# *ChainFS: Blockchain-Secured Cloud Storage*

Qiwu Zou[1], Yuzhe Tang[1], Ju Chen[1], Kai Li[1],
Charles A. Kamhoua[2], Kevin Kwiat[3], Laurent Njilla[3]
[1] *Department of EECS, Syracuse University, New York*
[2] *Army Research Lab, Adelphi, Maryland*
[3] *Cyber Assurance Branch, Air Force Research Laboratory, Rome, New York*

***Abstract*—This work presents ChainFS, a middleware system that secures cloud storage services using a minimally trusted Blockchain. ChainFS hardens the cloud-storage security against forking attacks. The ChainFS middleware exposes a file-system interface to end users. Internally, ChainFS stores data files in the cloud and exports minimal and necessary functionalities to the Blockchain for key distribution and file operation logging. We implement the ChainFS system on Ethereum and S3FS and closely integrate it with FUSE clients and Amazon S3 cloud storage. We measure the system performance and demonstrate low overhead.**

## I. INTRODUCTION

Cloud security continues to raise concerns as new security applications (e.g., IoT clouds, healthcare clouds, etc) become popular and new attack incidents (e.g., data breaches) emerge. The root cause is that clients lack trust to the cloud as a third party. The trust problem adversely affects cloud adoption rate in emerging application domains. On the other hand, the Blockchain technology has recently been adopted for supporting the world's first successful cryptocurrency, BitCoin [3], followed by many others, such as Ethereum [4], Litecoin [5], etc. Through the cryptocurrency applications, the Blockchain design has shown the great potential to behave as the first practical trusted third-party (TTP).

In this work, we propose to use Blockchain as a trusted third party (TTP) to harden the security of cloud storage, and to defend forking attacks. Suppose the data stored on the cloud is end-to-end encrypted. The attack surface is reduced to and refocuses on the planes of key management and meta-data about encrypted data. A forking attack [10] is such that a malicious

cloud server can present different views (of the same query) to different clients. Fork consistency states that an untrusted server can launch forking attacks but cannot evade detection by merging forks.

Blockchain can be repurposed as a log of cloud-storage operations, a scheme inspired by Catena [11]. With Blockchain, it hardens the security of cloud storage, as it prevents forking attacks in the cloud file system. The prevention is due to the fact that forking file-system views is as hard as double-spending transactions in Blockchain.

In our Blockchain-secured cloud storage, Blockchain is involved in securing A1) a key directory and A2) an operational log server. For A1), a public-key directory, as specified in the key transparency scheme [12], helps manage user identities in the cloud and is the foundation of establishing trust among users for data-sharing applications. Had such public-key directory served by an untrusted cloud, the directory cloud can fork directory views and present different public keys to different users regarding the same request to obtain Alice's public key. For A2), the operational log, as specified in the SUNDR protocol [10], records the client-server interactions when accessing a remote file system. A malicious storage server may launch a forking attack to present different views to different clients. Note that forking attack on encrypted file content is feasible, as a stale ciphertext block is still legitimate and can be decrypted by an intended client (see Sec.III.D for a detailed discussion).

We build a system instantiating the design. Our implementation is based on the S3FS project [13], which hooks client file-system operations with Amazon S3 cloud storage using Filesystem in Userspace or FUSE [14], [15]. We systematically examine the fuse operations and extract state information relevant to possible forking attacks. We design Blockchain logging schemes for storing two states, a public key directory and a file operational history. The stateful Blockchain logging is implemented by the smart-contract states on Blockchain, and stateless logging is implemented by the transactional interface of Blockchain. Our implementation is functional on Ethereum.

The contribution made in this paper includes the following:

1. We apply the Catena scheme [11] in two cloud-storage contexts: key distribution and file-system operations. We consider both stateless log and stateful index.
2. We implement a functional system and integrate it transparently with client-end Linux file systems, cloud-end Amazon S3 storage[1], and Ethereum Blockchain[4]. We demonstrate the performance overhead of ChainFS is low in practice.

