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

Abstract

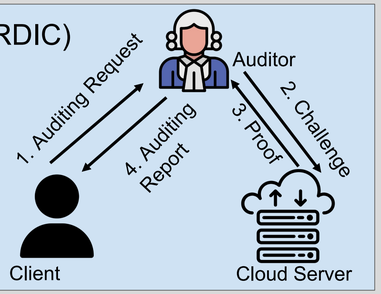
- Ensuring integrity of the data outsourced to a decentralized cloud storage system is a critical but challenging problem.
- Current decentralized cloud storage systems rely on blockchain to establish a trusted entity which can audit the storage peers using smart contracts. This brings significant overhead 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.

The Move to Decentralized Cloud

Centralized cloud, having only a few physical data centers, results in data being stored further away from users, slowing down data access. Having all resources maintained in limited physical locations is vulnerable to large outages and failures. Thus, cloud providers today have turned to a decentralized architecture.

Remote Data Integrity Checking (RDIC)

- Auditor can issue a challenge to the server
- Server computes a proof based on the challenge and the stored data
- Auditor verifies whether the data are correctly stored by checking the proof



Smart Contract Based Auditing

- Current decentralized storage solutions rely on blockchain based smart contracts
- There are several limitations to this approach:
  - ➢ The smart contracts are stored and ran on all miners, increasing the system burden.
  - ➢ The smart contracts are immutable, so cannot adapt to change.

Flash Translation Layer (FTL)

- The FTL is a firmware layer built into solid state drives (SSD)
- 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.

Trusted Execution Environment (TEE)

- Support secure computation in modern computing devices through hardware to create a secure memory area that is isolated from the normal OS
- Examples: Intel software-guard extensions (SGX), AMD secure encrypted virtualization (SEV), ARM TrustZone.

Adversarial Model

- We consider a decentralized cloud storage system which consists of storage peers where each storage peer is a computer owned by any individual or an organization who wants to join the storage network and provide storage services to earn profits.
- The computer is equipped with processors (with TEE enabled), RAM, as well as SSDs as external storage.
- Each storage peer is untrusted
  - ➢ The 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

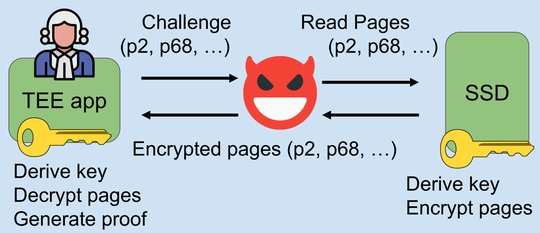
Design

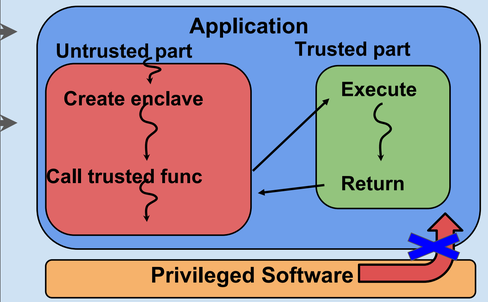
- An RDIC runs in the TEE. The TEE generates a challenge, requesting the associated data from the SSD. The TEE also verifies the data integrity upon receiving it.
- Two additional research challenges need to be addressed in our setting considering the OS is untrusted:

1. Ensuring that the data being challenged are really from the local SSD:
   - ➢ A master key is shared between FTL and TEE, then an ephemeral key for each integrity checking is generated via a key derivation function (KDF) based on the master key.
   - ➢ The FTL will encrypt the data from each page read from the SSD, using the ephemeral key (an optimization is to only encrypt a randomly selected portion of data).
   - ➢ TEE decrypts the ciphertext using the same key. If the ciphertext cannot be decrypted properly, the RDIC will fail.
2. Ensuring the challenged data come from the desired block locations on the disk:
   - ➢ We observe that the each data block is stored on a disk location which is mapped deterministically to a logical address (corresponding to one or more logical page numbers) visible to the FTL
   - ➢ We can use the corresponding logical page numbers as input to the KDF to derive the ephemeral key

(1) is necessary so the peer cannot profit from storing the data in a cheaper, less reliable store.

(2) is necessary so the peer cannot send a different block than requested while the correct one is corrupted.





Preliminary Results

We have implemented a prototype of our design using the open-source FTL firmware OpenNFM [1] and Intel SGX. Further, preliminary evaluations of our prototype during three essential phases demonstrate the low overhead. The setup phase implements the 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 the following table:

	Key share (one time)	Prepare file (one time)	Audit file
FTL time (s)	7.73	0.004	0.1
SGX time (s)	1.15	0.006	0.05

Acknowledgments/References

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

[1] Opennfm.https://code.google.com/p/opennfm/, 2011.