Fall 2017

Analysis of outsourcing data to the cloud using autonomous key generation

Mortada Abdulwahed Aman

Department:

ANALYSIS OF OUTSOURCING DATA TO THE CLOUD USING AUTONOMOUS KEY GENERATION

by

MORTADA ABDULWAHED AMAN

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

2017

Approved by

Dr. Egemen K. Çetinkaya, Advisor
Dr. Maciej J. Zawodniok
Dr. Sanjay K. Madria

ABSTRACT

Cloud computing, a technology that enables users to store and manage their data at a low cost and high availability, has been emerging for the past few decades because of the many services it provides. One of the many services cloud computing provides to its users is data storage. The majority of the users of this service are still concerned to outsource their data due to the integrity and confidentiality issues, as well as performance and cost issues, that come along with it. These issues make it necessary to encrypt data prior to outsourcing it to the cloud. However, encrypting data prior to outsourcing makes searching the data obsolete, lowering the functionality of the cloud. Most existing cloud storage schemes often prioritize security over performance and functionality, or vice versa. In this thesis, the cloud storage service is explored, and the aspects of security, performance, and functionality are analyzed in order to investigate the trade-offs of the service. DSB-SEIS, a scheme with encryption intensity selection, an autonomous key generation algorithm that allows users to control the encryption intensity of their files, as well as other features is developed in order to find a balance between performance, security, and functionality. The features that DSB-SEIS contains are deduplication, assured deletion, and searchable encryption. The effect of encryption intensity selection on encryption, decryption, and key generation is explored, and the performance and security of DSB-SEIS are evaluated. The MapReduce framework is also used to investigate the DSB-SEIS algorithm performance with big data. Analysis demonstrates that the encryption intensity selection algorithm generates a manageable number of encryption keys based on the confidentiality of data while not adding significant overhead on encryption or decryption.

ACKNOWLEDGMENTS

I would like to thank Dr. Egemen K. Çetinkaya, Dr. Sanjay K. Madria, and Dr. Maciej J. Zawodniok for their feedback and continuous help with this work. Furthermore, I thank the CoNetS research group for listening and giving feedback to the ideas presented in this document.

This thesis work was supported by the Department of Electrical and Computer Engineering at Missouri University of Science and Technology by providing funding through a graduate teaching and research assistantship.

I would like to thank Missouri University of Science and Technology's IT database team and Perry Koob for extensive help with providing required equipment and support during the experimentation phase of this thesis. I would also like the doctoral student, Katrina Ward for supporting the ideas presented in this work.

Finally, I would like to thank ACM, NSF/CANSec, and NSF/Raytheon BBN Technologies for providing travel grants to attend DCC 2016, CANSec 2016, and GENI-NICE 2016, respectively.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF ILLUSTRATIONS viii

LIST OF TABLES ix

SECTION

1. INTRODUCTION AND MOTIVATION 1
   - 1.1. CONTRIBUTIONS 5
   - 1.2. PUBLICATIONS 6
   - 1.3. ORGANIZATION OF THESIS 7

2. BACKGROUND AND RELATED WORK 8
   - 2.1. PRELIMINARIES 8
     - 2.1.1. Cryptography 8
     - 2.1.2. Deduplication 9
     - 2.1.3. Searchable Encryption 9
   - 2.2. CLOUD STORAGE SERVICE 9
   - 2.3. DEDUPLICATION 14
   - 2.4. SEARCHING ENCRYPTED DATA 15
   - 2.5. MAPREDUCE 18
   - 2.6. CLOUDLAB TESTBED 19

3. ARCHITECTURE 20
3.1. WORKFLOW 20
3.2. ENCRYPTION INTENSITY SELECTION 23
3.2.1. Low Intensity 24
3.2.2. Medium Intensity 24
3.2.3. High Intensity 25
3.3. DEDUPLICATION 25
3.4. ASSURED DELETION 27
3.5. SEARCHING ENCRYPTED DATA 28
3.5.1. Keyword Search in Encrypted Index 29
3.5.2. Duplicate Checking of Encrypted Data 30
3.6. MAPREDUCE APPLICATIONS 30
3.6.1. Indexing 31
3.6.2. Index Search 32
3.6.3. Disk Search 34

4. RESULTS 36
4.1. WIKIMEDIA DUMP 37
4.2. DEDUPLICATION PERFORMANCE 39
4.3. KEY GENERATION ANALYSIS 39
4.4. ENCRYPTION PERFORMANCE 41
4.5. DECRYPTION PERFORMANCE 42
4.6. INDEXING PERFORMANCE 44
4.7. SEARCH PERFORMANCE 44
4.7.1. Index Search 44
4.7.2. Disk Search 46
4.8. MAPREDUCE APPLICATIONS 47

4.8.1. Indexing Performance 47
4.8.2. Index Search Performance 49
4.8.3. Disk Search Performance 51

5. CONCLUSIONS 53

6. FUTURE WORK 56

APPENDIX 58

REFERENCES 220

VITA 227

LIST OF ILLUSTRATIONS

Figure Page

3.1 DSB-SEIS architecture 21
3.2 Client application flowchart 21
3.3 Cloud application flowchart 22
3.4 Encryption intensity selection example 25
3.5 MapReduce indexing application workflow 31
3.6 MapReduce index search application workflow 33
3.7 MapReduce disk search application workflow 34
4.1 WikiMedia word distribution 37
4.2 Deduplication overhead 38
4.3 Size of data 38
4.4 Key generation 40
4.5 Encryption performance 41
4.6 Decryption performance 43
4.7 Index performance 43
4.8 Index search performance 45
4.9 Search hits 45
4.10 Disk search performance 46
4.11 MapReduce indexing performance 48
4.12 MapReduce index search performance 49
4.13 Index search hits 50
4.14 MapReduce disk search performance 51

LIST OF TABLES

Table Page

2.1 Comparison of cloud storage schemes 10

4.1 Testing variables..... 36

1. INTRODUCTION AND MOTIVATION

Cloud computing is a service widely used due to the many features it provides to its users. Data storage and computation are two important examples of services that can be provided to users through cloud computing [1, 2, 3, 4, 5, 6]. The cloud services can reduce the amount of money spent on personal computing power and maintenance. For this reason, many individuals and enterprises turn to cloud computing for their data storage and computation needs such as data storage.

However, outsourcing data to the cloud concerns a majority of cloud's customers: How secure is the data stored on the cloud? How long does the transmission take? How much control does the customer have on their data? To solve the issue of data security (confidentiality and integrity), users tend to encrypt their data prior to outsourcing to the cloud. Even though encryption provides confidentiality for data, it also limits the capabilities of the cloud. Since the data is encrypted, the cloud cannot compute this data, and therefore the utility of the service is diminished.

Recently, attacks launched by adversaries have been getting stealthier and more complex, which increases the necessity of a more secure storage scheme that eliminates some of the current issues in data storage. Some examples of attacks directed towards cloud storage are hash value manipulation attacks and stolen host ID attacks [7]. These attacks aim to either steal, modify, or forge the client's data.

To explain some of these attacks, previous attacks of one of the most popular cloud storage providers, Dropbox, are analyzed [7]. Dropbox can be used to outsource data and gain access to it later on from any device that is connected to the Internet. A client application is installed on the personal device that synchronizes all changes to the data automatically. Data across multiple computers owned by the same user can also be synchronized. When the client application adds a file to the local Dropbox folder located

on the personal device, it splits the file into chunks up to 4 MB. A hash is then calculated from each chunk and sent to the Dropbox servers where it is compared to the hashes of the data stored there. If the chunk exists on the server, the user is granted access to it. If the calculated hash is not found, the server requests the chunk to be transmitted. After the chunk is sent, Dropbox calculates the hash on its cloud servers to compare it to the hash sent from the client application for validation. This process is referred to as deduplication. Dropbox's scheme is believed to have been changed since the attacks were discovered so that the deduplication is not performed across all users globally [8]. Although if deduplication remained global, unauthorized users could get access to data.

An attacker could gain unauthorized access to files by manipulating the hash value calculated for chunks by the client application. One way the calculated hash value can be manipulated is by altering the cryptography libraries and replacing the default libraries shipped with the Dropbox client application. If the hash is found on the cloud servers once it is manipulated and sent to the cloud, the attacker gains unauthorized access to files or chunks on the cloud even if the attacker does not own any of the files. On the other hand, if the server does not find a match for the hash, the chunks will be sent to the Dropbox servers, where they will be hashed and compared to the manipulated hash sent previously. Since they will not match, an error will be detected and the file will not be stored. However, this will not affect the attacker in any way. This attack is called the hash value manipulation attack. This attack is undetectable by the data owner or the server since Dropbox does not validate ownership of data or notify the user when another user gains access to their data.

During the installation of Dropbox on a device, it creates a host ID for that computer and links it to the owner's Dropbox account. If an attacker steals the host ID of a different Dropbox account, he can simply replace his own host ID with the stolen one and download the victim's files. This is referred to as stolen host ID attack [7].

The previous Dropbox example is but one scenario to many issues facing outsourcing data on the cloud, although Dropbox has changed its algorithms to avoid some of the attacks in the previous example. Many research papers have been written on the issues of the cloud [2, 3, 4, 9, 10]. This work is focused on investigating three main issues: security of data, performance of the service, and functionality of data stored on the cloud. The main features investigated with this work are data integrity and confidentiality during transit and rest, assured deletion, encryption intensity selection, deduplication of data, and searching encrypted data.

Data confidentiality is one of the issues that many research papers discuss due to the significant damage it can have on cloud customers. For instance, if a hospital's health records were to be revealed on the cloud, many patients' privacy would also be exploited. Therefore, confidentiality of data must be retained during transit and rest. This means that the data must be encrypted while being transmitted from the customer to the cloud, as well as while it is stored on the cloud. Returning to the previous example, Dropbox secures the transmission of data using SSL and encrypts the data with an AES key once it reaches its servers. The problem with this scenario is that the encryption and storage are done at the same place. Additionally, the keys are also generated on the cloud. This raises the question of how secure this encryption really is. To solve this issue, customers encrypt their data prior to outsourcing it to the cloud, but the pre-encryption of data can significantly limit the functionality of this data because the cloud will not be able to compute the encrypted data. Also, how secure is this encryption anyway? If the customer generates one key and encrypts all of their files with the same key, then breaking that key will reveal all of the data. On the other hand, if the customer generates a key for each file, even though this increases the complexity of revealing the data, it also makes key management a lot more complex. For this reason, a feature called encryption intensity selection [11, 12] is proposed, where

keys are generated dynamically based on the confidentiality of each file. The features are compared to the main two methods used in the service: using a single key to encrypt all data, and using a separate key to encrypt each file.

In addition to encryption intensity selection that generates keys autonomously to pre-encrypt data, a feature is proposed to increase the functionality of the encrypted data: searching encrypted data. This feature will allow documents that contain a certain word to be found, as well as check if a piece of data exists on the cloud. For example, a hospital outsources their files that contain billing information, patient health records, and appointment schedules to a cloud. This hospital might later want to retrieve only the files that contain a patient's health records. If the files are encrypted on the cloud, the hospital will have to retrieve all of their data, decrypt them, and find the ones they are seeking. For this reason, searchable encryption is implemented, where an encrypted index is generated from all documents and outsourced to the cloud to be searched later on. Another use for searchable encryption in this scheme is to check if a piece of data exists on the cloud. Similar to the Dropbox example, one should be able to check if a chunk of data exists on the cloud. Although since this data is pre-encrypted, it will be unintelligible if an attacker gains unauthorized access to this data. This work analyzes searching encrypted data and how encryption intensity selection affects this feature.

The performance and cost of outsourcing data to the cloud is another concern of cloud customers. How long will the client machine take to prepare the data for transmission? How long will it take to encrypt the data? How much bandwidth will it take to transmit the data to the cloud? How much will it cost to store data on the cloud? Is there a way to reduce the costs of this service? The highest cost of the service is the transmission of data. To reduce some of this cost, deduplication of redundant data before transmission is proposed [11, 12]. Eliminating redundant data can reduce the size of the data to be outsourced significantly. A simple deduplication scheme in this work is implemented to analyze its performance and how other features and attributes affect its performance. Its

overhead versus the amount of data it eliminates is analyzed, and additional features are implemented that enhance the performance of deduplication by implementing multi-aspect awareness, a feature that looks into different aspects of files while deduplicating [13, 14].

Keeping all of the issues discussed above, this work investigates the security, performance, and functionality of the cloud storage service. In order to do so, a cloud storage scheme is implemented to investigate the trade-offs between security, performance, and functionality for outsourcing data to the cloud. This scheme is called DSB-SEIS, a deduplicating secure backup scheme with encryption intensity selection. All the features discussed above are implemented in DSB-SEIS, and each feature is analyzed, as well as the trade-off of encryption intensity selection. The security and performance of the scheme, as well as the storage service, are investigated in this work. Additionally, MapReduce framework in the data storage service is explored. Three MapReduce applications are developed: indexing, index search, and disk search applications. The three algorithms evaluate the performance when a larger scale of data is used. The WikiMedia dump [15] is used as input for all applications in this thesis work.

Preliminary results for this work were generated using the CloudLab [16] infrastructure. A cloud was created on CloudLab using OpenStack [17]. As for the final results, a local cloud created by the Missouri University of Science and Technology (Missouri S&T) database team was used. The reason for switching to a local cloud was to simplify the setup step.

1.1. CONTRIBUTIONS

The contributions of this work are summarized as follows:

- The trade-offs of cloud storage service are discussed and analyzed.
- An autonomous encryption key generation algorithm, encryption intensity selection, based on confidentiality of data is developed.

- DSB-SEIS, a working cloud-based storage scheme is implemented in Java.
- DSB-SEIS includes features such as deduplication, assured deletion, searching encrypted data, and encryption intensity selection to balance the trade-off between security, performance, and functionality.
- MapReduce applications of indexing, and searching encrypted data are developed to investigate performance of Hadoop MapReduce in cloud data storage.
- The implementation of DSB-SEIS is evaluated, and the performance of the scheme, and overall, the storage service is analyzed.

1.2. PUBLICATIONS

This thesis work allowed for the following publications:

- Mortada A. Aman and Egemen K. Çetinkaya, "Towards Cloud Security Improvement with Encryption Intensity Selection," in *Proceedings of the 13th IEEE/IFIP International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 55–61. Munich, March 2017. (Best Student Paper Award).
- Mortada A. Aman and Egemen K. Çetinkaya, "A Secure Backup System with Encryption Intensity Selection and Deduplication," in *The Network Innovators Community Event (GENI NICE) Poster Session*, Irvine, CA, December 2016.
- Mortada A. Aman and Egemen K. Çetinkaya, "A Secure Backup System with Encryption Intensity Selection and Deduplication," in *10th Central Area Networking and Security Workshop (CANSec) Poster Session*, St. Louis, MO, October 2016.
- Mortada A. Aman and Egemen K. Çetinkaya, "DSB-SEIS: A Deduplicating Secure Backup System with Encryption Intensity Selection," in *Proceedings of the 4th ACM PODC Workshop on Distributed Cloud Computing (DCC)*, Chicago, IL, July 2016.

1.3. ORGANIZATION OF THESIS

The rest of this thesis is organized as follows. In Section 2, the preliminary knowledge needed for this work including deduplication, searching on encrypted data, MapReduce, and the CloudLab testbed is explained. Additionally, a list of related work in the cloud storage service is presented. In Section 3, the architecture of the DSB-SEIS scheme and all of its features are explained in detail. In Section 4, the results for the DSB-SEIS scheme are shown and evaluated. In Section 5, final thoughts and a conclusive analysis are presented. Finally, possible solutions to issues of the cloud storage service are proposed in Section 6.

2. BACKGROUND AND RELATED WORK

Preliminary concepts for this work are introduced in this section, and related work used to motivate the ideas is presented. The preliminaries include cryptography, deduplication, and searchable encryption. Additionally, the related work is divided into the following sections: work on the cloud storage service, deduplication, searching encrypted data, MapReduce, and CloudLab testbed. Each of the sections is listed here to show existing work and explain technologies used in this work.

2.1. PRELIMINARIES

This section explains necessary preliminary knowledge including cryptography, deduplication, and searchable encryption.

2.1.1. Cryptography. Cryptography is the study of secure information by changing the true meaning of a message to a seemingly arbitrary, unintelligible message. In the computing field, this can be done with methods such as encryption and hashing. Encryption and hashing differ such that encryption can be reversed to retrieve the original message, while hashing is a one-way function that cannot be reversed. Both encryption and hashing are used in data storage schemes. Encryption is used to keep data unintelligible on the server; hashing is used in deduplication, which is explained in the next subsection, and to check the integrity of data. The encryption algorithm used in data storage needs to be relatively secure to maintain an acceptable level of security. The security provided by encryption can be measured by the key size, block size, number of keys, and the algorithm used. As for hashing, the security of the algorithm depends on the probability of producing a hash collision. This means that the algorithm should have minimum to no chance of producing the same hash for two different messages.

2.1.2. Deduplication. Deduplication is defined as the elimination of redundant data. It is often done prior to transmitting data through a network to reduce the size of data, and therefore reduces the time taken to transmit the data. Deduplication is the opposite of replication, which makes multiple copies of data and distributes them to other nodes of the cloud. In most deduplication schemes, a hash algorithm is used to identify duplicates. For instance, hashes for two files are computed, A and B. If the hash of A is equal to the hash of B, then B is a duplicate of A. Some of the most used hash algorithms in deduplication are MD5, SHA, and Rabin Fingerprints [18**?**]. Additionally, a lot of research has been done to improve the efficiency and accuracy of deduplication [13, 14, 19, 20, 21].

2.1.3. Searchable Encryption. Searchable encryption adds the functionality of searching for a piece of data while encrypted. That means that no point of time during the search is data decrypted. This feature allows the user to search for a word contained in documents. An encrypted index can be generated from documents and then sent to the cloud. The cloud can then search the encrypted index to find requested data and return it to the client. Searchable encryption can be used to enable secure deduplication as well by allowing the search of chunks of data (instead of a word) on disk (instead of an index). Similar to searching an index for a word, the disk is encrypted to make the search secure.

2.2. CLOUD STORAGE SERVICE

In this section, existing cloud storage services and the features they provide are analyzed and compared to DSB-SEIS. In Table 2.1, some of the most popular storage service providers are listed, and their security, performance, and functionality features are compared.

Table 2.1. Comparison of cloud storage schemes

Service provider	Encrypted transmission	Encrypted storage	Encryption keys location	Encryption algorithm	Encryption intensity selection	Search on encrypted data	Assured deletion	Deduplication
Dropbox [22]	yes	yes	cloud	AES-256	no	no	no	server-side
Google Drive [23]	yes	yes	cloud	AES	no	no	no	files with the same name
SpiderOak ONE [24]	yes	yes	client machine	AES-256	no	no	no	yes
Amazon S3 [25]	yes	yes	cloud or client machine	AES	no	no	no	yes
Box [26]	yes	yes	separate cloud	AES-256	no	no	no	no
iCloud [27]	yes	yes	cloud	AES	no	no	no	no
Microsoft Cloud [28]	yes	yes	cloud	AES-256	no	no	no	no
Cumulus [29]	yes	yes	client machine	gzip, bzip2, & gpg	no	no	no	local
Bacula [30]	yes	yes	client machine	AES, blowfish & RSA	no	no	no	previously backed-up files
FadeVersion [31]	yes	yes	trusted third-party	AES	no	no	yes	yes
DSB-SEIS [11]	yes	yes	client machine	AES-256	yes	yes	yes	local and secure server-side

Some of the most popular storage service providers in the industry are Dropbox [22], Google Drive [23], SpiderOak [24], Amazon Web Services [25], Box [26], Apple iCloud [27], and Microsoft Cloud [28]. In addition, some schemes can be manually installed on any platform-as-a-service (Paas) cloud and used as a service. In this thesis, three schemes that can be installed manually are looked at: Cumulus [29], Bacula [30], and FadeVersion [31]. All of these schemes aim to achieve one main goal: securely store the user's data on the cloud until a user requests to view, retrieve, or delete it. All of these schemes and how they relate to DSB-SEIS will be discussed in this section.

The previous section presented an example of how attackers could exploit the Dropbox algorithm to access data they are not authorized to. Since then, customers have complained about this issue, which Dropbox has solved by disabling global deduplication across all users. Dropbox uses 128-bit SSL/TLS to achieve secure encrypted transmission of data. Once the data is stored on the server, it uses a single 256-bit key to encrypt data at rest. The encryption keys are generated and stored on the cloud, which can be exploited easily because both the data and keys are stored on the same cloud. Dropbox does not provide any control of the encryption algorithm since it is programmed to use one 256-bit AES key. Searching encrypted data is not possible on Dropbox. The cloud will have to decrypt the data in order to search it. Dropbox does not provide assured deletion, either. An experiment is conducted to observe how long a file is stored on Dropbox after a delete request is sent [7, 22]. It was found that the files remained on Dropbox six months after the delete request. This means that the data could still be stolen after the user decides to discontinue service. Finally, Dropbox still provides deduplication, but within the same user's data.

Google Drive is another big provider of the storage service. Google Drive uses 256-bit SSL/TLS [23, 32, 33] to secure data in transit. Once the data reaches the cloud, it is encrypted with a 128-bit AES key [**?**] and stored on the cloud. Similar to Dropbox, the keys are generated and stored on the cloud. Google Drive does not provide encryption intensity

selection, searching encrypted data, or assured deletion. Google Drive's deduplication differs from all other kinds of deduplication. Google Drive looks at the name of files, and if it finds a match from the files previously stored by the same user, Google Drive finds the difference between the previously uploaded version and the new version and uploads the difference to the cloud. Users can then view both versions.

SpiderOak One could possibly be one of the most secure storage service providers. It uses a combination of 2048-bit RSA and 256-bit AES layered encryption to secure transmission and storage [24**? ?**]. The encryption is done on the client machine prior to transmitting the data. This means that once the data leaves the client machine, the data is unintelligible. The encryption keys are stored on the client machine and never sent to the cloud. Hence, if an attacker gains access to the cloud, the data will still be secure since the keys are not stored in the same place. The drawback to this is that the functionality of data is nonexistent because any computation of the data requires decryption and therefore will require the user to retrieve the data. SpiderOak One does not provide encryption intensity selection, searching encrypted data, assured deletion, or deduplication.

Amazon S3 is used by many enterprises for data storage and computation, although it is by no means perfect [25]. Amazon S3 secures data in transit using SSL-encrypted endpoints using HTTPS [34]. To secure data at rest, Amazon offers some options. The customer can choose server-side encryption that uses a 256-bit AES key generated and stored on the cloud, or provide their own keys that will be used to encrypt the data on the cloud and then discarded after use. Amazon also allows client-side encryption by generating any kind of encryption key, symmetric or asymmetric, on the client machine and never sending the key to the cloud. There is a trade-off for both server-side and client-side encryption. Choosing server-side encryption reduces the security of data, similar to Dropbox and Google Drive, while choosing client-side encryption reduces functionality, similar to SpiderOak One. Amazon provides some control over the encryption of data, but it does not allow the user to select the intensity of encryption unless they choose to client-side encryption.

Amazon does not support assured deletion or searching encrypted data. Amazon offers StorReduce, an on-cloud block-level deduplication software that deduplicates data before migrating it to the cloud. This service is extra and not provided to all customers.

Box provides both encrypted transmission and storage [26]. Transmission is encrypted using TLS, while data at rest is encrypted using a 256-bit AES key that is then encrypted using another 256-bit AES key [**?**]. Box also partnered with Amazon Web Services to use their key management services (KWS) and provide layered encryption. The keys are generated and stored on an Amazon cloud separate from the data. Box does not allow encryption intensity selection, searching encrypted data, assured deletion, or deduplication.

Apple iCloud also provides encrypted transmission and storage [27]. Data in transit is encrypted using SSL, and data at rest is encrypted using a 128-bit AES key and stored on the cloud along with the data. Apple iCloud does not provide encryption intensity selection, searching encrypted data, assured deletion, or deduplication.

Microsoft Cloud provides security of data in transit by using TLS/SSL [28]. Data at rest is encrypted using a 256-bit AES key and stored on the cloud with the data. Microsoft Cloud does not provide encryption intensity selection, searching encrypted data, assured deletion, or deduplication.

Cumulus is developed to be installed on any thin cloud (providing minimal interface of get, put, delete, and list) that provides storage service [29]. Cumulus encrypts the data using compression algorithms .gzip, .bzip2, or .gpg prior to uploading it to the cloud in order to secure the data in transit and at rest. The encryption keys are stored on the client machine. Cumulus does not provide encryption intensity selection, searching encrypted data, or assured deletion. Cumulus provides deduplication of local data on a user machine, but not across multiple users.

Bacula is a network client/server application that is developed to enable the user to store their data on any server [30]. Bacula uses TLS to encrypt during transmission. It also offers a layered encryption approach to encrypt data prior to transmitting it. It first generates an RSA asymmetric key to encrypt the AES session keys, which are used to encrypt the data. Bacula gives the option of choosing the key size of the session keys but does not offer encryption intensity selection. It does not offer searching encrypted data or assured deletion. The deduplication offered by Bacula eliminates the non-modified, previously-sent files. If a file is modified after it has been stored, the difference is calculated and sent to the cloud.

FadeVersion serves as a secure cloud storage service that adds a security layer to existing cloud storage services [31]. It is an extension from Cumulus that aims to add additional performance and security features. Its main goal is to make assured deletion and version-control compatible. The data is encrypted using a layered encryption approach using the AES algorithm. FadeVersion generates a data key for each file and encrypts the data using these keys. The data keys are then encrypted using policy-based keys called control keys that define how each file is accessed. The encryption keys are stored and managed by a third-party key management system. FadeVersion does not provide encryption intensity selection or searching encrypted data, although it does provide assured deletion by deleting the control keys once a policy is revoked, making the data unrecoverable. It also provides deduplication of data by only transmitting one copy of the same object and creating pointers to that object once a duplicate is found.

2.3. DEDUPLICATION

The following schemes improve the deduplication efficiency of cloud data storage systems, thereby improving the service's performance. AA-Dedupe [14] proposes an application-aware local deduplication scheme that would use multiple methods of deduplication based on file size and application type. Local deduplication means that the source of

the data is deduplicated, as opposed to where the data is stored. Its main goal is to improve the efficiency and throughput of deduplication. It uses chunk-based deduplication and takes many attributes such as chunk size and hash function into account, making the deduplication dynamic based on the type of application (.rar, .mp3, .txt, etc.) and file size. SAM [13] is an extension of AA-Dedupe that combines file-level deduplication and chunk-level deduplication to achieve better deduplication efficiency and throughput. Similar to AA-Dedupe, SAM exploits file semantics such as file size, application type, and modification timestamp. SAM also combines deduplication across multiple users, as well as within the same user. CABDedupe [19] attempts to boost the deduplication performance by scanning only for modified and new files to store or restore. CABDedupe is unique in that it deduplicates both store and restore operations, which reduces the size of data on both the incoming and outgoing channels of the user and the cloud. A single-server scheme for secure deduplication across multiple users is presented [20]. It uses additively homomorphic encryption to enable the search for a piece of data. It also uses the password-authenticated key exchange (PAKE) to enable users to share keys. In this scheme, the application-awareness feature proposed in AA-Dedupe and the file semantic aspect introduced in SAM are utilized, and deduplication on both store and restore are deployed, similar to CABDedupe. This research also utilizes searchable encryption to achieve secure cloud-side duplicate checks. The deduplication algorithm used in this thesis work is simple and minimal in its functionality. Deduplication in this scheme, as well as in the previously mentioned work, is classified as archival storage deduplication [21].

2.4. SEARCHING ENCRYPTED DATA

Related work presents two major ways of searching encrypted data: general-purpose schemes (such as fully-homomorphic encryption [FHE] or oblivious RAMs [ORAMs]), and special-purpose schemes (such as searchable symmetric encryption, SSE). A study finds

that even though general-purpose solutions such as FHE and ORAMs are more general in their use, they can be expensive [35]. SSE provides the best trade-off between security, performance, and functionality.

A hybrid cloud approach for secure and authorized deduplication that uses convergent encryption is presented [36]. Even though the scheme achieves what the authors aimed for, the scheme has two main weaknesses. First, the use of convergent encryption makes this scheme less secure than what one hopes for because the data is encrypted with its own contents. Second, the need for a hybrid cloud can be impractical in many cases. Another scheme that uses additively homomorphic encryption and password-authenticated key exchange to enable secure deduplication is presented [20].

Controlling data in the cloud without losing computation is discussed [37]. The paper also proposes some possible methods of controlling data. The authors present current issues with the cloud and what makes customers afraid of using its services to their full potential. The authors also explain how computation-supporting encryption can be used to improve the advantages of cloud computing and make these services more appealing to users.

The following schemes are general-purpose and use public-key encryption to achieve keyword search encrypted data. A scheme that uses hidden vector encryption in order to achieve conjunctive, subset, and range queries on encrypted data is presented [38]. Additionally, a scheme that can provide sum and average queries using homomorphic encryption is constructed [39]. The authors claim that the performance is comparable with traditional encryption schemes, although this could be improved by replacing homomorphic encryption with searchable symmetric encryption.

A scheme that uses homomorphic encryption algorithm to run queries on the encrypted data after user authentication is developed [40]. The author also explores authorization and collusion issues. Most schemes that utilize homomorphic encryption can be expensive and thus can be improved by utilizing SSE.