## II. RELATED WORK

**Blockchain** is a readable, append-only, and distributed storage system for storing the history of transactions (or ledger), which is materialized on an open-membership P2P network where peers are not required to register an account for joining the Blockchain network. The open-membership design is essential to large-scale deployment of Blockchain on the planet. Furthermore, this design is secured by a mining mechanism, where adding a transaction to Blockchain requires so-called mining, which is intensive computations that solve puzzles [16].

In addition, Blockchain ensures no double spending transactions are included, that is, there are no two transactions that spend the same set of coins. This is realized by transaction validation work in Blockchain.

**Secure remote file systems**: SUNDR [10] is a remote file system (in a one-server-multi-client setting) that ensures fork consistency to the clients. IPFS [17] is a P2P file system based on DHT. Storj [18] is a secure P2P file system by file encryption and authentication. In addition, Storj support [19] based on a Merkle tree design. CONIKS [12] is a key transparency scheme which maintains a public-key directory for secure messaging. To prevent forking attacks against the key directory, CONIKS requires identity providers to periodically publish the log of directory snapshot digested by a Merkle trie. The log is published to peer identity providers. The peer providers are untrusted but are assumed not to collude with the owner providers.

**Non-cryptocurrency applications of Blockchain:** TPAD[32] supports the logging of cloud-client operations on Blockchain in a way to support data and query authenticity. ProvChain [34] stores the trace of cloud data operations on Blockchain in the hope of enhancing transparency for data auditability. ProvChain models the operation trace by

provenance data and supports data validation. Work [35] models the block-withholding attack (a variant of self-mining attack) and analyzes the security of existing Blockchain systems. CloudPoS[36] is a proof-of-stake consensus protocol tailored for a client-cloud system running as the substrate of a private Blockchain. IncBM cloud storage [33] ensures the data freshness and authenticity by an authenticated data structure based on Merkle tree and Bloom filters. Other works support data indexing of cloud storage system [28] and P2P storage systems [29]–[31].

## III. ChainFS System Design

### A. *System Overview*

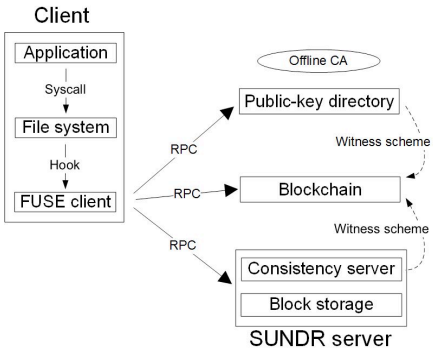


Figure 1. Overview of system architecture (CA represents certificate authority)

Overall, our system consists of three parties: a client, a server hosted by untrusted cloud, and a Blockchain. The client machine runs applications and operating systems for the end user; it manages user's private files and stores these data securely in remote parties, through a FUSE [20] client. The system architecture is as illustrated in Figure 1. The FUSE client interacts remote parties in two planes: key management and data storage.

On the key-management plane, a public-key directory stores the binding between public key and identity. With an external offline certificate authority, it also keeps the certificate for the key-identity binding signed by the authority.

On the data plane, we consider the SUNDR protocol[10] which dictates two server components: a block store that stores

the file contents and a consistency server which is an auditable log of RPC operations.

Both SUNDR server and public-key directory are hosted by untrusted platforms, such as public cloud services. These servers can launch forking attacks to present different views to different clients. For instance, for the public-key directory, forking attack can manifest in a man-in-the-middle attack [21] where the untrusted directory can play the role of the man in the middle and attack by presenting a fake key of Alice to Bob, while presenting Alice's true public key to Alice. This can lead to the consequence that Bob get connected to a person who is not Alice. For SUNDR server, the untrusted consistency server can present to Alice a copy of global operational history to conceal from her consistency violation in her operation history, while presenting to Bob another copy of the history.

In theory, with untrusted servers, fork consistency is the best one can achieve and it states that the untrusted server can launch a forking attack (i.e., it cannot be prevented without a trusted third party) which can, however, be detected given an external client-gossiping mechanism.

