

# Distributed Encrypted File System

Jiarui Gao<sup>1</sup>

<sup>1</sup>School of Computer Science, Shanghai Key Laboratory of Intelligent Information Processing,  
Fudan University, China  
jrgao14@fudan.edu.cn

## ABSTRACT

In this paper, I propose a Distributed Encrypted File System (DEFS) which allows clients to store large scale data on a cluster of untrusted servers. Files in DEFS are stored in a distributed and encrypted manner. It provides exclusive account for each client, as well as full recoveries for possible damaged files.

## KEYWORDS

Distributed File System; Distributed Data Storage; Encrypted File System.

### ACM Reference format:

Jiarui Gao<sup>1</sup>. 2017. Distributed Encrypted File System. In *Proceedings of*, , , 9 pages.  
DOI: <http://dx.doi.org/xxxx>

## 1 INTRODUCTION AND RELATED WORK

Distributed Encrypted File System (DEFS) is designed to take advantages of both encryption technic and distributed storage. Through DEFS, clients are able to store large scale data on untrusted servers. DEFS provides encryption services to prevent servers from stealing any clients' information. Also, it offers full recoveries if any of the clients' file is damaged by malicious servers. The distributed storage gives DEFS the capacity of hosting large scale of data. LockServer grants read-write locks for clients, which keeps a high consistency for DEFS.

### 1.1 Encrypted File System

Many non-networked file systems aims at keeping clients' data secure and checking integrity. An early example designed for Unix is Cryptographic File System (CFS)[2]. A modern classic encrypted file system is SUNDR[5], which is a general-purpose, multi-user network file system. The whole design in DEFS is largely inspired by this work.

### 1.2 Distributed File System

Typical traditional distributed file systems are NFS[6] and AFS[4]. Main difference between them is that AFS obtains better performance by caching the files, and it sacrifices the synchronization. NFS is later optimized to be Serverless Network File Systems[1].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2017 ACM. ACM ISBN xxx...\$15.00  
DOI: <http://dx.doi.org/xxxx>

Hadoop Distributed File System[7] and Google File System[3] concentrates on the support for large distributed data-intensive applications.

## 2 SYSTEM OVERVIEW

This section introduces the overview design of Distributed Encrypted File System (DEFS). As illustrated in Fig. 1, DEFS is composed of two parts, namely Distributed File System (Sec. 4) and Encrypted Layer (Sec. 3).

**Encrypted Layer** guarantees that all the data (including file content ,filename and client's directory structure) in DEFS is encrypted. All file contents are signed by clients, which insures that any modification will be detected. Before executing requests, firstly it examines authority for every command carefully. Detected illegal attempts will be logged. The layer keeps clients' accounts secure from illegal operations from possible malicious users.

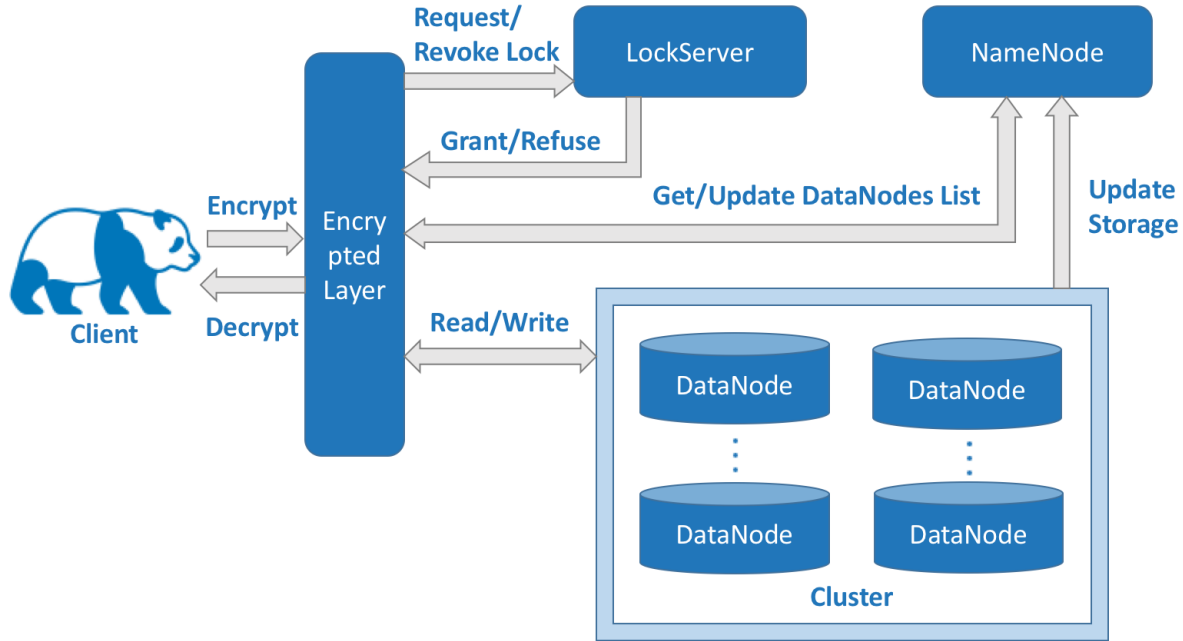
**Distributed File System** consists of LockServer, one trusted NameNode and a cluster of many untrusted DataNodes. LockServer provides read-write locks to maintain consistency. The NameNode/DataNode design is largely inspired by Hadoop [7] and Google File System[3]. In general, DEFS stores all metadata on one trusted server (NameNode), and stores application data blocks on others untrusted servers (DataNodes). NameNode records mappings with regards to file blocks and DataNodes, and hosts location related information. DataNodes stores the assigned block replicas.

