CHAPTER 2

PROVISIONING MAP-REDUCE FOR IMPROVING SECURITY OF CLOUD DATA

2.1 INTRODUCTION

The advent of Web 2.0 has made organizations move towards cloud with computing models with Information Technology (IT) as a Service. The emerging fields in information technology focus on applications running over the Internet such as (Software as a Service (SaaS)) or hardware systems available in datacenters. In both the cases management of data and services rendered by datacenters are not fully trustworthy. Enterprises moving towards emerging cloud or cloud computing model while expanding their infrastructure, face risks due to security of their data (Chen et al 2010). The major attributes that need to be resolved in cloud are data integrity, recovery, privacy as well as legal issues in areas like regulatory compliance and auditing (Yeh et al 2012; Kaufman 2009).

Kermani & Masoleh(2012) proposed AES applications are power and resource constrained and require efficient hardware implementations. But, the Application-Specific Integrated Circuit (ASIC) architectures of building blocks of the AES algorithms are evaluated and optimized to identify the high-performance and low-power architectures. Bouillaguet et al (2012) presented attacks on up to four rounds of AES that require at most three known/chosen plaintexts. An attack applied to cryptanalyze an AES-
based stream cipher shows the best known plaintext attack on six-round AES. Kim (2012) devised a differential fault analysis on AES key schedule and proved the work need one pair of correct and faulty cipher texts for retrieving AES-128 bit key. El-Fotouh & Diepold (2008) proposed the narrow block methodology to encrypt data and proved XTS is slower than narrow block. But narrow block requires more time when dealing with large datasets (Keller & Hall 2011).

The storage encryption proposed in this research invokes confidentiality of data by encrypting it. Since encryption is a time consuming process, it is controlled by an efficient application of the process in parallel mode. Hadoop's Map-Reduce seems to be an attractive, cost effective solution for large scale data processing services like securing data in cloud through block encryption in parallel mode (Map-Reduce Apache Hadoop 2012). Liu & Baas (2013) explored different levels of parallelism while implementing AES (key expansion) and proved higher energy efficiency while running it in parallel mode.

The research discussed in this chapter, proposes a method to perform encryption in parallel through Map-Reduce using standard XTS-AES approach for securing cloud information. The contribution of this chapter is summarized as proposing an appropriate algorithm for encrypting large dataset, analyzing the best possible modes that support the encryption process to run in parallel and attaining parallelism of the encryption process through Map Reduce paradigm.

2.2 ENCRYPTION ALGORITHM

Towards the end of the 20th century many backdoors and flaws were found in the existing symmetric algorithm like DES (Data Encryption
Standard) which is vulnerable to brute force attacks because of its relatively small 56 bits key size. To meet better security standards, the US Government agency National Institute of Standards and Technology (NIST) selected Rijndael's Algorithm as Advanced Encryption Standard (AES) and it has now become anndael's Algorithm as Advanced Encryption Standard (AES) and it has now become an industry standard (Rewagad & Pawar 2013). The algorithm is capable of encrypting bulk data on top-end 32 bit and 64 bit CPUs. It has proved to be an efficient algorithm for encrypting all sorts of data deployed in cloud from text to audio and video (Schneier & Whiting 2000). The performance of AES algorithms vary on various CPU based on key size (which is the forthcoming work) and its performance in the current scenario is evaluated in this research. Parallelizing the block contents involves various modes of operation that handles fixed block and variable block encryption through single and different keys.

