Secure Distributed Storage System

Snehil Suresh Wakchaure, Simrit Kaur Arora

Abstract—Secure Distributed Storage techniques are gaining importance with the increase of data centers, mobile devices and the need to save encrypted data in a distributed environment. Transformations such as Encryption [1, 10, 15], Secret sharing [2, 5, 20], Erasure Coding [3] are used while implementing security and Error correction techniques while implementing secure distributed storage systems. Secure Mobile Distributed File system (MDFS) by Huchton [4] is an example of an implementation of such a system. The current implementation schemes of distributed storage system as mentioned in the report by GMU, CERG [5] are used in our system, which are implemented on the PC platform and for Android devices. Previous implementations of such a system had a weakness in the sense that the security of the systems really depended on the security of the underlying Encryption algorithm e.g. AES. In our scheme, we estimate the performance of the system with All or Nothing Transform (AONT) [6,7,8] included in the system in order to eradicate the security weakness. No documented implementation of AONT existed according to previous survey, and hence our project implements the system on PC and Android and estimates the performance of the total scheme taking AONT execution time as a constant, as it is independent of n, t (number of fragments and threshold, as explained in Section II). We have developed a working implementation of such a system on PC and Android and evaluated the performance of various Erasure coding algorithms, Secret sharing algorithms and estimated performance results for the proposed scheme.

I. INTRODUCTION

A secure distributed storage system consists of four basic transformations, namely, Encryption [1, 10, 15], Secret Sharing [2, 5, 20], Erasure Coding [3, 5, 12] and All-Or-Nothing transform [6,7,8]. In this project, we implement a software system as shown in Section 3 of this report. We perform benchmarking and measure the execution times [11, 9] of Shamir’s Secret sharing technique [5] and that of Erasure coding techniques such as Reed Solomon Vandermonde, Cauchy good, Cauchy original [5]. We also estimate the coding time of the entire scheme from various measurements made in our software implementation of this scheme.

In Section II, we discuss the background knowledge needed to understand our implementation of the secure distributed system and we discuss some distributed storage basics, Advanced Encryption Standard (AES) [10, 15], Erasure Codes [5, 12], Shamir’s Secret sharing scheme [5], All-Or-Nothing Transform [6,7,8]. In Section III, we discuss the scheme of the implemented system. In Section IV, we discuss the devices used, the experimental setup, device limitations, application and implementation details. In Section V, we discuss various experimental results.

II. BACKGROUND

A. Distributed Storage Basics

Distributed storage techniques [4] involve storing data in distributed storage or network environments. Some of the transformations used in distributed storage systems are Encryption [1, 10, 15], Secret Sharing [2, 5, 20], Erasure Coding [3, 5, 12] and All-Or-Nothing transform [6,7,8]. Encryption allows incorporating security in the system, while Erasure codes are examples of error correcting code implementation which incorporate redundancy in the distributed data in order to allow complete recovery of data even after data loss of distributed data up to a certain threshold.

Example applications of distributed storage systems could be found in data centers, Local and network storage along with ubiquitous mobile environments. Google uses Bigtable and Mapreduce to store, access, and synchronize multiple copies of data across multiple data centers. [13] [14] There are many approaches to distributed storage in the data center scenario. In table 2.1, Four different projects are listed across the top. The vertical axis lists characteristics that can be used to differentiate each implementation. Each approach aims to solve a different problem and each has strengths and weaknesses that make it suitable for the given solution. Table 2.1 provides a quick comparison of different approaches to distributed storage. Pre-share key refers to whether the implementation requires a key to be pre-shared before encryption. Distributed storage techniques could be devised for mobile phone applications such as Android for storing data in a distributed fashion. Other example of distributed storage system is that which is used in data storage in physical hard disk drives.

![Table 2.1: Distributed Storage Comparison](image)
B. Advanced Encryption Standard (AES)

AES [15, 10, 1] is a Federal Information Processing Standard (FIPS) approved cryptographic block cipher standard used to encrypt and decrypt electronic data. It processes 128 bit data blocks using cipher keys with lengths of 128, 192, or 256 bits. AES encryption takes a plaintext message and a key to encrypt the plaintext message into an unintelligible stream called cipher text. AES decryption takes the cipher text and the same key to decrypt the cipher text into the original plaintext message [15]. C++ implementation of AES is available from the Crypto++ library. [16]

C. Erasure Codes

Erasure codes [3,5,12] are forward error correction (FEC) codes that translate some message M of length |M| into a coded message with a length greater than |M| such that M can be recovered from some subset of the coded message. In 1960, Reed and Solomon introduced a Maximum Distance Separable (MDS) algorithm [17]. An MDS erasure codes stores a message M in n fragments of size |M|/k such that any k fragments is sufficient to reconstruct M [18].