## 3 ENCRYPTED LAYER DESIGN

As shown in Fig. 2, the Encrypted Layer provides all encryption-related services. It is transparent for both clients and servers, and it insures all the information stored on servers are encrypted. The login manner offers separate accounts for each client, which prevents clients from possible malicious users as well. Illegal attempts made by clients will be recorded in a special log file by the Encrypted Layer. The following subsections introduces detailed designs in this layer.

### 3.1 Account Initialization

The Encrypted Layer generates one pair of RSA key  $(n_1, e_1, d_1)$  and one AES key  $K$  for each newly registered client. The private key  $(n_1, d_1)$  is written at the client's designated input location, and the client is fully responsible for keeping it safe. The public key  $(n_1, e_1)$  is stored in the system. The the system uses it  $(n_1, e_1)$  to encrypt the AES key  $K$  to be  $K_{en}$ , and stores it in the system as well. To be specific, the system maintains a mapping between clients' names and their corresponding keys' locations. RSA public key  $(n_1, e_1)$  is later used for client authentication and encrypting file full paths (filename included). AES key  $K$  is taken to encrypt file contents.



**Figure 1: An overview of Distributed Encrypted File System (DEFS).** The system contains an Encrypted Layer (left part) and a Distributed File System (right part). The Encrypted Layer provides encryption-related services. Generally, each request in Distributed File System requires getting the corresponding lock from LockServer first, and is responsible to revoke it when finished. After granted a lock, the client asks NameNode for IP addresses where the file blocks are stored (a list of DataNode IDs). Then the client writes or reads data blocks from DataNodes directly. To ensure that NameNode offers reasonable allocations of DataNodes, each DataNode sends Update Storage message to NameNode after any modification immediately.

When a client wants to login to DEFS, he must provide both the user name and the exact location of his RSA private key  $(n_1, d_1)$  correctly. If it matches with  $(n_1, e_1)$ , then user authentication is granted.

### 3.2 File Uploading/Writing

When a client requests to Upload/Write a file, the Encrypted Layer makes sure all the information is encrypted according to the following procedures.

- File full path with filename included (to insure uniqueness of each file) is encrypted with  $(n_1, e_1)$  as its encrypted filename on the server.
- The system uses client's provided private key  $(n_1, d_1)$  to decrypt  $(K_{en})$  to get AES key  $K$ . File content is encrypted with  $K$  as encrypted data stored on servers.
- Client's private key  $(n_1, d_1)$  is used to sign the hash value of the whole encrypted content. Then the signature is

added at the beginning of the encrypted file. By now, the encryption process is finished.

- A backup of the encrypted content is stored under its encrypted filename in the system for possible need of future recovery.
- Send Upload/Write request to the Distributed File System.

### 3.3 File Downloading/Reading

When a client requests to Download/Read a file, the Encrypted Layer decrypted the received file from Distributed File System in following steps.

- First, Encrypted Layer uses  $(n_1, e_1)$  to encrypt the file's full path (filename included) to get the encrypted filename.
- Then it checks the signature. The system decrypted the signature stored at the beginning of the encrypted data with public key  $(n_1, e_1)$ . And if the result equals to the hash value of the whole content, authentication of this file is granted. Otherwise, the system considers the data

illegally modified, and after sending notification to the client, the backup version will be uploaded automatically to the Distributed File System.