The encryption takes plaintext and a key as input, with the ciphertext being the output. The AES block cipher design is based on substitution permutation computation. AES is typically designed to accept three different key sizes like 128, 192 or 256 bits. The keylength represents the 32-bit words. The input, key, intermediate cipher and output have 128 bits. The number of rounds in AES is determined by its key size, which is AES-128 for 10 rounds, AES-192 for 12 rounds and AES-256 for 14 rounds. The AES algorithm is shown in the Figure 2.1. After the initial round key, each round function, except the final round contains four transformations which are subbytes, shiftrows, mixcolumns and addroundkey. The final round doesn’t include the mixcolumns operations. $N_b$ represents words added during each round.
CipherCode(bytein[4*Nb], byte out[4*Nb], word w[Nb*(Nr +1)])
begin
byte state[4*Nb]
state = in
AddRoundKey(state, w[0..Nb-1])
for round = 1 step 1 to Nr-1
Sub Bytes(state)
Shift Rows(state)
MixColumns(state)
AddRound Key
(state, w[round*Nb,(round+1)*Nb-l])
end for
Sub Bytes(state) ShiftRows(state)
AddRoundKey(state, w[Nr*Nb,(Nr+l)*Nb-l])
out = state
end

Figure 2.1 AES algorithm

SubBytes() Transformation

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the state using a substitution table (S-box).

ShiftRows() Transformation

In the ShiftRows() transformation, the bytes in the last three rows of the state are cyclically shifted over different numbers of bytes (offsets).

MixColumns() Transformation

The MixColumns() transformation operates on the state, column-by-column, treating each column as a four-term polynomial. The columns are
considered as polynomials over GF(2^8) and multiplied modulo x^4 + 1 with a fixed polynomial a(x), given by

\[ a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \]  \hspace{1cm} (2.1)

**AddRoundKey() Transformation**

In the AddRoundKey() transformation, a RoundKey is added to the state by a simple bitwise XOR operation. Each RoundKey consists of \(N_b\) words from the key schedule. \(N_b\) represents words each added into the columns of the state.

### 2.3 ENCRYPTION MODE

In this research, AES-128 bit keysizes is utilized for securing data at rest. The efficiency of cryptographic algorithm that runs in parallel mode can be improved by the introduction of several modes such as *Electronic Code Book (ECB)* (Dworkin2005) and *XTX (Xclusive XOR)* (Keller & Hall 2011). Martin (2010) suggested a mode that is widely supported parallel encryption mechanisms. *ECB* supports the kind of encryption in which the plaintext consists of a sequence of bit blocks like \(b_1, b_2, \ldots, b_n\) that is converted into corresponding cipher text blocks such as \(c_1, c_2, \ldots, c_n\) through the same key that acts on all the blocks. ECB mode suffers from the fact that if the message has repetitive elements in different blocks, all of them produce the same cipher text which is possible to substitute and find the actual plaintext. The other widely used IEEE 1619-2007 standard is *(XTS- TCB-CTS)* *(XTS- XEX Tweakable Block Cipher with Ciphertext Stealing)* mode in which the key material for *XTS-AES* consists of an *encryption key* as well as a *tweak key* that is used to incorporate the logical position of the data block into the
encryption. XTS outputs tend to produce independent outputs which lead to the parallelization of such methodology. XTS is a concrete instantiation of the class of tweakable block ciphers.

The XTS mode uses two keys K1 and K2. Since single key may weaken the security process two keys are introduced.

1. \( T \leftarrow \text{AES}(K_2, i) \otimes \alpha j \)
2. \( P P \leftarrow p \oplus T \)
3. \( CC \leftarrow \text{AES}(K_1, P P) \)
4. \( C \leftarrow CC \oplus T \)

The XTS mode allows parallelization and pipelining in cipher implementations. It enables the encryption of last incomplete block of data while other modes are not having this facility. The parallelization of cipher implementations is achieved through Map-Reduce algorithm which is discussed in the next section. Figure 2.2 describes the AES-XTS Mode.

![Figure 2.2 AES-XTS Mode](image)
2.4 HADOOP'S MAP-REDUCE PARADIGM

Google proposed Map-Reduce to simplify data processing on large clusters. Hadoop is the most popular open source implementation of the Map-Reduce framework developed by Yahoo! and the apache software foundation (Map-Reduce Apache Hadoop 2012). Map Reduce is deployed as a powerful data processing service over open source systems which has become increasingly popular for its parallel programming framework. Hadoop's Map-Reduce seems to be an attractive cost effective solution for large scale data processing services like securing data in cloud through block encryption (Pavlo et al 2009).