The following schemes propose searchable symmetric encryption (SSE) schemes to provide different functionalities. A new kind of attack on SSE schemes, called the adaptive chosen keyword attack IND-CKA, is defined [41]. This attack takes advantage of the access pattern of the keyword searches and attempts to figure out what words are contained in the respective files. Additionally, the author develops a scheme that can defend against IND-CKA and IND2-CKA that uses bloom filters and pseudo-random functions, although this scheme has the drawback of generating false positives. Later on, improved definitions and efficient constructions of SSE schemes are defined [42]. The authors revisit the IND2-CKA introduced previously and attempt to provide security for the user queries on top of the IND2-CKA security. They also define the nonadaptive and adaptive attacker models and mention that all previous work falls within the nonadaptive settings. They continue to construct two schemes to tackle the adaptive and nonadaptive settings. It is worth mentioning that to solve the adaptive attacker model, their scheme requires higher communication overhead and more storage at the server. They also add the ability to use SSE across multiple users, where one user can own the data and generate trapdoors for other users to search the data. The drawback to this scheme is that the search time is sequential and not dynamic. Furthermore, security definitions for ciphers that permit length-preserving encryption of a data stream with only a single pass through the data are explained [43]. They provide two implementations: HCBC1 and HCBC2. A scheme that solves the sequential SSE problems and parallelizes the keyword search encrypted data is presented [35]. The authors offer three main properties with their SSE construction: CKA security, no info leakage on updates, and the ability to implement their scheme efficiently in external memory. They use a red-black tree-based data structure to find the desired keyword in the encrypted data. This implementation also offers dynamic keyword search and secure data updates, although this scheme can be extended to reduce the size of the tree, perform the searches in external memory, and verify the results of the search. A parallelizable and authenticated scheme for online ciphers is proposed [44]. The authors

conclude that it is about five times faster than previous work. However, the scheme does not achieve functionality of the keyword search desired. A highly-scalable SSE scheme that supports Boolean queries is constructed [45]. The scheme provides a realistic and practical trade-off between performance and privacy by efficiently supporting very large databases. A scheme that supports ranked search for multiple keywords is introduced [46]. It also preserves the privacy of the search by limiting information leakage. However, the scheme uses inner product similarity coordinate matching, which can be expensive. A dynamic SSE scheme in very large databases is introduced [47]. The scheme is dynamic in the sense that the data is updatable after encryption, parallel, and tackles the adaptive and nonadaptive attacker settings. It also offers a small index size and short search time. An implementation that enables sub-string search while data is encrypted is introduced [48]. The authors develop an SSE algorithm called queryable encryption. They use suffix trees and achieve asymptotic efficiency comparable to unencrypted suffix trees. The scheme defines what will be leaked prior to the implementation to make it relatively secure. An efficient SSE scheme that supports multiple kinds of searches of encrypted data is proposed [49]. The searches offered from their scheme are the wild-card search, similarity search, fuzzy-keyword search, and disjunctive keyword search. Their scheme supports dynamic operations such as addition and removal of data. This scheme is secure against nonadaptive chosen keyword attacks but not adaptive keyword attacks. A memory-leakage-resilient SSE scheme is introduced [50]. The authors mention that attackers can often obtain some or even all secret keys from fast side-channel attacks. They investigate SSE methods to prevent this leakage.

2.5. MAPREDUCE

MapReduce is a model for distributed processing of large data across multiple nodes or clusters. It aims to simplify the development of distributed applications and enable big-data management [51, 52, 53, 54]. In order to use MapReduce, two functions need to be developed: a map and a reduce function. Once these functions are developed, the model

automatically parallelizes the processing and computation across the nodes or clusters. The model also handles node failures automatically and ensure task completion. If a node fails, the master node detects this failure and reassigns that node's responsibilities to a different node. A related work evaluated the scalability and efficiency of indexing strategies using MapReduce [55]. This paper shows that the most efficient algorithm for indexing using MapReduce is the per-posting algorithm. Although, in this work, the per-token algorithm is used for simplicity.

2.6. CLOUDLAB TESTBED

CloudLab is a scientific instrument that allows researchers and students to create their own clouds free of charge. It is built to minimize the impact of experiments interfering with each other. Hence, the hardware is isolated as much as possible to ensure the confidence and validity of tests. At the core, CloudLab is distributed across three main locations: Utah (University of Utah), Wisconsin (University of Wisconsin-Madison), and South Carolina (Clemson University). The three sites provide diversity for all kinds of research related to the cloud. They communicate with each other through IP and layer-two links such as the SDN-based 100 Gbps network.

The software provided by CloudLab is also diverse and flexible enough to allow for different cloud stack deployments. The user must first create a profile and a description for the hardware and software aspects of the cloud, and then proceed to gain full control of the resources [56, 57]. The preliminary results of this project were generated on the CloudLab testbed [11].

3. ARCHITECTURE

This section explains how DSB-SEIS functions in detail, including all the features proposed previously. Shown first is the overall architecture of the scheme and the work flow of both the client and the cloud. DSB-SEIS's features, encryption intensity selection, deduplication, assured deletion, and searching encrypted data, are then explained. Furthermore, MapReduce applications were developed for indexing, searching the index, and searching the disk to evaluate how the algorithms of DSB-SEIS perform with a larger scale of data using Hadoop MapReduce. The explanation for the MapReduce applications are explained at the end of this section. The cloud model considered in this work is an honest-but-curious cloud. This mean that the cloud will perform all computation requests, but might read the information while doing so.

3.1. WORKFLOW

Figure 3.1 shows the overall architecture of DSB-SEIS. On the left side, users request store, restore, and search from DSB-SEIS's client application. The scenario here is that all users have DSB-SEIS installed on their machines. The client then connects to the cloud application, which will then fulfill the request and send back the requested data to the client. The assumption here is that the users share their encryption keys if they would like to share data. The reason for this assumption is that sharing and mobility is limited since the scheme retains all encryption keys on the client machine and never allows for decryption outside this machine. Evaluation of shared schemes is considered part of the future work.

User 3 in Figure 3.1 requests data to be stored on the cloud. DSB-SEIS will then scan the user's filesystem for data, extract text from documents and file names, generate an encrypted inverted index, split the data into a fixed-size chunks, encrypt the data using the keys assigned by the encryption intensity selection feature, transmit the data to the cloud,

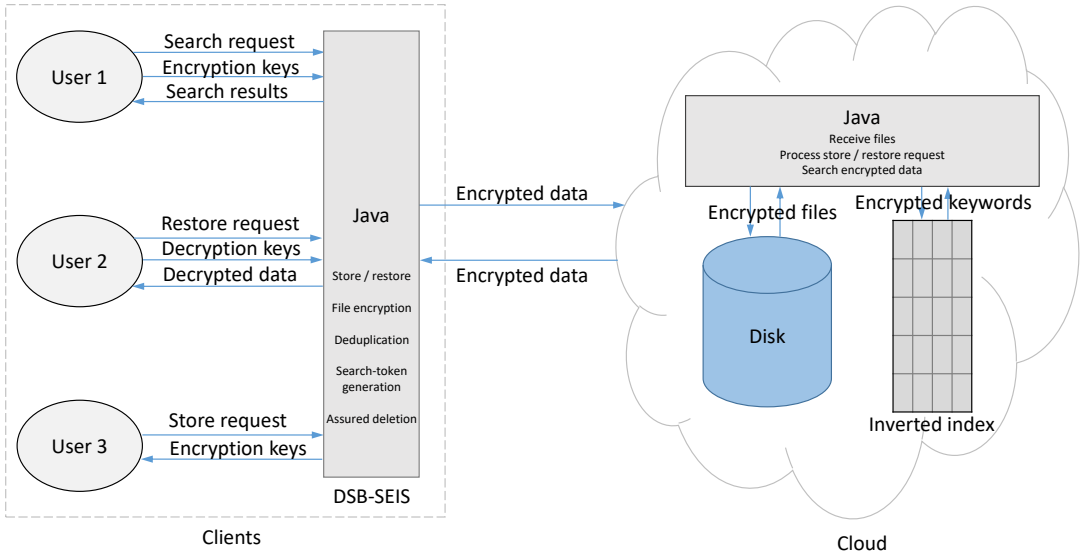


Figure 3.1. DSB-SEIS architecture

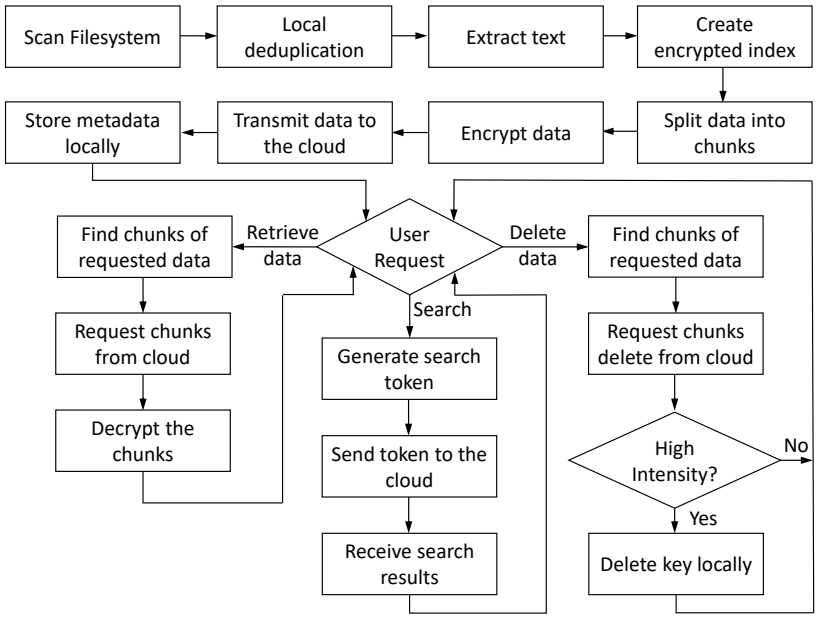


Figure 3.2. Client application flowchart

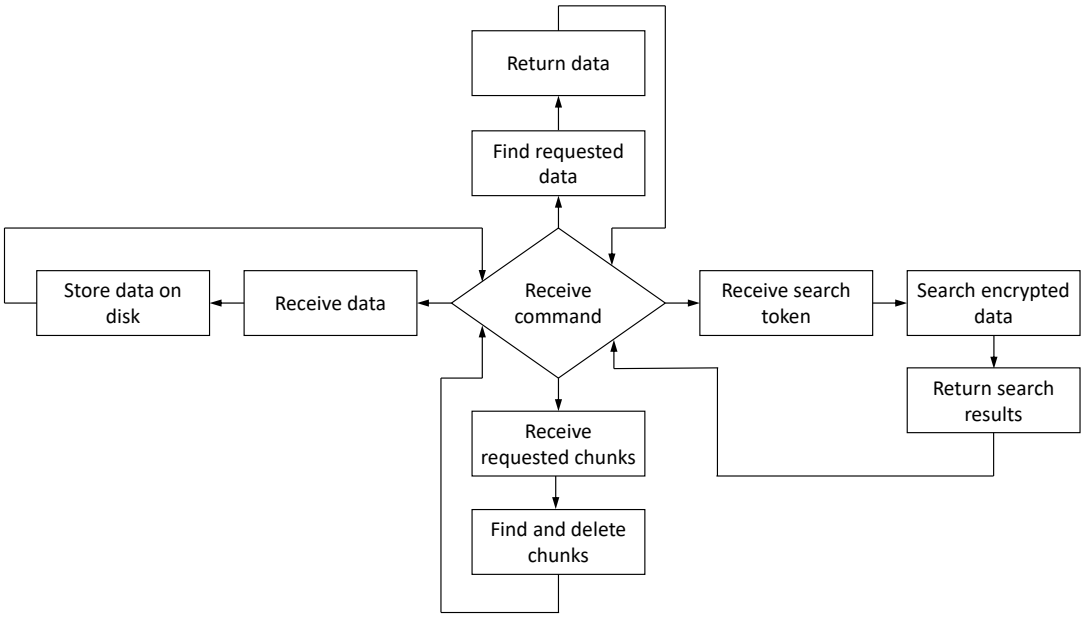


Figure 3.3. Cloud application flowchart

and finally store the metadata of all files sent on the client machine. This process is shown as a flow chart in Figure 3.2. After this is done, the client application then waits for a request from the user. The user can request to retrieve data from the cloud, search over encrypted data, or assuredly delete data. User 2 in Figure 3.1 requests to retrieve data from the cloud. The respective encrypted chunks of data is requested to be retrieved from the cloud, decrypted using the correct keys, and then given back to the user. User 1 in Figure 3.1 requests to search over the encrypted data. DSB-SEIS allows two kinds of search on the cloud: keyword search on the encrypted inverted index stored on disk, and duplicate checks on the encrypted chunks of data stored on disk. Both kinds of search follow a similar procedure: A search token is generated from the data to be searched for by encrypting and then hashing the data. The search token is then sent to the cloud, and the results are returned and decrypted. In order to find data stored on the cloud, the user must use the same key to generate the search token as the key used to encrypt the data initially. Hence, if a search token is generated for data using key B after the same data was encrypted with

key A and sent to the cloud, then the cloud will not return any results. Another feature not shown in Figure 3.1 is assured deletion. If assured deletion is requested and the respective chunks of data on the cloud are found to be deleted, then the key is deleted locally only if the encryption intensity of the data is set to the high intensity level.

Figure 3.3 shows the work flow of the cloud. The cloud waits for a request from the client, and once a command is received, it proceeds to perform the task requested. If a store request is received, the cloud receives the data and stores it on its disk. Alternatively, if a restore request is received, the cloud finds the data and returns it to the client. The cloud finds data and deletes it from its storage disk if a delete request is received. Finally, if the client requests to search the data, it proceeds to scan the inverted index or storage disk depending on the kind of search request (keyword or data).

3.2. ENCRYPTION INTENSITY SELECTION

The encryption intensity selection in DSB-SEIS is implemented to reduce the amount of encryption keys produced from the filesystem and increase security for confidential data. In other works, two main scenarios are used: one key is generated for all the files, or one key for each file in the filesystem. This feature is proposed to dynamically generate keys based on the confidentiality of a file. The higher the confidentiality, the higher the encryption intensity assigned to it.

The way this feature works is as follows. Two lists of keywords are assigned, one for the medium encryption intensity and another for the high encryption intensity. The user can add keywords to these lists to choose encryption intensity dynamically. While DSB-SEIS scans the file system for data, it uses the two lists to look for the keywords in the file names. An example is shown in Figure 1. If the medium keyword list contains "grades," and the high keyword list contains "health." A file system that contains the files "hello.txt," "groceries.zip," "FS16_grades.pdf," "SP16_grade.pdf," "health_records.txt," and "health_doctors.pdf" is scanned. The intensities assigned to these files respectively are

Algorithm 1: Encryption intensity algorithm

```
1  Initialize two lists of words;
   Input: encryption keywords for medium intensity;
   encryption keywords for high intensity;
2  foreach file scanned in filesystem;
3  do
4      if file path or name contains a keyword from medium intensity list;
5      then
6          assign the file medium encryption intensity;
7      else
8          if file path or name contains a keyword from high intensity list;
9          then
10             assign the file high encryption intesity;
11         else
12             assign the file low encryption entensity;
```

low, low, medium, medium, high, and high. A total of three encryption keys are generated and used. The first two files did not contain any of the keywords and therefore assigned the low intensity by default, the next two contained the keyword "grades", and last two contained the keyword "health." This algorithm is shown in Algorithm 1. The differences between the three intensity levels are as follows:

3.2.1. Low Intensity. This intensity level provides the minimum security to data. One AES key of size 128 bits is used to encrypt all data assigned to this intensity. After data in a file system is scanned, all data is assigned to this intensity by default until the user specifies the encryption intensity.

3.2.2. Medium Intensity. The medium intensity provides slightly more complex security to data by generating an AES key of size 256 bits and encrypting all data assigned to this level using this key. Unlike the low intensity level, this intensity level is not selected by default. The user needs to select the data to be encrypted with this intensity; this is either done manually or using the keyword lists mentioned previously.

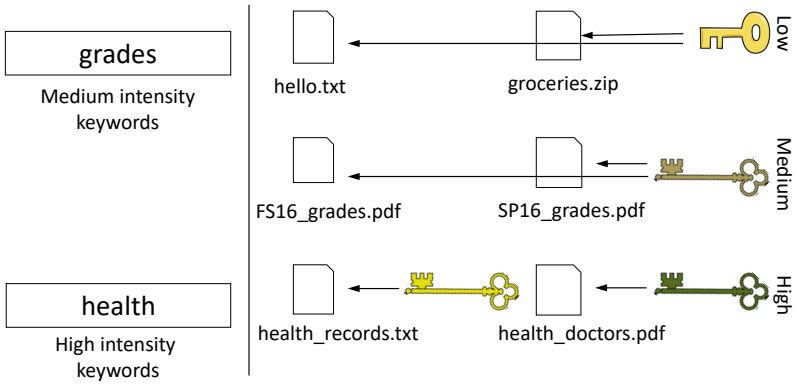


Figure 3.4. Encryption intensity selection example

3.2.3. High Intensity. Using the list of keywords for this intensity, an AES key of size 256 bits is generated for each file containing one or more keywords in its name. This provides further encryption complexity by using different keys for most confidential data and is therefore the highest encryption intensity in the scheme. This also allows for better key management due to the lesser amount of keys generated dynamically. It is important to mention that the protocols used for encryption in this work (AES) can be substituted with any other protocol. The protocols in this work are used to show applicability of the dynamic key generation algorithm.

3.3. DEDUPLICATION

Deduplication is proposed in this scheme in order to provide a better use of network bandwidth and lower the cost of data storage and transmission to and from the cloud. The deduplication algorithm uses the multi-aspect awareness feature that takes into account file semantics and application type, although it remains simple and basic. The aim is

Algorithm 2: Deduplication algorithm

```
1  if no previous scanned files;
2   then
3      initialize an empty list of files to be stored on the cloud;
4  else
5      read the previously scanned list of files;
6  foreach file in filesystem;
7   do
8      foreach file in list of files;
9       do
10         if file sizes are equal AND files are of the same application type;
11          then
12             if file names are equal OR file modification date are equal;
13              then
14                 the files are duplicates;
15         else
16             if file hashes are equal;
17              then
18                 the files are duplicates;
19         if file is previously scanned;
20          then
21             if file modification date is more recent AND hash is not equal;
22              then
23                 file is modified;
24         if file is not a duplicate OR file is modified;
25          then
26             add file to the list to be stored;
```

not to enhance the efficiency of the deduplication algorithm itself, but only for the basic functionality of removing redundant data. The algorithm used is shown in Algorithm 2. The initial step to the algorithm is to read the metadata of previously scanned files, if any. The filesystem is scanned for files. The scanned files are then compared with previously scanned files. The application and file-semantic awareness are first used to check for duplicates. Application awareness checks if the two files are of the same application (e.g., txt), while file semantic awareness checks the size, name, and modification date of the files. If the multi-aspect awareness fails to detect a duplicate, the files are hashed and compared. Additionally, the application checks for modified files and file updates. The modification date and the hash are checked for modification. Finally, the modified files and non-duplicates are added to the list of files to be stored on the cloud. The deduplication algorithm can be improved to achieve better efficiency and accuracy. This is considered for future work.

3.4. ASSURED DELETION

Assured deletion is implemented in this scheme to ensure safe deletion of confidential data. This feature is enabled along with encryption intensity selection. Since different keys are generated for confidential data (high intensity), the encryption keys of data can be simply removed for the data to be assuredly deleted. Since the low and medium encryption intensities use one key for all encrypted data, data encrypted using these intensities cannot be assuredly deleted. However, assured deletion of most confidential data is essential since leakage of this data will have the most impact. Moreover, leakage of less confidential data after deletion requests has much less impact on the user.

This algorithm is not based on policies and is left completely to the user to select when to assuredly delete a file. The algorithm used for assured deletion is shown in Algorithm 3. After the file system has been scanned and data is sent to the cloud, the user can select a file to be assuredly deleted. DSB-SEIS finds all the chunks of data from the file and requests data to be deleted from the cloud. If the file was encrypted with its own

encryption key (hence high intensity level) the key is deleted along with the file's metadata. This way, even if the cloud retains the data on its servers, the data is unintelligible and cannot be read because it cannot be decrypted.

Algorithm 3: Assured deletion algorithm

```
1 select file to be deleted;
2 find chunks of data that make the file;
3 foreach chunk do
4     send delete request to the cloud;
5 if file encrypted using high intensity then
6     delete the encryption key;
7     remove file's metadata;
```

3.5. SEARCHING ENCRYPTED DATA

Storing encrypted data on clouds normally limits the functionality of the data. Hence, the data is useless unless restored and decrypted first, which limits the functionality of storage schemes that encrypt data before sending it to the cloud. For instance, if a user encrypts data, stores it on the cloud, and later would like to retrieve only the files related to their health records without searchable encryption, the user must restore all of the data, decrypt it, and then search. This can be costly and time-consuming. A Searchable Symmetric Encryption (SSE) scheme is implemented in DSB-SEIS that creates an encrypted index for all documents backed up to enable keyword search. Chunk-based duplicate checks is enabled on the cloud using the same SSE scheme. The main features of SSE are the following:

- Key generation: use a secret to generate a private symmetric key
- Generate index: extract the text from documents and file names, then generate an index

- Encryption: use the key generated from the previous property to encrypt the data and generate ciphertext and an encrypted index
- Generate search token: encrypt and then hash the data to generate a token that can be used to search for the data
- Search: use the search token generated previously to search for the data by comparing the token to the hash of the encrypted data
- Decrypt: use the same key generated previously to decrypt the ciphertext to retrieve the plaintext

Both features of the SSE scheme are explained in the following sections.

3.5.1. Keyword Search in Encrypted Index. The SSE properties are used to extract all text from documents and file names. An index that references all words to their respective files is then created. Stop words are removed to only maintain important keywords. Additionally, the same word is only added once to the index. Hence, if a word is repeated multiple times in the same document, it is only referenced to that document once. If it occurs in multiple documents, it is referenced twice, once for each file.

After generating the index, the file pointers and the words are encrypted and hashed. A new key of size 128 bits is generated to encrypt the index to reduce the complexity of encrypting words contained in different files. The encrypted index is then sent to the cloud to be searched later on. The index is stored on disk and read into memory line by line. The assumption here is that each line can be read into memory. Otherwise, the search fails.

Once the index is stored on the cloud, the generated search token property can be used for any word, and sent to the cloud to be searched in the index. The search allows for Boolean expressions to be searched as well. After the search is completed on the cloud, the encrypted file pointers will be returned to the client. The client will decrypt the file pointers to find what files contained the word searched.

3.5.2. Duplicate Checking of Encrypted Data. This feature can be used to check if a copy of the data already exists on the cloud and therefore avoids duplicates (deduplication). Additionally, this feature can be used to check if the data has been modified legitimately or maliciously. If the user has modified the file between the time of storage and duplicate check, then some chunks of data are expected to be different than what is on the cloud. If so, only the modified chunks can be replaced. However, if no changes were made to the file and the cloud fails to find a match, the chunks were maliciously modified.

Similar to searching for keywords, the SSE properties are used to enable this feature. Once the data is encrypted and sent to the cloud, a search token can be generated for the data to be searched by encrypting it and computing the hash. The token can be sent to the cloud to be searched. The cloud hashes all of the encrypted chunks of data sent by the user and stored on its disk and compares them to the token sent by the client. Once the search is complete, the cloud returns the result of the search (either found or not found). The user then decides what needs to be done. If the data has not been modified, no further action is needed. If the data has been modified legitimately, the data can be updated by replacing the modified chunks. If the data has been modified maliciously, the data can be assuredly deleted and completely replaced with a legitimate copy.

3.6. MAPREDUCE APPLICATIONS

In order to test performance of DSB-SEIS with a larger scale of data, three MapReduce applications are developed: document index application, index search application, and disk search application. The WikiMedia dump is acquired [15] and used as input for the MapReduce Applications. The main files used in these experiments are the abstract and article page files. Overall, these applications are only used to evaluate the algorithms used in this thesis work with big data and are not included with the automated DSB-SEIS application (client and cloud applications).

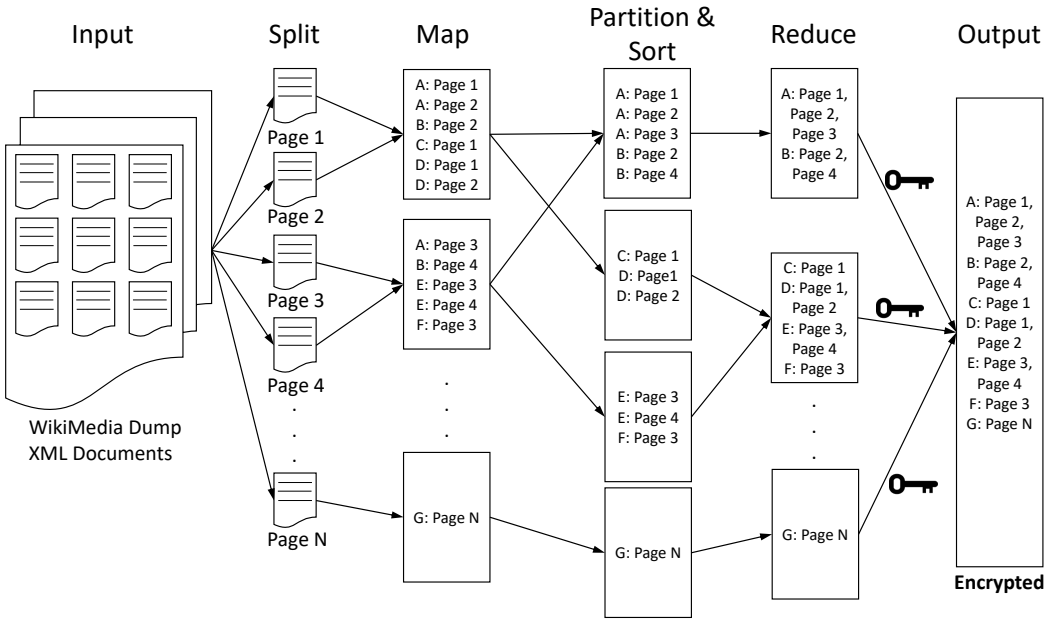


Figure 3.5. MapReduce indexing application workflow

3.6.1. Indexing. Figure 3.5 shows the work flow of the MapReduce index application. MapReduce requires two main interfaces to be implemented: mapper and reducer. In addition, developers are allowed to implement an interface for splitting the input since the input is too large to be read into memory at the same time. By default, MapReduce reads documents line by line and feeds them into the mapper. In the case of this experiment, an input split interface is developed to read one article document at a time. The reason for this is that the WikiMedia dump is in XML structure that contains many documents within it. Each document must be passed whole to the mapper. Since the documents are separated by tags and not newline characters, a custom input split interface must be developed. Once the input split passes a document, the mapper proceeds to extract text from each document, append the name of the page to the word, and pass the key value pair to the next phase. Stopwords are also removed from the text in the mapper. During this phase, memory ends

up running out and data is spilled to disk until it is ready to be processed again. In Figure 3.5, words are represented using letters such as A, B, and C, and documents are represented with page numbers such as page 1, page 2, and page 3.

Once the map phase is over, data is then read from disk, partitioned, and sorted to be passed to different reducers. This is done by hashing each word passed from the mapper, then these hashes are assigned to reducers and key value pairs are passed to the corresponding reducers. In the reducer phase, all data is encrypted using an encryption key, AES128, and passed to the output writer where the index is created. As the amount of data input increases, there will eventually be some entries that do not fit in memory since some words will occur in numerous documents that will exceed the size of memory. For this reason, a custom streaming output writer is developed. This output writer receives the encrypted values from the reducers and streams it to disk without having to store it in memory. This eliminates the possibility of running out of memory at the reducer step, which means that this algorithm can handle large amounts of data with no memory issues. It is important to mention that each reducer will write a chunk of the index. This means that the higher number of reducers used will split the index into more chunks. However, increasing the number of reducers is necessary as the size of data increases since some reducers might be overwhelmed and produce longer processing time. An assumption for this application is that it is run on secure resources such as a private cloud or private machines. The reason for this assumption is that the application needs access to the encryption key to generate an encrypted index.

3.6.2. Index Search. After the indexing application is complete and the index chunks are written to disk, its output is used as input for the keyword search application to find what documents contain certain words. Figure 3.6 shows the work flow of the index keyword search application. However, the index is split into chunks equal to the number of reducers of the index application. Each line in the index represents an entry for a word. Therefore, the default input split is used to read one line at a time. Each line is passed to

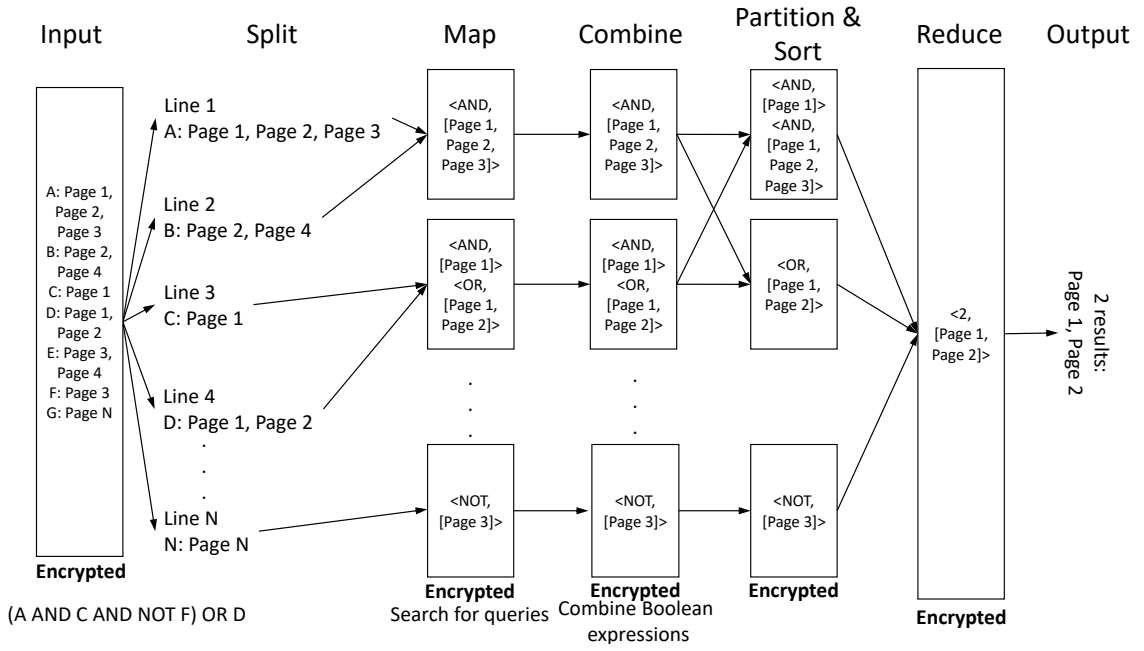


Figure 3.6. MapReduce index search application workflow

a mapper where the words searched for are found and passed to the next phase. Boolean expressions are also supported with this application. The Boolean operation the word is involved in is appended to the result of the entry before being passed to the the next phase. For example, in Figure 3.6, the search operation is (A and C and not F) or D. Since A and C are in an and operation, and is appended to the two entries. Additionally, F is in a not operation, and D is in a or operation. An optional combiner is implemented as the next phase to combine the Boolean operations. This means that the and, not, and or operations will be performed where appropriate in this step. Once combiner passes its value to the next phase, the data is sorted and sent to the reducer. It is important to use only one reducer for this application in order to acquire correct results since the data needs to be in one container to perform the Boolean operations correctly. This is not an issue for most cases since the data searched for will usually not be very large due to the filtering done in the mapper. Hence, only the words searched for are passed to this step. However, once entries are too large to hold in memory, the reducer resorts to write values to disk until they are ready to

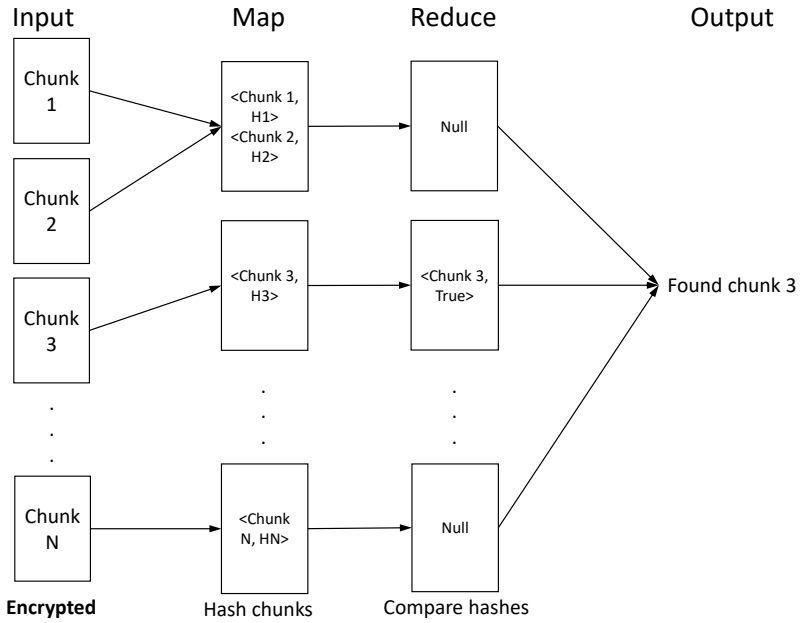


Figure 3.7. MapReduce disk search application workflow

be read again, which can create large overhead and make the application very unstable and unpredictable due to disk I/O overhead. The reducer combines the final computation for the Boolean expressions and writes the final results to disk. This application can be run on a public cloud without sacrificing security since the data is encrypted at any point.