The blockchain is used as a trusted third party to proactively detect or even prevent forking attacks. That is, the Blockchain, as a cryptocurrency ledger with the security of no double spending, can be repurposed as a witness scheme. In Blockchain witness, forking application views is made as hard as forking Blockchain. Due to the double-spending security, it prevents, not only detect, forking attacks at the application layer. In other words, the malicious server cannot record the two forks of a log in the Blockchain as the forks will be two double-spending transactions and Blockchain’s validity logic will invalidate at least one of them.

### B. *Blockchain Witness for Operation Log*

In SUNDR [10], the consistency server maintains file-system operation history. The history can be materialized as a log or a version structure (the vector of latest per-user versions). The untrusted consistency server can launch fork attacks by simply presenting client Alice one copy of the history and client Bob another copy of the history. SUNDR achieves fork consistency which makes forking attack detectable but does not prevents such attacks.

To prevent forking attacks, we propose to store the dynamic SUNDR operation history in Blockchain. For instance, when Alice reads file foo from the block server, a new log entry is generated that binds with log index i the following operation <*Alice,read,foo,foo's content*>. The log server sends a transaction encoding the new log entry and its index to the Blockchain.

When generating the transaction, the sender is the full UTXO[16] (or unspent transaction output) of the previous transaction (i.e., corresponding to the log entry of index i-1), the receiver is the account address of the log server and the transacted coins are the full amount of UTXO from the sender "account" (minus the transaction fee). By this means, the UTXO of the transaction of log entry i-1 is fully spent, which is essential to the forking attack security as analyzed below.

Alice finishing the read operation would then proceed to verify the operation is reflected in the Blockchain by checking three conditions: C1) whether transaction i is included in Blockchain, C2) whether transaction i spends the full UTXO of transaction i-1, C3) whether transaction i is signed by SUNDR consistency log. Alice then checks the three conditions for all transactions before transaction i. If all transactions satisfy the three conditions, Alice can be assured that the linear chain of transactions is the same chain that will be verified by all other clients. She then checks if the transactions semantically match her local view, that is, C4) if transaction i's OP_RETURN matches with what she receives from the log server. In addition, she will check C5) if any operation before i is semantically consistent with operation i. For instance, if Bob queries the same file at position i-1 and the content he got is different from Alice, this may violate the fork consistency. After the five conditions, namely C1,...C5, are met, Alice can be convinced that the read result is globally consistent and there is no forking attack. The security analysis of this process is below.

Suppose there is a successful forking attack that can bypass the client verification. This means the log server needs to return to Bob value X while return to Alice value Y ($X \neq Y$). If Alice's verification passes, the log that Alice sees must be that Bob and Alice have the same value, say value Y. In other words, Blockchain returns to Alice log entries Bob|Y,Alice|Y and returns to Bob log entry Bob|X. This is equivalent to say that in order to have a successful forking attack on the log, the

Blockchain itself needs to be forked (at the time Alice requests the log and the time Bob requests), which is difficult.

### C. *Blockchain Witness for Key Directory*

Key transparency schemes, such as CONIKS[12], maintain a dynamic key directory (subject to key insertions and revocations) with global consistency. The consistency is about prevention of forking attacks --- when a CONIKS client receives the request key bindings; it can be assured that the same key bindings are returned to other clients. To verify the consistency, CONIKS leverages a Merkle-trie authenticated data structure and publishes the root digest among all clients using a Gossiping protocol.

We propose to leverage a Blockchain witness scheme to replace gossiping protocol in the CONIKS directory service.

To set up the stage, there are three properties a CONIKS client are required to verify: D1) the digests are linearly chained, D2) non-equivocation among all CONIKS clients, D3) correct keys are included in the CONIKS bindings. The trail of a dynamic CONIKS directory consists of a series of directory snapshots, each digested by a Merkle trie, and the series of Merkle root digests.

To map CONIKS to Blockchain that satisfies the three requirements, we propose the following scheme: Every time the CONIKS directory produces a digest (or STR), the digest is sent over to the Blockchain in a transaction. The format of the transaction is the following: sender and receiver are fixed to the account of CONIKS directory. The coin amount is the minimal transaction fee. The digest is embedded in the transaction in OP_RETURN [11], [16] field.

