Design and Implementation of Data Encryptionin Cloud based on HDFS

Zhonghan Cheng
Nanjing University,Nanjing,China
e-mail: chengzhonghan47@gmail.com

Hao Huang
Nanjing University,Nanjing,China
e-mail: hhuang@nju.edu.cn

Diming Zhang
Jiangsu University of Science and Technology,
Zhenjiang, China
e-mail: zhangdiming@gmail.com

Zhenjiang Qian
Nanjing University,Nanjing,China
e-mail: zhenjiang.qian@gmail.com

***Abstract*—As an open-source distributed programming framework, Hadoop has gradually become popular in industry recently. Its distributed file system (HDFS) enables storing large data with advantages of high fault tolerance and throughput. However, the fact that the current HDFS does not support intra-cloud data encryption yet makes data privacy becomes a key security issue. This paper presents ahybrid encryption method based on HDFS. We adopt symmetric encryption to encrypt and decrypt file blocks at datanodes and use asymmetric encryption scheme to protect the symmetric keys. By this method, we can prevent datanode intruders from stealing user data, while ensuring that clients are lightweight. The experiments show that with and without block encryption algorithm, our solution brings43% and 2% performance degradation compared to the generic HDFS.**

Keywords-Hadoop; Distributed File System; Data Encryption;

I. Introduction

Hadoop [1] is a distributed platform developed by the Apache Foundation for reliable, efficient and scalable storage and computing on large data. It provides a unified interface to users so that they can develop and enjoy services without knowing the details of cloud. It is an open-source project implemented in Java, and originates from Google's distributed computing framework. Due to its features of high reliability, scalability, supporting for cross-platform, it has been widely studied and applied by academia and industry. The Hadoop project contains distributed storage service HDFS (Hadoop Distributed File System), large-scale parallel computing framework MapReduce, distributed database HBase, data warehouse tools Hive, distributed lock facilities Zookeeper and other subprojects.

The main design principles of HDFS can be tracedback to GFS (Google File System) [2]. It utilizes storage capacity of large scale clusters, and provides users with a transparent interface similar totraditional file system owing to master-slave structure. The master (called namenode) is used to keep metadata of HDFS and interact with users to manage their files in cloud. The slave (called datanode) is responsiblefor storing files.

At the beginning of HDFS design phase, the main consideration was to improve data reliability and storage efficiency but data privacy was overlooked. Nevertheless, when HDFS provides storage services to users as a public cloud, files belonging to different users may be uploaded to the same machine. If there is no data protection measure, data leakage is likely to occur since datanode intruderscan obtain file plaintext directly. Currently, the security work of Hadoop project is still at its infancy stage, andonly simple access control mechanisms and file permission are employed [3, 4]. Therefore, most of the current commercial companies just use Hadoop to build private clusters.

This paper proposes a hybrid encryption method for HDFS to protect intra-cloud data privacy while keeping client lightweight. From the experiment results, we present that even if the performance degradation caused by our solution reaches 43%, we can parallel the data encryptionalgorithm by extra computing units to reduce the overall overhead substantially since the unavoidable architecture overhead is only2%.

The remainder of this paper is organized as follows. In Section 2 and 3, we present the related work and the background. Design and implementation of our approach for HDFS is described in Section 4. Followed by our evaluation in Section 5, we summarize our paper in Section 6.

II. Related Work

There has merged a significant body of studies that proposed measures to secure HDFS with data privacy protection. SAPSC[5], a secure architecture has been proposed, including data isolation service, intra-cloud data migration service and inter-cloud data migration service. In hybrid cloud environment, it places sensitive data separately at private storage cluster and normal data at public cluster. Different from such data isolation solution for hybrid cloud, we focus our attention on data encryption for single cloud.