3.6.3. Disk Search. An assumption for this application is that the client already chunked, encrypted, and sent the data to the cloud prior to running this application. It can be run on a public cloud since the disk is contained of encrypted chunks of data with the size of a specific chunk size or smaller. Figure 3.7 shows the work flow of this application. A custom input split is implemented to read whole chunks and pass them to the mappers since the data is split previously and does not need to be split any further. This also simplifies the hashing of chunks at the map phase. In the map phase, the chunks are hashes and then appended to the chunk identifier to create a key value pair. These key value pairs are then passed to reducer where the hashes of the chunks on disk are compared to the chunks searched for. The partition and sort phase is omitted since it does not make much of a

difference in this application. However, similar to the previous applications, the mapper could write data to disk once it runs out of memory. The final result is written to disk after the reducer finishes its computation to display what chunks were found. In Figure 3.7, chunk 3 was searched for and found on disk.

4. RESULTS

To test and evaluate DSB-SEIS, the WikiMedia dump is retrieved [15] and used as the filesystem. Additionally, two virtual machines are created by the Missouri S&T database team for the purpose of testing DSB-SEIS. One of the virtual machines is used as a client, and the other as the cloud. The two machines contain the following resources: Red Hat 4.4.7 running on Linux 2.6.32 operating system, 8 GB of RAM, and 100 GB of storage capacity. The two differ in the CPU; the client machine uses an Intel Xeon E5-2680 v2 running at 2.8 GHz frequency, and the cloud machine runs an Intel Xeon E5-2650 v3 running at 2.3 GHz. Both processors contain 25 MB of cache memory. The client machine connects to the cloud machine in order to run the tests. Both machines are located in Rolla, MO. In the following sections, different metrics of DSB-SEIS are evaluated. The preliminary results for this work are run on the CloudLab testbed [16]. The variables used for testing in this work are shown in Table 4.1. The chunk size is varied to show the effects chunking adds on encryption, decryption, and search. To show the overhead of encryption intensity selection on encryption and decryption, the encryption intensity is varied. For MapReduce applications, the number of reducers is varied. All data points shown in this section are averages of five runs with the 95th percentile confidence interval.

Table 4.1. Testing variables

Variable	**Values**
Chunk size	4 MB, 20 MB, 50 MB, 500 MB
Encryption intensity	low, medium, high, dynamic
Number of reducers	10, 25, 50, 100

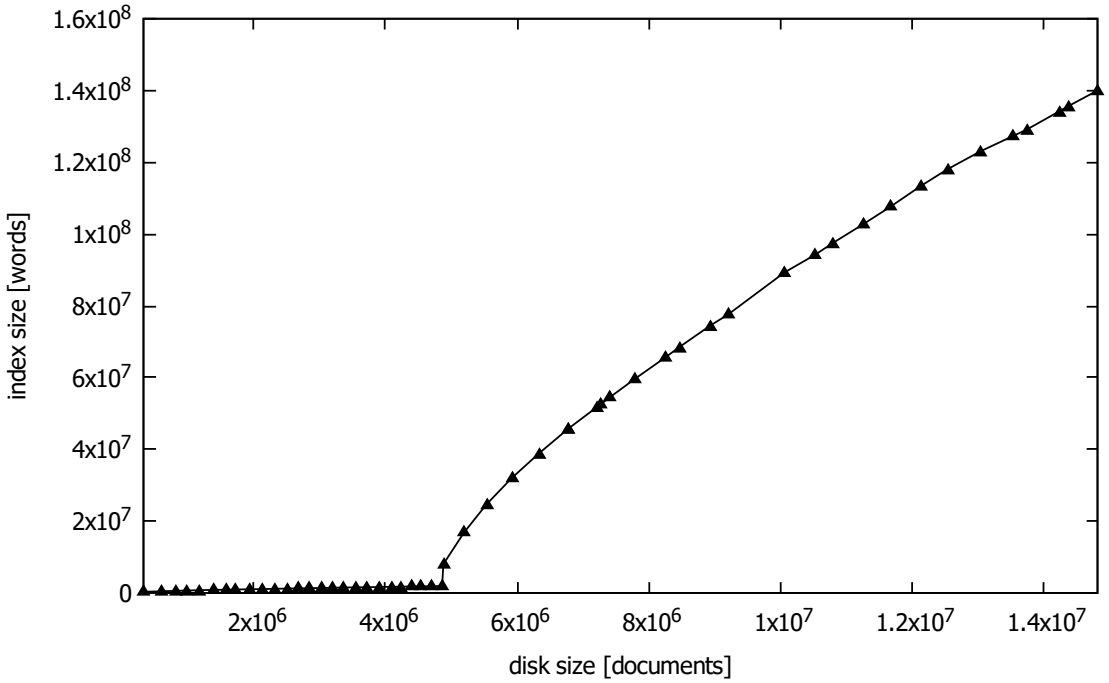


Figure 4.1. WikiMedia word distribution

4.1. WIKIMEDIA DUMP

The latest WikiMedia dump [15] is used as the filesystem for the experiments in this thesis work. The dump consists of all current and archived article data on WikiMedia including abstracts, hyperlinks, body text, and metadata. The dump decompresses to more than five terabytes of data in XML format. Due to space limitations in machines used in this work, these experiments only use a portion of the abstract and body text from the dump. It is important to note that the abstract files are much smaller than the body text files because the abstracts are usually much shorter than the body. This can be seen in Figure 4.1, where the incline starts increasing around five million documents. This point is also the end of the abstract files and start of the body text files.

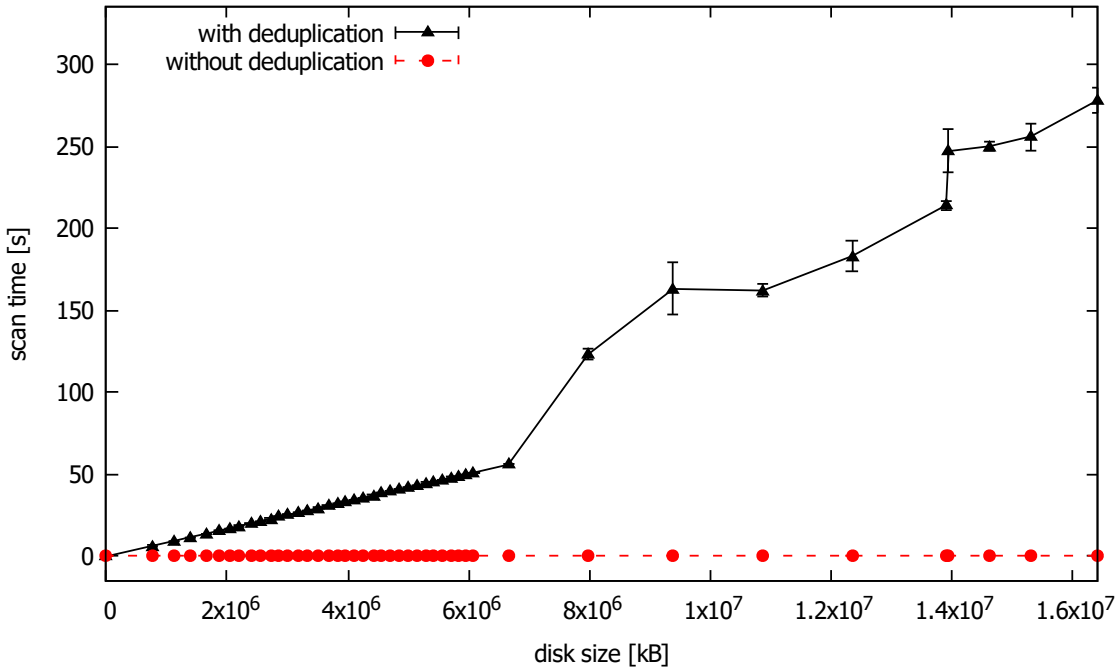


Figure 4.2. Deduplication overhead

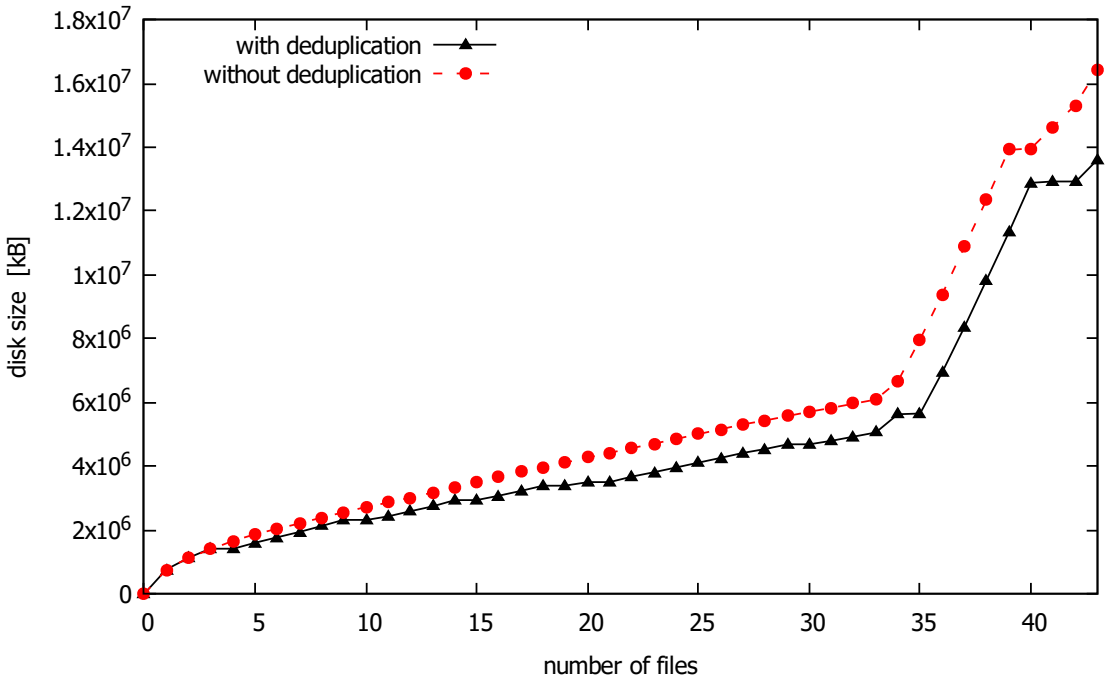


Figure 4.3. Size of data

4.2. DEDUPLICATION PERFORMANCE

In this section, the deduplication overhead is analyzed using two metrics: scan time (time taken to generate a list of files to be sent to the cloud), and the size of data to be transmitted. Figure 4.2 shows the deduplication overhead in scan time. It is important to mention that 42 files were used from the WikiMedia dump including 27 abstract files, seven body text files, and some duplicates. The size difference between abstract files and article files is noticeable in Figure 4.2 at the 6 GB mark, where the size of each file added increases. Scanning the filesystem without deduplication took less than one second across for all files. It is important to note that scan time without deduplication did increase slightly as the size of data increased. However, the incline was so small that the curve appears as a straight line. When deduplication was added, a somewhat linear overhead to the scan time increased to up to around 270 seconds. At first glance, this might be considered a significant increase. However, when considering the amount of redundant data removed from the filesystem, deduplication could still reduce the cost of the service by having less data to handle in the tasks that follow, such as transmission to and from the cloud, indexing, and searching. Figure 4.3 shows the size of data with and without deduplication with each file added. Size of data with deduplication is less than without deduplication by about 2 GB as shown in Figure 4.3. The advantage heavily depends on the amount of redundant data in the filesystem. If there is little to no redundancy in the filesystem, the overhead of deduplication will be larger than the advantage.

4.3. KEY GENERATION ANALYSIS

Key management is very important to avoid data leaks. If a key were to be lost, stolen, or broken, data could be stolen. To make key management easier, the number of keys should be reduced. On the other hand, if a single key is used to encrypt all files in a filesystem containing confidential data, breaking this key means a leak of all data on that

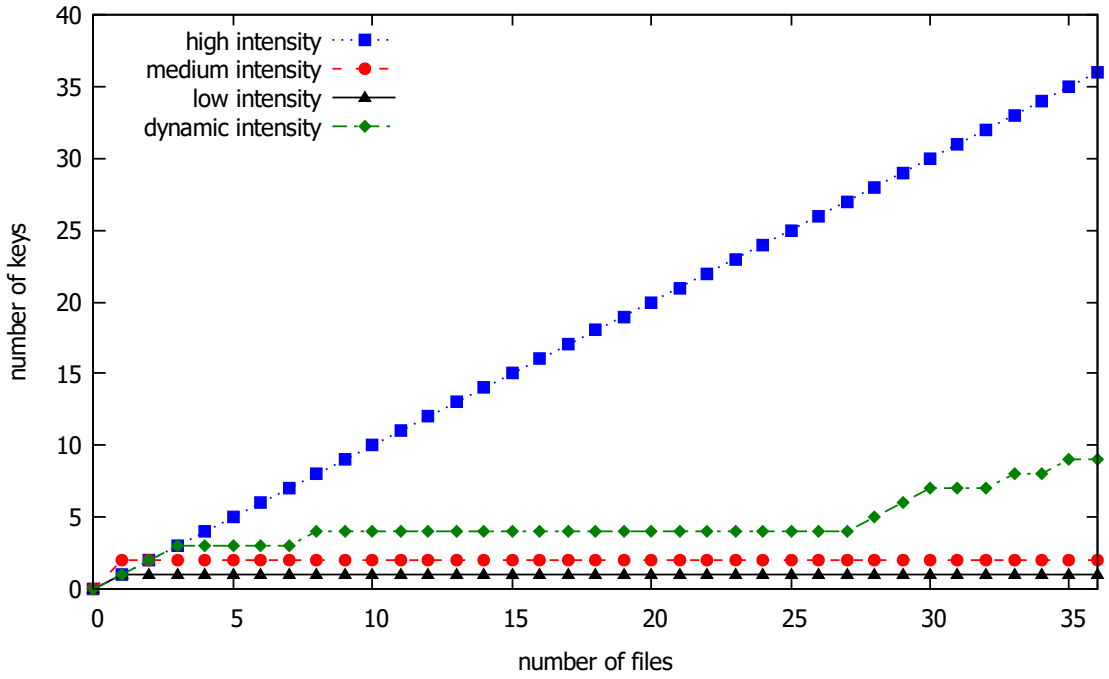
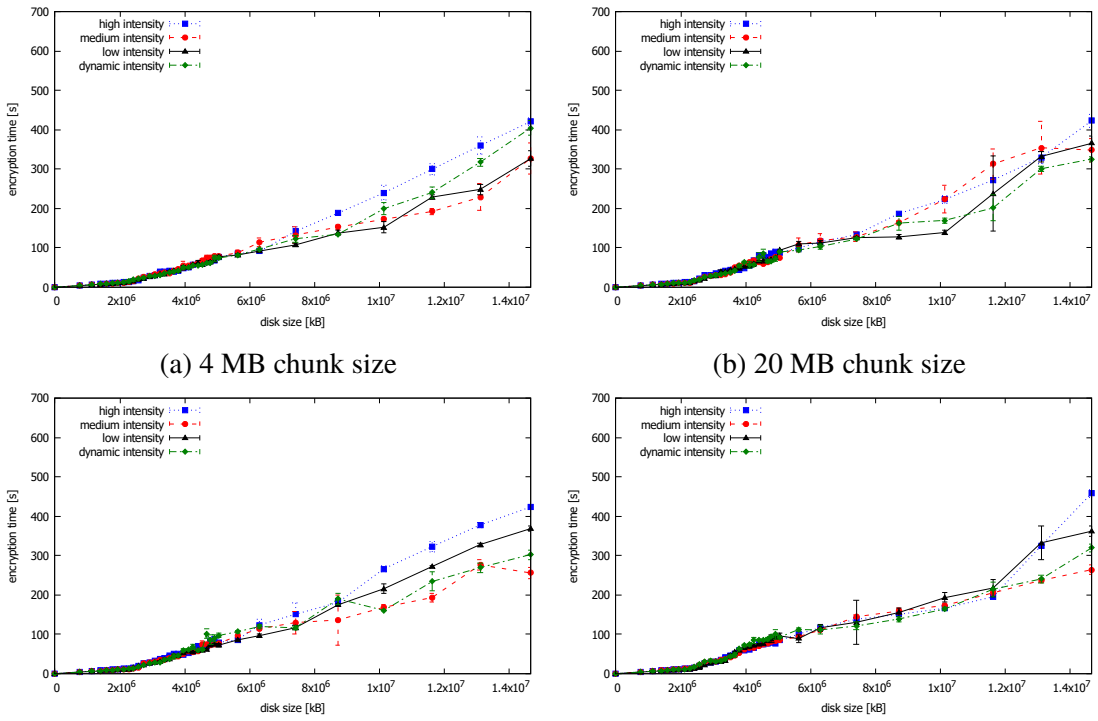


Figure 4.4. Key generation

filesystem. For this reason, the autonomous key generation algorithm proposed in this thesis work is evaluated in this section. Figure 4.4 shows the number of keys generated when all files are assigned to the low, medium, high, and dynamic intensity levels. To make the plot easier to understand, the number of keys in the medium intensity level is set to two since the low intensity key is generated by default even though only the medium intensity key is used to encrypt data. The number of keys generated using the low and medium intensities remain consistent across the plot. When the high intensity is used for all files, a key is generated for each additional file. The number of keys in this case increased linearly. This means that if the filesystem contains a million files, there would be one million keys to manage. Key management can get out of hand easily. In the case of dynamic intensity, the keywords "medium" and "high" were assigned to random files in the filesystem and then passed to the autonomous key generation algorithm. The number of keys increased as the confidential files are added. This reduces the number of encryption keys to be managed when compared to the high intensity while separating the encryption of confidential files. The number



(a) 4 MB chunk size

(b) 20 MB chunk size

(c) 50 MB chunk size

(d) 500 MB chunk size

Figure 4.5. Encryption performance

of keys generated highly depends on the number of confidential files in the filesystem. The worst case scenario for the dynamic intensity is when all files in the filesystem are confidential. In this case, the number of keys would be equal to the number of files, similar to the high intensity. Another case to consider is when all files are non-confidential. The number of keys would be equal to one, similar to the low intensity.

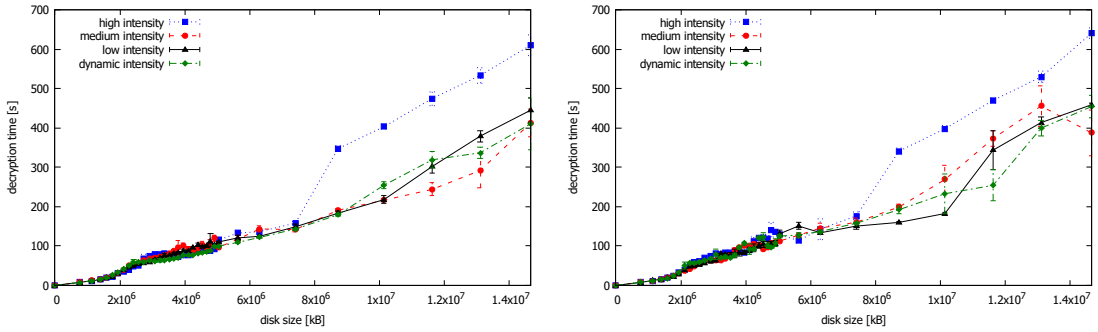
4.4. ENCRYPTION PERFORMANCE

Alongside key generation, the encryption performance is inspected to evaluate the four intensities and observe any overhead added to encryption. Figure 4.5 shows the time taken to encrypt the same files from the key generation section using their assigned keys. Note that the files are chunked prior to encryption. The time taken to chunk the files is included in the encryption time. To inspect the effect of chunk size on encryption, four

chunk sizes are used: 4 MB, 20 MB, 50 MB, and 500 MB. When comparing the encryption performance of the four chunk sizes, no significant overhead is observed for any of the intensities. However, it is observed that a larger chunk size takes slightly longer. On the other hand, the encryption intensity effects are inspected. As expected, the low intensity generally takes the least time, because a smaller key is used. However, the difference between this intensity and the others is not significant. When comparing all four intensities, the performance falls in the same range. This means that dynamic intensity selection does not add significant overhead to encryption. In addition, it makes the number of keys more manageable.

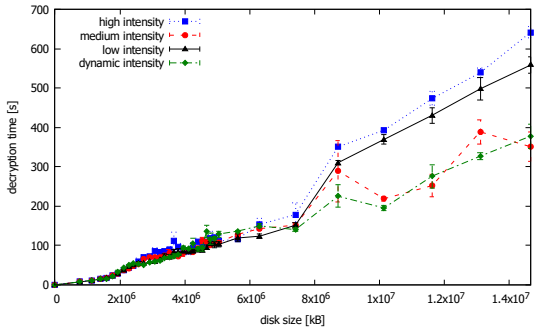
4.5. DECRYPTION PERFORMANCE

To retrieve the data to its original form, it must be decrypted. The decryption performance is evaluated in the same way encryption is. The respective chunks are decrypted and then merged in the correct order to recreate the file. The chunk merge time is included in the decryption time shown in Figure 4.6. The four chunk sizes are used once again: 4 MB, 20 MB, 50 MB, and 500 MB. Similar to encryption, the chunk size does not add significant overhead. Larger chunk size takes slightly longer than smaller size. The low intensity generally shows the least decryption time similar to encryption; however, the difference is not significant. The decryption performance of all intensities falls in the same range. This means that dynamic intensity selection does not add significant overhead to decryption. The most important metric for both encryption and decryption is size of data processed. It is also observed that decryption takes slightly longer than encryption due to the decryption sequential nature.

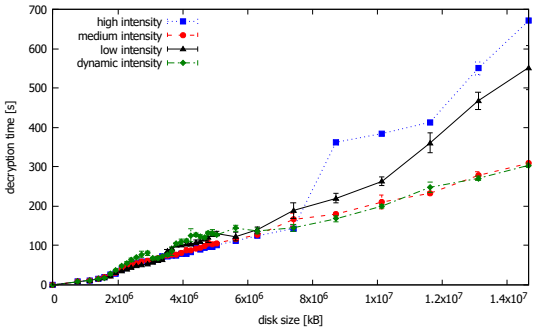


(a) 4 MB chunk size

(b) 20 MB chunk size



(c) 50 MB chunk size



(d) 500 MB chunk size

Figure 4.6. Decryption performance

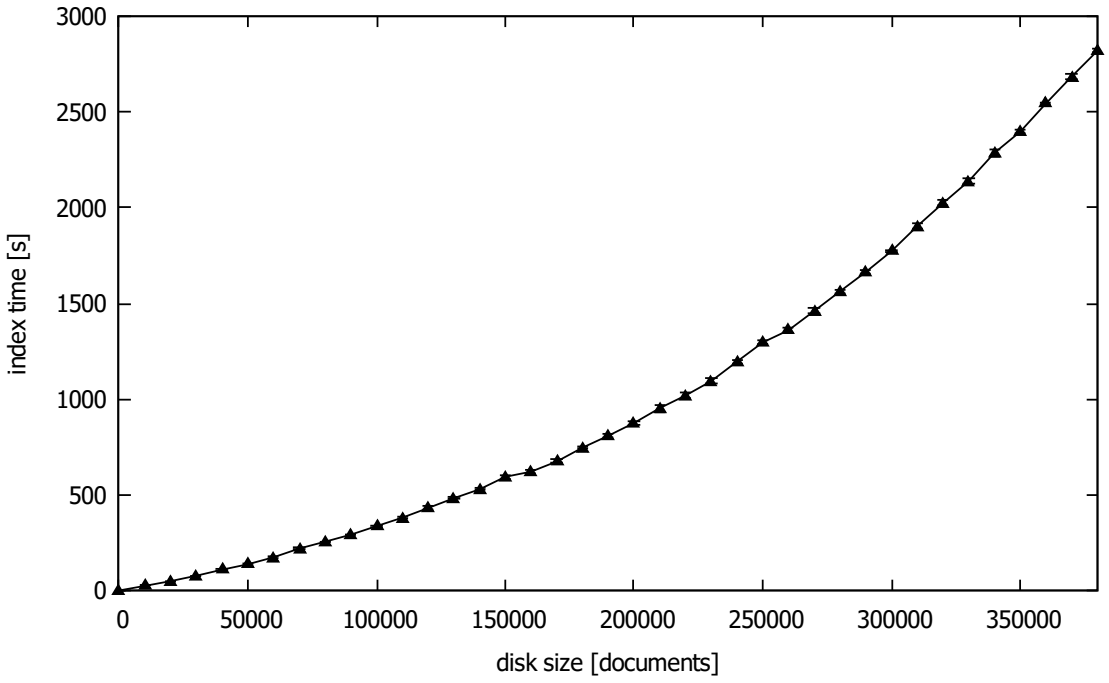


Figure 4.7. Index performance

4.6. INDEXING PERFORMANCE

Figure 4.7 shows the performance of the indexing algorithm as the number of documents increases. It is observed that the time taken to create an encrypted index grows exponentially as the size of data increases. This is due to the time taken to scan the index as new entries added to the index to avoid duplication. As the size of index increases in memory, the time taken to scan the index increases as well, creating the exponential behavior. The data points shown on the figure include time taken to extract text, encrypt words and document pointers, and write index on disk. The drawback of this indexing algorithm is that the index must be stored in memory while creating the index. Once memory runs out, no more data can be indexed until data in memory is spilled to disk. Additionally, since the curve grows exponentially, the cost of the algorithm can become expensive quickly.

4.7. SEARCH PERFORMANCE

The performance of searching for data on disk or an index is inspected and evaluated. Searching on encrypted data requires the client to create a search token by encrypting and then hashing the data to be searched. In addition, the results found from the search must be decrypted to acquire useful information. The time taken to generate a search token, search disk or index, transmit data through a network, and decrypt the results are included in the respective plots in the following sections.