- After checking integrity, Encrypted Layer uses private key  $(n_1, d_1)$  to decrypt  $(K_{en})$  and get AES key  $K$ . Then it uses  $K$  to decrypt file content.
- The decrypted file is downloaded and is ready to be read.

### 3.4 Other Commands

The system provide other commands including *Move*, *Copy*, *Remove*, etc. Those commands could be converted to Upload/Write (Sec. 3.2) and Download/Read (Sec. 3.3). For example, command *Move* includes

- Download and remove the original file.
- Move and rename the backup version.
- Upload the file.

### 3.5 Layer Content

As introduced formerly in Sec. 3, there is some client-related data stored in directories in the Encrypted Layer. DEFS insures that even the Encrypted Layer is invaded entirely by hackers somehow, nothing will be leaked.

All data stored in Encrypted Layer includes:

- Mapping of (client name, location of corresponding RSA public key).
- Mapping of (client name, encrypted AES key).
- A log file recording all illegal attempts.
- Clients' RSA public keys.
- Clients' backup file with encrypted filename and encrypted file content.

As shown above, the Encrypted Layer is totally secure. The purpose of requiring RSA private key directly to login and persevering only the encrypted version of backup file is exactly to insure that there is no sensitive information in the Encrypted Layer.

## 4 DISTRIBUTED FILE SYSTEM DESIGN

Fig. 3 demonstrates the design of Distributed File System. It mainly consists of three parts, namely LockServer, NameNode, and a cluster of DataNodes. DEFS provides a distributed storage schema for large scale data.

### 4.1 LockServer

Generally, I consider the synchronization problem in DEFS as a typical readers-writers problem[8]. The LockServer provides services following readers-preference solution. That is to say once the first reader is in the entry section, LockServer will lock the resource. Doing this will prevent any writers from accessing it. Subsequent readers can just utilize the locked (from writers) resource. The very last reader (indicated by the reader-counting variable) must unlock the resource, thus making it available to writers.

When requesting for read or write locks, the client sends a control message (*encryptedfilename, locktype*) to LockServer, and in return he gets the granted lock or a waiting instruction (1/0). After the operation, the client revokes the lock via another control message to the server (*encryptedfilename, locktype*).

### 4.2 NameNode

The single NameNode in Distributed File System maintains all the metadata and is in charge of allocating data blocks to specific DataNodes. DEFS considers NameNode to be entirely trustworthy.

As illustrated in Fig. 3, there are two mappings. The mappings are kept both in RAM and in json files to improve durability. The data will not be lost if the server program crashes.

A DEFS client wanting to download/read a file first contacts with the NameNode for the locations of data blocks (DataNode IDs) comprising the data and then reads block contents from the DataNode closest to the client. When a client requests to upload/write a file, a message asking for file allocation or relocation is sent to NameNode. Then the NameNode nominate three DataNodes for each block and return their IDs as a list back to the client.

To guarantee the selected DataNodes have the capacity to store the assigned block replicas, NameNode updates the storage mapping (DataNode ID, Storage Left) immediately upon receipt of control messages from DataNodes. During the nomination, NameNode randomly selects three DataNodes for each block among which have enough storage.

### 4.3 DataNodes

Distributed File System contains a cluster of DataNodes to store data blocks. In DEFS, all DataNodes are considered not trustworthy, and the data sent to them are only clients' instructions and encrypted data blocks. Data pipeline goes between clients and DataNodes directly.

During the startup process of each DataNode, it performs a handshake with NameNode to inform the connection and its current storage capacity. And after processing storage-changing requests such as block deletion or allocation, DataNodes send control messages (*DataNodeID, Storage*) to NameNode to update the space left.

### 4.4 Fault Tolerance

DEFS provides a robust data storage on untrusted servers. As long as the NameNode is trustworthy, the DataNodes can be untrusted. Block replicas hosted by DataNodes may be modified or deleted at anytime. DEFS has two methods in dealing with such situation.

