloud then attests (D,`success`,store code) to the data owner.

Accessing data. Using the same pk the data owner requests reading the data by encrypting the block id b, the access code a and a public key pk_o. The code checks that the corresponding data does not occur in the data revocation tree and verifies the tree (by verifying the root and every accessed node). If the data was not deleted the storage server attests ($\{b, a, pk_o\}_{pk}$, D, read code). Here D can be decrypted using pk_o leading to the requested data.

The executed read program also checks whether the block id b is marked as to be deleted by the NVRAM and only attests ($\{b, a, pk_o\}_{pk}$, `fail`, read code) if any check fails.

Deletion. Given the deletion request for a block with id b we first check whether it was already deleted in the data revocation tree, if so, we output `deleted`. If not, we execute the following:

(1) Test whether some block id b' is stored in the NVRAM as under deletion. If so, finish the deletion process first, i.e., execute the rest from step (3) with respect to the block id b'. Instead of outputting `deleted`, we continue the execution in step (2).
(2) Store the block id b in the NVRAM indicating that it is supposed to be deleted.
(3) Choose a new random value N for the root node.
(4) Update the root node with N and the path to add the hash of b and the data to the tree.
(5) Remove block id b from the NVRAM.
(6) Output `deleted`.

The attestation of Flicker, i.e., the attestation of the triple (delete block b, `deleted`, deletion code) now gives certainty to the receiver that the data has been deleted.

IV. Conclusion

We presented a concept for attesting data deletion that can be used as a building block for more complex systems than cloud storage. This is the main advantage over simply storing encrypted data in the cloud.

One application for such a cloud storage are computations and data transfer based on policies. The policies can implement arbitrary access control policies, even such policies that have future obligations such as deletion. For example, the policy could state that whenever data was used for a statistic the data is deleted thereafter.

Another interesting application are different trusted hardware assumptions, for example TrInc [8] is a trusted counter that can only be incremented. TrInc can be used to improve performance of distributed cryptographic protocols such as multi-party computation [3] and TrInc can be implemented using the same mechanism as used for the data revocation tree.

In addition, the system could be implemented on the next generation of trusted hardware by Intel, the Intel Software Guard eXtension (Intel SGX). The main advantage is that the overhead of 10–47% that is induced by Flicker can be reduced and it can be run easily in parallel with other applications.

A drawback of the solution is that in addition to the encryption, we would need to implement an oblivious RAM in order to not reveal information about the data given the access patterns observed by the malicious OS.

References

[1] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 831–843. ACM, 2014.

[2] J. Backes, M. Backes, M. Dürmuth, S. Gerling, and S. Lorenz. X-pire! - a digital expiration date for images in social networks. *CoRR*, abs/1112.2649, 2011.

[3] M. Backes, F. Bendun, A. Choudhury, and A. Kate. Asynchronous MPC with a strict honest majority using non-equivocation. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 10–19, 2014.

[4] M. Backes, S. Gerling, S. Lorenz, and S. Lukas. X-pire 2.0: a user-controlled expiration date and copy protection mechanism. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1633–1640, 2014.

[5] A. Choudhury, M. Hirt, and A. Patra. Unconditionally secure asynchronous multiparty computation with linear communication complexity. *IACR ePrint Archive*, 2012:517, 2012.

[6] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *NDSS*, volume 3, pages 131–145, 2003.

[7] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[8] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *NSDI*, volume 9, pages 1–14, 2009.

[9] U. Maheshwari, R. Vingralek, and W. Shapiro. How to build a trusted database system on untrusted storage. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, pages 10–10. USENIX Association, 2000.

[10] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 315–328, New York, NY, USA, 2008. ACM.

[11] P. F. Oliveira, L. Lima, T. T. Vinhoza, J. Barros, and M. Médard. Trusted storage over untrusted networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[12] P. F. Oliveira, L. Lima, T. T. Vinhoza, J. Barros, and M. Medard. Coding for trusted storage in untrusted networks. *Information Forensics and Security, IEEE Transactions on*, 7(6):1890–1899, 2012.