Figure 2.1 shows an example of an erasure coding where n = 5 and k = 3. The original file is encoded into 5 fragments. Fragments 3 and 4 are lost, but fragments 1, 2, and 5 remain. Since the threshold for recover is 3, there are sufficient fragments remaining to recover the original file.

Erasure codes have the desired threshold recovery property, meaning that k or more fragments out of n enable the reconstruction of M, but they do not offer security of the individual fragments. One of the requirements of MDFS is that any fragment mi or subset of fragments less than k is insufficient to reconstruct M or infer any information about M. Unfortunately, some of the fragments generated by Reed-Solomon encoding are just blocks of the original data. Therefore, if an adversary were to capture some of the fragments, they would be able to infer partial information about M, which does not meet the requirements of the system.

![Figure 2.1: (3,5) Erasure Code](image)

D. Shamir's Secret Sharing Scheme

Fragment destinations are not predetermined, so public key management would be incredibly difficult. Public key dissemination and management between arbitrary nodes is both a complex process and a likely point of failure. Furthermore, it would be an undesirable side effect to require the receiving node to decrypt the fragments and store them unencrypted on their physical memory. We would like to take advantage of the efficiency of symmetric encryption, without the complication added by public-key distribution.

Shamir’s Secret Sharing Algorithm is closely related to Reed-Solomon and other MDS erasure codes. First discussed by Adi Shamir in 1979, he explains how to distribute information using threshold security. In his example, he describes a situation where a company, with multiple authorized executives, would require at least three signatures to sign a corporate check. This system is an example of (3, n) threshold security.

Shamir’s algorithm has the properties of security and resiliency required of our implemented scheme, but is not efficient. It suffers from the fact that each fragment is larger than the original message. This would increase network activity to unacceptable levels for relatively small files.

E. All-Or-Nothing Transform

The concept of an All-or-Nothing Transform (AONT) [6,7,8] was introduced by Rivest [18] to increase the cost of brute force attacks on block ciphers without changing the key length. As defined in that paper, an AONT is an efficiently computable transformation f, mapping sequences of blocks (i.e., fixed length strings) to sequences of blocks, which has the following properties:

- Given all of f(x1,..., xn) = (y1,..., yn), it is easy to compute x1,..., xn.
- Given all but one of the blocks of the output (i.e., given y1,..., yj-1, yj+1, ... , yn for any 1 ≤ j ≤ n), it is infeasible to find out any information about any of the original blocks xi.

An AONT itself does not perform any encryption, since there is no secret key information involved. However, if its output is encrypted, block by block, with a block cipher, the resulting scheme will have the following interesting property: An adversary cannot find out any information about any block of the message without decrypting all the blocks of the cipher text. Now if the adversary attempts to do an exhaustive search for the key, she will need to perform ‘n’ decryptions before determining whether a given key is correct. Thus, the attack will be slowed down by a factor of ‘n’, without any change in the size of the secret key. This is particularly important in scenarios where the key length is constrained to be insecure or marginally secure. (e.g., because of export regulations)

III. Scheme of the implemented system

Some of the transformations used in distributed storage systems are Encryption [1, 10, 15], Secret Sharing [2, 5, 20], Erasure Coding [3, 5, 12] and All-Or-Nothing transform [6,7,8].

A. Encoder

Figure 3.1 shows the system diagram of the new proposed scheme for the Encoder part.
As shown in Figure 3.1, the data of size ‘m’ bytes, which can be a file or a memory buffer, is encrypted using encryption algorithm such as AES [5,10,1]. The shared key of length ‘k’ bytes undergoes the Shamir’s secret sharing process [2,5,20] to generate n shares of ‘k’ bytes each. Also, the cipher text of length ‘m’ bytes undergoes AONT transformation [6,7,8] to get ‘m’ bytes of output data which is fed as an input to the Erasure coding algorithm, producing ‘n’ shares of length ‘m/t’ bytes each.

**B. Decoder**

Figure 3.2 shows the system diagram of the new proposed scheme for the Decoder part.