**Random Selection of DataNodes.** As mentioned in Sec. 4.2, the NameNode nominate DataNodes randomly for each block. Assume there are totally  $n$  DataNodes, and  $m$  of them are malicious. Here we suppose that malicious DataNodes damage every data block sent to it. We also assume that the client's file contains  $k$  blocks, and the number of replicas in DEFS is  $r$ . Then the possibility of unrecoverable damage to one data block is  $\frac{C_m^r}{C_n^r}$ . The possibility of unrecoverable damage to the whole file is:

$$1 - (1 - \frac{C_m^r}{C_n^r})^k \quad (1)$$

As designed in DEFS,  $r = 3$ . And if  $\frac{m}{n} = \frac{1}{3}$ , which means one third of the DataNodes are malicious, the possibility of recovering the full file still remains to be:

$$(1 - \frac{C_m^r}{C_n^r})^k > (1 - (\frac{m}{n}))^k > (\frac{26}{27})^k \quad (2)$$

As calculated, DEFS keeps a high tolerance of fault.

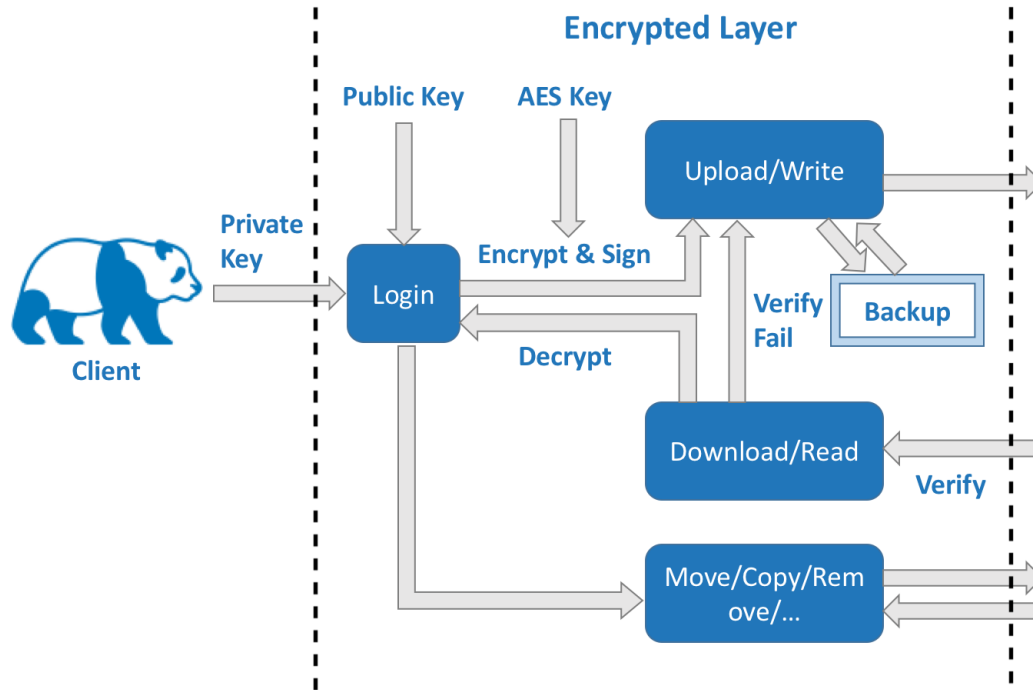


Figure 2: An overview of the Encrypted Layer. The client needs to provide the location of RSA private key to login to the system. The public key and the encrypted AES key is kept by the system. A simple work flow is shown here.

**Encrypted File Backup.** As introduced in Sec. 3.2, the Encrypted Layer maintains a backup for every encrypted file, which is absolutely practical when the file size is small. In the decryption procedure, if the retrieved file is found to be modified, the backup version will be used to correct the data on DataNodes automatically. This backup method provides a 100% percent recovery of any file.

## 5 IMPLEMENTATION

All the discussed design above about DEFS is fully implemented. I choose programming language Python, and library *Crypto* is used for encryption. Source code is available at <https://github.com/kittenish/Distributed-Encrypted-File-System>. Screen shots for command line interfaces are presented in Appendix Sec. A. This section will introduce detailed settings and side information in implementing DEFS.

### 5.1 Settings

The number of DataNodes is set to be 5, and the initial storage is 100MB. The number of replicas is 3. All communication is made through socket. For simplicity, clients consider the first DataNode (of the three IDs provided) to be the nearest one in transporting data. Besides user name and the private key, each client needs to provide

an available socket port in logging-in process. The encrypted file backup method is adopted for fault tolerance. And all metadata are stored as *json* format in local file system. Also, DEFS tries to catch exceptions as much as possible to avoid system crash.