When a client or a third-party wants to audit, she downloads the log of STRs from the Blockchain. She then goes through the chain and ensures the total order of STRs (C1).

When a client wants to monitor, she downloads her bindings from CONIKS directory with Merkle proofs. She also downloads the STR log from the Blockchain, such that she can verify the authenticity of the CONIKS bindings. She then checks the equality of the CONIKS keys and her local keys.

### D. *Security Analysis*

ChainFS leverages the Blockchain to add forking-attack resilience to an encrypted file system on the cloud. For both public key directory and file system operations, it maintains a linear log of application-specific entries, and maps the log to Blockchain.

For forking attack in the public directory, the cloud presents different key-name bindings (of the same person) to different clients. For the client to accept the binding, recall that it checks the inclusion of the binding in the snapshot of the directory with digest as a log entry that is audited in the Blockchain. For two clients to accept two forked bindings, there must be two log entries audited in the Blockchain at the same day. In other words, the log in Blockchain must be forked, which is as hard as double spending the transaction in the Blockchain.

For forking attack in the SUNDR operational log, the storage server can present two encrypted and different files to two different clients. These two events are recorded in these two clients' local logs. Also, the cloud server will record its own version of global log to the Blockchain. During the log attestation and auditing, local logs will be compared against the global log to test if the local log is a subset of the global log, and optionally, if there is any violation of storage consistency[22]. For the forking attack to bypass detection, one needs to force Blockchain to fork itself. Forking blocks that are confirmed (e.g., after 6 times) is difficult in a public, large-scale Blockchain.

## IV. SYSYTEM IMPLEMENTATION

A FUSE client exposes the internal of a file-system to the user space and allows to custom a file system in a transparent way. Concretely, there are two APIs, respectively at the syscall level (e.g., stats, write, etc.) and at the kernel level (e.g., lookup, etc.).

Each Linux file is stored by data blocks (for storing its content) and by inode (for storing metadata). The metadata of a file includes modes (i.e., the permission list), the owner user, group user (roles) and access dates (e.g., modify date, create date, etc.). In addition to the persistent data, opening a file in

Linux will create dynamic data in memory, such as file descriptor, which binds an open file with a process.

We implement our system design based on the S3FS project[13], which allows a Linux user to mount a remote file system in an Amazon-S3 alike cloud. Briefly, S3FS overwrites the syscall FUSE APIs (e.g., stats, write, open, etc.) and redirect the file storage to the cloud --- Both inode and content are stored remotely.

To build an authentication layer on S3FS, we first use a hash function to digest file content and then use an intra-file Merkle tree to digest the inode metadata and content hash. To manage files in a directory tree, we build an inter-file Merkle tree where each leaf is the root hash of an intra-file Merkle tree.

We store the root hash of inter-file Merkle tree in Blockchain and keep the Merkle tree itself off-chain in the cloud. The off-chain Merkle tree itself is stored as a shadow file on the Blockchain.

For each read operation (e.g., stats or read), we verify the authenticity and membership of query result using the Merkle proof against the latest hash stored on Blockchain. For each write operation (e.g., write, chmod, chown), we first verify the result from the cloud using the Merkle proof and then update the local state with the new root hash generated before sending it to the Blockchain.

In the setting of multiple clients, file-system operations are logged in a global state stored in Blockchain. Each client accessing a shared file would download the entire log chain to check following facts: F1) the operations/transactions in the log are linearly chained in a total order, F2) strong consistency (i.e., every read reflects the latest write in the total-order log) can be checked by iterating through the chain. Note that fork consistency is assured in these checks because the mapping scheme to transaction renders forking file-system operation log to be as hard as forking the Blockchain itself, as analyzed before.

In our current implementation, the FUSE-blockchain interaction is realized by having an intermediary web-server process that relay the message from FUSE to the Blockchain. The FUSE client in ChainFS, upon sending requests to the remote cloud, would send a CURL [23] request to the web-

server process. The web server runs javascript that translates the received CURL request to a Geth request and relays the request to the Ethereum Blockchain. Implementing the hash function is based on the SHA256 algorithm. We note that our implementation may not be performance optimal as it relies on this web-server intermediary process. Nevertheless, we demonstrate the performance efficiency in our performance evaluation.