4.7.1. Index Search. After transmitting the index to the cloud, the index is stored on disk to be searched. The client creates search tokens of words and sends them to the cloud along with Boolean operations to return the documents that contain these keywords. The search operation in this work is “fat” and “cat.” Figure 4.8 shows the performance of this search. The data points shown in Figure 4.8 include the time taken to generate a search token, transmit the index, search, and decrypt results. It is observed that the search time grows linearly as the size of the index increases. This is expected, as the time taken to read

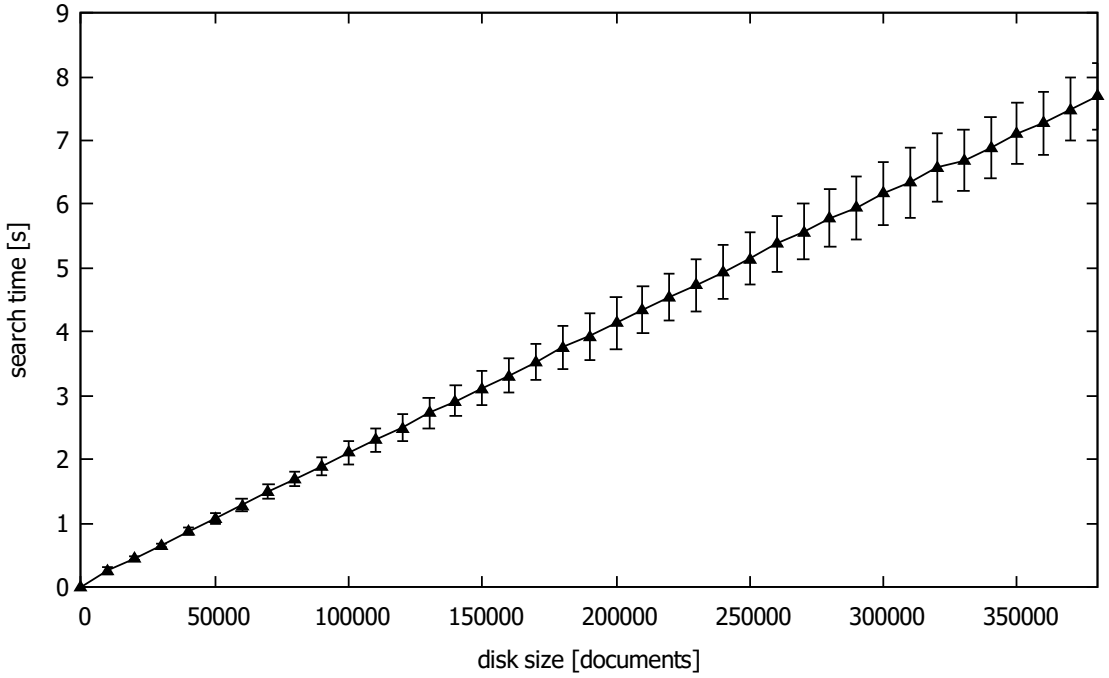


Figure 4.8. Index search performance

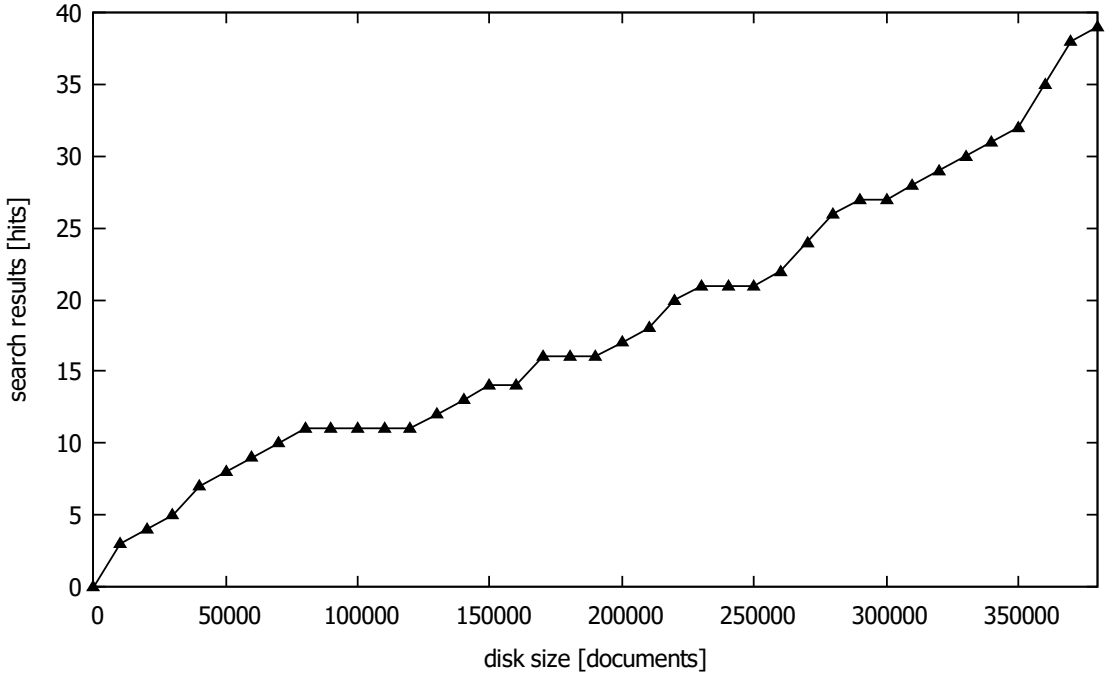


Figure 4.9. Search hits

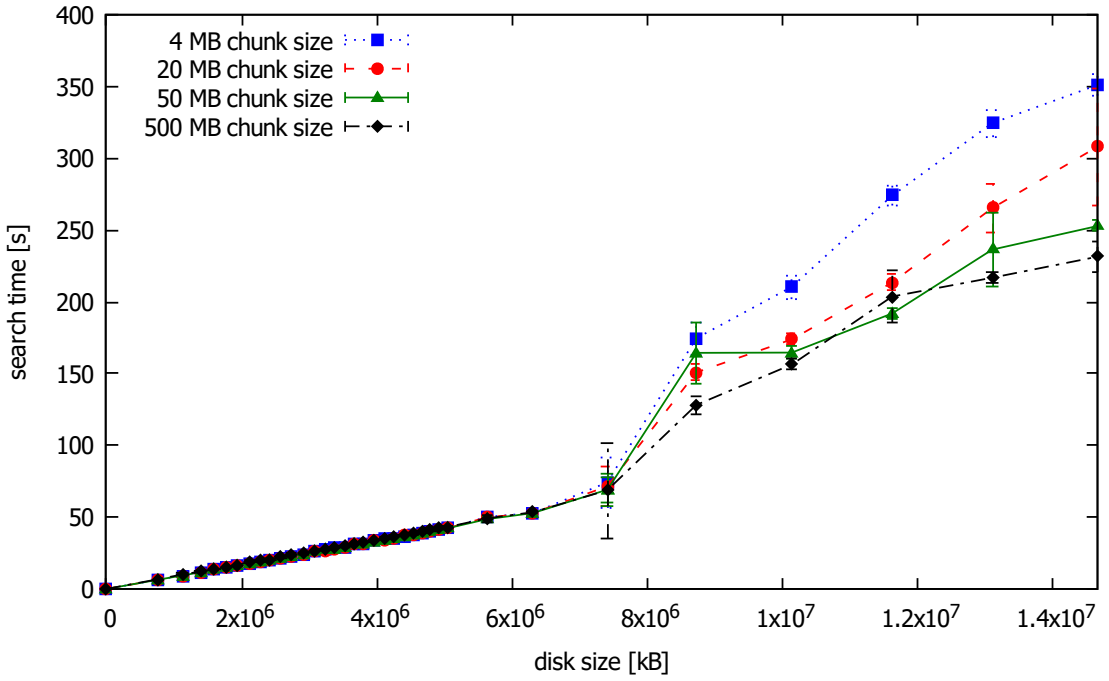


Figure 4.10. Disk search performance

the index from disk grows. However, as the size of the index grows, the confidence intervals become larger as well. The main reason for this error is the overhead accessing the disk adds. The confidence intervals can be decreased by storing the index in memory. However, this limits the maximum size of the index. Another solution is to use a different data structure, such as tree structure, to save the index. The security of the indexing algorithm can be improved by adding randomization. However, this can increase the complexity of storing the index, and decrease the performance due to adding randomization when creating the index, and remove the randomization when decrypting the results. Overall, the encrypted index search eliminates the cost of retrieving and decrypting data in order to search.

4.7.2. Disk Search. Searching the cloud disk for a chunk or a piece of data can significantly reduce the cost of finding the required data by not having to retrieve and decrypt all data from the cloud. Figure 4.10 shows the disk search performance, including the time taken to generate a search token from the data, transmit the token and results, and decrypt results. Four chunk sizes are used to show the effect of chunk size on the search

performance because a larger chunk size will take longer to encrypt. It is observed from the plot that the difference found from larger chunk size is minimal and insignificant for the most part. However, a larger chunk size takes a slightly shorter to complete. Initially, it was believed that a smaller chunk size would add a much larger overhead to the search. However, this is not the outcome shown in the figure. It is believed that explanation for this is that the disk is accessed only once at the same time to find the required data. Therefore, disk I/O is not overloaded, which leads to searching all data on disk at about the same time for all chunk sizes.

4.8. MAPREDUCE APPLICATIONS

To test the MapReduce applications, the Missouri S&T database team set up a four-node cluster and a single node machine containing Hadoop. The index and index search applications are run on the cluster, and the disk search application is run the single node machine to evaluate the algorithms using a larger scale of data. Originally, all three applications were supposed to be run on both the cluster and the single node to compare performance. Due to time limitation, it is decided to compare performance with different cluster sizes as future work. The WikiMedia dump is used once again, but in this case, more data is able to be processed. The cluster machines contain the following resources: Red Hat 4.4.7 running on Linux 2.6.32, 8 GB of RAM, 490 GB of storage space. Two of the cluster machines run an Intel Xeon E5-2680 v2 at 2.8 GHz frequency, and the other two run an Intel Xeon E5-2683 v3 at 2.0 GHz frequency. The single node machine contains the same operating system and amount of RAM, but it contained an Intel Xeon E5-2680 v2 at 2.8 GHz frequency, and 1.4 TB of storage space.

4.8.1. Indexing Performance. For the index application, four numbers of reducers are used to evaluate the indexing algorithm and observe the performance: 10 reducers, 25 reducers, 50 reducers, and 100 reducers. Figure 4.11 shows the performance of the MapReduce indexing application using the four reducer values. In the case of 100 reducers,

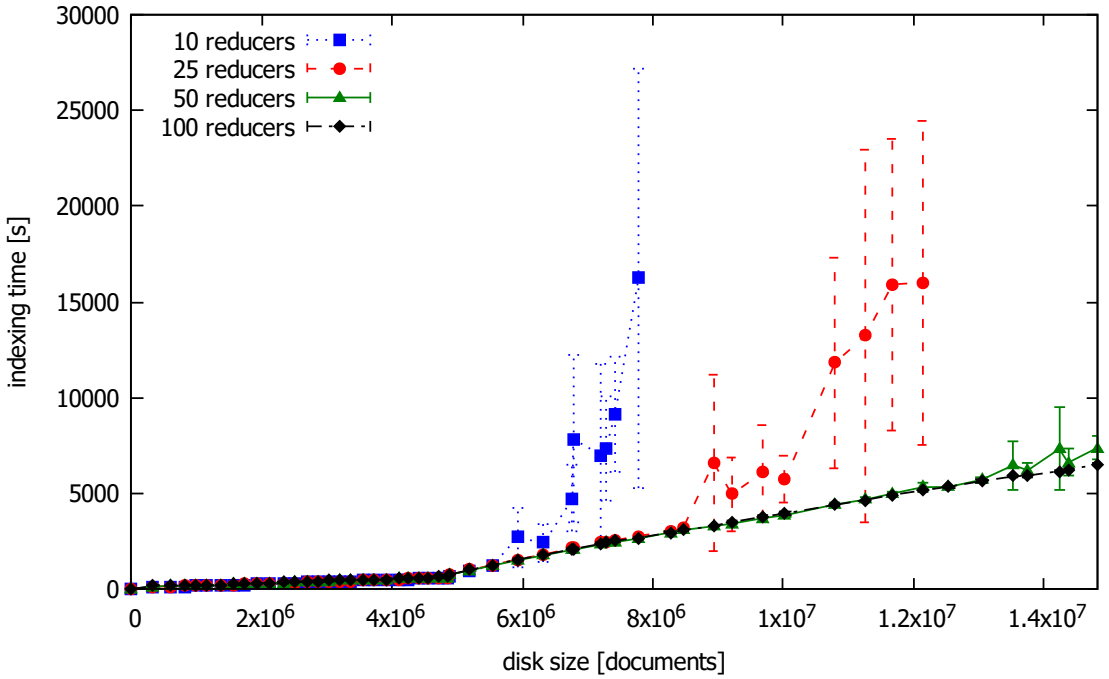


Figure 4.11. MapReduce indexing performance

the index time remains consistent and increases linearly as the size of data increases. As for the 50 reducers, the index time increases linearly as the size of data increases until it reaches the 12 million documents mark. At this point, the performance starts to be unstable and unpredictable. It is observed that the time taken for the reducers to finish is much longer than the other phases. The time taken for the reducers to finish is also unstable, while the other phases remains somewhat stable. This means that the reducers are overwhelmed with data, and the sort function is assigning uneven amounts of data to different reducers, resulting in unstable behavior. This can be seen again at the unstable points for the 25 and 10 reducers. The unstable behavior can ultimately result in failure of execution if the reducers cannot handle the amount of data. This can be seen at the 8 million document point for the 10 reducer curve, and 12 million document for the 25 reducer curve. At these points, the reducers can no longer handle the large uneven amount of data and eventually fail. It is important to mention that the algorithm used in this work contains a streaming output for the reducer to eliminate the issue of running out of memory. This means that if

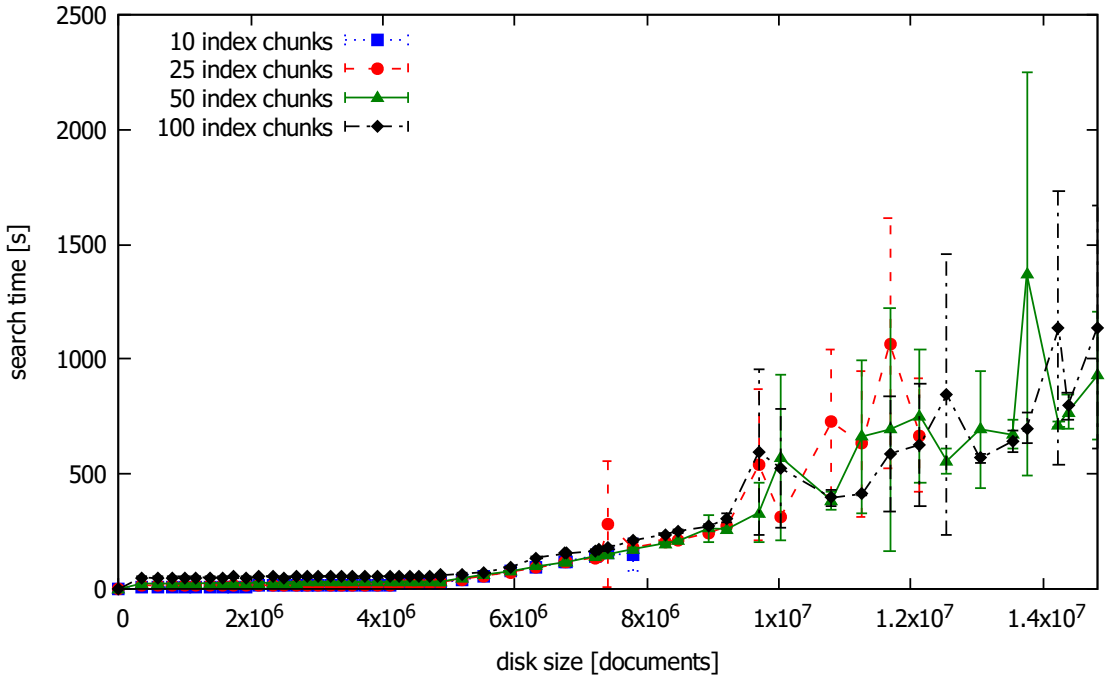


Figure 4.12. MapReduce index search performance

the data is evenly distributed, all reducers should finish around the same time. However, if that is not the case, the bottleneck of the application will be the reducer that is overwhelmed with data. On the other hand, when considering the performance of a lesser amount of data, it is observed that a smaller number of reducers performed slightly better. This means that selecting an optimal number of reducers is crucial to obtain the best performance. The number of reducers should be selected based on the size of data to be processed. Compared with Figure 4.7, it be observed that MapReduce can enhance the performance of generating an encrypted index, assuming that the number of reducers is set optimally.

4.8.2. Index Search Performance. After the index application has finished, the index is written to disk in chunks equal to the number of reducers used in the index application. Hence, if 100 reducers are used, the index is split into 100 chunks. The output of the index application is used as input for the search application, resulting in four values for the number of index chunks: 10, 25, 50, and 100 chunks. Since the 25 and 10 reducer curves failed in the previous application at 8 and 12 million document, respectively, no

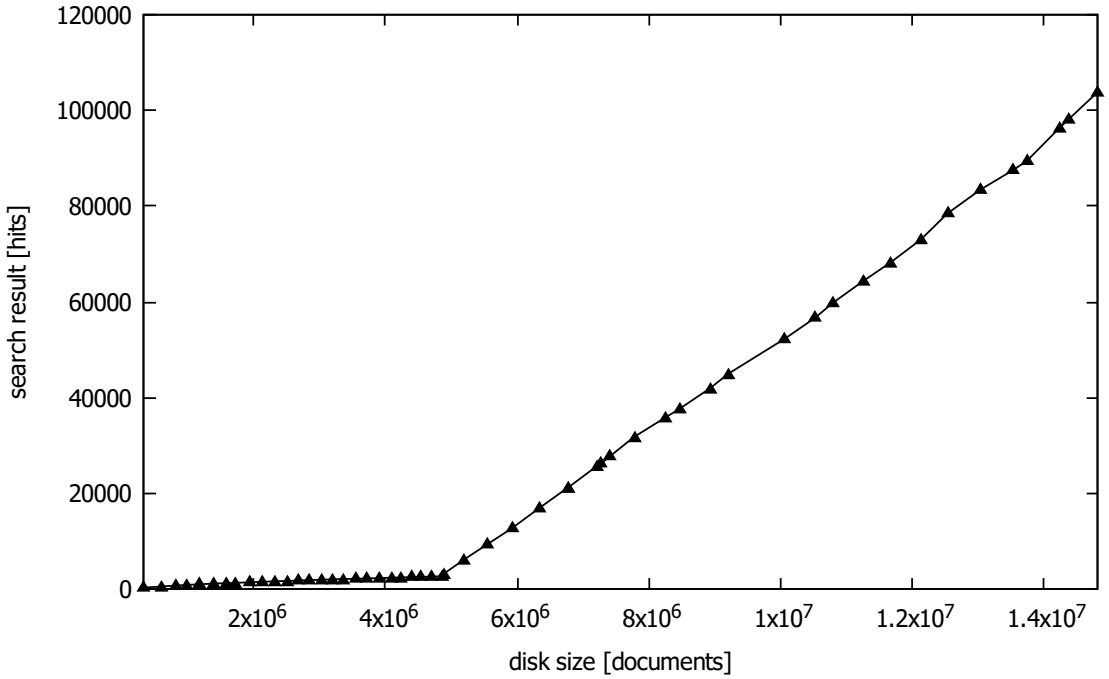


Figure 4.13. Index search hits

search data is obtained for the points past these points. The search operation for this search is “fat” and “cat”. Figure 4.12 shows the performance of index search using the three metrics. All curves act similarly until the 10 million document mark, where the performance became unstable. Prior to this point, the search time remains consistent and increases linearly for all four curves. The lesser number of index chunks perform slightly better due to the lesser disk I/O overhead. An important thing to observe is that the number of index chunks does not affect the unstable point for the curves. After further investigation of this problem, it is observed that this is highly related to the amount of data searched for. For example, when searching for a word that occurs in a large number of documents, the size of this entry in the index will be large as well. Hence, if the size of the entry is larger than the size of main memory, MapReduce will resort to accessing the disk to temporarily store this data until needed once again. MapReduce will have to access the disk again once the data is needed for processing. This will continue to happen until MapReduce is finished executing the Boolean operations and the search overall. The number of disk accesses

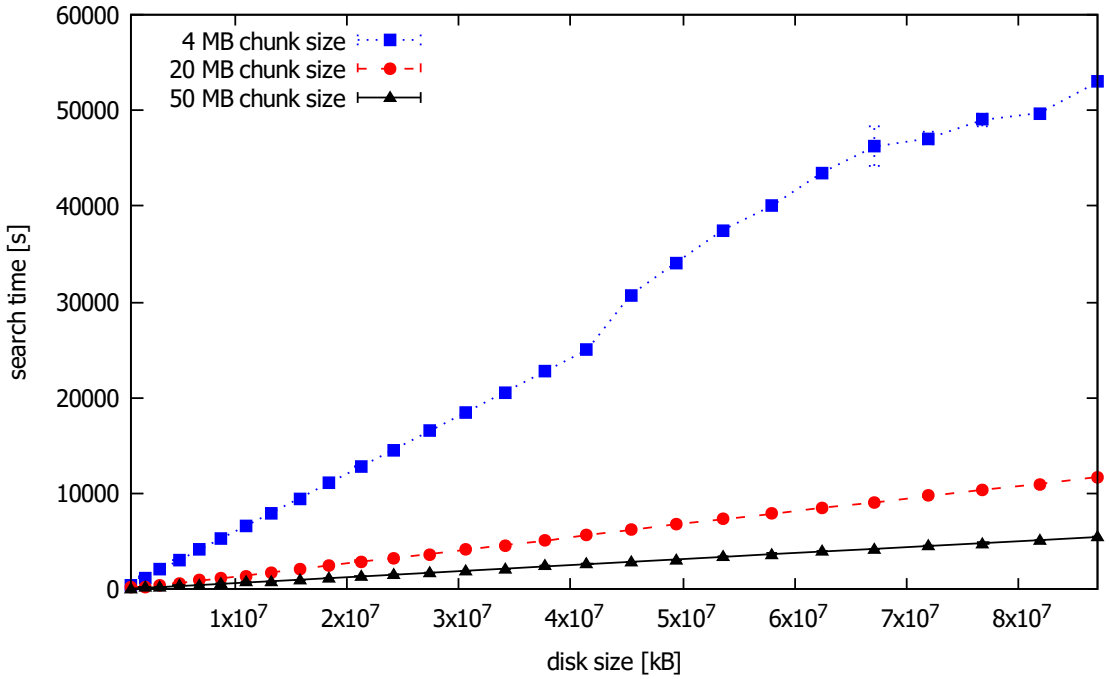


Figure 4.14. MapReduce disk search performance

needed by MapReduce is not consistent, which creates the unstable behavior in the curves. Figure 4.13 shows the number of hits found by the search application. MapReduce can perform as well or worse depending on the overhead of disk I/O, memory size, and index size when compared with Figure 4.8.

4.8.3. Disk Search Performance. To check if a chunk of data exists on the cloud disk, it is assumed that the disk contains encrypted chunks of data prior to running this application. The chunks could be encrypted using any intensity and can be equal to or less than the chunk size. Three chunk sizes used to evaluate this application are 4 MB, 20 MB, and 50 MB. Figure 4.14 shows the performance of disk search using MapReduce. At first glance, it is observed that the chunk size creates a large difference in performance using MapReduce. The 50 MB chunk size performed the best, while the 4 MB chunk size performed the worst. When compared to the other disk search application that does not use MapReduce, a large difference in performance is observed. The non-MapReduce application is not affected by chunk size, while the MapReduce application is. After

investigation, it is found that using the custom input format that reads whole chunks as input creates this overhead. Since a smaller chunk size generates a larger number of chunks, the input format assigns a single mapper for each chunk, resulting in a larger number of mappers to complete. This adds overhead to the performance, making smaller chunk sizes perform worse. Hence, disk search using MapReduce works best with a larger chunk size. The performance can be improved if the input of the application is changed to a list of hashes instead of chunks stored on disk. This would eliminate the overhead of creating a single mapper for each chunk by replacing the input with one large file containing the hashes of the chunks. However, this would require the cloud to compute the hashes regularly to ensure that the list is up-to-date.

5. CONCLUSIONS

Cloud computing provides many great services to users that can reduce the cost of purchasing and maintaining resources. Some of the great services provided by cloud computing is data storage and computation. Along with the benefits of these services come some issues that worry most users. Cost of the service, security of data, and functionality of stored data are some of these issues. Users often resort to encrypting data prior to outsourcing it to the cloud. Although this increases the security of data, it also limits the functionality of the cloud by not allowing for computation since the data is encrypted and unintelligible. Additionally, this encryption is often constricted by not allowing the user to control the key generation. In this thesis work, the service of data storage is analyzed and evaluated by developing a functional scheme called DSB-SEIS that allows users to autonomously generate encryption keys for confidential data to separate the encryption from less confidential data. The scheme also utilizes searchable symmetric encryption to allow for searching encrypted data without having to retrieve or decrypt data. The scheme also contains features widely used by service providers, such as deduplication, to evaluate the service. The algorithms used in this scheme are also developed as MapReduce applications to evaluate the performance of the algorithms with larger scale of data.

To evaluate DSB-SEIS, the WikiMedia dump is used to generate results to evaluate features in the scheme, including deduplication, number of keys generated, encryption, decryption, indexing, and searching. The indexing and searching features are also evaluated using MapReduce. Deduplication adds overhead related to the size of data to scan time due to the time needed to hash data and compare multiple aspects of the files. However, assuming that duplicate data exists in the filesystem, deduplication reduces the amount of data to be transmitted and processed, which could ultimately reduce the cost of the service. Although if no duplicates exist, the drawback of deduplication exceeds its benefit.

The number of keys generated by the autonomous key generation, encryption intensity selection, is more manageable. This feature generated keys for more confidential files and separated the encryption from less-confidential data. This feature does not add much overhead to encryption or decryption. However, no noticeable overhead is added by using a single key for each file, either. Therefore, the main advantage of encryption intensity selection is generating a more manageable number of keys. Chunking data to different sizes also does not add much overhead to encryption or decryption. Indexing performance is observed to become expensive quickly. This performance can be enhanced by using the MapReduce application developed in this work. Searching the index is observed to take longer linearly as the size of data increases. Searching the disk for a chunk of data works successfully with encryption intensity selection. Chunking data to different sizes does not add noticeable overhead to searching the disk. However, when this algorithm is tested with MapReduce, it is observed that a larger chunk size results in higher performance due to the lesser number of mappers created by the chunks. The indexing algorithm is also tested using MapReduce. It is observed that when an insufficient number of reducers is used, the reducers are overwhelmed and the performance becomes unstable and eventually fails. Therefore, selecting a sufficient number of reducers is essential for higher performance. After the indexing application is complete, the output is used as input for the index search application. It is found that the performance of this search becomes unstable when the size of the results exceeds the size of memory, resulting in high disk I/O overhead.

The service of data storage on the cloud is beneficial to customers when resource purchase and maintenance is a concern. However, to keep the cost at a minimum, the trade-offs of features used in the service must be considered. Deduplication works best when duplicate data exists in the filesystem. Encryption intensity selection has a good potential to generate a manageable amount of keys while increasing the security of confidential data, with no noticeable overhead. Indexing and searching for keywords eliminates the need for retrieving and decrypting data in order to search. Searching the disk has the advantage of

enabling the user to check if a chunk of data exists on the cloud without having to decrypt or retrieve data. Finally, MapReduce has the potential of processing a larger scale of data if the number of reducers is selected optimally. Additionally, searching an index using MapReduce can be improved by splitting the index to prevent entries from exceeding the size of memory.

6. FUTURE WORK

In this section, drawbacks of the thesis work are discussed, and future work to tackle these drawback are proposed. First of all, DSB-SEIS does not allow for mobility of data. Since the encryption keys are stored on the client machine at all times, encrypted data cannot be retrieved and decrypted on any other machine, and therefore the mobility of data is highly constricted. A possible solution for this is to allow for password-based key generation. If a password is used to generate encryption keys in each intensity level, the keys can be recreated on different machines, therefore allowing the decryption of data on trusted machines that have DSB-SEIS installed. However, this can be less secure than generating completely random keys due to the possibility of having the passwords stolen. This also add the complexity of memorizing the passwords needed to generate the required key, which makes it much easier to lose keys by forgetting passwords. Another possible solution to this drawback is to enable transmitting keys from machine to machine through a secure channel. This can also solve another drawback of DSB-SEIS, data sharing across multiple users. Since encryption intensity selection generates different keys for confidential data, these keys can be shared with other authorized users to enable data sharing. For instance, if a group in a company is working on a project that needs everyone to retrieve the data, a key can be generated using autonomous key generation and then shared securely with the rest of the group. These solutions will be analyzed in future work.

Additionally, adding randomization to the encrypted index algorithm is considered for future work. Currently, encrypted indexes generated by DSB-SEIS do not add randomization to entries. This means that the document pointers will be exactly the same across the index. This also means that if the index is split into chunks, encrypting the same word will generate the same output. This enables attackers to launch attacks such as adaptive chosen keyword attacks [58, 59] to gain unauthorized information about data. To avoid

these attacks, randomization must be added to words and document pointers to make the encrypted data appear completely random. However, this requires the randomization to be saved on disk in order to retrieve the original data. Other index data structures effects on performance, such as trees, can also be investigated as part of future work.

In future work, MapReduce applications developed in this work are further investigated and analyzed. The indexing application performance is tested with an even larger scale of data extending to terabytes to find its capabilities of handling big data. The index search application is tested after splitting entries in the index that do not fit in memory and adding randomization to the index. The disk search application is tested after changing the input from chunks of data stored on disk to a list of hashes of chunks generated prior to launching the application. Performance of applications are tested with different cluster sizes as part of future work as well.