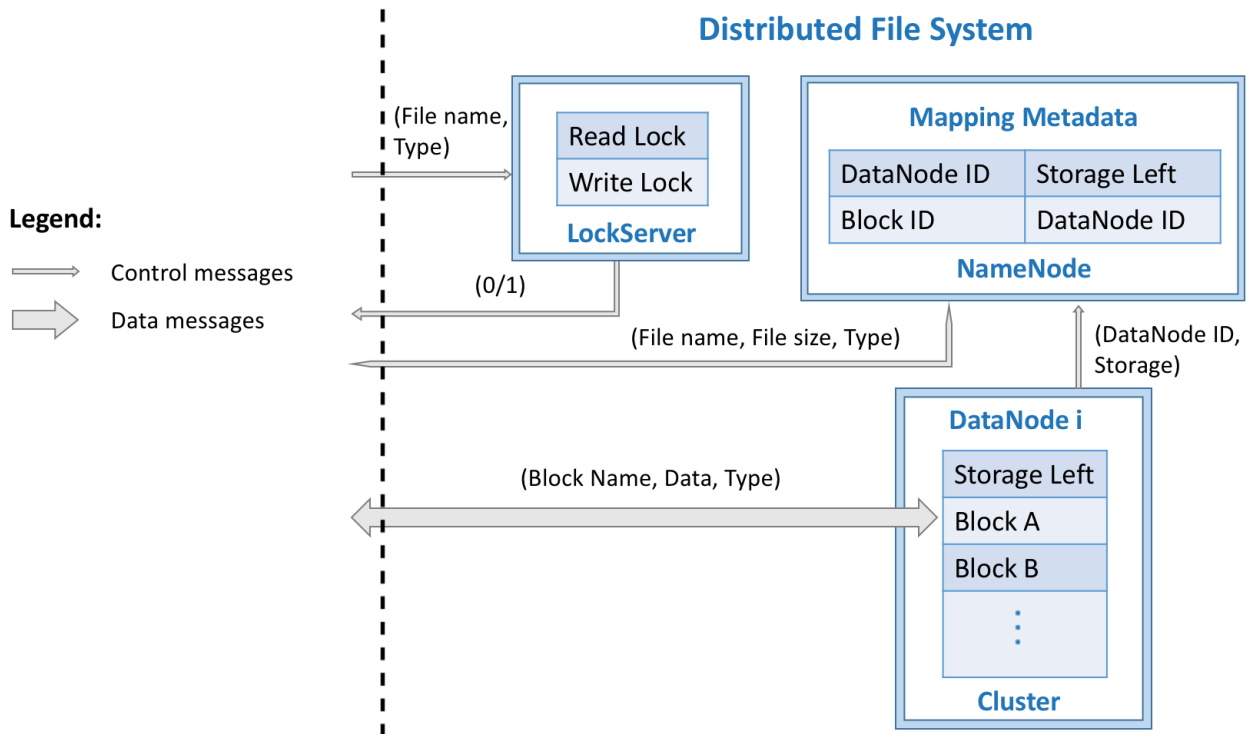
### 5.2 Side Information

Ideally, socket connection between clients and DataNodes are supposed to be established when the transportation of data stream begins. But for simplicity, in my implementation, all socket connections between clients and other servers are built upon *login* command. The *login* request will fail if the client's socket port is unavailable or any of the initial connections fail.

Both the Encrypted Layer and the Distributed File System is transparent to clients. Commands of reading and writing is the same as traditional file systems. The command line interface is designed to be user-friendly, and clear help information is available for every function.

## 6 FUTURE WORK

Lots of extensions could be made to DEFS to achieve better performance. In this section, both design improvements and implementation modification will be discussed.



**Figure 3: An overview of Distributed File System (DFS). The LockServer grants read-write locks. The NameNode stores the mapping metadata shown in the table. Data pipelines goes directly between clients and DataNodes.**

From design aspect:

- Clients' separate accounts prevent files from being shared, the sharing function should be added.
- When file damage is detected, the client should report to NameNode. And more functions could be added to NameNode to locate malicious servers and disconnect with them when necessary.
- More NameNodes and more clusters of DataNodes can be added to improve storage capacity.
- The LockServer currently utilizes a readers-preference solution, which may cause severe starvation for writers. Other lock granting and revoking algorithms can be adopted.

From implementation aspect:

- Reduce the hard coded definitions in the system, and make DEFS more configurable.
- More rules in Encrypted Layer for examining commands' authority to avoid malicious users.
- When file damage happens, Encrypted Layer should try to recover the data from blocks inside DEFS first to get the latest version, before uploading the backup version.

## 7 CONCLUSION

In conclusion, DEFS is a file system designed to store large scale data on untrusted servers. Clients' data is hosted in a distributed and encrypted manner. And all the information is protected from both servers and other malicious users.