As shown in Figure 3.2, the decoder receives the ‘t’ shares of ‘m/t’ bytes and feeds it as an input to the Erasure decoder. The ‘m’ byte output of the Erasure decoder goes to the AONT⁻¹. The ‘m’ byte output of AONT decoder goes to the decryption block which generates the original ‘m’ byte data. Shamir’s Secret sharing scheme decoder is used to get back the ‘k’ byte key necessary for decryption and is generated from the ‘n’ shares of ‘k’ byte each received as an input to the system.

### IV. EXPERIMENTAL SETUP

The experiment was carried out in stages parallel for PC based and Android based implementations:

**Stage 1:** Performance evaluation of Erasure coding, Secret key sharing and implementation of AONT schemes

The experimental setup for Stage 1 comprised of a C++/Inline assembly code to measure the execution time of the encoding/decoding process for performance evaluation of the available algorithms.

**Stage 2:** Java implementation of the Secure Distributed Storage System

#### A. Devices and tools used for PC

- **Language:** Java
- **Compiler/ debugger:** Eclipse along with Android Development Tool (ADT) [22] Java and Native Development Kit (NDK) plugin [23] provided by Google.
- **Execution Platform:** Nexus 7

**Hardware Specifications:**
- Internal Memory: 16 GB storage, 1GB RAM
- OS: Android, v4.1(Jelly Bean)
- Chipset: Nvidia Tegra

**C. Libraries used**

Table 4.1 gives a list of libraries used to implement the system.

#### D. Assumptions and restrictions

Memory assumptions for PC: Heap: Dynamic memory is allocated on the heap the available memory for the process can be measured programatically. I am currently using Linux VM and using ‘ulimit -a’ returns the memory available as ‘unlimited’. So, a code was written to calculate the approximate available memory per process in my system and gives the following result. Max Usable Memory per process = 6396313600 Bytes=6GB approximately. The code uses ‘malloc’ to allocate memory to the current process until no more memory is available, and thus finds out the memory available per process for the current hardware configuration.

For android devices: The size of your APK file is limited to...
50MB to ensure secure on-device storage, but expansion files can be attached to the APK. Each app can have two expansion files, each one up to 2GB, in whatever format we choose.

E. Application and Implementation details

A GUI was developed both the PC and Android platforms, and was coded in Java. The GUI had 5 screens, each for the main encoder/decoder, Encryption, Secret Sharing, Erasure Coding and AONT. Sub-modules were written to implement all the four transformations on PC Figure 4.1 and Android Figure 4.2. Programs to measure execution times were written and the encoding/decoding times for the selected transformations were written. Selection of Erasure code algorithms depend on the performance of the algorithm obtained from the benchmarking results mentioned in Section V, for the target platforms. The sub-modules were then integrated with the GUI to achieve desired system functionality. Benchmarking programs were also written to get a database of measurements for different input parameters, such as n, t and w for Erasure codes. Various wrappers were written to use the specified libraries for both target platforms.

![Figure 4.1 shows the UI of PC](image1)

![Figure 4.2 shows the UI of android](image2)

F. Files, testing and verification

Two kinds of inputs could be used as an input to the system as shown in Figure 4.1 and 4.2. These are the inputs either from memory or from a physical file on the hard disk. The experimental setup for the PC takes physical files as system input/output while the Android device uses memory buffers to do the same. The execution times for the system which takes input from the memory is definitely faster as it does not involve the file I/O operation times.

![Figure 4.3](image3)

![Figure 4.4](image4)

Test cases involve the schemes shown in figures 4.3 and 4.4 respectively, where the data is input/output from/to either RAM (i.e. memory) or actual data files. We have implemented both these schemes each on the PC and the Android device.

G. Results for the PC

Analysis is done as follows: The setup for measurement of execution times of various Erasure coding algorithms was made considering an input file of size 2 MB, and the execution time was measured for Reed Solomon Vandermonde, Cauchy original and Cauchy good versions of the Erasure Codes using the Jerasure [25] library. Also measurements were made for program execution times for the Shamir’s secret sharing scheme was made using the Libgfsahre [26] library. Program execution times for AES Encryption with 128 bit key size was also measured. The execution times of these algorithms contribute to the total system execution time. This time is in fact without the program initiation time.

The results shown below are as expected. The expected results should have program execution times directly proportional to the word size, confirmed as per observations.
Figure 4.5 shows the program execution time results for (n,t) pair values for Reed Solomon Vandermonde erasure coding algorithm along for w=8,16,32 respectively.