Map-Reduce can fit into any kind of environment like closed or open systems. The framework is designed for writing applications that rapidly process vast amounts of data during runtime on compute clusters. The code automatically partitions input data, perform scheduling, monitoring and has the fault tolerance mechanism through which it executes the failed tasks in a commodity of large cluster machines(Dean& Ghemawat 2008; Michael Stonebraker et al 2010).

2.4.1 Definition of Map-Reduce Programming

Map-Reduce is designed on a simple model with two keys such as map and reduce derived from functional programming languages (Map-Reduce Apache Hadoop 2012). For any kind of application to run using Map-Reduce model, the input should be a set of key value pairs and the mapper produces an intermediate key pair value that is sent to the reducer to carry out the rest of the application process. The explanation includes defining mappers and reducers. A description of how the system executes these two functions is also presented. The fundamental unit of data in Map-Reduce computations is the <key; value> pair, where keys and values are always just binary strings.
Theorem 2.1. A mapper is a randomized function that takes as input one ordered \(<key; value>\) pair of binary strings. As output the mapper produces a finite multiset of new key; value pairs. It is important that the mapper operates on one \(<key; value>\) pair at a time.

Theorem 2.2. A reducer is a randomized function that takes as input a binary string which is the key, and a sequence of values \(v_1, v_2\) which are also binary strings. As output, the reducer produces a multiset pair of binary strings \(<k_1; v_1>, <k_2; v_2>, <k_3; v_3>…\). The key in the output tuples is identical to the key in the input tuple.

2.4.2 Parallelizing Encryption

As stated before, the main benefit of this programming paradigm is the ease of parallelization. Since each mapper \(\mu_t\) operates only on one tuple at a time, the system can have many instances of \(\mu_t\) operating on different tuples in \(U_{r-1}\) in parallel. After the map step, the system partitions the set of tuples output throughout the instances of \(\mu_t\) which are created based on their key. That is, part i of the partition has all key; value pairs that have key \(k_i\). Since reducer \(\rho_t\) only operates on one part of this partition, the system based on the application creates many instances of \(\rho_r\) running on different parts in parallel. Data will be stored as contiguous blocks and every block is represented by unique \(block id\) (I0, I1, I2...I_{p-1}). A Map-Reduce includes set of mappers (\(M_1,M_2...M_r\)) and reducers (\(R_1,R_2...R_r\)). The input is given to mapper in the form of \(<block id, object>\). This object is the content of the HDFS block or the data stored in the corresponding \(block id\). During encryption the mapper and the reducer function based on the key, value pair as discussed above.
2.4.2.1 Execution of mapper

Block \(<I_{r-1}, \text{object\_id}_{i}>\) is given to mapper \(M_r\). The mapper will generate the corresponding encrypted output \(I'R\) and send it to the reducer \(R_r\) let

\[ W = \{<I_0, \text{object}_1>, <I_1, \text{object}_2>, <I_2, \text{object}_3>, ..., <I_{n-1}, \text{object}_n>\} \]

It can be generalized as

\[ I'_r = I_{r, i} \in W(\text{blockid, object}) \quad Mr(\text{<blockid, Enc-comobject>}) \]

2.4.2.2 Execution of reducer

The collected outputs from various mappers are written to the disk in the sequential order \((I_1, M_2, ..., M_r, I'_1, I'_2, ..., I'_n)\). In the proposed work the algorithm is designed in such a way for securing large data sets by performing block encryption followed by compression under each mapper and combining the result and storing it on HDFS (Hadoop Distributed File System). The Mapper reads block of equal size which can be optimized based on the available free nodes in the cloud environment as shown below:

\[ \text{Block Size} = \frac{\text{Input Data Size}}{\text{Number of nodes}} \quad (2.2) \]