Hearn et al. [6] presenteda distributed storage prototype named Tahoe that integrates access control, encryption and erasure coding for fault-tolerance for secure. Tahoe-LAFS [7] is a middle layer between Hadoop and Tahoe that makes MapReduce be able torun on Tahoe seamlessly. Lin et al [8] proposed two hybrid encryption schemes—HDFS-RSA and

HDFS-Paring, by modifying the fuse-dfs module in HDFS. Methods in [6, 8] both deploy the encryption/decryption modules at clientsin order to easeimplemented. However, the cloud client should be lightweight, inserting encryption, decryption and key management intoit would lead to heavier load and more complexity. Seonyoung et al. [9] employed AES algorithm to achieve encryption at clients and decryption at datanodes, but its main drawback is that the session keys are not protectedand consequently invaders can still steal the data plaintext easily by obtaining them.We deploy security modules at datanodesand leave the namenode unchanged toavoid it becoming a single point of failure. We let the client keeps the secret key so that the client is the ultimate controller of the data privacy. On the other hand, our solution keeps the client lightweight and fits the mobile environment.

III. Background

In the early design stage of HDFS, the major assumptions would be the following key points:

- There are a large number of nodes in the system. Failures of hardware, programs or operating system bugs, or human errorsmay lead to nodes fail.
- The system is mainly used to store large files.
- It is assumed that reading frequency is much higher than writing, and appending new data is supported rather than overwriting existing data in writing case.
- HDFS pays more attention on sustained bandwidth than latency because it assumes that there are few applications have stringent response time requirements.

To meet the above requirements, HDFS splits files stored into fixed-size blocks. The cluster is divided into a namenode to manage the metadata and multiple datanodes to store blocks. The namenode is the entry of the whole cloud, the metadata contains namespace, access control information, mapping from files to file blocks and block location information. The other nodes are datanodes thatoperate files in terms of blocks that each block has a unique ID. The namenode provides a publicand transparent interface to clients. A client would visit the namenode first to get the metadata of files that it needs, and then communicates with datanodes to download and upload blocks.

On the abovementioned fundamentalmechanisms, HDFS presentssome measures to improve data reliability and throughput of reading and writing.First, it keeps multiple replicas for each block across different datanodes toavoid data lost. Second, the namenode must check the validity of all datanodes periodically and keepsenough number of block replicas. Third, by choosing a large block size(default 64MB), it can reduce the size of the metadata storing on the namenode and cut downnetwork overhead caused by block mechanism. Forth, considering the assumption that HDFS doesn't need to support inserting and modifying datain the middle of a file, each block belonging to the same file can be distributed across different datanodes.

The steps of reading a file in HDFS are as below:

1) The client sends the path, offset and length of the file portionthat it needs to read.
2) The namenode searches the metadata by the information sent by the client, andpasses back the location lists of the blocks that belong to the file.
3) For each block location list, the client connects the closest datanode and creates a data stream with it to download block data.If an error occurs during creating connection or downloading, the client would try to connect to the other datanodes in the list.
4) The reading processends upuntilthe client receives all blocks of the file over.

The steps of writing are similar to reading, except that when the client receives the block location lists from the namenode, it writes data to all datanodes in the list to ensure enough block replicas. More specifically,the client createsa pipeline with these datanodes, uploads data to the first datanode, and then the first datanode delivers to the second one and so on. When the pipelineprocess ends, the client notifies the namenode to update the relevant metadata.

IV. Data Encryption based on HDFS

A. System Architecture

From the above description of HDFS, we can draw suchtwo conclusions: 1) the namenode provides clients with a unique entry that is responsible to manage all datanodes within HDFS and exchange metadata with clients. It is likely to become aperformance bottleneck of the whole cluster.2) The client should be lightweight, andits function should be as simple as possible and the information relevant to users should be placed in cloud.

Our security demand is to prevent attackers stealing file data after they intruding into datanodes. We insert encryption and decryption modules into datanodes, as shown in Figure 1. The modules use AES algorithm to encrypt blocks before writing and decrypt before reading. One data block is encrypted and decrypted with a key which is also stored on datanode. As encryption, decryption and key management modules are deployed at datanodes, the modifications to the original protocol between datanodes and namenode remains unchanged. The authentication protocol and key exchange protocol is implemented and deployed at clients and datanodes.

