Poster: A Self-auditing Protocol for Decentralized Cloud Storage via Trusted Hardware Components

Josh Dafoe, Niusen Chen, Bo Chen
Department of Computer Science, Michigan Technological University, Houghton, MI, USA
{jwdafoe, niusenc, bchen}@mtu.edu

***Abstract*—Ensuring integrity of the data outsourced to a decentralized cloud storage system is a critical but challenging problem. To provide this guarantee, current decentralized cloud storage systems rely on blockchain and smart contracts to establish a trusted entity which can audit the storage peers. This would result in a significant overhead as each smart contract is run on all the miners of the blockchain. By leveraging trusted hardware components equipped with the storage peer, this work has designed a unique self-auditing protocol which can ensure data integrity in the decentralized cloud without relying on the blockchain and smart contracts.**

***Index Terms*—decentralized cloud, trusted execution environment, flash translation layer, integrity auditing, self auditing**

I. Introduction

Cloud storage systems have been used extensively to store data outsourced by corporate entities and individuals. To build cloud storage systems, a centralized architecture is widely used, but many concerns have arisen with its efficacy. For example, having a few physical data centers store all the outsourced data results in data being stored further away from most users, slowing down data access; also, having all resources maintained in limited physical locations is vulnerable to large outages and failures. Therefore, cloud providers today have turned to a new decentralized architecture, in which any entity can join the system as a peer, providing storage service to earn rewards. Essentially, all the peers form "virtual" data centers. Mainstream decentralized cloud storage providers include Storj [3], Sia [1], Filecoin [6], etc.

To promote storage outsourcing, we should allow data owners to ensure integrity of their outsourced data over time. Towards this guarantee, an essential step is to detect any data corruption securely and timely. In the centralized cloud, this issue is not significant as data are stored in trustworthy data centers physically possessed and protected by the cloud provider. In the decentralized cloud however, this issue is significant as data are stored in arbitrary machines hosted by untrusted peers which may misbehave. An integrity auditing scheme is therefore highly desired in the decentralized setting. When designing the auditing scheme, we should avoid imposing too much burden on the data owners to conform to the notion of storage outsourcing [5].

An immediate solution is to maintain a blockchain, and to implement a smart contract in the blockchain which can act as a trusted entity to verify integrity of the data stored in untrusted peers [3], [6]. This solution suffers from a few limitations: First, it is inflexible. After a smart contract is committed to the blockchain, it cannot be updated due to the immutability nature of blockchain. Second, it is highly inefficient. A smart contract is typically stored across all miners, and will be executed on all of them upon integrity auditing, incurring a significant overhead in storage, communication and computation.

This work thus investigates a new integrity auditing protocol without relying on the blockchain. Our key idea is to leverage trusted hardware components equipped with each local storage peer to establish a root of trust, and to develop a self auditing protocol which can be seamlessly integrated with those trusted components. Two broadly available hardware-level trusted components, the trusted execution environment and the flash translation layer, have been explored to build our protocol. Note that we only focus on the static archival data.

II. Background

Trusted execution environment (TEE). Many processors used in modern computers are equipped with TEE features including Intel SGX, AMD SEV/SME/TSME, etc. SGX is broadly available in Intel Xeon processors and SEV/SME/TSME are broadly available in AMD Ryzen, RyzenPro and EPYC processors. The TEE allows a critical application to be run inside a secure memory area (e.g., an SGX enclave), in which the critical data and execution can be *isolated* (at the hardware level) from the untrusted world and hence remain protected even if the OS is compromised.

Flash translation layer (FTL). The FTL is a firmware layer built into main-stream solid state drives (SSD), which have occupied more than half [2] of the external storage of modern computers. The FTL is *isolated* from the OS by the storage hardware, so that even if the OS is compromised, the FTL can remain intact. Such a hardware-level isolation can ensure the security of the computation performed in the FTL even if the OS is compromised. The FTL will be responsible for handling the NAND flash memory that is structured into flash blocks, with each block being composed of flash pages.

Remote data integrity checking (RDIC). To allow a trusted party to check the integrity of data stored in an untrusted server, remote data integrity checking has been designed [4], [7]. In an RDIC, an auditor can issue a challenge to the server; the server then computes a proof based on the challenge and the stored data, and the auditor can verify whether the data are correctly stored by checking the proof. To facilitate integrity checking, a tag is computed for each data block and, both tags and data blocks are outsourced to the server.

III. Threat Model

We consider a decentralized cloud storage system consisting of storage peers. Each peer is a computer (e.g., a laptop, a desktop, a server computer, etc) owned by an individual or an organization who wants to join the storage network and to provide services. The computer is equipped with processors (with TEE enabled), RAM, as well as an SSD (with FTL built inside) as external storage. The storage peer is untrusted. Especially, the operating system (OS) of the peer may be compromised by a remote hacker, or infected by a piece of malware which is able to gain the root privilege. Both the SGX and the FTL are assumed to be trusted.