Figure 2.3 shows how HDFS (Hadoop Distributed File System) contents got assigned to mapper where mapper performs encryption followed by compression and the reducer is used to write the contents on to HDFS (i.e., output). Though HDFS supports record level and block level compression on input data in the proposed method after encryption under each mapper a compression through gzip is done inorder to reduce the storage space.
2.5 RESULTS AND DISCUSSION

2.5.1 Experimental Setup

The AES algorithm running under Map-Reduce is tested on a campus environment that is created as a HadoopTestbed. This testbed comprised of a 32 node cluster each of which has a Intel Xeon 1.6Ghz processor with 500 GB of local storage running on Hadoop 0.20. The initial
tests are made with earlier versions of Hadoop 19. The testbed for protecting data in cloud environment is experimented with series of small datasets. The experimental results showed that encryption followed by compression mechanism using such a parallel technique required less time in the order of secs for large datasets. Experimental results are explained with the time lapsed for running the same process with reducer and without reducer. A comparison graph is generated for encrypting different datasets and for different types of files like audio, video, text and image. The testbed is being utilized to perform encryption followed by compression while storing different datasets as in the cloud environment.

### 2.5.1.1 Modes of parallelism with AES

Figure 2.4 AES encryption with ECB mode and XTS mode. The results prove that the time difference between the two modes are very less which implies that AES with XTS is adaptable for protecting data in cloud. While performing encryption using XTS mode, it produced a different output for the same data from two different mappers whereas EBC produced the same cipher text from different mappers. Such a concept is found to be relatively insecure. The time difference for 15 GB text data is 3 minutes which indicates that XTS with AES suits for securing large datasets in cloud environment.
Figure 2.4 AES encryption with ECB and XTS mode

2.5.1.2 Encryption with and without reducer

Figure 2.5 shows the performance of Map-Reduce Algorithm using AES (under XTS Mode) with reducer and without reducer. When it is used without reducer all the nodes of the cluster are assigned as mapper and hence increases the speed of encryption followed by compression process.
2.5.1.3 AES+XTS encryption

Figure 2.6 gives the evaluation of the proposed approach for different datasets. It is inferred from the results generated that the proposed approach shows remarkable difference for large size of text, audio, video and image files than for small size user generated content files. It is identified that the time taken for encrypting text is relatively high and that for image is less. Audio and video file required relatively equal time for performing the process.

![Figure 2.6 AES+XTS on different datasets](image)

2.5.1.4 Compression

From the previous experiment, it is concluded that without reducer the performance of the algorithm is better. So, compression on large datasets is experimented with a setup where all the nodes of the cluster act as mapper. Figure 2.7 showed the result that encryption with compression takes relatively less time. From the experiment with various datasets the compression ratio obtained for text and image data are in the order of 1:10 and 1:2. But for audio and video files, no remarkable difference in the ratio is noticed.
Figure 2.7 Compression on different dataset

2.6 SUMMARY

In this chapter, protecting cloud data through encryption and to efficiently utilize the resources of cloud, compression using Map-Reduce model is proposed and implemented. The goal is derived with a view to provide improved level of security through Map-Reduce framework that is being widely used in cloud world in recent years. This model is designed with 1) ECB and XTS modes that support parallelism, 2) XTS mode with AES running under Mapper and Reducer inorder to provide security to the users data in and 3) the evaluation of the performance of the entire framework with and without using reducer. Finally the hybrid framework is tested for different datasets and the evaluation is done. It is clear from the results that AES with XTS mode running under Mapper gives better result than using mapper and reducer together. The experimental result of this proposed technique shows that encryption methodology achieves relatively less time for large datasets in a cluster of machines. The compression is done at the expense of time in order to utilize the resources of the storage environment. The next chapter discusses about securing the MapReduce process.