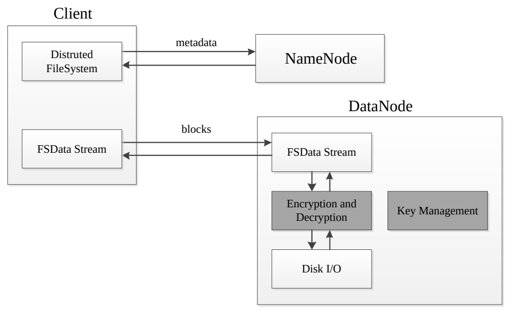

Figure 1. Overview of encrypted HDFS

B. Key Management

We can’t guarantee data privacy by just encrypting file block since if the AES keys used to encrypt the data block is stored on datanode in plaintext, attackers can decrypt blocks by stealing the AES key first in plaintext after they permeating into the datanode.In order to solve this problem, we employ asymmetric encryption algorithm RSA to encrypt AES keys, and save the encrypted AES key on the datanode. The RSA key pair is generated by client, and the private key is maintained on client, and the public key is stored on namenode as metadata. The private key can be encrypted by the key generated from the hashed user password and stored as file.We can also save the user private key in the USB key. Figure 2 shows the abstract of hybrid encryption scheme.

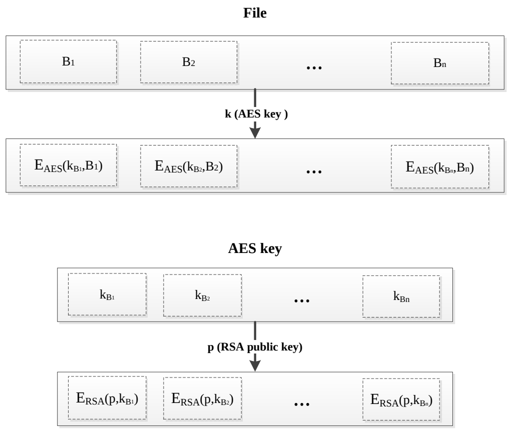

Figure 2. Abstract of hybrid encryption scheme

C. Implementation

We have implemented the two encryption algorithms by JEC (Java Cryptography Extension). In AES encryption, we split block ciphertext into 16 bytes sections, and we don’t padding the last section when it is less than 16 bytes. CFB mode [9] is applied that the length of block ciphertext would not increaseafter encryption.

1) Encryption

We add encryption module intothe code of creating file indatanode. Figure 3depicts the timing diagram of the procedures during encryption. The client generates an RSA key pair when it initializes, then the public key is uploaded to the namenode. When it requeststhe namenode to createa file, the namenode sends back the metadata including the list of datanodes used to store the new file and the client’s RSA public key. For each file block, the client sends the public key to the first datanode in pipeline then delivers the block data in packet. Then the datanode generates an AES session key after receiving the new block, uses the key to encrypt the block data and the RSA public key to encrypt AES key. Finally, the datanode savesthe encrypted file block and the encrypted AES session keyon the local disk as a separate file. We set the name suffix of a key file as its block ID so that datanode can search itby its filename. The other datanodesused to savethe block replica in pipeline get theencrypted block and the encrypted sessionkey directly from the previous onewithout re-encrypting.

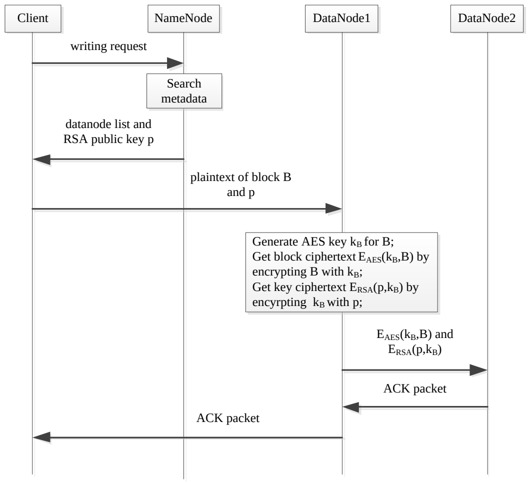
Client
NameNode
DataNode1
DataNode2
writing request
Search metadata
datanode list and RSA public key p
plaintext of block B and p
Generate AES key k_B for B; Get block ciphertext $E_{AES}(k_B,B)$ by encrypting B with k_B; Get key ciphertext $E_{RSA}(p,k_B)$ by encyrpting k_B with p;
$E_{AES}(k_B,B)$ and $E_{RSA}(p,k_B)$
ACK packet
ACK packet