APPENDIX

DSB-SEIS CODE

DSBSEIS.java contains the driver code for the client application of DSB-SEIS. It displays the menu to the user, reads input, and executes the requested command. Deduplication.java contains functions to find duplication in a filesystem. KeywordDictionary.java contains a custom implementation of a hash map that is used for an index. Cryptography functions such as hashing, encryption, and decryption functions are defined in Cryptography.java. ProcessedFile.java contains a custom class that stores information about files processed in DSB-SEIS. Receiver.java contains functions that read data from a connected socket. Scanner.java contains functions that scan a directory for files and generate a list of files to be processed. Utilities.java contains supporting functions for the application. Page.java is a class defined for the documents extracted from WikiMedia XML files. AbstractHandler.java and PageHandler.java contain implementations of classes that handle the extraction of WikiMedia abstracts and body text documents. PageProcessor.java contains an interface to process and store the documents extracted from the WikiMedia XML files. DSB-SEIServer.java is the implementation of the cloud application for DSB-SEIS. XMLInput.java is the driver for the Hadoop MapReduce Indexing application. TextArrayWritable.java is class implementation that allows Hadoop MapReduce to pass arrays as values between interfaces. XMLInputFormat.java is an implementation of a custom input format for Hadoop MapReduce that reads XML files. XMLMapper.java is an implementation of a mapper for the indexing application. XMLReducer.java is the reducer for the indexing application. StreamingTextOutputFormat.java is a custom output format that streams the index to disk without having to save it in memory. HadoopSearchIndex.java is the driver class for the Hadoop MapReduce index search application. HadoopSearchMap-

per.java is the mapper for the index search application. HadoopSearchCombiner.java is the combiner for the index search application. HadoopSearchReducer.java is the reducer for the index search application. HashFiles.java is the driver class for the Hadoop MapReduce disk search application. WholeFileInputFormat.java is a custom input format to read and pass a whole file as input. HashFilesMapper.java is the mapper for the disk search application. HashFilesReducer.java is the reducer for the disk search application.

```
/*
 * DSBSEIS.java
 */
package dsbseis;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.nio.ByteBuffer;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.List;


public class DSBSEIS {
    // client application driver function
```

```
public static void main(String[] args) throws IOException,
   ParseException, Exception {
   int CHUNKSIZE = 1024 * 1024;
   Scanner scanner = new Scanner();
   int choice, choice2;
   Sender sender = new Sender();
   Socket sock = sender.Connect();
   DataInputStream inChannel = new DataInputStream(sock.
      getInputStream());
   DataOutputStream outChannel = new DataOutputStream(sock.
      getOutputStream());
   Receiver receiver = new Receiver();
   BufferedReader buffer = new BufferedReader(new
      InputStreamReader(System.in));
   String keyword;
   String tok;

   do {
      System.out.println("_____________________ DSB-SEIS
         ______________________________");
      System.out.println("Please choose an option: ");
      System.out.println("\t 1. Scan a directory for files to
         backed up");
      System.out.println("\t 2. Select encryption intensity for
         files");
      System.out.println("\t 3. Start backup");
```

```
System.out.println("\t 4. Parse text from files and send
   to the cloud");
System.out.println("\t 5. Request a keyword search from
   the cloud");
System.out.println("\t 6. Request a file duplicate check")
   ;
System.out.println("\t 7. Restore a specific or all files
   ");
System.out.println("\t 8. Request a file to be deleted");
System.out.println("\t 9. Exit");

choice = Integer.parseInt(buffer.readLine());
switch (choice) {
   case 1:
      System.out.print("Please enter the directory you
         would like to scan: \n");
      String path = buffer.readLine();
      System.out.print("\n_________________ " + path + "
         _____________\n\n");
      scanner.ScanDirectory(path);
      break;
   case 2:
      System.out.println("1. Automatic selection using
         keywords");
      System.out.println("2. Specify an intensity for
         individual files");
      choice2 = Integer.parseInt(buffer.readLine());
```

```
switch (choice2) {
    case 1:
        System.out.println("1. Medium Intensity");
        System.out.println("2. High Intensity");
        choice2 = Integer.parseInt(buffer.readLine()
           );
        switch (choice2) {
            case 1:
                System.out.println("Please enter a
                   keyword you would like to be added
                    to the medium intensity");
                keyword = buffer.readLine();
                scanner.SetIntensityKeywords(keyword,
                    2);
                break;
            case 2:
                System.out.println("Please enter a
                   keyword you would like to be added
                    to the high intensity");
                keyword = buffer.readLine();
                scanner.SetIntensityKeywords(keyword,
                    3);
                break;
        }
        break;
    case 2:
```

```
            System.out.println("Select the file you
               would like to intensify: ");
            for(int i = 1; i <= scanner.files.size(); i
               ++) {
               System.out.println("\t" + i + ". " +
                  scanner.files.get(i-1).Name);
            }
            int selection = Integer.parseInt(buffer.
               readLine());
            if(selection > scanner.files.size() ||
               selection <= 0)
               System.out.println("Incorrect input");
            else {
               System.out.println("Choose a intensity: (
                  Low - 1, Medium - 2, High - 3)");
               int selection2 = Integer.parseInt(buffer.
                  readLine());
               if(selection2 > 3 || selection2 <= 0) {
                  System.out.println("Incorrect input")
                     ;
               }
               else
                  scanner.files.get(selection - 1).
                     chooseKey(selection2);
            }
            break;
         default:
```

```
                System.out.println("ERROR: INCORRECT OPTION
                    ");
                break;
            }
            break;
        case 3:
            int currentFile = 0;
            System.out.println("--- Backing up " + scanner.
                files.size() + " files ---");
            for (ProcessedFile file : scanner.files) {
                if ((file.filePath == null))
                    continue;
                file.setChunks(Utilities.ChunkFile(file.filePath
                    , CHUNKSIZE));
                file.setNumChunks(file.chunks.length);
                List<String> encryptedChunks = new ArrayList<>()
                    ;
                for (String chunk : file.chunks) {
                    encryptedChunks.add(Cryptography.encryptFile
                        ("Data/" + chunk, file.EncryptionKey));
                }
                String[] chunks = new String[encryptedChunks.
                    size()];
                chunks = encryptedChunks.toArray(chunks);
                file.setChunks(chunks);
                for(String chunk : file.chunks) {
```

```
                sender.SendFile("Encrypted/" + chunk,
                    outChannel, 1);
            }
            Utilities.progressBar(currentFile++, scanner.
                files.size());
        }
        scanner.writeFiles();
        break;
    case 4:
        KeywordDictionary dic = new KeywordDictionary();
        int currentFileWord = 0;
        String[] contents;
        for (ProcessedFile file : scanner.files) {
            contents = Utilities.ExtractWords(file.filePath)
                ;
            String fileName = Utilities.BytesToString(
                Cryptography.encryptString(file.Name, "Keys/
                Level1key.dat"));
            for(String k : contents) {
                tok = Utilities.BytesToString(Cryptography.
                    GenerateStringToken(k, "Keys/Level1key.
                    dat"));
                dic.Add(tok, fileName);
            }
            Utilities.progressBar(currentFileWord++, scanner
                .files.size());
        }
```

```
            try(PrintWriter buffWriter = new PrintWriter(new
               FileWriter("index/index", false))){
               for(String key : dic.words.keySet()) {
                  buffWriter.print(key + ",");
                  for(String val : dic.words.get(key)) {
                     buffWriter.print(val + ",");
                  }
                  buffWriter.println();
               }
               dic.words.clear();
            }
            System.out.println("Transmitting keywords");
            byte[] com = ByteBuffer.allocate(4).putInt(2).array
               ();
            outChannel.write(com, 0, 4);
            sender.SendFile("index/index", outChannel, 1);
            break;
         case 5:
            System.out.println("Please enter the word to search
                for: ");
            String query = buffer.readLine();
            int type = 1;
            String[] words = query.split("/s+");
            com = ByteBuffer.allocate(4).putInt(3).array();
            outChannel.write(com, 0, 4);
            com = ByteBuffer.allocate(4).putInt(words.length).
               array();
```

```
outChannel.write(com, 0, 4);
for(String word : words) {
    if(word.startsWith("!")) {
        type = 1;
        word = word.replaceAll("!", "").toLowerCase
            ();
    }
    else if(word.startsWith("|")){
        type = 2;
        word = word.replaceAll("/|", "").toLowerCase
            ();
        System.out.println(word);
    }
    else { type = 3;
          word = word.toLowerCase();
    }
    tok = Utilities.BytesToString(Cryptography.
        GenerateStringToken(word, "Keys/Level1key.dat
        "));
    outChannel.writeInt(type);
    sender.SendString(tok, outChannel);
}
int numFile = receiver.ReceiveInt(inChannel);
System.out.println("Results: " + numFile);
byte[][] found = new byte[numFile][];
for (int i = 0; i < numFile; i++) {
    found[i] = receiver.ReceiveBytes(inChannel);
```

```
                System.out.println("Encrypted file name received
                    from the cloud: " + new String(found[i], "
                   UTF-8"));
            }
            for (byte[] res : found) {
                byte[] temp = Utilities.StringToBytes(Utilities.
                   BytesToASCII(res));
                System.out.println(Cryptography.decryptString(
                   temp, "Keys/Level1key.dat"));
            }
            break;
        case 6:
            System.out.println("Enter the name of the file you
               would like to check if it exists on the cloud:
               ");
            String fToC = buffer.readLine();
            String[] chuns = scanner.findFile(fToC).chunks;
            byte[] hash;
            for (String chu : chuns) {
                hash = Cryptography.hashFile("Encrypted/" + chu,
                    "SHA-256");
                com = ByteBuffer.allocate(4).putInt(4).array();
                outChannel.write(com, 0, 4);
                sender.SendBytes(hash, outChannel);
                int dup = receiver.ReceiveInt(inChannel);
                if (dup == 1) {
```

```
                System.out.println("The chunk " + chu + "
                    exists on the cloud");
            }
            else {
                System.out.println("The chunk" + chu + "
                    does not exist on the cloud");
            }
        }
        break;
    case 7:
        System.out.println("Enter the name of the file you
            would like to retrieve: ");
        String fToR = buffer.readLine();
        ProcessedFile file = scanner.findFile(fToR);
        String[] chunks = file.chunks;
        for(String ch : chunks) {
            com = ByteBuffer.allocate(4).putInt(5).array();
            outChannel.write(com, 0, 4);
            sender.SendString(ch, outChannel);
            receiver.ReceiveFile(inChannel);
            Cryptography.decryptFile("DSBase/" + ch, "Keys/
                Level1key.dat");
        }
        Utilities.GroupFileChunks(chunks, fToR + "encrypted
            ");
        break;
    case 8:
```

```
                System.out.println("Enter the name of the file you
                   would like to delete: ");
                String fToD = buffer.readLine();
                String[] fs = scanner.AssuredDeletion(fToD);
                for (String f : fs) {
                    com = ByteBuffer.allocate(4).putInt(6).array();
                    outChannel.write(com, 0, 4);
                    sender.SendString(f, outChannel);
                }
                break;
            case 9:
                sender.Shutdown(sock);
                System.out.println("Thank you for using DSB-SEIS");
                break;
            default:
                System.out.println("Error: The option selected is
                   not valid");
                break;
          }
       } while (choice != 9);
   }
}

/*
 * Deduplication.java
 */
package dsbseis;
```

```
import java.util.Date;

public class Deduplication {
    // detects duplicates
    public static boolean IsDuplicate(String name1, String name2,
        long size1, long size2, Date date1, Date date2, String hash1,
        String hash2) {
        if ((CompareFileName(name1, name2) && CompareSize(size1,
            size2)) ||
            (CompareSize(size1, size2) && CompareDateOfModification(
                date1, date2)))
            return true;
        else return CompareHash(hash1, hash2);
    }

    public static boolean CompareFileName(String filename1, String
        filename2) {
        return filename1 == null ? filename2 == null : filename1.
            equals(filename2);
    }

    public static boolean CompareSize(long size1, long size2) {
        return size1 == size2;
    }

    public static boolean CompareDateOfModification(Date date1, Date
        date2) {
```

```
        return date1.compareTo(date2) == 0;
    }

    public static boolean CompareHash(String hash1, String hash2){
        return hash1 == null ? hash2 == null : hash1.equals(hash2);
    }
}

/*
 * KeywordDictionary.java
 */
package dsbseis;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class KeywordDictionary {
    // map to store index in memory
    public Map<String, List<String>> words = new HashMap<>();

    // scans the index to avoid duplicates
    public void Add(String key, String value) {
        if(words.containsKey(key)) {
            if (!words.get(key).contains(value)) {
                words.get(key).add(value);
            }
```

```
        }
        else {
            List<String> files = new ArrayList<>();
            files.add(value);
            words.put(key, files);
        }
    }
}

/*
 * Cryptography.java
 */
package dsbseis;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
```

```
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.lang.RandomStringUtils;

// cryptography functions
public class Cryptography {
    // constructor for cryptography class
    public Cryptography() throws IOException,
        UnsupportedEncodingException, NoSuchAlgorithmException{
        generateKey(1);
        generateKey(2);
    }
    // test function that encypts and decrypts a string
    public void test() {
        String plaintext = "Hello There";
        String KeyPath = "Keys/Level1key.dat";
        try {
            byte[] cipher = encryptString(plaintext, KeyPath);
            System.out.print("cipher: ");
            for (int i=0; i<cipher.length; i++)
                System.out.print(new Integer(cipher[i])+":");
            System.out.println("");
            System.out.println("decrypt: " + decryptString(cipher,
                KeyPath));
```

```
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}

// returns a cipher java object for AES
public Cipher getCipher(){
    try {
        return Cipher.getInstance("AES/CBC/PKCS5PADDING", "SunJCE
            ");
    } catch (NoSuchAlgorithmException | NoSuchProviderException |
         NoSuchPaddingException ex) {
        Logger.getLogger(Cryptography.class.getName()).log(Level.
            SEVERE, null, ex);
    }
    return null;
}

// encrypts a file and using an AES key stored on disk
public static String encryptFile(String path, String keyPath)
    throws FileNotFoundException, IOException {
    String encryptedName = RandomStringUtils.randomAlphanumeric
        (8) + "." + RandomStringUtils.randomAlphanumeric(3);
    File directory = new File("Encrypted/");
    directory.mkdir();
    byte[] buffer = new byte[1048576];
    int readBytes = 0;
```

```
    try (FileInputStream stream = new FileInputStream(path);
       FileOutputStream outStream = new FileOutputStream("
       Encrypted/" + encryptedName);){
       SecretKey KEY = readKey(keyPath);
       IvParameterSpec IV = readIV(keyPath);
       Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING")
          ;
       cipher.init(Cipher.ENCRYPT_MODE, KEY, IV);
       while((readBytes = stream.read(buffer)) > 0)
       {
          byte[] encBuffer = new byte[readBytes];
          System.arraycopy(buffer, 0, encBuffer, 0, readBytes);
          byte[] tmp = cipher.update(encBuffer);
          outStream.write(tmp);
       }
       byte[] finalBuffer = cipher.doFinal();
       if(finalBuffer != null)
          outStream.write(finalBuffer);
       outStream.flush();
    } catch(Exception e) {
       System.err.println(e.getMessage());
    }
    return encryptedName;
}

// decrypts a file using an AES key stored on disk
```

```
public static void decryptFile(String path, String KeyPath)
    throws FileNotFoundException, IOException {
    byte[] buffer = new byte[1048576];
    int readBytes = 0;
    try(FileInputStream stream = new FileInputStream(path);
        FileOutputStream outStream = new FileOutputStream("
            decrypted/" + new File(path).getName());) {
        SecretKey KEY = readKey(KeyPath);
        IvParameterSpec IV = readIV(KeyPath);
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING")
            ;
        cipher.init(Cipher.DECRYPT_MODE, KEY, IV);
        try(CipherInputStream cin = new CipherInputStream(stream,
            cipher)){
            while((readBytes = cin.read(buffer)) > 0) {
                byte[] encBuffer = new byte[readBytes];
                System.arraycopy(buffer, 0, encBuffer, 0, readBytes
                    );
                outStream.write(encBuffer);
            }
            outStream.flush();
        }
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

```
// hashes a file stored on disk
public static byte[] hashFile(String filePath, String Algorithm)
   throws NoSuchAlgorithmException, IOException
{
   MessageDigest md = MessageDigest.getInstance(Algorithm);
   byte[] checksum = new byte[1024];
   int read = 0;
   try (FileInputStream stream = new FileInputStream(filePath))
      {
          while((read = stream.read(checksum)) != -1){
             md.update(checksum, 0, read);
          }
          byte[] checksumFinal = md.digest();
          return checksumFinal;
   }
}

// hashes a byte array
public static byte[] hashBytes(byte[] data) throws
   NoSuchAlgorithmException {
   MessageDigest digest = MessageDigest.getInstance("SHA-256");
   byte[] hash = digest.digest(data);
   return hash;
}

// encrypts a string using a key stored on disk
public static byte[] encryptString(String plain, String KeyPath){
```

```
    try {
        SecretKey KEY = readKey(KeyPath);
        IvParameterSpec IV = readIV(KeyPath);
        byte[] cipher = encrypt(plain.getBytes(), KEY, IV);
        return cipher;
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
    return null;
}

// decrypts a string using a key stored on disk
public static String decryptString(byte[] cipher, String KeyPath)
    {
    try {
        SecretKey KEY = readKey(KeyPath);
        IvParameterSpec IV = readIV(KeyPath);
        byte[] decrypted = decrypt(cipher, KEY, IV);
        String decipher = new String(decrypted);
        return decipher;
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
    return "";
}

// generates a key for encryption intensity selection
```

```
public static void generateKey(int Intensity) throws
   UnsupportedEncodingException, NoSuchAlgorithmException,
   IOException
{
   int keySize = 128;
   String keyPath = "";
   switch (Intensity) {
      case 1:
         keySize = 128;
         keyPath = "Keys/Level1key.dat";
         break;
      case 2:
         keySize = 256;
         keyPath = "Keys/Level2key.dat";
         break;
      case 3:
         keySize = 256;
         keyPath = "Keys/" + "Level3key.dat";
         break;
      default:
         break;
   }
   KeyGenerator keyGen = KeyGenerator.getInstance("AES");
   keyGen.init(keySize);
   byte[] IV = generateIV();
   saveKey(keyPath, keyGen.generateKey(), IV);
}
```

```
// generate a level 3 key for encryption intensity selection
public static void generateKey(int Intensity, String
   fileToBeEncrypted) throws UnsupportedEncodingException,
   NoSuchAlgorithmException, IOException
{
   int keySize = 128;
   String keyPath = "";
   switch (Intensity) {
      case 1:
         keySize = 128;
         keyPath = "Keys/Level1key.dat";
         break;
      case 2:
         keySize = 256;
         keyPath = "Keys/Level2key.dat";
         break;
      case 3:
         keySize = 256;
         keyPath = "Keys/" + fileToBeEncrypted + "Level3key.dat
            ";
         break;
      default:
         break;
   }
   KeyGenerator keyGen = KeyGenerator.getInstance("AES");
   keyGen.init(keySize);
```

```
    byte[] IV = generateIV();
    saveKey(keyPath, keyGen.generateKey(), IV);
}

// generate an initialization vector in memory
private static byte[] generateIV() {
    byte[] b = new byte[16];
    new SecureRandom().nextBytes(b);
    return b;
}

// read a key from disk
public static SecretKey readKey(String path) throws
   FileNotFoundException, IOException{
    SecretKey KEY;
    int keySize = (int)new File(path).length()- 16;
    try (FileInputStream stream = new FileInputStream(path)){
        byte[] key = new byte[keySize];
        stream.read(key, 0, keySize);
        KEY = new SecretKeySpec(key, 0, keySize, "AES");
    }
    return KEY;
}
// reads an initialization vector from disk
public static IvParameterSpec readIV(String path) throws
   FileNotFoundException, IOException{
    IvParameterSpec IV;
```

```
    int keySize = (int)new File(path).length()- 16;
    try (FileInputStream stream = new FileInputStream(path)) {
        byte[] iv = new byte[16];
        stream.read(new byte[keySize], 0, keySize);
        stream.read(iv, 0, 16);
        IV = new IvParameterSpec(iv);
    }
    return IV;
}

// saves a key to disk
public static void saveKey(String path, SecretKey key, byte[] IV)
     throws IOException {
    FileOutputStream stream = new FileOutputStream(path);
    try {
        stream.write(key.getEncoded());
        stream.write(IV);
    } finally {
        stream.close();
    }
}

// encrypts byte array with key in memory
private static byte[] encrypt(byte[] plainText, SecretKey Key,
    IvParameterSpec iv) throws Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
```

```
    return cipher.doFinal(plainText);
}

// decrypts byte array with key in memory
private static byte[] decrypt(byte[] cipherText, SecretKey Key,
    IvParameterSpec iv) throws Exception{
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.DECRYPT_MODE, Key , iv);
    return cipher.doFinal(cipherText);
}

// generates a search token from byte array using a key stored on
     disk
public static byte[] GenerateToken(byte[] keyword, String KeyPath
    ) throws Exception {
    SecretKey KEY = readKey(KeyPath);
    IvParameterSpec IV = readIV(KeyPath);
    byte[] EncBytes = encrypt(keyword, KEY, IV);
    return hashBytes(EncBytes);
}

// generates a search token from string using a key stored on
    disk
public static byte[] GenerateStringToken(String keyword, String
    KeyPath) throws UnsupportedEncodingException, Exception
{
    byte[] keywordBytes = keyword.getBytes("US-ASCII");
```

```
        return GenerateToken(keywordBytes, KeyPath);
    }

}

/*
 * ProcessedFile.java
 */
package dsbseis;

import java.io.File;
import java.io.IOException;
import java.security.NoSuchAlgorithmException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;

// data structure to store metadata of a file
public class ProcessedFile {
    public String Name;
    public String Hash;
    public long Size;
    public Date ModTime;
    public String EncryptionKey;
    public int chunkNum;
```

```
public String filePath;
public String[] chunks;
public List<String> keywords = new ArrayList<>();

public void setName(String name) {
    Name = name;
}

public void setHash(String hash) {
    Hash = hash;
}

public void setSize(String size) {
    Size = Long.parseLong(size);
}

public void setDate(String time) throws ParseException {
    ModTime = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy
       ").parse(time);
}

public void setDate(long time) {
    ModTime = new Date(time);
}

public void setKey(String key) {
    EncryptionKey = key;
```

```
}

public void setNumChunks(String number) {
    chunkNum = Integer.parseInt(number);
}

public void setNumChunks(int number) {
    chunkNum = number;
}

public void setChunks(String[] c) {
    chunks = c;
}

public void setPath(String path) {
    filePath = path;
}

public void chooseKey(int intensity) throws
   NoSuchAlgorithmException, IOException {
    switch(intensity)
    {
        case 1:
            if(new File("Keys/Level1key.dat").exists()) {
                EncryptionKey = "Keys/Level1key.dat";
            }
            else {
```

```
                Cryptography.generateKey(1);
                EncryptionKey = "Keys/Level1key.dat";
            }
            break;
        case 2:
            if (new File("Keys/Level2key.dat").exists()) {
                EncryptionKey = "Keys/Level2key.dat";
            }
            else {
                Cryptography.generateKey(2);
                EncryptionKey = "Keys/Level2key.dat";
            }
            break;
        case 3:
            if (new File("Keys/" + Name + "Level3key.dat").exists()
                ) {
                EncryptionKey = "Keys/" + Name + "Level3key.dat";
            }
            else {
                Cryptography.generateKey(3, Name);
                EncryptionKey = "Keys/" + Name + "Level3key.dat";
            }
            break;
    }
}

public void setKeywords(String[] words) {
```

```
        keywords.addAll(Arrays.asList(words));
    }
}

/*
 * Receiver.java
 */
package dsbseis;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;

// receives data through a socket
public class Receiver {
    // integer
    public int ReceiveInt(DataInputStream netStream) throws
        IOException {
        byte[] array = new byte[4];
        netStream.read(array, 0, 4);
        int i = ByteBuffer.wrap(array).getInt();
        return i;
    }

    // long integer
```

```
public long ReceiveLong(DataInputStream netStream) throws
    IOException {
    byte[] array = new byte[8];
    netStream.read(array, 0, 8);
    long l = ByteBuffer.wrap(array).getLong();
    return l;
}

// byte array
public byte[] ReceiveBytes(DataInputStream netStream) throws
    IOException {
    byte[] h;
    int length = ReceiveInt(netStream);
    h = new byte[length];
    netStream.read(h, 0, length);

    return h;
}

// string
public String ReceiveString(DataInputStream netStream) throws
    IOException {
    String s;
    int length = ReceiveInt(netStream);
    byte[] sBytes = new byte[length];
    netStream.read(sBytes, 0, length);
    s = Utilities.BytesToASCII(sBytes);
```

```
    return s;
}

// file
public String ReceiveFile(DataInputStream netStream) throws
    IOException {
    int length = ReceiveInt(netStream);
    byte[] filenamebytes = new byte[length];
    netStream.read(filenamebytes, 0, length);
    String filename = new String(filenamebytes);
    System.out.println("Receiving " + filename);

    long fileSize = ReceiveLong(netStream);
    new File("DSBase/").mkdirs();
    try (FileOutputStream output = new FileOutputStream("DSBase/"
        + filename)) {
        byte[] buffer = new byte[1024];
        // Read the incoming stream
        int bytesRead;
        long totalBytes = 0;
        while (totalBytes != fileSize) {
            int remaining = 1024;
            if((fileSize - totalBytes) < 1024)
            {
                remaining = (int)(fileSize - totalBytes);
            }
            bytesRead = netStream.read(buffer, 0, remaining);
```

```
                output.write(buffer, 0, bytesRead);
                totalBytes += bytesRead;
                Utilities.progressBar(totalBytes, fileSize);
            }
            output.close();
        }
        catch(Exception ex) {
            System.out.println(ex.getMessage());
        }
        return filename;
    }
}

/*
 * Scanner.java
 */
package dsbseis;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.security.NoSuchAlgorithmException;
```

```
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;


public class Scanner {
    // functions that scan disk to process files

    public List<ProcessedFile> files;

    // lists of keywords of intensity keywords
    public List<String> mediumKeywords;
    public List<String> highKeywords;

    // constructor that reads previosuly stored metadata
    public Scanner() throws IOException, ParseException {
        files = new ArrayList<>();
        mediumKeywords = new ArrayList<>();
        highKeywords = new ArrayList<>();
        if(new File("files.csv").exists()) {
            try(BufferedReader reader = new BufferedReader(new
                FileReader("files.csv"))) {
                String line;
                while((line = reader.readLine()) != null) {
                    String[] values = line.split (",");
                    ProcessedFile newFile = new ProcessedFile();
```

```
                newFile.setName(values[0]);
                newFile.setDate(values[1]);
                newFile.setSize(values[2]);
                newFile.setHash(values[3]);
                newFile.setKey(values[4]);
                newFile.setNumChunks(values[5]);
                List<String> chunks = new ArrayList<>();
                for(int i = 0; i < newFile.chunkNum; i++) {
                    chunks.add(values[6 + i]);
                }
                String[] chunksArray = new String[newFile.chunkNum
                   ];
                chunksArray = chunks.toArray(chunksArray);
                newFile.setChunks(chunksArray);
                files.add(newFile);
            }
        }
    }
}

// scans a path and generates a list of files
static String[] ScanPath(Path directory) throws IOException {
    String[] res;
    List<String> files = new ArrayList<>();
    try (DirectoryStream<Path> ds = Files.newDirectoryStream(
       directory)) {
        for (Path child : ds) {
```

```
            if (!Files.isDirectory(child)) {
                files.add(child.toAbsolutePath().toString());
            }
        }
    }
    res = new String[files.size()];
    res = files.toArray(res);
    return res;
}

// scans a directory with deduplication and stores the metadata
    in memory
public void ScanDirectory(String path) {
    try {
        for (String file : new File(path).list()) {
            File f = new File(path + "/" + file);
            if(f.isDirectory()){ ScanDirectory(f.toString());}
            else {
                boolean duplicate = false;
                byte[] hashToCompare = Cryptography.hashFile(path +
                    "/" + file, "SHA-256");
                if (files.size() > 0) {
                    for (int i = 0; i < files.size(); i++) {
                        ProcessedFile temp = files.get(i);
```

```
                duplicate = Deduplication.IsDuplicate(f.
                    getName(), temp.Name, f.length(), temp.
                    Size, new Date(f.lastModified()), temp.
                    ModTime, Utilities.BytesToString(
                    hashToCompare), temp.Hash);
                if (Deduplication.CompareFileName(f.getName
                    (), temp.Name) && (new Date(f.
                    lastModified()).compareTo(temp.ModTime) >
                     0 && !Deduplication.CompareHash(
                    Utilities.BytesToString(hashToCompare),
                    temp.Hash))) {
                    files.remove(i);
                    duplicate = false;
                }
                if (duplicate == true) {
                    files.get(i).setPath(f.getCanonicalPath()
                        );
                    break;
                }
            }
        }
        if (duplicate == false) {
            ProcessedFile newFile = new ProcessedFile();
                                newFile.setName(f.getName()
                                    );
```

```
                                newFile.setHash(Utilities.
                                    BytesToString(
                                    hashToCompare));
                                newFile.setSize(String.
                                    valueOf(f.length()));
                                newFile.setDate(f.
                                    lastModified());
                boolean notDefault = false;
                for(String keyword : mediumKeywords) {
                    if (file.contains(keyword)) {
                        newFile.chooseKey(2);
                        notDefault = true;
                    }
                }
                for(String keyword : highKeywords) {
                    if(file.contains(keyword)) {
                        newFile.chooseKey(3);
                        notDefault = true;
                    }
                }
                if(notDefault == false) {
                    newFile.chooseKey(1);
                }
                newFile.setPath(path + "/" + file);
                files.add(newFile);
            }
        }
```

```
        }
    }
    catch(IOException | NoSuchAlgorithmException ex) {
        System.out.println(ex.getMessage());
    }
}

// scans directory without deduplication and stores metadata in
   memory
public void ScanDirectoryND(String Path) throws
   NoSuchAlgorithmException, IOException, ParseException {
   for (String file : new File(Path).list()) {
       File f = new File(Path + "/" + file);
       if(f.isDirectory()){ ScanDirectoryND(f.toString());}
       else {
           File Info = new File(Path + "/" + file);
           ProcessedFile newFile = new ProcessedFile();
           newFile.setName(Info.getName());
           newFile.setSize(String.valueOf(Info.length()));
           newFile.setDate(Info.lastModified());
           boolean notDefault = false;
           for (String keyword : mediumKeywords) {
               if (file.contains(keyword)) {
                   newFile.chooseKey(2);
                   notDefault = true;
               }
           }
```

```
            for (String keyword : highKeywords) {
                if (file.contains(keyword)) {
                    newFile.chooseKey(3);
                    notDefault = true;
                }
            }
            if (notDefault == false) {
                newFile.chooseKey(1);
            }
            newFile.setPath(file);
            files.add(newFile);
        }
    }
}

// sets the keywords in the lists
public void SetIntensityKeywords(String keyword, int intensity)
    throws NoSuchAlgorithmException, IOException{
    if (intensity == 2) {
        mediumKeywords.add(keyword);
    }
    else if (intensity == 3) {
        highKeywords.add(keyword);
    }
    for (ProcessedFile f : files) {
        boolean notDefault = false;
        for (String word : mediumKeywords) {
```

```
            if (f.Name.contains(word)) {
                f.chooseKey(2);
                notDefault = true;
            }
        }
        for (String word : highKeywords) {
            if (f.Name.contains(word)) {
                f.chooseKey(3);
                notDefault = true;
            }
        }
        if (notDefault == false) {
            f.chooseKey(1);
        }
    }
}

// writes metadata of files to disk
public void writeFiles() throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new
        FileWriter("files.csv", true))) {
        ProcessedFile[] fi = new ProcessedFile[files.size()];
        fi = files.toArray(fi);
        for (ProcessedFile file : fi) {
            writer.write(file.Name + ',');
            writer.write(file.ModTime.toString() + ',');
            writer.write(String.valueOf(file.Size) + ',');
```

```
            writer.write(file.Hash + ',');
            writer.write(file.EncryptionKey + ',');
            writer.write(String.valueOf(file.chunkNum) + ',');
            for(String chunk : file.chunks) {
                writer.write(chunk + ',');
            }
            writer.write("\n");
        }
    }
}

// assuredly delete a file
public String[] AssuredDeletion(String file) {
    String[] chunks = null;
    for (ProcessedFile f : files) {
        if(f.Name == null ? file == null : f.Name.equals(file)) {
            if (f.EncryptionKey.contains("Level3key.dat")) {
                new File(f.EncryptionKey).delete();
            }
            chunks = f.chunks;
            files.remove(f);
            break;
        }
    }
    return chunks;
}
```

```
    public ProcessedFile findFile(String file) {
        ProcessedFile found = null;
        for (ProcessedFile f : files) {
            if (f.Name == null ? file == null : f.Name.equals(file)) {
                found = f;
            }
        }
        return found;
    }
}

/*
 * Utilities.java
 */
package dsbseis;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Arrays;
import org.apache.commons.codec.binary.Base64;
```

```
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.apache.commons.io.FilenameUtils;
import org.xml.sax.SAXException;

public class Utilities {
   // list of stop words to eliminate from documents
   private static final ArrayList<String> stopwords = new ArrayList
      <>(Arrays.asList(
             "a", "as", "able", "about", "above", "according", "
                accordingly", "across", "actually", "after",
             "afterwards", "again", "against", "aint", "all", "allow
                ", "allows", "almost", "alone", "along",
             "already", "also", "although", "always", "am", "among",
                 "amongst", "an", "and", "another", "any",
             "anybody", "anyhow", "anyone", "anything", "anyway", "
                anyways", "anywhere", "apart", "appear",
             "appreciate", "appropriate", "are", "arent", "around",
                "as", "aside", "ask", "asking", "associated",
             "at", "available", "away", "awfully", "be", "became", "
                because", "become", "becomes", "becoming", "been",
```

```
"before", "beforehand", "behind", "being", "believe", "
   below", "beside", "besides", "best", "better", "
   between",
"beyond", "both", "brief", "but", "by", "cmon", "cs", "
   came", "can", "cant", "cannot", "cant", "cause", "
   causes",
"certain", "certainly", "changes", "clearly", "co", "
   com", "come", "comes", "concerning", "consequently
   ",
"consider", "considering", "contain", "containing", "
   contains", "corresponding", "could", "couldnt", "
   course",
"currently", "definitely", "described", "despite", "did
   ", "didnt", "different", "do", "does", "doesnt", "
   doing",
"dont", "done", "down", "downwards", "during", "each",
   "edu", "eg", "eight", "either", "else", "elsewhere
   ",
"enough", "entirely", "especially", "et", "etc", "even
   ", "ever", "every", "everybody", "everyone", "
   everything",
"everywhere", "ex", "exactly", "example", "except", "
   far", "few", "ff", "fifth", "first", "five", "
   followed", "following",
"follows", "for", "former", "formerly", "forth", "four
   ", "from", "further", "furthermore", "get", "gets",
    "getting", "given",
```