## A APPENDIX

Here are some screen shots of the fully implemented DEFS.

### A.1 Client

Fig. 4, Fig. 5 and Fig. 6.

### A.2 LockServer

Fig. 8

### A.3 NameNode

Fig. 7

### A.4 DataNode

In sum there are 5 DataNodes, as shown in Fig. 9 to Fig. 13.

```

python (Python)  361  python (Python)  362  python (Python)  363  python (Python)  364  python (Python)  365  python (Python)  366  python (Python)  367  python (Python)  368
→ src git:(master) X python client.py
Distributed Encrypted File System Start...
Type 'help' for user guide...
DEFS/> NONE/> help

      help                                print user-guide
      quit()                             quit the efs

      cd                                change directory (absolute/relative path are supported)
      pwd                                show current absolute path
      ls [mode]                          list all files
      mkdir                              make a new directory (both absolute/relative path are supported)
      rm [mode]                          remove file (add "-r" to remove directory, both absolute/relative path are supported, recursion not supported)
      mv [src] [dest]                    move the file from src to dest (only file move is supported, both absolute/relative path are supported)
      cp [src] [dest]                    copy file from src to dest (only file copy is supported, both absolute/relative path are supported)

      register [user_name] [PRK_loc]     register with user name and specific location to store your private key
      login [user_name] [PRK_loc] [socket_port] login with user name , private key location and an available socket port
      upload [src] [dest]                upload the src file to the dest directory (both absolute/relative path are supported)
      download [src] [dest]              download the src file to the dest directory (both absolute/relative path are supported)

DEFS/> NONE/> login test /Users/mac/Desktop/test.pem 7984
Login succeeded.
DEFS/> test/> ls
['.DS_Store', 'dict1', 'dict2']
DEFS/> test/> cd ./dict1
Change directory ./dict1 successfully.
DEFS/> dict1/> cd ..
Change directory .. successfully.
DEFS/> test/> pwd
/test
DEFS/> test/> upload /Users/mac/Desktop/1.png
Too few arguments for command upload.
Type 'help upload' for more information.
DEFS/> test/> upload /Users/mac/Desktop/1.png dict1
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Begin to encrypt: 0xNG

IHDR0gi#iCCPICC Profi...
Send chunk 1 to DataNode 7005...
Send chunk 1 to DataNode 7003...
Send chunk 1 to DataNode 7002...
Send chunk 2 to DataNode 7005...
Send chunk 2 to DataNode 7003...
Send chunk 2 to DataNode 7001...

```

Figure 4

## REFERENCES

- [1] Thomas E Anderson, Michael D Dahlin, Jeanna M Neefe, David A Patterson, Drew S Roselli, and Randolph Y Wang. 1995. Serverless network file systems. In *ACM SIGOPS Operating Systems Review*, Vol. 29. ACM, 109–126.
- [2] Matt Blaze. 1993. A cryptographic file system for UNIX. In *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 9–16.
- [3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *ACM SIGOPS operating systems review*, Vol. 37. ACM, 29–43.
- [4] John H Howard and others. 1988. *An overview of the andrew file system*. Carnegie Mellon University, Information Technology Center.
- [5] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. 2004. Secure Untrusted Data Repository (SUNDR). In *OSDI*, Vol. 4. 9–9.
- [6] Spencer Shepler, Mike Eisler, David Robinson, Brent Callaghan, Robert Thurlow, David Noveck, and Carl Beame. 2003. Network file system (NFS) version 4 protocol. *Network* (2003).
- [7] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE, 1–10.
- [8] Wikipedia. 2017. Readers writers problem. (2017). [https://en.wikipedia.org/wiki/Readers-writers\\_problem](https://en.wikipedia.org/wiki/Readers-writers_problem)

```

python (Python)  361  python (Python)  362  python (Python)  363  python (Python)  364  python (Python)  365  python (Python)  366  python (Python)  367  python (Python)  368
Upload file /Users/mac/Desktop/1.png successfully.
DEFS/> test/> mv dict1/1.png dict2/2.png
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Get chunk 1 from DataNode 7005...
Get chunk 2 from DataNode 7005...
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Remove chunk 1 from DataNode 7005...
Remove chunk 1 from DataNode 7003...
Remove chunk 1 from DataNode 7002...
Remove chunk 2 from DataNode 7005...
Remove chunk 2 from DataNode 7003...
Remove chunk 2 from DataNode 7001...
Remove file name from NameNode...
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Begin to encrypt: PNG

IHDRg#iCCP ICC Profi...
Send chunk 1 to DataNode 7005...
Send chunk 1 to DataNode 7003...
Send chunk 1 to DataNode 7002...
Send chunk 2 to DataNode 7003...
Send chunk 2 to DataNode 7001...
Send chunk 2 to DataNode 7004...
Move file dict1/1.png successfully to dict2/2.png .
DEFS/> test/> cp dict1/1.png dict1/copy.png
Fail to copy file dict1/1.png because no such file.
DEFS/> test/> cp dict2/2.png dict1/copy.png
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Get chunk 1 from DataNode 7005...
Get chunk 2 from DataNode 7003...
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Begin to encrypt: PNG

IHDRg#iCCP ICC Profi...
Send chunk 1 to DataNode 7003...
Send chunk 1 to DataNode 7005...
Send chunk 1 to DataNode 7002...
Send chunk 2 to DataNode 7002...
Send chunk 2 to DataNode 7004...
Send chunk 2 to DataNode 7003...
Copy file dict2/2.png successfully to dict1/copy.png .
DEFS/> test/> rm dict1/copy.png
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Remove chunk 1 from DataNode 7003...
Remove chunk 1 from DataNode 7005...