IV. Design

Our goal is to design a self-auditing protocol, which functions in each storage peer and can detect if any data have been corrupted locally. Simply adopting RDIC would not work, as a trusted third party is not immediately available in the setting of decentralized cloud. We therefore rely on the TEE equipped within the storage peer to act as the trusted auditor. The TEE can always ensure that the auditor program is running in an isolated memory region even if the OS is compromised. Periodically, the auditor will issue a challenge to the local SSD. For efficiency, a random checking technique [4], [7] is used, in which the auditor only challenges a random subset of the stored data. The OS will return this random subset of data, together with the corresponding tags, and the auditor will check their integrity. Two additional research challenges need to be addressed in our setting considering the OS is untrusted:

First, how can the auditor ensure that the data being challenged are really from the local SSD? The storage peer may outsource it's data to some less expensive store, with no security guarantee, for financial gain. This data may be far away from the peer (slowing data access) and less reliable. Thus, we must guarantee that the data are actually stored in the SSD of the storage peer. Our observation is, the data stored in the local SSD is ultimately managed by the FTL, which remains trusted even if the OS is compromised. Therefore, upon integrity checking, if the OS does not fetch the data from the local SSD, the FTL will be aware of such a misbehavior and inform the auditor "somehow". The SSD access from the OS eventually will result in the access of the flash memory pages. Therefore, the FTL can embed some "secret" into the page being accessed, and the auditor can extract this secret and confirm that the data really comes from the FTL. Specifically, the FTL can encrypt the data from each flash page which have been read by the auditor in the TEE, using a symmetric key shared between the TEE and the FTL. Upon receiving the data, the TEE can decrypt the ciphertext using the same key. If the ciphertext cannot be decrypted properly, the RDIC will fail as the tags will not be corresponding to the challenged data. To prevent the replay attack, the symmetric key should be changed upon each integrity check. This can be addressed by sharing a master key between the TEE and the FTL, and generating an ephemeral key for each integrity checking via a key derivation function (KDF) based on the master key. One optimization towards reducing the encryption/decryption computation is to only encrypt/decrypt a portion of data randomly selected from each page, and the random location can be determined by the ephemeral key. To share the master key securely between the TEE and the FTL, a key sharing protocol (e.g. ECC based Diffie-Hellman) can be used when the peer is initialized.

Second, how can the auditor ensure the challenged data come from the desired block locations on the SSD? For example, the auditor randomly picks a data block at block address 2,000 and checks its integrity; however, the untrusted OS controls the read system call, so may return the data at block address 20,000 as the data at 2,000 have been corrupted. Our observation is, different blocks at the OS level will correspond to different logical flash addresses controlled by the trusted FTL. Therefore, if we bind each data with the corresponding logical flash page numbers, the adversary will be unable to perform the aforementioned attack. Our solution is, upon deriving the ephemeral key via the KDF and the master key shared between TEE and FTL, we also use the corresponding logical page number as input to the KDF (typically, the logical flash page numbers can be computed from a given block address).

V. Preliminary Implementation and Evaluation

We have implemented a prototype of our design using the open-source FTL firmware OpenNFM (ported [8] to an electronic development board LPC-H3131 with 180MHz ARM micro-controller, 32 MB SDRAM, and 512MB SLC NAND flash) and Intel SGX (equipped in Lenovo Yoga C940 with Intel Core i7-1065G7 1.3 GHz and 12GB LPDDR4 3733 MHz RAM). Further, preliminary evaluations of our prototype during three essential phases demonstrate the low overhead. The setup phase implements a key sharing protocol between SGX and FTL. During file preparation, the SGX creates tags used for the auditing process. The auditing process, managed by the SGX requests the necessary data from the untrusted OS, and evaluates the integrity and source (from FTL). Our preliminary results are summarized in Table I.

	Key Sharing (one time)	Prepare File (one time)	Audit File
time in FTL (s)	7.73	.004	.1
time in SGX (s)	1.15	.006	.05

TABLE I
TIME FOR DIFFERENT PHASES WITH SMALL (10 BLOCKS) FILES

Acknowledgment. This work was supported by US National Science Foundation under grant number 2225424-CNS, 1928349-CNS, and 2043022-DGE.

References

[1] Sia. http://sia.tech/.
[2] Ssd market share. https://www.t4.ai/industry/ssd-market-share.
[3] Storj - decentralized cloud storage. https://storj.io/.
[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *CCS '07*.
[5] B. Chen and R. Curtmola. Towards self-repairing replication-based storage systems using untrusted clouds. In *Proc. of CODASPY '13*.
[6] Filecoin. Filecoin. https://filecoin.io/.
[7] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proc. of Asiacrypt '08*.
[8] D. Tankasala, N. Chen, and B. Chen. A step-by-step guideline for creating a testbed for flash memory research via lpc-h3131 and opennfm. 2020.