```
"gives", "go", "goes", "going", "gone", "got", "gotten
   ", "greetings", "had", "hadnt", "happens", "hardly
   ", "has", "hasnt",
"have", "havent", "having", "he", "hes", "hello", "help
   ", "hence", "her", "here", "heres", "hereafter", "
   hereby", "herein",
"hereupon", "hers", "herself", "hi", "him", "himself",
   "his", "hither", "hopefully", "how", "howbeit", "
   however", "i", "id",
"ill", "im", "ive", "ie", "if", "ignored", "immediate",
    "in", "inasmuch", "inc", "indeed", "indicate", "
   indicated", "indicates",
"inner", "insofar", "instead", "into", "inward", "is",
   "isnt", "it", "itd", "itll", "its", "its", "itself
   ", "just", "keep",
"keeps", "kept", "know", "knows", "known", "last", "
   lately", "later", "latter", "latterly", "least", "
   less", "lest", "let",
"lets", "like", "liked", "likely", "little", "look", "
   looking", "looks", "ltd", "mainly", "many", "may",
   "maybe", "me", "mean",
"meanwhile", "merely", "might", "more", "moreover", "
   most", "mostly", "much", "must", "my", "myself", "
   name", "namely",
"nd", "near", "nearly", "necessary", "need", "needs", "
   neither", "never", "nevertheless", "new", "next", "
   nine", "no",
```

```
"nobody", "non", "none", "noone", "nor", "normally", "
   not", "nothing", "novel", "now", "nowhere", "
   obviously", "of",
"off", "often", "oh", "ok", "okay", "old", "on", "once
   ", "one", "ones", "only", "onto", "or", "other", "
   others", "otherwise",
"ought", "our", "ours", "ourselves", "out", "outside",
   "over", "overall", "own", "particular", "
   particularly", "per",
"perhaps", "placed", "please", "plus", "possible", "
   presumably", "probably", "provides", "que", "quite
   ", "qv",
"rather", "rd", "re", "really", "reasonably", "
   regarding", "regardless", "regards", "relatively",
   "respectively",
"right", "said", "same", "saw", "say", "saying", "says
   ", "second", "secondly", "see", "seeing", "seem", "
   seemed",
"seeming", "seems", "seen", "self", "selves", "sensible
   ", "sent", "serious", "seriously", "seven", "
   several", "shall",
"she", "should", "shouldnt", "since", "six", "so", "
   some", "somebody", "somehow", "someone", "something
   ", "sometime",
"sometimes", "somewhat", "somewhere", "soon", "sorry",
   "specified", "specify", "specifying", "still", "sub
   ", "such",
```

```
"sup", "sure", "ts", "take", "taken", "tell", "tends",
   "th", "than", "thank", "thanks", "thanx", "that", "
   thats",
"thats", "the", "their", "theirs", "them", "themselves
   ", "then", "thence", "there", "theres", "thereafter
   ", "thereby",
"therefore", "therein", "theres", "thereupon", "these",
    "they", "theyd", "theyll", "theyre", "theyve", "
   think", "third",
"this", "thorough", "thoroughly", "those", "though", "
   three", "through", "throughout", "thru", "thus", "
   to", "together",
"too", "took", "toward", "towards", "tried", "tries", "
   truly", "try", "trying", "twice", "two", "un", "
   under",
"unfortunately", "unless", "unlikely", "until", "unto",
    "up", "upon", "us", "use", "used", "useful", "uses
   ", "using",
"usually", "value", "various", "very", "via", "viz", "
   vs", "want", "wants", "was", "wasnt", "way", "we",
   "wed", "well",
"were", "weve", "welcome", "well", "went", "were", "
   werent", "what", "whats", "whatever", "when", "
   whence", "whenever",
"where", "wheres", "whereafter", "whereas", "whereby",
   "wherein", "whereupon", "wherever", "whether", "
   which", "while",
```

```
            "whither", "who", "whos", "whoever", "whole", "whom", "
               whose", "why", "will", "willing", "wish", "with", "
               within",
            "without", "wont", "wonder", "would", "would", "wouldnt
               ", "yes", "yet", "you", "youd", "youll", "youre", "
               youve",
            "your", "yours", "yourself", "yourselves", "zero"));

// displays progress of application
public static void progressBar(int current, int total) {
    float progress = ((float)current/total) * 100;
    String progressBar = "| | " + (int) progress + "%\r";
    if(progress > 5 && progress < 10) {
        progressBar = "|= | " + (int) progress + "%\r";
    }
    else if(progress > 10 && progress < 15) {
        progressBar = "|== | " + (int) progress + "%\r";
    }
    else if(progress > 15 && progress < 20) {
        progressBar = "|=== | " + (int) progress + "%\r";
    }
    else if(progress > 20 && progress < 25) {
        progressBar = "|==== | " + (int) progress + "%\r";
    }
    else if(progress > 25 && progress < 30) {
        progressBar = "|===== | " + (int) progress + "%\r";
    }
```

```
else if(progress > 30 && progress < 35) {
   progressBar = "|====== | " + (int) progress + "%\r";
}
else if(progress > 35 && progress < 40) {
   progressBar = "|======= | " + (int) progress + "%\r";
}
else if(progress > 40 && progress < 45) {
   progressBar = "|======== | " + (int) progress + "%\r";
}
else if(progress > 45 && progress < 50) {
   progressBar = "|========= | " + (int) progress + "%\r";
}
else if(progress > 50 && progress < 55) {
   progressBar = "|========== | " + (int) progress + "%\r";
}
else if(progress > 55 && progress < 60) {
   progressBar = "|=========== | " + (int) progress + "%\r";
}
else if(progress > 60 && progress < 65) {
   progressBar = "|============ | " + (int) progress + "%\r";
}
else if(progress > 65 && progress < 70) {
   progressBar = "|============= | " + (int) progress + "%\r
      ";
}
else if(progress > 70 && progress < 75) {
```

```
    progressBar = "|============== | " + (int) progress + "%\r
        ";
}
else if(progress > 75 && progress < 80) {
    progressBar = "|=============== | " + (int) progress + "%\
        r";
}
else if(progress > 80 && progress < 85) {
    progressBar = "|================ | " + (int) progress +
        "%\r";
}
else if(progress > 85 && progress < 90) {
    progressBar = "|================= | " + (int) progress +
        "%\r";
}
else if(progress > 90 && progress < 95) {
    progressBar = "|================== | " + (int) progress +
        "%\r";
}
else if(progress > 95 && progress < 100) {
    progressBar = "|=================== | " + (int) progress +
         "%\r";
}
else if(progress == 100) {
    progressBar = "|====================| " + (int) progress +
         "%\n";
}
```

```
    System.out.print(progressBar);
}

public static void progressBar(long current, long total) {
    float progress = ((float)current/total) * 100;
    String progressBar = "| | " + (int) progress + "%\r";
    if(progress > 5 && progress < 10) {
        progressBar = "|= | " + (int) progress + "%\r";
    }
    else if(progress > 10 && progress < 15) {
        progressBar = "|== | " + (int) progress + "%\r";
    }
    else if(progress > 15 && progress < 20) {
        progressBar = "|=== | " + (int) progress + "%\r";
    }
    else if(progress > 20 && progress < 25) {
        progressBar = "|==== | " + (int) progress + "%\r";
    }
    else if(progress > 25 && progress < 30) {
        progressBar = "|===== | " + (int) progress + "%\r";
    }
    else if(progress > 30 && progress < 35) {
        progressBar = "|====== | " + (int) progress + "%\r";
    }
    else if(progress > 35 && progress < 40) {
        progressBar = "|======= | " + (int) progress + "%\r";
    }
```

```
else if(progress > 40 && progress < 45) {
   progressBar = "|======== | " + (int) progress + "%\r";
}
else if(progress > 45 && progress < 50) {
   progressBar = "|========= | " + (int) progress + "%\r";
}
else if(progress > 50 && progress < 55) {
   progressBar = "|========== | " + (int) progress + "%\r";
}
else if(progress > 55 && progress < 60) {
   progressBar = "|=========== | " + (int) progress + "%\r";
}
else if(progress > 60 && progress < 65) {
   progressBar = "|============ | " + (int) progress + "%\r";
}
else if(progress > 65 && progress < 70) {
   progressBar = "|============= | " + (int) progress + "%\r
      ";
}
else if(progress > 70 && progress < 75) {
   progressBar = "|============== | " + (int) progress + "%\r
      ";
}
else if(progress > 75 && progress < 80) {
   progressBar = "|=============== | " + (int) progress + "%\
      r";
}
```

```
    else if(progress > 80 && progress < 85) {
        progressBar = "|================ | " + (int) progress +
            "%\r";
    }
    else if(progress > 85 && progress < 90) {
        progressBar = "|================= | " + (int) progress +
            "%\r";
    }
    else if(progress > 90 && progress < 95) {
        progressBar = "|================== | " + (int) progress +
            "%\r";
    }
    else if(progress > 95 && progress < 100) {
        progressBar = "|=================== | " + (int) progress +
             "%\r";
    }
    else if(progress == 100) {
        progressBar = "|====================| " + (int) progress +
             "%\n";
    }
    System.out.print(progressBar);
}

// safely converts byte array to a string
public static String BytesToString(byte[] bytes) {
    return new String(Base64.encodeBase64(bytes));
}
```

```
// safely converts a string to a byte array
public static byte[] StringToBytes(String s) {
    return Base64.decodeBase64(s);
}

// converts string to byte array
public static byte[] ASCIItoBytes(String s) throws
   UnsupportedEncodingException {
    return s.getBytes("US-ASCII");
}

// converts a byte array to a string
public static String BytesToASCII(byte[] b) throws
   UnsupportedEncodingException {
    return new String(b, "US-ASCII");
}

// extract words from a document
public static String[] ExtractWords(String file) throws
   IOException, SAXException, ParserConfigurationException {
    String line;
    List<String> words = new ArrayList<>();
    if(FilenameUtils.getExtension(file).equals("txt")) {
        try (BufferedReader reader = new BufferedReader(new
           FileReader(file))) {
            while ((line = reader.readLine()) != null) {
```

```
                String[] items = TokenizeWords(line);
                for (String item : items) {
                    if (item.length() <= 1)
                        continue;
                    if (!words.contains(item) && !stopwords.contains
                        (item)) {
                        words.add(item);
                    }
                }
            }
        }
    }
    String[] res = new String[words.size()];
    res = words.toArray(res);
    return res;
}

// extract documents from XML container to disk
public static void ExtractXMLDocs(String file) throws
    SAXException, ParserConfigurationException, IOException {
    SAXParser parser = SAXParserFactory.newInstance().
        newSAXParser();
    if(file.contains("abstract")){
        AbstractHandler handler = new AbstractHandler(new
            PageProcessor() {
            @Override
            public void process(Page page){
```

```
            String title = page.Title.replaceAll("\\s+", "_").
               replaceAll("/", "_");
            try (PrintWriter buffWriter = new PrintWriter(new
               FileWriter("disk/" + title + ".txt", true))){
               buffWriter.println(page.Text);
            } catch (IOException ex) {
               Logger.getLogger(Utilities.class.getName()).log(
                  Level.SEVERE, null, ex);
            }
         }
      });
      parser.parse(file, handler);
   }
   else if(file.contains("page")) {
      PageHandler handler = new PageHandler(new PageProcessor()
         {
         @Override
         public void process(Page page){
            String title = page.Title.replaceAll("\\s+", "_").
               replaceAll("/", "_");
            try (PrintWriter buffWriter = new PrintWriter(new
               FileWriter("disk/" + title + ".txt", true))){
               buffWriter.println(page.Text);
            } catch (IOException ex) {
               Logger.getLogger(Utilities.class.getName()).log(
                  Level.SEVERE, null, ex);
            }
```

```
            }
        });
        parser.parse(file, handler);
    }
}

// tokenize words
public static String[] TokenizeWords(String s) {
    if (s != null) {
        String filtered = s.toLowerCase().replaceAll("[^a-z\\s]",
            "");
        String[] tokenized = filtered.split("\\s+");
        return tokenized;
    }
    else
        return null;
}

public static String stripIllegalPathCharacters(String path) {
    return path.replaceAll("[^a-zA-Z0-9_/-/.]", "");
}

// chunk a file on disk
public static String[] ChunkFile(String path, int chunkSize)
    throws IOException {
    final int BUFFER_SIZE = 20 * 1024;
    byte[] buffer = new byte[BUFFER_SIZE];
```

```
int index = 0;
long position = 0;
long size = new File(path).length();
List<String> chunks = new ArrayList<>();
try (FileInputStream fis = new FileInputStream(path)) {
    while(position < size) {
        new File("Data/").mkdir();
        try (FileOutputStream output = new FileOutputStream("
           Data/" + FilenameUtils.getName(path) + "_" + index
           + ".chk")) {
            chunks.add(FilenameUtils.getName(path) + "_" +
               index + ".chk");
            int remaining = chunkSize;
            int bytesRead;
            while(remaining > 0 && (bytesRead = fis.read(buffer
               , 0, Math.min(remaining, BUFFER_SIZE))) > 0) {
               output.write(buffer, 0, bytesRead);
               remaining -= bytesRead;
               position += bytesRead;
            }
        }
        index++;
    }
}
String[] res = new String[chunks.size()];
res = chunks.toArray(res);
return res;
```

```
}

// group chunks together
public static void GroupFileChunks(String[] chunkList, String
    fileName){
    try {
        new File("Files/").mkdir();
        try (FileOutputStream output = new FileOutputStream("Files
            /" + new File(fileName).getName().replace("encrypted",
            ""))){
            for(String chunk : chunkList) {
                int bytesRead = 0;
                byte[] buffer = new byte[1024];
                try (FileInputStream input = new FileInputStream("
                    decrypted/" + chunk)) {
                    while((bytesRead = input.read(buffer, 0, buffer.
                        length)) > 0) {
                        output.write(buffer, 0, bytesRead);
                    }
                }
            }
        }
    catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
```

```
}

/*
 * Page.java
 */
package dsbseis;

// data structure to store a page in memory
public class Page {
    public String Title = "";
    public String Text = "";
    public Page() {
        Title = "";
        Text = "";
    }
}

/*
 * AbstractHandler.java
 */
package dsbseis;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

// handler of xml abstract documents
public class AbstractHandler extends DefaultHandler {
```

```
private boolean bTitle = false, newPage = true, bAbstract = false
    , bLinks = false;
Page page;
private final PageProcessor processor;

public AbstractHandler(PageProcessor processor) {
    this.processor = processor;
}

@Override
public void startElement(String uri, String localName, String
   qName, Attributes attributes) throws SAXException {
    if(qName.equals("doc")) {
        newPage = false;
        page = new Page();
    }
    else if(qName.equals("title")){
        bTitle = true;
    }
    else if(qName.equals("abstract")) {
        bAbstract = true;
    }
    else if(qName.equals("anchor")) {
        bLinks = true;
    }
}
```

```
@Override
public void endElement(String uri, String localName, String qName
   ) throws SAXException {
   if(!newPage) {
       if (qName.equals("doc")) {
           newPage = true;
           processor.process(page);
           page = null;
       }
       else if(qName.equals("title")){
           bTitle = false;
       }
       else if(qName.equals("abstract")) {
           bAbstract = false;
       }
       else if(qName.equals("anchor")) {
           bLinks = false;
       }
   }
}

@Override
public void characters(char ch[], int start, int length) throws
   SAXException {
   if(bTitle){
       page.Title = new String(ch, start, length);
   }
```

```
        else if(bAbstract) {
            page.Text += new String(ch, start, length);
        }
        else if(bLinks) {
            page.Text += " " + new String(ch, start, length);
        }
    }
}

/*
 * PageHandler.java
 */
package dsbseis;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

// handles body text documentsfrom xml containers
public class PageHandler extends DefaultHandler {
    private boolean newPage = true, bTitle = false, bComment = false,
         bText = false;
    private Page page;
    private final PageProcessor processor;

    public PageHandler(PageProcessor processor) {
        this.processor = processor;
    }
```

```
@Override
public void startElement(String uri, String localName, String
   qName, Attributes attributes) throws SAXException {
   if(qName.equals("page")) {
      newPage = false;
      page = new Page();
   }
   else if(qName.equals("title")){
      bTitle = true;
   }
   else if(qName.equals("comment")) {
      bComment = true;
   }
   else if(qName.equals("text")) {
      bText = true;
   }
}

@Override
public void endElement(String uri, String localName, String qName
   ) throws SAXException {
   if(!newPage) {
      if(qName.equals("page")) {
         newPage = true;
         processor.process(page);
         page = null;
```

```
            }
            else if(qName.equals("title")){
                bTitle = false;
            }
            else if(qName.equals("comment")) {
                bComment = false;
            }
            else if(qName.equals("text")) {
                bText = false;
            }
        }
    }

    @Override
    public void characters(char ch[], int start, int length) throws
        SAXException {
        if(bTitle){
            page.Title += new String(ch, start, length);
        }
        else if(bComment) {
            page.Text += new String(ch, start, length);
        }
        else if(bText) {
            page.Text += new String(ch, start, length);
        }
    }
}
```

```
/*
 * PageProcessor.jar
 */
package dsbseis;

// interface to process pages
public interface PageProcessor {
    void process(Page page);
}

/*
 * DSB-SEIServer.java
 */
package dsb.seiserver;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
```

```
import java.net.SocketException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;


public class DSBSEIServer {
    // index data structure in memory
    static HashMap<String, List<String>> keywords = new HashMap<>();

    // handle requests from client
```

```
public static void main(String[] args) throws IOException,
   NoSuchAlgorithmException {
   int Port = 2390;
   ReadIndex("HSHDS");
   while(true){
      System.out.println("waiting on connection...");
      try (ServerSocket listener = new ServerSocket(Port);
         Socket client = listener.accept();) {
         try (DataInputStream netStream = new DataInputStream(
            client.getInputStream()); DataOutputStream out =
            new DataOutputStream(client.getOutputStream())){
            while(!client.isClosed()){
               System.out.println("Waiting on command");
               try {
                  byte[] array = new byte[4];
                  netStream.read(array, 0, 4);
                  int command = ByteBuffer.wrap(array).getInt
                     ();
                  System.out.println("Received command: " +
                     command);
                  // Process the commands
                  switch(command) {
                     // if == 1: receive files
                     case 1:
                        readFile(netStream);
                        break;
                     // if == 2: receive keywords
```

```
                        case 2:
                            netStream.readInt();
                            readIndexFile(netStream);
                            break;
// // if == 3: search for a keyword
                        case 3:
                            // receive the token
                            int numQueries = netStream.readInt();
                            Map<String, Integer> Queries = new
                               HashMap<>();
                            List<List<String>> andRes = new
                               ArrayList<>();
                            List<List<String>> orRes = new
                               ArrayList<>();
                            List<List<String>> notRes = new
                               ArrayList<>();
                            List<String> result = new ArrayList
                               <>();
                            for(int i = 0; i < numQueries; i++) {
                                int type = netStream.readInt();
                                int len = netStream.readInt();
                                byte[] token = new byte[len];
                                netStream.read(token, 0, len);
                                Queries.put(new String(token),
                                   type);
                            }
                            System.out.print("Searching..");
```

```
for(String tok : Queries.keySet()){
    if(null != Queries.get(tok)) {
        switch (Queries.get(tok)) {
            // Not query
            case 1:
                notRes.add(searchIndex(
                    tok));
                break;
            // Or query
            case 2:
                orRes.add(searchIndex(
                    tok));
                break;
            // And Query
            case 3:
                andRes.add(searchIndex(
                    tok));
                break;
            default:
                break;
        }
    }
}
if(!andRes.isEmpty()) {
    for(List<String> aR : andRes) {
        if(aR != null){
            if(result.isEmpty()){
```

```
                    result.addAll(aR);
                }
                else {
                    result = intersection(
                        result, aR);
                }
            }
            else {
                result = new ArrayList<>();
                break;
            }
        }
    }
    if(!notRes.isEmpty()) {
        for(List<String> nR : notRes) {
            if(nR != null)
                result = difference(result,
                    nR);
        }
    }
    if(!orRes.isEmpty()) {
        for(List<String> oR : orRes) {
            if(oR != null){
                System.out.println(oR.
                    toString());
                result = union(result, oR);
            }
```

```
    }
}
if(result == null)
    out.writeInt(0);
// return found
else {
    int size = result.size();
    byte[] sizeBytes = ByteBuffer.
        allocate(4).order(ByteOrder.
        BIG_ENDIAN).putInt(size).array
        ();
    System.out.println(ByteBuffer.wrap
        (sizeBytes).order(ByteOrder.
        BIG_ENDIAN).getInt());
    out.write(sizeBytes);

    for (String f : result) {
        byte[] tmp = f.getBytes("US-
            ASCII");
        int length = tmp.length;
        sizeBytes = ByteBuffer.allocate
            (4).order(ByteOrder.
            BIG_ENDIAN).putInt(length).
            array();
        out.write(sizeBytes);
        out.write(tmp);
    }
```

```
            }
            out.flush();
            break;
        // if == 4: search for a chunk
        case 4:
            // receive a token
            int len2 = netStream.readInt();
            byte[] token2 = new byte[len2];
            netStream.read(token2, 0, len2);
            // get a list of files
            List<File> files = ScanDirectory(
                Paths.get("DSBase/"));
            // hash each one
            byte[] hash;
            int exists = 0;
            MessageDigest SHA256 = MessageDigest.
                getInstance("SHA-256");
            System.out.println("Searching...");
            int currentFile = 0;
            for(File f : files)
            {
                hash = hashFile(SHA256, f);
                if(Arrays.equals(hash, token2)) {
                    exists = 1;
                }
                progressBar(currentFile++, files.
                    size());
```

```
            }
            byte[] existsBytes = ByteBuffer.
               allocate(4).order(ByteOrder.
               BIG_ENDIAN).putInt(exists).array()
               ;
            out.write(existsBytes);
            out.flush();
            // return true if found and false if
               not
            break;
        // if == 5: send back a file
        case 5:
            int length2 = netStream.readInt();
            byte[] filenamebytes2 = new byte[
               length2];
            netStream.read(filenamebytes2, 0,
               length2);
            String filename2 = new String(
               filenamebytes2);
            System.out.println("Sending back " +
               filename2);
            byte[] bytes = new byte[1024];
            try (FileInputStream input = new
               FileInputStream("DSBase\\" +
               filename2))
            {
```

```
            byte[] length2Bytes = ByteBuffer.
               allocate(4).order(ByteOrder.
               BIG_ENDIAN).putInt(length2).
               array();
            out.write(length2Bytes);
            out.write(filenamebytes2);
            long fileSize = (new File("DSBase
               \\" + filename2).length());
            out.writeLong(fileSize);
            int bytesSent;
            long totalSent = 0;
            while ((bytesSent = input.read(
               bytes)) > 0) {
               out.write(bytes, 0, bytesSent);
               totalSent += bytesSent;
               progressBar(totalSent, fileSize
                  );
            }
         }
         break;
      // if == 6: delete a file
      case 6:
         int length3 = netStream.readInt();
         byte[] filenamebytes3 = new byte[
            length3];
         netStream.read(filenamebytes3, 0,
            length3);
```

```
                String filename3 = new String(
                    filenamebytes3);
                File f = new File(filename3);
                boolean deleted = f.delete();
                if(deleted) {
                    System.out.println("Deleted " +
                        filename3);
                }
                break;
            case 7:
                int currentDelete = 0;
                if((new File("DSBase/")).exists()) {
                    File[] toBeDeleted = (new File("
                        DSBase/")).listFiles();
                    for (File file: toBeDeleted) {
                        file.delete();
                        progressBar(currentDelete++,
                            toBeDeleted.length);
                    }
                }
                keywords = new HashMap<>();
                break;
            default:
                System.out.println("Wrong Command");
                System.exit(1);
                break;
        }
```

```
                }
                catch(SocketException | EOFException e) {
                    client.close();
                    System.out.println("\tConection closed...");
                }
            }
        } catch (IOException ex) {
            Logger.getLogger(DSBSEIServer.class.getName()).log(
                Level.SEVERE, null, ex);
        }
    }
}

public static void progressBar(int current, int total)
{
    float progress = ((float)current/total) * 100;
    String progressBar = "| | " + (int) progress + "%\r";
    if(progress > 5 && progress < 10) {
        progressBar = "|= | " + (int) progress + "%\r";
    }
    else if(progress > 10 && progress < 15)
    {
        progressBar = "|== | " + (int) progress + "%\r";
    }
    else if(progress > 15 && progress <20)
    {
```

```
    progressBar = "|=== | " + (int) progress + "%\r";
}
else if(progress > 20 && progress < 25)
{
    progressBar = "|==== | " + (int) progress + "%\r";
}
else if(progress > 25 && progress < 30)
{
    progressBar = "|===== | " + (int) progress + "%\r";
}
else if(progress > 30 && progress < 35)
{
    progressBar = "|====== | " + (int) progress + "%\r";
}
else if(progress > 35 && progress < 40)
{
    progressBar = "|======= | " + (int) progress + "%\r";
}
else if(progress > 40 && progress < 45)
{
    progressBar = "|======== | " + (int) progress + "%\r";
}
else if(progress > 45 && progress < 50)
{
    progressBar = "|========= | " + (int) progress + "%\r";
}
else if(progress > 50 && progress < 55)
```

```
{
    progressBar = "|========== | " + (int) progress + "%\r";
}
else if(progress > 55 && progress < 60)
{
    progressBar = "|=========== | " + (int) progress + "%\r";
}
else if(progress > 60 && progress < 65)
{
    progressBar = "|============ | " + (int) progress + "%\r";
}
else if(progress > 65 && progress < 70)
{
    progressBar = "|============= | " + (int) progress + "%\r
        ";
}
else if(progress > 70 && progress < 75)
{
    progressBar = "|============== | " + (int) progress + "%\r
        ";
}
else if(progress > 75 && progress < 80)
{
    progressBar = "|=============== | " + (int) progress + "%\
        r";
}
else if(progress > 80 && progress < 85)
```

```
    {
        progressBar = "|=============== | " + (int) progress +
            "%\r";
    }
    else if(progress > 85 && progress < 90)
    {
        progressBar = "|================ | " + (int) progress +
            "%\r";
    }
    else if(progress > 90 && progress < 95)
    {
        progressBar = "|================= | " + (int) progress +
            "%\r";
    }
    else if(progress > 95 && progress < 100)
    {
        progressBar = "|================== | " + (int) progress +
             "%\r";
    }
    else if(progress == 100)
    {
        progressBar = "|====================| " + (int) progress +
             "%\n";
    }
    System.out.print(progressBar);
}
```

```
public static void progressBar(long current, long total) {
    float progress = ((float)current/total) * 100;
    String progressBar = "| | " + (int) progress + "%\r";
    if(progress > 5 && progress < 10) {
        progressBar = "|= | " + (int) progress + "%\r";
    }
    else if(progress > 10 && progress < 15) {
        progressBar = "|== | " + (int) progress + "%\r";
    }
    else if(progress > 15 && progress <20) {
        progressBar = "|=== | " + (int) progress + "%\r";
    }
    else if(progress > 20 && progress < 25) {
        progressBar = "|==== | " + (int) progress + "%\r";
    }
    else if(progress > 25 && progress < 30) {
        progressBar = "|===== | " + (int) progress + "%\r";
    }
    else if(progress > 30 && progress < 35) {
        progressBar = "|====== | " + (int) progress + "%\r";
    }
    else if(progress > 35 && progress < 40) {
        progressBar = "|======= | " + (int) progress + "%\r";
    }
    else if(progress > 40 && progress < 45) {
        progressBar = "|======== | " + (int) progress + "%\r";
    }
```

```
else if(progress > 45 && progress < 50) {
   progressBar = "|========= | " + (int) progress + "%\r";
}
else if(progress > 50 && progress < 55) {
   progressBar = "|========== | " + (int) progress + "%\r";
}
else if(progress > 55 && progress < 60) {
   progressBar = "|=========== | " + (int) progress + "%\r";
}
else if(progress > 60 && progress < 65) {
   progressBar = "|============ | " + (int) progress + "%\r";
}
else if(progress > 65 && progress < 70) {
   progressBar = "|============= | " + (int) progress + "%\r
      ";
}
else if(progress > 70 && progress < 75) {
   progressBar = "|============== | " + (int) progress + "%\r
      ";
}
else if(progress > 75 && progress < 80) {
   progressBar = "|=============== | " + (int) progress + "%\
      r";
}
else if(progress > 80 && progress < 85) {
   progressBar = "|================ | " + (int) progress +
      "%\r";
```

```
    }
    else if(progress > 85 && progress < 90) {
        progressBar = "|================ | " + (int) progress +
            "%\r";
    }
    else if(progress > 90 && progress < 95) {
        progressBar = "|================= | " + (int) progress +
            "%\r";
    }
    else if(progress > 95 && progress < 100) {
        progressBar = "|================== | " + (int) progress +
             "%\r";
    }
    else if(progress == 100) {
        progressBar = "|====================| " + (int) progress +
             "%\n";
    }
    System.out.print(progressBar);
}

public static List<String> searchIndex(String token) throws
    FileNotFoundException, IOException {
    List<String> res = new ArrayList<>();
    File[] indexes = new File("index/").listFiles();
    String line;
    String[] l;
    for(File index : indexes){
```

```
        try(BufferedReader br = new BufferedReader(new FileReader(
            index.getAbsolutePath()))){
            while((line = br.readLine())!= null) {
               l = line.split(",");
               if(token.equals(l[0])){
                   for(int i = 1; i < l.length; i++) {
                      res.add(l[i]);
                   }
               }
            }
        }
    }
    return res;
}

public static String readIndexFile(DataInputStream netStream)
    throws IOException {
    byte[] array = new byte[4];
    netStream.read(array, 0, 4);
    int length = ByteBuffer.wrap(array).getInt();
    byte[] filenamebytes = new byte[length];
    netStream.read(filenamebytes, 0, length);
    String filename = new String(filenamebytes);
    System.out.println("Receiving " + filename);
    long fileSize = netStream.readLong();
    new File("index/").mkdirs();
```

```
    try (FileOutputStream output = new FileOutputStream("index/"
       + filename)) {
       byte[] buffer = new byte[1024];
       // Read the incoming stream
       int bytesRead;
       long totalBytes = 0;
       while (totalBytes != fileSize) {
           int remaining = 1024;
           if((fileSize - totalBytes) < 1024) {
              remaining = (int)(fileSize - totalBytes);
           }
           bytesRead = netStream.read(buffer, 0, remaining);
           output.write(buffer, 0, bytesRead);
           totalBytes += bytesRead;
           progressBar(totalBytes, fileSize);
       }
    }
    catch(Exception ex) {
       System.out.println(ex.getMessage());
    }
    return filename;
}

public static String readFile(DataInputStream netStream) throws
    IOException {
    byte[] array = new byte[4];
    netStream.read(array, 0, 4);
```