```

Figure 5

```

Remove chunk 1 from DataNode 7002...
Remove chunk 2 from DataNode 7002...
Remove chunk 2 from DataNode 7004...
Remove chunk 2 from DataNode 7003...
Remove file name from NameNode...
Remove file dict1/copy.png successfully.
DEFS/> test/> download dict1/1.png /Users/mac/Desktop
Fail to download file dict1/1.png because no such source file.
DEFS/> test/> download dict2/2.png /Users/mac/Desktop
Begin to encrypt: /Users/mac/Desktop/Distributed-Encrypted-File-Syst...
Get chunk 1 from DataNode 7005...
Get chunk 2 from DataNode 7003...
Download file dict2/2.png successfully, already at path /Users/mac/Desktop.
DEFS/> test/> quit()
Quit DEFS...
src git:(master) x

```

Figure 6

```

..le-System/src (zsh) 361 ..le-System/src (zsh) 362 ..le-System/src (zsh) 363 ..le-System/src (zsh) 364 ..le-System/src (zsh) 365 ..le-System/src (zsh) 366 ..le-System/src (zsh) 367 ..le-System/src (zsh) 368
Last login: Wed Dec 20 10:19:40 on console
➔ ~ cd ../Desktop/Distributed-Encrypted-File-System/src
➔ src git:(master) x python start_NameNode.py
NameNode server started on port 5000

```

Figure 7



```

python (Python) 第1 python (Python) 第2 python (Python) 第3 python (Python) 第4 python (Python) 第5 python (Python) 第6 python (Python) 第7 python (Python) 第8
Last login: Wed Dec 20 14:00:58 on ttys005
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) # cd ./src
➔ src git:(master) # python start_Lock.py
LockNode server started on port 9090
Grant write lock to HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Release write lock for HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Grant read lock to HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Release read lock for HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Grant write lock to HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Release write lock for HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd
Grant write lock to cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP
Release write lock for cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP
Grant read lock to cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP
Release read lock for cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP
Grant write lock to dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc
Release write lock for dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc
Grant write lock to dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc
Release write lock for dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc
Grant read lock to cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP
Release read lock for cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP

```

Figure 8

```

python (Python) 第1 python (Python) 第2 python (Python) 第3 python (Python) 第4 python (Python) 第5 python (Python) 第6 python (Python) 第7 python (Python) 第8
Last login: Wed Dec 20 14:00:55 on ttys000
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) # cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) # cd ./src
➔ src git:(master) # python start_DataNode_1.py
DataNode server started on port 7001
Update size to 100000000.0 kb...
Finish writing HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd_2
Update size to 99992261.0 kb...
Delete file HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd_2
Update size to 100000000.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP_2
Update size to 99992261.0 kb...