Figure 4.6 shows the program execution time results for (n,t) pair values for Cauchy good erasure coding algorithm along for w=8,16,32 respectively.

Figure 4.7 shows the program execution time results for (n,t) pair values for Cauchy original erasure coding algorithm along for w=8,16,32 respectively.

Figure 4.8 shows the program execution time results for (n,t) pair values for Shamir’s Secret Sharing algorithm along for w=8,16,32 respectively.
Figure 4.9

Figure 4.9 shows the program execution time results for (n,t) pair values and w=32 for the encoding process of the entire system. The choice of Erasure code made is Cauchy original since it is the best choice from the benchmarking results mentioned below. The various parameters are compared to get the following observations. The AONT program execution time is a constant for all (n,t) paris and the above chart shows total system time-AONT time, which still gives us as pretty good idea about performance of the system at different (n,t) pair values, since AONT time is a constant.

The above charts represent program execution times for Reed Solomon Vandermonde, Cauchy Good & Cauchy Original Erasure coding algorithms for different word sizes i.e. w=8,16,32 respectively. In the following observations, we notice the effect of increase in the word size. Also, the graphs represent the program execution times measured against different value pairs of (n,t). Choice of Erasure code shall depend on the number of shares & word size can be selected as per performance requirements, w=32 being the best, as observed from above results for the implementation for the PC. For example, we can see that for w=32, Cauchy original is a good choice of Erasure code for all (n,t) pairs. If we compare the results column-wise, we find that Cauchy Original is a good choice of Erasure code. Some of the other observations are as mentioned below.

For Reed Solomon Erasure codes, Execution time increases linearly with increase in the values of n,t. Higher word size, decreases the performance. Greater the number of shares, slower is the performance.

For Cauchy good erasure coding, with increasing n,t, execution time increases for except low values (4,3) of n,t. Greater the number of shares, slower is the performance. Higher the word size, better the performance. For (n,t)=(4,3), Cauchy Good Erasure code algorithm shows relatively higher execution time than for (n,t)=(8,6), the reason for the same remains uninvestigated, and as part of an extension to the project & as further work on the project. However, the results serve the purpose & goals of our project for choosing the algorithm.

For Cauchy original erasure coding, with increasing n,t execution time increases. Greater the number of shares, slower is the performance. Higher word size, decreases the performance.

For high values of (n,t), Cauchy_good is the choice of code for low word sizes (w=8,16). For high word sizes, Cauchy_orig has better performance by a factor of 2. For low values of (n,t), Cauchy_good is the choice of code for low word sizes (w=8,16). Cauchy_orig has better performance (by a factor of 2) for higher word sizes (w=32).

H. Results for Android

Benchmarking the performance of various algorithms available for the 4 different transformations (using a file of 1MB and Libraries: GFshare and Jerasure):

Figure 4.10

Figure 4.10 shows the program execution time results for (n,t) pair values for Shamir’s Secret Sharing Scheme for n=4,8,16,32 and t=3,6,12,24 respectively.

Figure 4.11

Figure 4.11 shows the encoding time results for (n,t) pair values for the 3 erasure codes for w=8.

Figure 4.12

Figure 4.12 shows the encoding time results for (n,t) pair values for the 3 erasure codes for w=16.
For Cauchy original erasure coding, with increasing n, t execution time increases. Performance decreases with increase in word size. Greater the number of shares, slower is the performance.

For Cauchy good erasure code, with increasing n, t execution time increases. Greater the number of shares, slower is the performance, which further decreases with increase in word size.

V. CONCLUSION

In this paper, we have presented a secure implementation of a distributed system. For this, performance of various erasure codes like Reed Solomon Vandermonde, Cauchy good and Cauchy original, Shamir’s secret sharing algorithm, AES Encryption algorithm was benchmarked. Also, AONT was implemented in java using Blake512 as hash. Based on the results obtained from benchmarking the erasure codes and performance of AONT, the proposed scheme was implemented on both PC and android and the results obtained were documented.

It was found that Cauchy original is a good choice of Erasure Code for PC while Reed Solomon Vandermonde is the choice of code for Android. This study will help future implementations a base reference for choice of algorithms based on the benchmarking data, & also it is a demonstration of a secure distributed system implementation on PC and Android.

REFERENCES