```
int length = ByteBuffer.wrap(array).getInt();
byte[] filenamebytes = new byte[length];
netStream.read(filenamebytes, 0, length);
String filename = new String(filenamebytes);
System.out.println("Receiving " + filename);
array = new byte[8];
netStream.read(array, 0, 8);
long fileSize = ByteBuffer.wrap(array).getLong();
new File("DSBase/").mkdirs();
try (FileOutputStream output = new FileOutputStream("DSBase/"
    + filename)) {
   byte[] buffer = new byte[1024];
   // Read the incoming stream
   int bytesRead;
   long totalBytes = 0;
   while (totalBytes != fileSize) {
       int remaining = 1024;
       if((fileSize - totalBytes) < 1024) {
           remaining = (int)(fileSize - totalBytes);
       }
       bytesRead = netStream.read(buffer, 0, remaining);
       output.write(buffer, 0, bytesRead);
       totalBytes += bytesRead;
       progressBar(totalBytes, fileSize);
   }
}
catch(Exception ex) {
```

```
            System.out.println(ex.getMessage());
        }
        return filename;
    }

    public static String readString(DataInputStream netStream) throws
         IOException {
        int length = netStream.readInt();
        byte[] tokenbytes = new byte[length];
        netStream.read(tokenbytes, 0, length);
        String token = new String(tokenbytes);
        return token;
    }

    private static byte[] hashFile(MessageDigest digest, File file)
        throws IOException {
        //Get file input stream for reading the file content
        try (FileInputStream fis = new FileInputStream(file)) {
            //Create byte array to read data in chunks
            byte[] byteArray = new byte[1024];
            int bytesCount;
            //Read file data and update in message digest
            while ((bytesCount = fis.read(byteArray)) != -1) {
                digest.update(byteArray, 0, bytesCount);
            }
        }
```

```
    return digest.digest();
}

static List<File> ScanDirectory(Path directory) throws
    IOException {
    List<File> files = new ArrayList<>();
    try (DirectoryStream<Path> ds = Files.newDirectoryStream(
        directory)) {
        for (Path child : ds) {
            if (!Files.isDirectory(child)) {
                files.add(child.toFile());
            }
        }
    }
    return files;
}

public static <T> List<T> difference(Collection<T> A, Collection<
    T> B) {
    List<T> diff = new ArrayList<>(A);
    diff.removeAll(B);
    return diff;
}


public static <T> List<T> union(Collection<T> list1, Collection<T
    > list2) {
```

```
    Set<T> set = new HashSet<>();

    set.addAll(list1);
    set.addAll(list2);

    return new ArrayList<>(set);
}

public static <T> List<T> intersection(Collection<T> list1,
   Collection<T> list2) {
    List<T> list = new ArrayList<>();
    for (T t : list1) {
        if(list2.contains(t)) {
            list.add(t);
        }
    }

    return list;
}

public static void ReadIndex(String IndexPath) throws IOException
     {
    BufferedReader buf;
    String line;
    String[] splitLine;
    String[] content;
    List<String> fi = new ArrayList<>();
```

```
    if(new File(IndexPath).exists()) {
        buf = new BufferedReader(new FileReader(IndexPath));
        line = buf.readLine();
        while(line != null){
            splitLine = line.split(":");
            content = splitLine[1].split(",");
            fi.addAll(Arrays.asList(content));
            keywords.put(splitLine[0], fi);
            line = buf.readLine();
            fi = new ArrayList<>();
        }
        buf.close();
    }
}

// Saves the index into a a file on disk
public static void saveOnDisk() {
    String indexSegment = "HSHDS";
    try (PrintWriter writer = new PrintWriter(indexSegment, "UTF
        -8")) {
        int i;
        for(String k : keywords.keySet()) {
            writer.print(k + ":");
            i = 0;
            for(String v : keywords.get(k)) {
                writer.print( i == 0 ? v : "," + v);
                if(i == 0)
```

```
                    i++;
                }
                writer.println();
            }
        } catch (IOException e) {
           System.out.println(e.getMessage());
        }
    }
}

/*
 * XMLInput.java
 */
package xmlinput;



import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.URI;
import java.net.URISyntaxException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
```

```
import javax.crypto.spec.IvParameterSpec;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class XMLInput {
    public enum OUTPUT_COUNTER {
        OUTPUT_RECORDS_COUNTER
    };

    static SecretKey KEY;
    static IvParameterSpec IV;
    // driver of MapReduce indexing application
    public static void main(String[] args) throws URISyntaxException,
         NoSuchAlgorithmException {
        try {
            Runtime rt = Runtime.getRuntime();
            Process proc;
            Configuration conf = new Configuration();
            String line;
```

```
String[] Args = new GenericOptionsParser(conf, args).
   getRemainingArgs();
FileSystem fs = FileSystem.get(conf);
int numReduce = Integer.parseInt(Args[0]);

// Other docs
conf.set("START_TAG_KEY", "<page>");
conf.set("END_TAG_KEY", "</page>");
//Abstracts
conf.set("START_TAG_KEY2", "<doc>");
conf.set("END_TAG_KEY2", "</doc>");

conf.set("mapreduce.map.output.compress", "true");
conf.set("mapred.map.output.compress.codec", "org.apache.
   hadoop.io.compress.SnappyCodec");
conf.set("mapreduce.output.fileoutputformat.compress", "
   false");

try {
    if(! fs.exists(new Path("key.dat"))) {
        System.out.println("generating a key");
        KEY = generateKey(128, "AES");
        byte[] iv = generateIV();
        IV = new IvParameterSpec(iv);
        saveKey(KEY, iv, "key.dat", fs);
    }
} catch (NoSuchAlgorithmException ex) {
```

```
    Logger.getLogger(XMLInput.class.getName()).log(Level.
        SEVERE, null, ex);
}

Job job = Job.getInstance(conf);
job.setJobName("XML Parser");
job.addCacheFile(new URI("key.dat#key"));
job.setJarByClass(XMLInput.class);
job.setMapperClass(XMLMapper.class);
job.setReducerClass(XMLReducer.class);
job.setNumReduceTasks(numReduce);

job.setInputFormatClass(XMLInputFormat.class);
job.setOutputFormatClass(StreamingTextOutputFormat.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(Args[1]));
FileOutputFormat.setOutputPath(job, new Path(Args[2]));
job.waitForCompletion(false);
System.out.print(job.getCounters().findCounter("org.apache
    .hadoop.mapreduce.TaskCounter", "MAP_INPUT_RECORDS").
    getValue() + ",");
System.out.print(numReduce + "," + (job.getFinishTime() -
    job.getStartTime())/1000 + ",");
```

```
            proc = rt.exec("hdfs dfs -du -s /wikiIndexTest");
            BufferedReader reader = new BufferedReader(new
               InputStreamReader(proc.getInputStream()));
            while ((line = reader.readLine())!= null) {
               System.out.print(line.split("\\s+")[0] + ",");
            }
            proc.waitFor();
            System.out.print(job.getCounters().findCounter("xmlinput.
               XMLInput$OUTPUT_COUNTER", "OUTPUT_RECORDS_COUNTER").
               getValue() + ",");
            System.exit(1);
        } catch (IOException | IllegalStateException |
            IllegalArgumentException | InterruptedException |
            ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }

    private static SecretKey generateKey(int size, String Algorithm)
        throws UnsupportedEncodingException, NoSuchAlgorithmException
        {
        KeyGenerator keyGen = KeyGenerator.getInstance(Algorithm);
        keyGen.init(size);
        return keyGen.generateKey();
    }

    private static byte[] generateIV() {
```

```
        byte[] b = new byte[16];
        new SecureRandom().nextBytes(b);
        return b;
    }

    public static void saveKey(SecretKey key, byte[] IV, String path,
         FileSystem fs) throws IOException {
        FSDataOutputStream stream = fs.create(new Path(path));
        try {
            stream.write(key.getEncoded());
            stream.write(IV);
        } finally {
            stream.close();
        }
    }
}

/*
 * TextArrayWritable.java
 */
package xmlinput;

import org.apache.hadoop.io.ArrayWritable;
import org.apache.hadoop.io.Text;

// enables mapreduce to store arrays as a writable object
public class TextArrayWritable extends ArrayWritable {
    public TextArrayWritable() {
```

```
        super(Text.class);
    }

    public TextArrayWritable(Text[] values) {
        super(Text.class, values);
    }

    @Override
    public Text[] get() {
        return (Text[]) super.get();
    }

    @Override
    public String toString() {
        Text[] values = get();
        String s = "";
        for(int i = 0; i < values.length; i++){
            s += values[i];
            if(i != (values.length - 1))
                s += " ";
        }
        return s;
    }
}

/*
 * XMLInputFormat.java
 */
```

```
package xmlinput;

import com.google.common.base.Charsets;
import com.google.common.io.Closeables;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DataOutputBuffer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

// custom input format to read documents from XML container
public class XMLInputFormat extends TextInputFormat {
   @Override
   public RecordReader<LongWritable, Text> createRecordReader(
           InputSplit split, TaskAttemptContext context) {
       try {
```

```
        return new XmlRecordReader((FileSplit) split, context.
            getConfiguration());
    } catch (IOException ex) {
        Logger.getLogger(XMLInputFormat.class.getName()).log(Level
            .SEVERE, null, ex);
        return null;
    }
}

public static class XmlRecordReader extends RecordReader<
    LongWritable, Text> {
    private byte[] startTag;
    private byte[] endTag;
    private long start;
    private long end;
    private FSDataInputStream fsin;
    private final DataOutputBuffer buffer = new DataOutputBuffer
        ();
    private final LongWritable key = new LongWritable();
    private final Text value = new Text();
    private Logger logger = Logger.getLogger(XmlRecordReader.
        class.getName());

    public XmlRecordReader(FileSplit split, Configuration conf)
        throws IOException {
        String name = split.getPath().getName();
        if(name.contains("pages")) {
```

```
        startTag = conf.get("START_TAG_KEY").getBytes(Charsets.
            UTF_8);
        endTag = conf.get("END_TAG_KEY").getBytes(Charsets.
            UTF_8);
    }
    else if(name.contains("abstract")){
        startTag = conf.get("START_TAG_KEY2").getBytes(Charsets
            .UTF_8);
        endTag = conf.get("END_TAG_KEY2").getBytes(Charsets.
            UTF_8);
    }
}

@Override
public void initialize(InputSplit is, TaskAttemptContext tac)
        throws IOException, InterruptedException {
    FileSplit fileSplit = (FileSplit) is;

    start = fileSplit.getStart();
    end = start + fileSplit.getLength();
    Path file = fileSplit.getPath();

    FileSystem fs = file.getFileSystem(tac.getConfiguration())
        ;
    fsin = fs.open(fileSplit.getPath());
    fsin.seek(start);
}
```

```
@Override
public boolean nextKeyValue() throws IOException,
   InterruptedException {
   if (fsin.getPos() < end) {
      if (readUntilMatch(startTag, false)) {
         try {
            buffer.write(startTag);
            if (readUntilMatch(endTag, true)) {

               value.set(buffer.getData(), 0, buffer.
                  getLength());
               key.set(fsin.getPos());
               return true;
            }
         } finally {
            buffer.reset();
         }
      }
   }
   return false;
}

@Override
public LongWritable getCurrentKey() throws IOException,
      InterruptedException {
```

```
    return key;
}

@Override
public Text getCurrentValue() throws IOException,
    InterruptedException {
    return value;

}

@Override
public float getProgress() throws IOException,
    InterruptedException {
    return (fsin.getPos() - start) / (float) (end - start);
}

@Override
public void close() throws IOException {
    Closeables.close(fsin, true);
}

private boolean readUntilMatch(byte[] match, boolean
   withinBlock)
       throws IOException {
    int i = 0;
    while (true) {
       int b = fsin.read();
```

```
                if (b == -1)
                    return false;

                if (withinBlock)
                    buffer.write(b);

                if (b == match[i]) {
                    i++;
                    if (i >= match.length)
                        return true;
                } else
                    i = 0;

                if (!withinBlock && i == 0 && fsin.getPos() >= end)
                    return false;
            }
        }
    }
}

/*
 * XMLMapper.java
 */
package xmlinput;

import java.io.ByteArrayInputStream;
import java.io.FileNotFoundException;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.StringTokenizer;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import org.apache.log4j.Logger;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
```

```
import org.apache.hadoop.fs.Path;

public class XMLMapper extends Mapper<LongWritable, Text, Text, Text
   > {
   private final Logger logger = Logger.getLogger(XMLMapper.class);

   // stop words to eliminate while extracting text
   private static final ArrayList<String> stopwords = new ArrayList
      <>(Arrays.asList(
   "a", "as", "able", "about", "above", "according", "accordingly",
      "across", "actually", "after",
   "afterwards", "again", "against", "aint", "all", "allow", "allows
      ", "almost", "alone", "along",
   "already", "also", "although", "always", "am", "among", "amongst
      ", "an", "and", "another", "any",
   "anybody", "anyhow", "anyone", "anything", "anyway", "anyways", "
      anywhere", "apart", "appear",
   "appreciate", "appropriate", "are", "arent", "around", "as", "
      aside", "ask", "asking", "associated",
   "at", "available", "away", "awfully", "be", "became", "because",
      "become", "becomes", "becoming", "been",
   "before", "beforehand", "behind", "being", "believe", "below", "
      beside", "besides", "best", "better", "between",
   "beyond", "both", "brief", "but", "by", "cmon", "cs", "came", "
      can", "cant", "cannot", "cant", "cause", "causes",
    "certain", "certainly", "changes", "clearly", "co", "com", "come
       ", "comes", "concerning", "consequently",
```

```
"consider", "considering", "contain", "containing", "contains",
   "corresponding", "could", "couldnt", "course",
"currently", "definitely", "described", "despite", "did", "didnt
   ", "different", "do", "does", "doesnt", "doing",
"dont", "done", "down", "downwards", "during", "each", "edu", "
   eg", "eight", "either", "else", "elsewhere",
"enough", "entirely", "especially", "et", "etc", "even", "ever",
    "every", "everybody", "everyone", "everything",
"everywhere", "ex", "exactly", "example", "except", "far", "few
   ", "ff", "fifth", "first", "five", "followed", "following",
"follows", "for", "former", "formerly", "forth", "four", "from",
    "further", "furthermore", "get", "gets", "getting", "given",
"gives", "go", "goes", "going", "gone", "got", "gotten", "
   greetings", "had", "hadnt", "happens", "hardly", "has", "
   hasnt",
"have", "havent", "having", "he", "hes", "hello", "help", "hence
   ", "her", "here", "heres", "hereafter", "hereby", "herein",
"hereupon", "hers", "herself", "hi", "him", "himself", "his", "
   hither", "hopefully", "how", "howbeit", "however", "i", "id",
"ill", "im", "ive", "ie", "if", "ignored", "immediate", "in", "
   inasmuch", "inc", "indeed", "indicate", "indicated", "
   indicates",
"inner", "insofar", "instead", "into", "inward", "is", "isnt", "
   it", "itd", "itll", "its", "its", "itself", "just", "keep",
"keeps", "kept", "know", "knows", "known", "last", "lately", "
   later", "latter", "latterly", "least", "less", "lest", "let",
```

"lets", "like", "liked", "likely", "little", "look", "looking",
"looks", "ltd", "mainly", "many", "may", "maybe", "me", "mean
",
"meanwhile", "merely", "might", "more", "moreover", "most", "
mostly", "much", "must", "my", "myself", "name", "namely",
"nd", "near", "nearly", "necessary", "need", "needs", "neither",
"never", "nevertheless", "new", "next", "nine", "no",
"nobody", "non", "none", "noone", "nor", "normally", "not", "
nothing", "novel", "now", "nowhere", "obviously", "of",
"off", "often", "oh", "ok", "okay", "old", "on", "once", "one",
"ones", "only", "onto", "or", "other", "others", "otherwise",
"ought", "our", "ours", "ourselves", "out", "outside", "over", "
overall", "own", "particular", "particularly", "per",
"perhaps", "placed", "please", "plus", "possible", "presumably",
"probably", "provides", "que", "quite", "qv",
"rather", "rd", "re", "really", "reasonably", "regarding", "
regardless", "regards", "relatively", "respectively",
"right", "said", "same", "saw", "say", "saying", "says", "second
", "secondly", "see", "seeing", "seem", "seemed",
"seeming", "seems", "seen", "self", "selves", "sensible", "sent
", "serious", "seriously", "seven", "several", "shall",
"she", "should", "shouldnt", "since", "six", "so", "some", "
somebody", "somehow", "someone", "something", "sometime",
"sometimes", "somewhat", "somewhere", "soon", "sorry", "
specified", "specify", "specifying", "still", "sub", "such",
"sup", "sure", "ts", "take", "taken", "tell", "tends", "th", "
than", "thank", "thanks", "thanx", "that", "thats",

```
 "thats", "the", "their", "theirs", "them", "themselves", "then",
     "thence", "there", "theres", "thereafter", "thereby",
 "therefore", "therein", "theres", "thereupon", "these", "they",
    "theyd", "theyll", "theyre", "theyve", "think", "third",
 "this", "thorough", "thoroughly", "those", "though", "three", "
    through", "throughout", "thru", "thus", "to", "together",
 "too", "took", "toward", "towards", "tried", "tries", "truly", "
    try", "trying", "twice", "two", "un", "under",
 "unfortunately", "unless", "unlikely", "until", "unto", "up", "
    upon", "us", "use", "used", "useful", "uses", "using",
 "usually", "value", "various", "very", "via", "viz", "vs", "want
    ", "wants", "was", "wasnt", "way", "we", "wed", "well",
 "were", "weve", "welcome", "well", "went", "were", "werent", "
    what", "whats", "whatever", "when", "whence", "whenever",
 "where", "wheres", "whereafter", "whereas", "whereby", "wherein
    ", "whereupon", "wherever", "whether", "which", "while",
  "whither", "who", "whos", "whoever", "whole", "whom", "whose",
     "why", "will", "willing", "wish", "with", "within",
  "without", "wont", "wonder", "would", "would", "wouldnt", "yes
     ", "yet", "you", "youd", "youll", "youre", "youve",
  "your", "yours", "yourself", "yourselves", "zero"));
private SecretKey KEY;
private IvParameterSpec IV;

// mapper implementation for indexing application
@Override
```

```
public void map(LongWritable key, Text value, Context context)
   throws IOException, InterruptedException {
   try {
      List<String> written = new ArrayList<>();
      InputStream is = new ByteArrayInputStream(value.toString()
         .getBytes());
      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
         newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse(is);
      String w;
      StringTokenizer itr;
      doc.getDocumentElement().normalize();

      // Abstract files
      NodeList AbstractList = doc.getElementsByTagName("doc");
      // All other files
      NodeList pageList = doc.getElementsByTagName("page");

      if(AbstractList.getLength() > 0) {
         for (int i = 0; i < AbstractList.getLength(); i++) {
            Node nNode = AbstractList.item(i);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
               Element eElement = (Element) nNode;
               String Title = eElement.getElementsByTagName("
                  title").item(0).getTextContent();
```

```
itr = new StringTokenizer(eElement.
   getElementsByTagName("abstract").item(0).
   getTextContent().replaceAll("[^A-Za-z\\s+]",
   "").toLowerCase());
while(itr.hasMoreTokens()) {
   w = itr.nextToken().trim();
   if(!stopwords.contains(w) && w.length() > 2
      && !written.contains(w)) {
      written.add(w);
      context.write(new Text(w), new Text(Title
         ));
   }
}
NodeList links = eElement.getElementsByTagName("
   links").item(0).getChildNodes();
for (int temp2 = 0; temp2 < links.getLength();
   temp2++) {
   Node sub = links.item(temp2);
   if(sub.getNodeType() == Node.ELEMENT_NODE) {
      Element e = (Element) sub;
      itr = new StringTokenizer(e.
         getElementsByTagName("anchor").item(0)
         .getTextContent().replaceAll("[^A-Za-z
         \\s+]", "").toLowerCase());
      while(itr.hasMoreTokens()) {
         w = itr.nextToken().trim();
```

```
                        if(!stopwords.contains(w) && w.length
                            () > 2 && !written.contains(w)) {
                            written.add(w);
                            context.write(new Text(w), new
                                Text(Title));
                        }
                    }
                }
            }
            written.clear();
        }
    }
    if(pageList.getLength() > 0) {
        for (int i = 0; i < pageList.getLength(); i++) {
            Node nNode = pageList.item(i);
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element eElement = (Element) nNode;
                String Title = eElement.getElementsByTagName("
                    title").item(0).getTextContent();
                NodeList revision = eElement.
                    getElementsByTagName("revision");
                for (int j = 0; j < revision.getLength(); j++) {
                    Node sub = revision.item(j);
                    if(sub.getNodeType() == Node.ELEMENT_NODE){
                        Element e = (Element) sub;
```

```
if(e.getElementsByTagName("comment").
   getLength() > 0 && e.
   getElementsByTagName("comment") !=
   null) {
   itr = new StringTokenizer(e.
      getElementsByTagName("comment").
      item(0).getTextContent());
   while(itr.hasMoreTokens()) {
      w = itr.nextToken().trim();
      if(!stopwords.contains(w) && w.
         length() > 2 && !written.
         contains(w)) {
         written.add(w);
         context.write(new Text(w), new
            Text(Title));
      }
   }
}
if(e.getElementsByTagName("text").
   getLength() > 0 && e.
   getElementsByTagName("text") != null)
   {
   itr = new StringTokenizer(e.
      getElementsByTagName("text").item
      (0).getTextContent());
   while(itr.hasMoreTokens()) {
      w = itr.nextToken().trim();
```

```
                                    if(!stopwords.contains(w) && w.
                                        length() > 2 && !written.
                                        contains(w)) {
                                        written.add(w);
                                        context.write(new Text(w), new
                                            Text(Title));
                                    }
                                }
                            }
                        }
                    }
                }
                written.clear();
            }
        } catch (ParserConfigurationException | SAXException |
            IOException | DOMException e) {
            //
        }
    }

    public void readKey(String path, FileSystem fs) throws
        FileNotFoundException, IOException{
        FSDataInputStream stream = fs.open(new Path(path));
        int keySize = (int) fs.getFileStatus(new Path(path)).getLen()
            - 16;
        try {
```

```
            byte[] key = new byte[keySize];
            byte[] iv = new byte[16];
            stream.read(key, 0, keySize);
            stream.read(iv, 0, 16);
            KEY = new SecretKeySpec(key, 0, keySize, "AES");
            IV = new IvParameterSpec(iv);

        } finally {
            stream.close();
        }
    }

    private static byte[] encrypt(byte[] plainText, SecretKey Key,
        IvParameterSpec iv) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
        return cipher.doFinal(plainText);
    }

    public static byte[] encryptString(String plain, SecretKey Key,
        IvParameterSpec iv){
        try {
            byte[] cipher = encrypt(plain.getBytes(), Key, iv);
            return cipher;
        } catch(Exception e) {
            System.err.println(e.getMessage());
        }
```

```
    return new byte[1];
}

public static byte[] hashBytes(byte[] data) throws
    NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(data);
    return hash;
}

public static String decryptString(byte[] cipher, SecretKey Key,
    IvParameterSpec iv){
    try {
        byte[] decrypted = decrypt(cipher, Key, iv);
        String decipher = new String(decrypted);
        return decipher;
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
    return "";
}

private static byte[] decrypt(byte[] cipherText, SecretKey Key,
    IvParameterSpec iv) throws Exception{
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.DECRYPT_MODE, Key , iv);
    return cipher.doFinal(cipherText);
```

```
    }
}

/*
 * XMLReducer.java
 */
package xmlinput;

import com.sun.istack.logging.Logger;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

// reducer implementation for MapReduce application
public class XMLReducer extends Reducer<Text, Text, Text, Text> {
```

```
private final Logger logger = Logger.getLogger(XMLReducer.class);
private SecretKey KEY;
private IvParameterSpec IV;

@Override
protected void setup(Context context) throws IOException,
   InterruptedException, UnsupportedEncodingException {
   if (context.getCacheFiles() != null && context.getCacheFiles
      ().length > 0) {
      readKey("key.dat", FileSystem.get(context.getConfiguration
         ()));
   }
}

@Override
public void reduce(Text key, Iterable<Text> values, Context
   context) throws IOException, InterruptedException {
   context.getCounter(XMLInput.OUTPUT_COUNTER.
      OUTPUT_RECORDS_COUNTER).increment(1);
   boolean firstKey = true;
   byte[] enc = encryptString(key.toString(), KEY, IV);
   Text encWord = new Text("");
   try {
      encWord = new Text(new String(Base64.encodeBase64(
         hashBytes(enc))));
   } catch (NoSuchAlgorithmException ex) {
```

```
            java.util.logging.Logger.getLogger(XMLReducer.class.
                getName()).log(Level.SEVERE, null, ex);
        }
        for (Text value : values) {
            context.write(firstKey ? encWord : null, new Text(new
                String(Base64.encodeBase64(encryptString(value.toString
                (), KEY, IV)))));
            firstKey = false;
        }
    }

    public void readKey(String path, FileSystem fs) throws
        FileNotFoundException, IOException{
        FSDataInputStream stream = fs.open(new Path(path));
        int keySize = (int) fs.getFileStatus(new Path(path)).getLen()
             - 16;
        try {
            byte[] key = new byte[keySize];
            byte[] iv = new byte[16];
            stream.read(key, 0, keySize);
            stream.read(iv, 0, 16);
            KEY = new SecretKeySpec(key, 0, keySize, "AES");
            IV = new IvParameterSpec(iv);

        } finally {
            stream.close();
        }
```

```
}

private static byte[] encrypt(byte[] plainText, SecretKey Key,
   IvParameterSpec iv) throws Exception {
   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
   cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
   return cipher.doFinal(plainText);
}

public static byte[] encryptString(String plain, SecretKey Key,
   IvParameterSpec iv){
   try {
      byte[] cipher = encrypt(plain.getBytes(), Key, iv);
      return cipher;
   } catch(Exception e) {
      System.err.println(e.getMessage());
   }
   return new byte[1];
}

public static byte[] hashBytes(byte[] data) throws
   NoSuchAlgorithmException {
   MessageDigest digest = MessageDigest.getInstance("SHA-256");
   byte[] hash = digest.digest(data);
   return hash;
}
```

```
    public static String decryptString(byte[] cipher, SecretKey Key,
        IvParameterSpec iv){
        try {
            byte[] decrypted = decrypt(cipher, Key, iv);
            String decipher = new String(decrypted);
            return decipher;
        } catch(Exception e) {
            System.err.println(e.getMessage());
        }
        return "";
    }

    private static byte[] decrypt(byte[] cipherText, SecretKey Key,
        IvParameterSpec iv) throws Exception{
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, Key , iv);
        return cipher.doFinal(cipherText);
    }
}

/*
 * StreamingTextOutputFormat.java
 */
package xmlinput;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
```

```
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.RecordWriter;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.util.ReflectionUtils;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.log4j.Logger;

// streams output of application without having to store it in
   memory
public class StreamingTextOutputFormat<K, V> extends
   TextOutputFormat<K, V> {
   protected static class StreamingLineRecordWriter<K, V> extends
      RecordWriter<K, V> {
      private static final String utf8 = "UTF-8";
      private static final byte[] newline;
      private final Logger log = Logger.getLogger(
         StreamingLineRecordWriter.class);
      static {
         try {
            newline = "\n".getBytes(utf8);
```

```
    } catch (UnsupportedEncodingException uee) {
        throw new IllegalArgumentException("can't find " + utf8
            + " encoding");
    }
}
protected DataOutputStream out;
private final byte[] keyValueSeparator;
private final byte[] valueDelimiter;
private boolean dataWritten = false;

public StreamingLineRecordWriter(DataOutputStream out, String
     keyValueSeparator, String valueDelimiter) {
    this.out = out;
    try {
        this.keyValueSeparator = keyValueSeparator.getBytes(
           utf8);
        this.valueDelimiter = valueDelimiter.getBytes(utf8);
    } catch (UnsupportedEncodingException uee) {
        throw new IllegalArgumentException("can't find " + utf8
            + " encoding");
    }
}

public StreamingLineRecordWriter(DataOutputStream out) {
    this(out, ",", ",");
}
```

```
private void writeObject(Object o) throws IOException {
    if (o instanceof Text) {
        Text to = (Text) o;
        out.write(to.getBytes(), 0, to.getLength());
    } else {
        out.write(o.toString().getBytes(utf8));
    }
}

@Override
public synchronized void write(K key, V value) throws
    IOException {
    boolean nullKey = (key == null || key instanceof
        NullWritable);
    boolean nullValue = (value == null || value instanceof
        NullWritable);
    if (nullKey && nullValue) {
        return;
    }

    if (!nullKey) {
        // if we've written data before, append a new line
        if (dataWritten) {
            out.write(newline);
        }
        // write out the key and separator
        writeObject(key);
```

```
            out.write(keyValueSeparator);
        } else if (!nullValue) {
            // write out the value delimiter
            out.write(valueDelimiter);
        }
        // write out the value
        writeObject(value);
        // track that we've written some data
        dataWritten = true;
    }

    @Override
    public synchronized void close(TaskAttemptContext context)
        throws IOException {
        // if we've written out any data, append a closing newline
        if (dataWritten) {
            out.write(newline);
        }
        out.close();
    }
}

@Override
public RecordWriter<K, V> getRecordWriter(TaskAttemptContext job)
     throws IOException {
    Configuration conf = job.getConfiguration();
    boolean isCompressed = getCompressOutput(job);
```

```
String keyValueSeparator = conf.get("mapreduce.output.
   textoutputformat.separator", ",");
String valueDelimiter = conf.get("mapreduce.output.
   textoutputformat.delimiter", ",");
CompressionCodec codec = null;
String extension = "";
if(isCompressed) {
       Class<? extends CompressionCodec> codecClass =
          getOutputCompressorClass(job, GzipCodec.class);
       // create the named codec
       codec = (CompressionCodec) ReflectionUtils.newInstance(
          codecClass, conf);
       extension = codec.getDefaultExtension();
}
Path file = getDefaultWorkFile(job, extension);
FileSystem fs = file.getFileSystem(conf);
if (!isCompressed) {
       FSDataOutputStream fileOut = fs.create(file, false);
       return new StreamingLineRecordWriter<>(fileOut,
          keyValueSeparator, valueDelimiter);
} else {
       // build the filename including the extension
       FSDataOutputStream fileOut = fs.create(file, false);
       return new StreamingLineRecordWriter<>(new
          DataOutputStream(
                     codec.createOutputStream(fileOut)),
                        keyValueSeparator,
```