```

Figure 9

```

python (Python) 第1 python (Python) 第2 python (Python) 第3 python (Python) 第4 python (Python) 第5 python (Python) 第6 python (Python) 第7 python (Python) 第8
Last login: Wed Dec 20 14:00:57 on ttys001
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) # cd ./Desktop/Distributed-Encrypted-File-System/src
cd: no such file or directory: ./Desktop/Distributed-Encrypted-File-System/src
➔ Distributed-Encrypted-File-System git:(master) # cd ./src
➔ src git:(master) # python start_DataNode_2.py
DataNode server started on port 7002
Update size to 100000000.0 kb...
Finish writing HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd_1
Update size to 99983616.0 kb...
Delete file HNWVmv+tp4zbA4WdBUUi79JHqFyv_sUSCL0mB9W1tHbRSMKCePi f0sa3DsV0m8FicZJcdep7c9c4tP6NdmSYjh_HTO+Yre943vwd_1
Update size to 100000000.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVnm41MzeOWnAg7Z28uOQRqRta16XSbWLSRNXkFWCR9Paem4+GeMeWSPEu3NhQP_1
Update size to 99983616.0 kb...
Finish writing dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc_1
Update size to 99967232.0 kb...
Finish writing dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc_2
Update size to 99959493.0 kb...
Delete file dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc_1
Update size to 99975877.0 kb...
Delete file dKTLWokd8b7GbI0YC3g_e4e62x1u6rPwMYsIprSsIlGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDJxYtzhk+hNGRK3XRMPvW94gYhc_2
Update size to 99983616.0 kb...

```

Figure 10



```
python (Python)  361  python (Python)  362  python (Python)  363  python (Python)  364  python (Python)  365  python (Python)  366  python (Python)  367  python (Python)  368
Last login: Wed Dec 20 14:00:57 on ttys002
➔ ~ git push origin mastercd
fatal: Not a git repository (or any of the parent directories): .git
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) X cd ./src
➔ src git:(master) X python start_DataNode_3.py
DataNode server started on port 7003
Update size to 100000000.0 kb...
Finish writing HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_1
Update size to 99983616.0 kb...
Finish writing HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_2
Update size to 99975877.0 kb...
Delete file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_1
Update size to 99992261.0 kb...
Delete file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_2
Update size to 100000000.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_1
Update size to 99983616.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_2
Update size to 99975877.0 kb...
Send file cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_2
Finish writing dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_1
Update size to 99959493.0 kb...
Finish writing dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_2
Update size to 99951754.0 kb...
Delete file dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_1
Update size to 99968138.0 kb...
Delete file dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_2
Update size to 99975877.0 kb...
Send file cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_2
|
```

Figure 11

```
python (Python)  361  python (Python)  362  python (Python)  363  python (Python)  364  python (Python)  365  python (Python)  366  python (Python)  367  python (Python)  368
Last login: Wed Dec 20 14:00:58 on ttys003
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) X cd ./src
➔ src git:(master) X python start_DataNode_4.py
DataNode server started on port 7004
Update size to 100000000.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_2
Update size to 99992261.0 kb...
Finish writing dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_2
Update size to 99984522.0 kb...
Delete file dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_2
Update size to 99992261.0 kb...
|
```

Figure 12

```
python (Python)  361  python (Python)  362  python (Python)  363  python (Python)  364  python (Python)  365  python (Python)  366  python (Python)  367  python (Python)  368
Last login: Wed Dec 20 14:00:58 on ttys004
➔ ~ cd ./Desktop/Distributed-Encrypted-File-System/
➔ Distributed-Encrypted-File-System git:(master) X cd ./src
➔ src git:(master) X python start_DataNode_5.py
DataNode server started on port 7005
Update size to 100000000.0 kb...
Finish writing HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_1
Update size to 99983616.0 kb...
Finish writing HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_2
Update size to 99975877.0 kb...
Send file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_1
Send file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_2
Delete file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_1
Update size to 99992261.0 kb...
Delete file HNWVm+tp4zbA4WdBUUi79JHqFyv_sUSCL0m89W1tHbRSMKCePi f0sa3DsV0m8fi cZJcdep7c9c4tP6Ndm5Yjh_HTO+Yre943vmD_2
Update size to 100000000.0 kb...
Finish writing cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_1
Update size to 99983616.0 kb...
Send file cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_1
Finish writing dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_1
Update size to 99967232.0 kb...
Delete file dKTLlWokd8b7Gbi0YC3g_e4e62x1u6rPwMYsIprSsilGKWfQkGSgr23eH0Ck6yLeteNeNPDgmDjXyTzhk+hN6RK3XRMFpvW94gYhc_1
Update size to 99983616.0 kb...
Send file cm2kbCvTL7lqQ08az46AreYy2PKLfuNviWX70zVrw41MzeOWnAg72Z8uOQRqRta16XSbwLsRNXkfWCR9Paem4+GeMeWSPEu3NhQP_1
|
```

Figure 13