## V. PERFORMANCE

In our experiment setup, we have three types of machines. First, we create an account on Amazon S3 AWS and run cloud instances. Second, we run our FUSE clients on a local machine. The client machine has the CPU of Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz and a 10-GB memory. Third, the Blockchain runs on three server machines with the following specification: Intel 8-core i7-6820HK CPU of 2.70GHz and 8MB cache, 32 GB Ram, and 1TB Disk.

**File Create/Write Performance**: In the experiments, we first use LFS [24] small file benchmark to generate 1000 small files with sizes varying from 1 KB to 100 KB. We use the Linux dd utility [25] for file generation and time measurement. We report the average time and standard deviation. Here, the files are generated using random contents such that the digests to put on Blockchain will change, and ChainFS will not have an unfair advantage.

The performance result of the small-file experiment is presented in Figure 2. Comparing the ideal case that runs an S3FS without Blockchain, our ChainFS has up to 35% performance overhead (with 10 KB files). The overhead decreases as the files grow large. In particular, when the files are too small, the performance becomes unstable. Involving Blockchain does not increase much standard deviation, and we suspect this is due to that the original cloud connection has a certain degree of uncertainty.

Following a similar procedure, we also conduct experiments with large files. We generate files with file size varying from 1 MB to 1 GB. We measure the execution time and report metrics in a similar fashion to the small file case. The result is reported in Figure 3. The Blockchain overhead increases as the file

grows large and reaches the maximal about 28% (at 1 GB file). In the large file setting, the system bottleneck is at transferring data across the Internet (e.g., between client machines and cloud).