```
                            valueDelimiter);
        }
    }
}

/*
 * HadoopSearchIndex.java
 */
package hadoopsearchindex;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.security.NoSuchAlgorithmException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class HadoopSearchIndex {
    public enum OUTPUT_COUNTER {
        OUTPUT_RECORDS_COUNTER
    };

    // driver for the MapReduce index search application
```

```
public static void main(String[] args) throws IOException,
   URISyntaxException, InterruptedException,
   ClassNotFoundException, NoSuchAlgorithmException {
   Configuration conf = new Configuration();
   String[] otherArgs = new GenericOptionsParser(conf, args).
      getRemainingArgs();

   if (otherArgs.length < 4 ) {
      System.err.println("Usage: Index <numReduce> <in> <out> <#
         queries> <q1> <q2> ... <qn> (use ! for not and | for or
         )");
      System.exit(2);
   }
   int numQueries = Integer.parseInt(otherArgs[3]);
   int numReduce = Integer.parseInt(otherArgs[0]);
   if (otherArgs.length != (4 + numQueries) ) {
      System.err.println("Missing queries");
      System.exit(2);
   }

   conf.setInt("numQueries", numQueries);
   for(int i = 4; i < (4 + numQueries); i++) {
      conf.set("q" + (i - 4), otherArgs[i]);
   }

   Job job = Job.getInstance(conf);
   job.setJobName("Query and decrypt");
```

```
        job.addCacheFile(new URI("key.dat#key"));
        job.setJarByClass(HadoopSearchIndex.class);
        job.setMapperClass(HadoopSearchMapper.class);
        job.setCombinerClass(HadoopSearchCombiner.class);
        job.setReducerClass(HadoopSearchReducer.class);
        job.setNumReduceTasks(numReduce);

        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(TextArrayWritable.class);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(TextArrayWritable.class);

        FileInputFormat.addInputPath(job, new Path(otherArgs[1]) {});
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));

        job.waitForCompletion(true);
        System.out.print(numReduce + "," + (job.getFinishTime() - job
           .getStartTime())/1000 + ",");
        System.out.println(job.getCounters().findCounter("
           hadoopsearchindex.HadoopSearchIndex$OUTPUT_COUNTER", "
           OUTPUT_RECORDS_COUNTER").getValue());
    }
}

/*
 * HadoopSearchMapper.java
 */
```

```
package hadoopsearchindex;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.commons.codec.binary.Base64;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

// mapper implementation for MapReduce index search application
public class HadoopSearchMapper extends Mapper<Object, Text,
   IntWritable, TextArrayWritable> {
   private SecretKey KEY;
```

```
private IvParameterSpec IV;
private final Map<String, Integer> queries = new HashMap<>();
private int numQueries;

@Override
protected void setup(Context context) throws IOException,
   InterruptedException, UnsupportedEncodingException {
   int typeQ;
   readKey("key.dat", FileSystem.get(context.getConfiguration())
      );
   Configuration conf = context.getConfiguration();
   numQueries = conf.getInt("numQueries", 1);
   for(int i = 0; i < numQueries; i++) {
      String q = conf.get("q" + i);
      if(q.startsWith("!")) typeQ = 1; // NOT Query
      else if(q.startsWith("|")) typeQ = 2; // OR Query
      else typeQ = 3; // AND Query
      String query = q.replaceAll("[^A-Za-z\\s+]", "").
         toLowerCase();
      try {
         String enc = new String(Base64.encodeBase64(hashBytes(
            encryptString(query, KEY, IV))));
         queries.put(enc, typeQ);
      } catch (NoSuchAlgorithmException ex) {
         //
      }
   }
```

```
}

@Override
protected void map(Object key, Text value, Context context)
   throws IOException, InterruptedException {
   String line = value.toString();
   String[] splitLine = line.split(",");
   String w = splitLine[0];
   if(queries.containsKey(w)){
      String[] res = Arrays.copyOfRange(splitLine, 1, splitLine.
         length);
      Text[] finalFiles = new Text[res.length];
      for(int i = 0; i < res.length; i++) {
         finalFiles[i] = new Text(res[i]);
      }
      context.write(new IntWritable(queries.get(w)), new
         TextArrayWritable(finalFiles));
   }
}

public void readKey(String path, FileSystem fs) throws
   FileNotFoundException, IOException{
   FSDataInputStream stream = fs.open(new Path(path));
   int keySize = (int) fs.getFileStatus(new Path(path)).getLen()
       - 16;
   try {
      byte[] key = new byte[keySize];
```

```
            byte[] iv = new byte[16];
            stream.read(key, 0, keySize);
            stream.read(iv, 0, 16);
            KEY = new SecretKeySpec(key, 0, keySize, "AES");
            IV = new IvParameterSpec(iv);

        } finally {
            stream.close();
        }
    }

    private static byte[] encrypt(byte[] plainText, SecretKey Key,
        IvParameterSpec iv) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
        return cipher.doFinal(plainText);
    }

    public static byte[] encryptString(String plain, SecretKey Key,
        IvParameterSpec iv){
        try {
            byte[] cipher = encrypt(plain.getBytes(), Key, iv);
            return cipher;
        } catch(Exception e) {
            System.err.println(e.getMessage());
        }
        return new byte[1];
```

```
}

public static byte[] hashBytes(byte[] data) throws
   NoSuchAlgorithmException {
   MessageDigest digest = MessageDigest.getInstance("SHA-256");
   byte[] hash = digest.digest(data);
   return hash;
}

public static String decryptString(byte[] cipher, SecretKey Key,
   IvParameterSpec iv){
   try {
       byte[] decrypted = decrypt(cipher, Key, iv);
       String decipher = new String(decrypted);
       return decipher;
   } catch(Exception e) {
       System.err.println(e.getMessage());
   }
   return "";
}

private static byte[] decrypt(byte[] cipherText, SecretKey Key,
   IvParameterSpec iv) throws Exception{
   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
   cipher.init(Cipher.DECRYPT_MODE, Key , iv);
   return cipher.doFinal(cipherText);
}
```

```
}

/*
 * HadoopSearchCombiner.java
 */
package hadoopsearchindex;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashSet;
import java.util.Set;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
/**
 * combine the list of result for each type and decrypt
 */
public class HadoopSearchCombiner extends Reducer<IntWritable,
   TextArrayWritable, IntWritable, TextArrayWritable>{
   List<String> AndRes = new ArrayList<>();
   List<String> OrRes = new ArrayList<>();
   List<String> NotRes = new ArrayList<>();
   List<List<String>> tempAndRes = new ArrayList<>();
   List<List<String>> tempOrRes = new ArrayList<>();
   List<List<String>> tempNotRes = new ArrayList<>();
   private SecretKey KEY;
   private IvParameterSpec IV;

   // Finds difference between set A and set B (A - B) and returns a
       new list
   public static <T> List<T> Difference(Collection<T> A, Collection<
      T> B) {
      List<T> diff = new ArrayList<>(A);

      diff.removeAll(B);
      return diff;
   }


   // Finds the union between two collections and returns a new list
```

```
public static <T> List<T> union(Collection<T> list1, Collection<T
   > list2) {
   Set<T> set = new HashSet<>();

   set.addAll(list1);
   set.addAll(list2);

   return new ArrayList<>(set);
}

// Finds the intersection between two collections and returns a
   new list
public static <T> List<T> intersection(Collection<T> list1,
   Collection<T> list2) {
   List<T> list = new ArrayList<>();

   for (T t : list1) {
      if(list2.contains(t)) {

         list.add(t);
      }
   }

   return list;
}

@Override
```

```
protected void setup(Context context) throws IOException,
   InterruptedException, UnsupportedEncodingException {
   //if (context.getCacheFiles() != null && context.
      getCacheFiles().length > 0) {
      readKey("key.dat", FileSystem.get(context.getConfiguration
         ()));
   //}
}

@Override
protected void cleanup(Context context) throws IOException,
   InterruptedException {
   int ind = 0;
   Text[] finalFiles = new Text[1];
   String[] res;
   if(!tempNotRes.isEmpty()){
      // combine not operator results
      for(List<String> nQ : tempNotRes) {
         if(ind == 0)
         {
            NotRes = union(NotRes, nQ);
            ind++;
         }
         NotRes = intersection(NotRes, nQ);
      }

      res = new String[NotRes.size()];
```

```
    res = NotRes.toArray(res);
    finalFiles = new Text[NotRes.size()];
    for(int i = 0; i < res.length; i++) {
        finalFiles[i] = new Text(res[i]);
    }
    context.write(new IntWritable(1), new TextArrayWritable(
        finalFiles));
}

if(!tempOrRes.isEmpty()){
    // combine or operator results
    for(List<String> oQ : tempOrRes) {
        OrRes = union(OrRes, oQ);
    }
    res = new String[OrRes.size()];
    res = OrRes.toArray(res);
    finalFiles = new Text[OrRes.size()];
    for(int i = 0; i < res.length; i++) {
        finalFiles[i] = new Text(res[i]);
    }
    context.write(new IntWritable(2), new TextArrayWritable(
        finalFiles));
}
if(!tempAndRes.isEmpty()) {
    ind = 0;
    // combine and operator results
    for(List<String> aQ : tempAndRes) {
```

```
            if(ind == 0) {
                AndRes = union(AndRes, aQ);
                ind++;
            }
            else {
                AndRes = intersection(AndRes, aQ);
            }
        }
        res = new String[AndRes.size()];
        res = AndRes.toArray(res);
        finalFiles = new Text[AndRes.size()];
        for(int i = 0; i < res.length; i++) {
            finalFiles[i] = new Text(res[i]);
        }
        context.write(new IntWritable(3), new TextArrayWritable(
            finalFiles));
    }
}

@Override
protected void reduce(IntWritable key, Iterable<TextArrayWritable
    > values, Context context) throws IOException,
    InterruptedException{
    String[] vals;
    List<String> temp;
    for(TextArrayWritable value : values) {
        vals = value.toStrings();
```

```
            temp = new ArrayList<>();
            temp.addAll(Arrays.asList(vals));
            switch (key.get()) {
                case 1:
                    tempNotRes.add(temp);
                    break;
                case 2:
                    tempOrRes.add(temp);
                    break;
                case 3:
                    tempAndRes.add(temp);
                    break;
                default:
                    break;
            }
        }
    }

    public void readKey(String path, FileSystem fs) throws
        FileNotFoundException, IOException{
        FSDataInputStream stream = fs.open(new Path(path));
        int keySize = (int) fs.getFileStatus(new Path(path)).getLen()
             - 16;
        try {
            byte[] key = new byte[keySize];
            byte[] iv = new byte[16];
            stream.read(key, 0, keySize);
```

```
        stream.read(iv, 0, 16);
        KEY = new SecretKeySpec(key, 0, keySize, "AES");
        IV = new IvParameterSpec(iv);

    } finally {
        stream.close();
    }
}

private static byte[] encrypt(byte[] plainText, SecretKey Key,
    IvParameterSpec iv) throws Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
    return cipher.doFinal(plainText);
}

public static byte[] encryptString(String plain, SecretKey Key,
    IvParameterSpec iv){
    try {
        byte[] cipher = encrypt(plain.getBytes(), Key, iv);
        return cipher;
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
    return new byte[1];
}
```

```
    public static byte[] hashBytes(byte[] data) throws
       NoSuchAlgorithmException {
       MessageDigest digest = MessageDigest.getInstance("SHA-256");
       byte[] hash = digest.digest(data);
       return hash;
    }

    public static String decryptString(byte[] cipher, SecretKey Key,
       IvParameterSpec iv){
       try {
          byte[] decrypted = decrypt(cipher, Key, iv);
          String decipher = new String(decrypted);
          return decipher;
       } catch(Exception e) {
          System.err.println(e.getMessage());
       }
       return "";
    }

    private static byte[] decrypt(byte[] cipherText, SecretKey Key,
       IvParameterSpec iv) throws Exception{
       Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
       cipher.init(Cipher.DECRYPT_MODE, Key , iv);
       return cipher.doFinal(cipherText);
    }
}

/*
```

```
 * HadoopSearchReducer.java
 */
package hadoopsearchindex;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
// reducer implementation for MapReduce index search application
public class HadoopSearchReducer extends Reducer<IntWritable,
   TextArrayWritable, IntWritable, TextArrayWritable> {
   List<String> Result = new ArrayList<>();
   List<String> NotResult = new ArrayList<>();
   List<String> OrResult = new ArrayList<>();
   List<String> AndResult = new ArrayList<>();
   private SecretKey KEY;
   private IvParameterSpec IV;
   @Override
   protected void setup(Context context) throws IOException,
      InterruptedException, UnsupportedEncodingException {
          readKey("key.dat", FileSystem.get(context.getConfiguration
             ()));
   }

   // Finds difference between set A and set B (A - B) and returns a
       new list
   private static <T> List<T> Difference(Collection<T> A, Collection
      <T> B) {
      List<T> diff = new ArrayList<>(A);
      diff.removeAll(B);
      return diff;
   }


   // Finds the union between two collections and returns a new list
```

```
private static <T> List<T> union(Collection<T> list1, Collection<
    T> list2) {
    Set<T> set = new HashSet<>();
    set.addAll(list1);
    set.addAll(list2);

    return new ArrayList<>(set);
}

@Override
protected void cleanup(Context context) throws IOException,
    InterruptedException {
    if(NotResult.isEmpty()) {
        Result.addAll(AndResult);
    }
    else {
        Result.addAll(Difference(AndResult, NotResult));
    }
    // Find union between the results and or result
    if(!OrResult.isEmpty() && OrResult != null) {
        Result = union(Result, OrResult);
    }
    String[] res = new String[Result.size()];
    res = Result.toArray(res);
    Text[] finalFiles = new Text[Result.size()];
    for(int i = 0; i < Result.size(); i++) {
```

```
            finalFiles[i] = new Text(decryptString(Base64.decodeBase64
                (res[i]), KEY, IV));
        }
        context.getCounter(HadoopSearchIndex.OUTPUT_COUNTER.
            OUTPUT_RECORDS_COUNTER).setValue(Result.size());
        context.write(new IntWritable(Result.size()), new
            TextArrayWritable(finalFiles));
    }

    @Override
    public void reduce(IntWritable key, Iterable<TextArrayWritable>
        values, Context context) throws IOException,
        InterruptedException{
        // 1 is not
        // 2 is or
        // 3 is and
        String[] vals;
        for(TextArrayWritable value : values) {
            vals = value.toStrings();
            switch (key.get()) {
                case 1:
                    NotResult.addAll(Arrays.asList(vals));
                    break;
                case 2:
                    OrResult.addAll(Arrays.asList(vals));
                    break;
                case 3:
```

```
                AndResult.addAll(Arrays.asList(vals));
                break;
            default:
                break;
        }
    }
}

public void readKey(String path, FileSystem fs) throws
    FileNotFoundException, IOException{
    FSDataInputStream stream = fs.open(new Path(path));
    int keySize = (int) fs.getFileStatus(new Path(path)).getLen()
        - 16;
    try {
        byte[] key = new byte[keySize];
        byte[] iv = new byte[16];
        stream.read(key, 0, keySize);
        stream.read(iv, 0, 16);
        KEY = new SecretKeySpec(key, 0, keySize, "AES");
        IV = new IvParameterSpec(iv);
    } finally {
        stream.close();
    }
}

private static byte[] encrypt(byte[] plainText, SecretKey Key,
    IvParameterSpec iv) throws Exception {
```

```
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.ENCRYPT_MODE, Key, iv);
    return cipher.doFinal(plainText);
}

public static byte[] encryptString(String plain, SecretKey Key,
    IvParameterSpec iv){
    try {
        byte[] cipher = encrypt(plain.getBytes(), Key, iv);
        return cipher;
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
    return new byte[1];
}

public static byte[] hashBytes(byte[] data) throws
    NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(data);
    return hash;
}

public static String decryptString(byte[] cipher, SecretKey Key,
    IvParameterSpec iv){
    try {
        byte[] decrypted = decrypt(cipher, Key, iv);
```

```
            String decipher = new String(decrypted);
            return decipher;
        } catch(Exception e) {
            System.err.println(e.getMessage());
        }
        return "";
    }

    private static byte[] decrypt(byte[] cipherText, SecretKey Key,
        IvParameterSpec iv) throws Exception{
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, Key , iv);
        return cipher.doFinal(cipherText);
    }
}

/*
 * HashFiles.java
 */
package hashfiles;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BooleanWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

// driver class for MapReduce disk search application
public class HashFiles {
    public enum OUTPUT_COUNTER {
        OUTPUT_RECORDS_COUNTER
    };

    public static void readHashes(String hashFile, Configuration conf
        ) throws FileNotFoundException, IOException{
        try(BufferedReader buf = new BufferedReader(new FileReader(
            hashFile))) {
            int numHashes = Integer.parseInt(buf.readLine());
            conf.setInt("NumHashes", numHashes);
            for(int i = 0; i < numHashes; i++) {
                conf.set("hash" + i, buf.readLine());
            }
        }
    }

    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
```

```
String[] Args = new GenericOptionsParser(conf, args).
   getRemainingArgs();
conf.set("mapreduce.map.output.compress", "true");
conf.set("mapred.map.output.compress.codec", "org.apache.
   hadoop.io.compress.SnappyCodec");
conf.set("mapreduce.output.fileoutputformat.compress", "
   false");
if(Args.length < 3) {
    System.err.println("Usage: Search <hashes> <in> <out>")
       ;
    System.exit(2);
}
readHashes(Args[0], conf);
Job job = Job.getInstance(conf);
job.setJobName("Search Disk");
job.setJarByClass(HashFiles.class);
job.setMapperClass(HashFilesMapper.class);
job.setReducerClass(HashFilesReducer.class);
job.setInputFormatClass(WholeFileInputFormat.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(BooleanWritable.class);

FileInputFormat.addInputPath(job, new Path(Args[1]));
FileOutputFormat.setOutputPath(job, new Path(Args[2]));
job.waitForCompletion(false);
```

```
            System.out.println((job.getFinishTime() - job.getStartTime
                ())/1000 + "," + job.getCounters().findCounter("
                hashfiles.HashFiles$OUTPUT_COUNTER", "
                OUTPUT_RECORDS_COUNTER").getValue());
            System.exit(1);
        } catch (IOException | IllegalStateException |
            IllegalArgumentException | InterruptedException |
            ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }

}

/*
 * WholeFileInputFormat.java
 */
package hashfiles;

import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.io.IOUtils;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

// custom input format that reads chunks whole
public class WholeFileInputFormat extends FileInputFormat<Text, Text
   > {
   @Override
   protected boolean isSplitable(JobContext context, Path filename)
      {
          return false;
   }

   @Override
   public RecordReader<Text, Text> createRecordReader(
                 InputSplit inputSplit, TaskAttemptContext context)
                    throws IOException,
                 InterruptedException {
          WholeFileRecordReader reader = new WholeFileRecordReader()
             ;
          reader.initialize(inputSplit, context);
          return reader;
   }
```

```
public class WholeFileRecordReader extends RecordReader<Text,
   Text> {
   private FileSplit split;
   private Configuration conf;
   private final Text currKey = new Text();
   private final Text currValue = new Text();
   private boolean fileProcessed = false;

   @Override
   public void initialize(InputSplit split, TaskAttemptContext
      context)
         throws IOException, InterruptedException {
      this.split = (FileSplit) split;
      this.conf = context.getConfiguration();
   }

   @Override
   public boolean nextKeyValue() throws IOException,
      InterruptedException {
      if (fileProcessed) {
         return false;
      }
      currKey.set(split.getPath().getName());
      int fileLength = (int) split.getLength();
      byte[] result = new byte[fileLength];
      FileSystem fs = FileSystem.get(conf);
      FSDataInputStream in = null;
```

```
    try {
        in = fs.open(split.getPath());
        IOUtils.readFully(in, result, 0, fileLength);
        currValue.set(result, 0, fileLength);

    } finally {
        IOUtils.closeStream(in);
    }
    this.fileProcessed = true;
    return true;
}

@Override
public Text getCurrentKey() throws IOException,
        InterruptedException {
    return currKey;
}

@Override
public Text getCurrentValue() throws IOException,
        InterruptedException {
    return currValue;
}

@Override
public float getProgress() throws IOException,
    InterruptedException {
```

```
            return 0;
        }

        @Override
        public void close() throws IOException {
        }
    }
}

/*
 * HashFilesMapper.java
 */
package hashfiles;

import java.io.IOException;
import java.security.NoSuchAlgorithmException;
import org.apache.commons.net.util.Base64;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.log4j.Logger;

// mapper implementation for MapReduce disk search application
public class HashFilesMapper extends Mapper<Text, Text, Text, Text>{
    Logger logger = Logger.getLogger(HashFilesMapper.class);
    @Override
    protected void map(Text key, Text value, Context context) throws
        IOException, InterruptedException {
        try {
```

```
            byte[] hash = Base64.encodeBase64(AES.hashBytes(value.
               getBytes()));
            context.write(key, new Text(new String(hash)));
        } catch (NoSuchAlgorithmException ex) {
            //
        }
    }
}

/*
 * HashFilesReducer.java
 */
package hashfiles;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.BooleanWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.log4j.Logger;

// reducer implementation for MapReduce disk search application
public class HashFilesReducer extends Reducer<Text, Text, Text,
   BooleanWritable> {
```

```
List<String> hashes = new ArrayList<>();
Map<String, Boolean> hashmap = new HashMap<>();
Logger logger = Logger.getLogger(HashFilesReducer.class);
@Override
protected void setup(Context context) {
    Configuration conf = context.getConfiguration();
    int numHashes = conf.getInt("NumHashes", 0);
    for(int i = 0; i < numHashes; i++) {
        hashes.add(conf.get("hash" + i));
        hashmap.put(conf.get("hash" + i), false);
    }
}

@Override
protected void cleanup(Context context) throws IOException,
   InterruptedException{
    for(String hash : hashmap.keySet()) {
        if(hashmap.get(hash)) context.getCounter(HashFiles.
           OUTPUT_COUNTER.OUTPUT_RECORDS_COUNTER).increment(1);
        context.write(new Text(hash), new BooleanWritable(hashmap.
           get(hash)));
    }
}

@Override
public void reduce(Text key, Iterable<Text> values, Context
   context) throws IOException, InterruptedException {
```

```
        for(Text value : values){
            for(String hash : hashmap.keySet()) {
                if(!hashmap.get(hash)) {
                    if(hash.equals(value.toString())) {
                        hashmap.replace(hash, true);
                        break;
                    }
                }
            }
        }
    }
}
```

REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[3] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583–592, 2012.

[4] Diogo A. B. Fernandes, Liliana F. B. Soares, João V. Gomes, Mário M. Freire, and Pedro R. M. Inácio. Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170, 2014.

[5] Egemen K. Çetinkaya. A Brief Review of Security in Emerging Programmable Computer Networking Technologies. *IEEE-HKN Bridge Magazine*, 112(2):27–34, May 2016.

[6] Jose Moura and David Hutchison. Review and analysis of networking challenges in cloud computing. *Journal of Network and Computer Applications*, 60:113–129, January 2016.

[7] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark Clouds on the Horizon: Using Cloud Storage As Attack Vector and Online Slack Space. In *Proceedings of the 20th USENIX Conference on Security*, pages 1–11, San Francisco, CA, August 2011.

[8] DropShip. https://github.com/driverdan/dropship, 2011.

[9] Siani Pearson and Azzedine Benameur. Privacy, Security and Trust Issues Arising from Cloud Computing. In *Proceedings of the Second IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 693–702, Indianapolis, IN, November 2010.

[10] Lori M. Kaufman. Data Security in the World of Cloud Computing. *IEEE Security Privacy*, 7(4):61–64, July 2009.

[11] Mortada A. Aman and Egemen K. Çetinkaya. Towards Cloud Security Improvement with Encryption Intensity Selection. In *Proceedings of the 13th IEEE/IFIP International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 55–61, Munich, March 2017.

[12] Mortada A. Aman and Egemen K. Çetinkaya. DSB-SEIS: A Deduplicating Secure Backup System with Encryption Intensity Selection. In *Proceedings of the 4th ACM PODC Workshop on Distributed Cloud Computing (DCC)*, Chicago, IL, July 2016.

[13] Yujuan Tan, Hong Jiang, Dan Feng, Lei Tian, Zhichao Yan, and Guohui Zhou. SAM: A Semantic-Aware Multi-tiered Source De-duplication Framework for Cloud Backup. In *Proceedings of the 39th International Conference on Parallel Processing (ICPP)*, pages 614–623, San Diego, CA, September 2010.

[14] Yinjin Fu, Hong Jiang, Nong Xiao, Lei Tian, and Fang Liu. AA-Dedupe: An Application-Aware Source Deduplication Approach for Cloud Backup Services in the Personal Computing Environment. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pages 112–120, Austin, TX, September 2011.

[15] WikiMedia Dump. https://dumps.wikimedia.org/enwiki/latest/, 2017.

[16] CloudLab. http://cloudlab.us/, 2017.

[17] OpenStack. https://www.openstack.org/, 2017.

[18] Andrei Z. Broder. *Some applications of Rabin's fingerprinting method*, pages 143–152. Springer New York, New York, NY, 1993.

[19] Yujuan Tan, Hong Jiang, Dan Feng, Lei Tian, and Zhichao Yan. CABdedupe: A Causality-Based Deduplication Performance Booster for Cloud Backup Services. In *Proceedings of the IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pages 1266–1277, Anchorage, AK, May 2011.

[20] Jian Liu, N. Asokan, and Benny Pinkas. Secure Deduplication of Encrypted Data Without Additional Independent Servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 874–885, Denver, CO, October 2015.

[21] João Paulo and José Pereira. A Survey and Classification of Storage Deduplication Systems. *ACM Computing Surveys*, 47(1):11:1–11:30, July 2014.

[22] Dropbox. https://www.dropbox.com/, 2017.

[23] Google Drive. https://www.google.com/drive/, 2017.

[24] SipderOak One. https://spideroak.com/solutions/spideroak-one, 2017.

[25] Amazon S3. https://aws.amazon.com/s3/, 2017.

[26] Box. https://www.box.com/, 2017.

[27] Apple iCloud. http://www.apple.com/icloud/, 2017.

[28] Microsoft Cloud. https://cloud.microsoft.com/, 2017.

[29] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. Cumulus: Filesystem Backup to the Cloud. *ACM Transactions on Storage (TOS)*, 5(4):14:1–14:28, December 2009.

[30] Bacula. http://www.bacula.org/, 2017.

[31] Arthur Rahumed, Henry C. H. Chen, Yang Tang, Patrick P. C. Lee, and John C. S. Lui. A Secure Cloud Backup System with Assured Deletion and Version Control. In *Proceedings of the 40th International Conference on Parallel Processing Workshops (ICPPW)*, pages 160–167, Taipei City, September 2011.

[32] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[33] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465.

[34] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFCs 5785, 7230.

[35] Seny Kamara and Charalampos Papamanthou. Parallel and Dynamic Searchable Symmetric Encryption. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, pages 258–274, Okinawa, April 2013.

[36] Jin Li, Yan Kit Li, Xiaofeng Chen, Patrick P. C. Lee, and Wenjing Lou. A Hybrid Cloud Approach for Secure Authorized Deduplication. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1206–1216, May 2015.

[37] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control. In *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*, pages 85–90, Chicago, IL, November 2009.

[38] Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Proceedings of the 4th Theory of Cryptography Conference (TCC)*, pages 535–554, Amsterdam, February 2007.

[39] Tingjian Ge and Stan Zdonik. Answering Aggregation Queries in a Secure System Model. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 519–530, Vienna, September 2007.

[40] Bharath K. Samanthula, Yousef Elmehdwi, Gerry Howser, and Sanjay Madria. A Secure Data Sharing and Query Processing Framework via Federation of Cloud Computing. *Information Systems*, 48:196–212, March 2015.

[41] Eu-Jin Goh. Secure Indexes. Cryptology ePrint Archive, Report 2003/216, 2003.

[42] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[43] M. Bellare, A. Boldyreva, L. Knudsen, and C. Namprempre. On-line Ciphers and the Hash-CBC Constructions. *Journal of Cryptology*, 25(4):640–679, October 2012.

[44] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In *Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security*, pages 424–443, Bengaluru, December 2013.

[45] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Proceedings of the 33rd Annual Cryptology Conference*, pages 353–373, Santa Barbara, CA, August 2013.

[46] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):222–233, January 2014.

[47] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. Cryptology ePrint Archive, Report 2014/853, 2014.

[48] Melissa Chase and Emily Shen. Substring-Searchable Symmetric Encryption. *Proceedings on Privacy Enhancing Technologies*, 2015(2):263–281, June 2015.

[49] Changhui Hu, Lidong Han, and Siu Ming Yiu. Efficient and secure multi-functional searchable symmetric encryption schemes. *Security and Communication Networks*, 9(1):34–42, 2016.

[50] Shuguang Dai, Huige Li, and Fangguo Zhang. Memory leakage-resilient searchable symmetric encryption. *Future Generation Computer Systems*, 62:76–84, September 2016.

[51] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[52] Jeffrey Dean and Sanjay Ghemawat. MapReduce: A Flexible Data Processing Tool. *Communications of the ACM*, 53(1):72–77, January 2010.

[53] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. Efficient Big Data Processing in Hadoop MapReduce. *Proceedings of the VLDB Endowment*, 5(12):2014–2015, August 2012.

[54] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–24, February 2007.

[55] Richard McCreadie, Craig Macdonald, and Iadh Ounis. MapReduce indexing strategies: Studying scalability and efficiency. *Information Processing and Management*, 48(5):873 – 888, 2012.

[56] Robert Ricci, Eric Eide, and the CloudLab Team. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *The Magazine of USENIX*, 39(6):36–38, December 2014.

[57] Aditya Akella. Experimenting with Next-Generation Cloud Architectures Using CloudLab. *IEEE Internet Computing*, 19(5):77–81, 2015.

[58] Chae Hoon Lim and Pil Joong Lee. *Another Method for Attaining Security Against Adaptively Chosen Ciphertext Attacks*, pages 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[59] Ronald Cramer and Victor Shoup. *A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack*, pages 13–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

VITA

Mortada Aman was born in Qatif, Saudi Arabia. He developed a passion for technology as he was growing up. He knew early in his life that he would like to pursue a career in computers. In his last year of high school, he decided to attend Missouri University of Science and Technology for a Bachelor of Science in Computer Engineering.

After graduating in December 2015, he enrolled in Missouri University of Science and Technology's computer engineering Master of Science program. During his studies, he worked as graduate research and teaching assistant at the Electrical and Computer Engineering department at Missouri S&T. He received his Master of Science in computer engineering from Missouri S&T in December 2017. Upon graduation, he will work for Cerner Corporation.