Figure 3. Timing diagram of the procedures during encryption

Note thatHDFS checks the integrity of file blocks by CRC(Cyclic Redundancy Check). The client calculates the CRC codes of a block by per 512 bytes and uploads them with block to datanode. The datanode recalculates the checksums periodically, checks whether the results equal to the original values find possible errors occurs in the contents of blocks. Client also needs to recalculate the checksums before reading blocks. Since the checksums on datanode are calculated by clients with the file plaintext, after adding encryption module, the thread named “DataBlockScanner” would calculate the checksumsperiodicallywith ciphertext, causing the results mismatch. Our solution is to recalculate the checksum by block ciphetext instead of the original values after encryption at datanode.

2) Decryption

As shown in Figure 4, after the datanode receiving readingblock request by client, it sends back the corresponding AES key ciphertext. The client decrypts it by its RSA private key and sends the AES key to the datanode. Finally, the datanode gets the block plaintext and transportsit to the client.

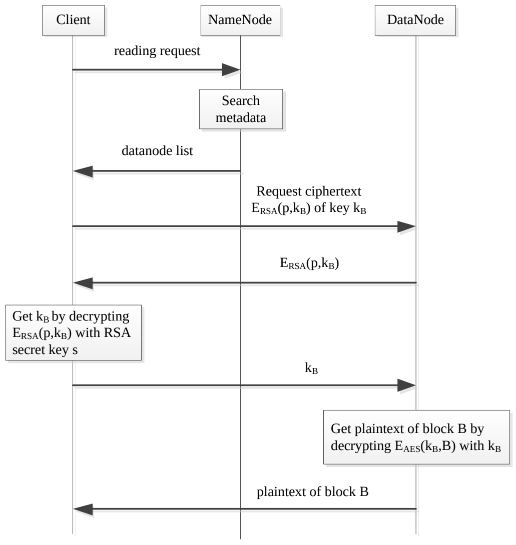
Client
NameNode
DataNode
reading request
Search metadata
datanode list
Request ciphertext $E_{RSA}(p,k_B)$ of key k_B
$E_{RSA}(p,k_B)$
Get k_B by decrypting $E_{RSA}(p,k_B)$ with RSA secret key s
k_B
Get plaintext of block B by decrypting $E_{AES}(k_B,B)$ with k_B
plaintext of block B

Figure 4. Timing diagram of the procedures during decryption

V. Evaluation

A. Environment

Werun experiments in a small testbed containing of two machines.The hardware configuration of the namenode is 8-core Intel Xeon 2.00 GHz, 10 GB memory, and 1 TB hard disk.The datanode has 16-core 2.2 GHz, 20 GB memory, and 2 TB hard disk. The configuration of replication is 1.

B. Encrypted HDFS with and without Data Encryption/Decryption Algorithms vs. Generic HDFS

Note that the performance overhead contains the portion of block encryption/decryption algorithms and the architecture overhead caused by key protection and management, modification to protocol among nodes, checksum recalculation.We expect to evaluate the overall overhead and the unavoidable architecture overheadseparately. In order to evaluate the latter one, we replace the AES algorithm with identity function in code and keep the other functions of our method unchanged.

1) Encryption

In the encryption case, we have compared the writing throughput of the encrypted HDFS with the generic HDFS by creating files of various sizes. As shown in Figure 5,the throughput reaches maximum when creatinga 512 MB filein generic HDFS and 256 MB in encrypted HDFS. The performance degradation varies with file size, it reaches maximum 53% when the file size is256 MB, and drops to minimum33% at 8 MB, which is 43% at 1GB.