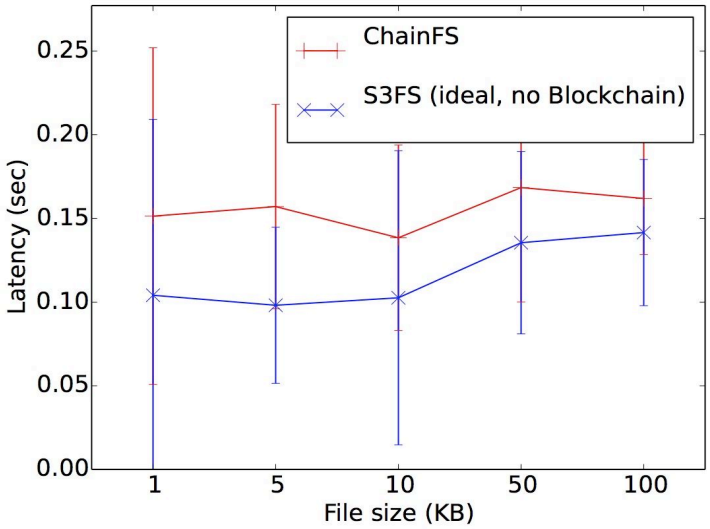


Figure 2. File write latency with small files

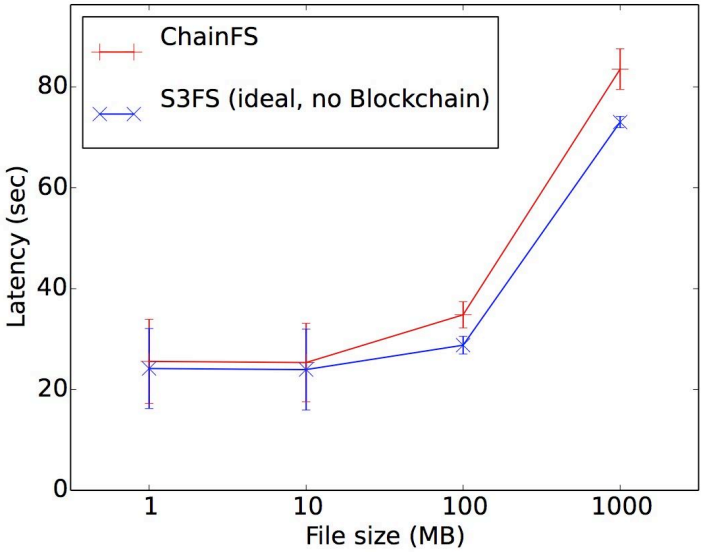


Figure 3. File write latency with large files

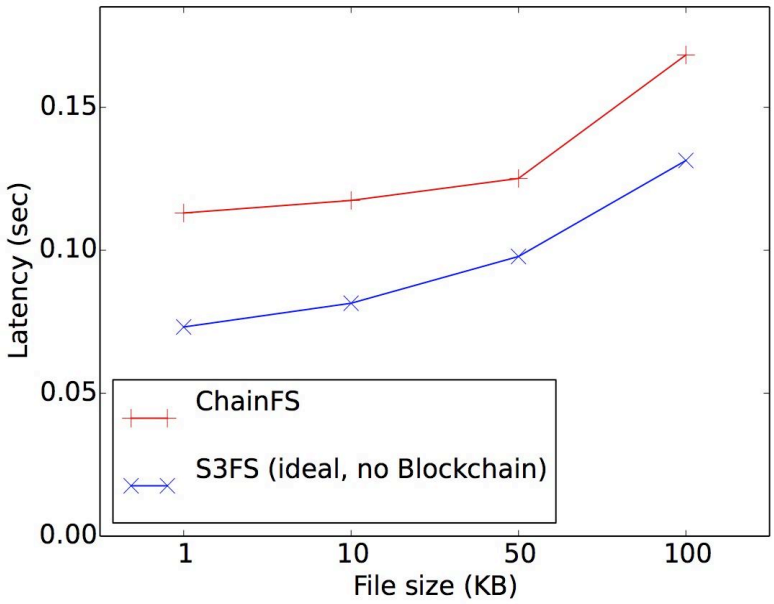


Figure 4. File read latency

**File Read Performance**: We conduct an experiment to evaluate the read latency of ChainFS. In the experiment, the client machine first runs a script to create 100 files of variable sizes (from 1 KB to 100 KB). It then launches a series of Linux CAT commands to repeatedly read the files. In this experiment, we measure the time spent on the second stage (i.e., running CAT commands). We report the performance result in Figure 4. It can be seen that on average, ChainFS adds about 30% overhead to the regular read path of a cloud file system. As the file grows large, the overhead stays const.

## VI. CONCLUSION

This paper presents ChainFS, a multi-client file system in the cloud whose security is hardened by leveraging Blockchain. ChainFS prevents forking attacks by the hardness of double spending in the Blockchain. It systematically applies the security design to the system of an encrypted cloud storage, on the planes of key management and file operation log. We implement a functional prototype of ChainFS based on S3FS and Ethereum and demonstrate practical performance. ChainFS enables end users to securely shares data in the cloud.

## REFERENCES

[1] “Amazon AWS.” [Online]. Available: https://aws.amazon.com/.
[2] G. Developers, “Google cloud computing, hosting services & apis.” 2015.
[3] “Bitcoin - Open source P2P money.” [Online]. Available: http://bitcoin.org. [Accessed: 29-Jan-2018].
[4] “Ethereum Project.” [Online]. Available: http://www.ethereum.org. [Accessed: 29-Jan-2018].
[5] “Litecoin - Open source P2P digital currency.” [Online]. Available: http://litecoin.org. [Accessed: 29-Jan-2018].
[6] “Signal >> Home.” [Online]. Available: https://signal.org. [Accessed: 31-Mar-2018].
[7] “WhatsApp,” WhatsApp.com. [Online]. Available: https://www.whatsapp.com. [Accessed: 31-Mar-2018].
[8] “Keybase.” [Online]. Available: https://keybase.io/. [Accessed: 31-Mar-2018].
[10] J. Li, M. N. Krohn, D. Mazieres, and D. E. Shasha, “Secure Untrusted Data Repository (SUNDR),” in OSDI, 2004, vol. 4, pp. 9–9.
[11] A. Tomescu and S. Devadas, “Catena: Efficient Non-equivocation via Bitcoin,” in 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 393–409.
[12] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, “CONIKS: Bringing Key Transparency to End Users,” in USENIX Security Symposium, 2015, vol. 2015, pp. 383–398.
[13] s3fs-fuse. Github.
[14] Wikipedia contributors, “Filesystem in Userspace,” Wikipedia, The Free Encyclopedia, 14-Mar-2018. [Online].
[15] libfuse. Github.
[16] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, “Bitcoin and Cryptocurrency