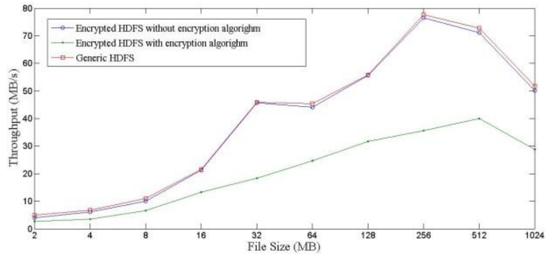

Figure 5. Writing throughput: encrypted HDFS vs. generic HDFS

It is obvious that the significant overhead is mainly caused by the encryptioncomputingwhich needs to be serialized with the network transport and disk operations at datanode.However, by observing the encrypted HDFS withoutdata encryption algorithm,we realize thatthe ratio of the architecture overhead is negligible (about 2%), so the encryption/decryption processes can be completely migrated to GPUs or encryption cards to reduce the overall overhead substantially.

2) Decryption

We have performed reading files of different sizes in cloud to evaluate the decryption overhead. As shown in Figure 6,the reading speed is 1.4 timeshigherthan writing when the file size is 1 GBsince the reading operation on disk is faster.Due to the efficiencyof AES decryption is close to encryption, the decryption module brings more significant performance degradation (75%) than encryption. But the architecture overhead of decryption equals to the one in the encryption case.

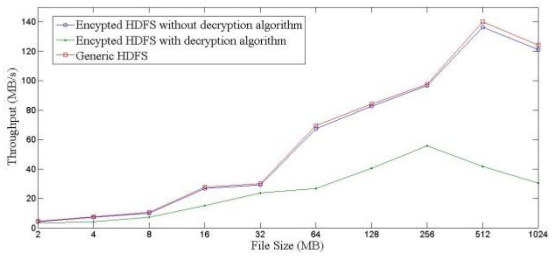

Figure 6. Reading throughput: encrypted HDFS vs. generic HDFS

VI. Conclusion

There is no effective mechanism for file privacy protection HDFS, so it is unsecure to apply it in real cloud environment. In this paper, a data encryption method based on HDFS is presented. We employed hybrid encryption scheme to protect file blocks and session keys, which can prevent datanode intruders from stealing user data. In contrast to the other similar works, we keep the advantage of lightweightness for client. The experiments show thatthe proposed method introduces 43% overhead, but the architecture overhead is negligible. Therefore, the future work is to take advantage of GPUs or multicore technology for paralleling the encryption/decryption modules to improve the overall performance.

Acknowledgment

The work is supported by the National High Technology Research and Development Program (863 Program) of China (No. 2011AA01A202), the National Science Foundation of China (No. 61021062),the “Six Talents Peak” High-Level Personnel Project of Jiangsu Province (No. 2011-DZXX-035).

References

[1] Tom White, Hadoop: The Definitive Guide. O’Reilly Media, 2009.

[2] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[3] The apache hadoop. http://hadoop.apache.org/.

[4] Owen O’ Malley, Kan Zhang, Sanjay Radia, Ram Marti, Christopher Harrell. Hadoop Security Design, Technical Report, 2009.

[5] Shen, Q., Yang, Y., Wu, Z., Yang, X., Zhang, L., Yu, X., ... & Long, M. (2012, March). SAPSC: Security Architecture of Private Storage Cloud Based on HDFS. In Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on (pp. 1292-1297). IEEE.

[6] Wilcox-O'Hearn, Z., & Warner, B. (2008, October). Tahoe: the least-authority filesystem. In Proceedings of the 4th ACM international workshop on Storage security and survivability (pp. 21-26). ACM.

[7] Tahoe-LAFS. https://tahoe-lafs.org/trac/tahoe-lafs.

[8] Lin, H. Y., Shen, S. T., Tzeng, W. G., & Lin, B. S. (2012, March). Toward Data Confidentiality via Integrating Hybrid Encryption Schemes and Hadoop Distributed File System. In Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on (pp. 740-747). IEEE.

[9] Park, Seonyoung, and Youngseok Lee. "Secure Hadoop with Encrypted HDFS." Grid and Pervasive Computing. Springer Berlin Heidelberg, 2013. 134-141.

[10] Advanced Encryption Standard. http://en.wikipedia.org/wiki/ Encryption_Standard.