Technology (manuscript). 2015," Retrieved 8/6, 2015.
[17] P. Labs, "IPFS is the Distributed Web," IPFS. [Online]. Available: https://ipfs.io/. [Accessed: 25-Mar-2018].
[18] "Storj - Decentralized Cloud Storage," Storj - Decentralized Cloud Storage. [Online]. Available: https://storj.io/. [Accessed: 25-Mar-2018].
[19] A. Juels and B. S. Kaliski Jr., "Pors: Proofs of Retrievability for Large Files," in Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, 2007, pp. 584–597.
[20] libfuse. Github.
[21] J. Bonneau, "EthIKS: Using Ethereum to Audit a CONIKS Key Transparency Log," in Financial Cryptography and Data Security, 2016, pp. 95–105.
[22] B. H. Kim and D. Lie, "Caelus: Verifying the Consistency of Cloud Services with Battery-Powered Devices," in 2015 IEEE Symposium on Security and Privacy, 2015, pp. 880–896.
[23] "curl." [Online]. Available: https://curl.haxx.se/. [Accessed: 28-Mar-2018].
[24] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-structured File System," ACM Trans. Comput. Syst., vol. 10, no. 1, pp. 26–52, Feb. 1992.
[25] Wikipedia contributors, "dd (Unix)," Wikipedia, The Free Encyclopedia, 24-Jan-2018. [Online].
[28] Y. Tang, A. Iyengar, w. Tan, l. Fong, l. Liu, and b. Palanisamy, "deferred lightweight indexing for log-structured key-value stores," in 2015 15th ieee/acm international symposium on cluster, cloud and grid computing, 2015, pp. 11–20.
[29] Yuzhe Tang, Jianliang Xu, shuigeng Zhou, wang-chien lee, dingxiong deng, and yue wang, "a

lightweight multidimensional index for complex queries over dhts," ieee tranS. parallel distrib. sysT., vol. 22, no. 12, pp. 2046–2054.
[30] Y. Tang, J. Xu, S. Zhou, and w. C. lee, "m-light: indexing multi-dimensional data over dhts," in 2009 29th ieee international conference on distributed computing systems, 2009, pp. 191–198.
[31] Y. Tang and S. Zhou, "lht: a low-maintenance indexing scheme over dhts," in 2008 the 28th international conference on distributed computing systems, 2008, pp. 141–151.
[32] Yuzhe (richard) Tang, Zihao xing, Cheng Xu, Ju chen, Jianliang Xu, "lightweight blockchain logging for data-intensive applications," in trusted smart contracts 2018.
[33] Y. Tang, t. wang, L. Liu, X. hu, and J. jang, "lightweight authentication of freshness in outsourced key-value stores," in proceedings of the 30th annual computer security applications conference, new orleans, louisiana, usa, 2014, pp. 176–185.
[34] X. liang, S. shetty, D. tosh, C. Kamhoua, k. Kwiat, and L. njilla, "provchain: a blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in proceedings of the 17th ieee/acm international symposium on cluster, cloud and grid computing, madrid, spain, 2017, pp. 468–477.
[35] D. k. tosh, S. shetty, X. liang, C. a. Kamhoua, K. a. Kwiat, and L. njilla, "security implications of blockchain cloud with analysis of block withholding attack," in 2017 17th ieee/acm international symposium on cluster, cloud and grid computing (ccgrid), 2017, pp. 458–467.
[36] Deepak kumar tosh, sachin shetty, peter foytik, charles Kamhoua and laurent njilla, "cloudpos: a proof-

of-stake consensus design for blockchain,” in ieee cloud 2018.