Crypto Security Layer for Healthcare Applications and Data Storage in a Multi-Cloud Environment

MSc Research Project
Cloud Computing

Achal Bhangre
Student ID: 22181946

School of Computing
National College of Ireland

Supervisor: Dr. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Achal Bhangre
Student ID:	22181946
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Punit Gupta
Submission Due Date:	14/12/2023
Project Title:	Crypto Security Layer for Healthcare Applications and Data Storage in a Multi-Cloud Environment
Word Count:	XXX
Page Count:	9

Signature:	Achal.Bhangre
Date:	24th January 2024

Crypto Security Layer for Healthcare Applications and Data Storage in a Multi-Cloud Environment

Achal Bhangre
22181946

Table of Contents

1 Introduction **2**

2 Requirements **2**

3 Installation **3**

4 Connect to Multi-cloud Environment **4**

5 Proposed Model Flow and Implementation **5**

5.1 Steps Involved in Proposed Model 5

5.2 Cipher Process 5

6 Obtained Results **6**

7 Evaluating the result **7**

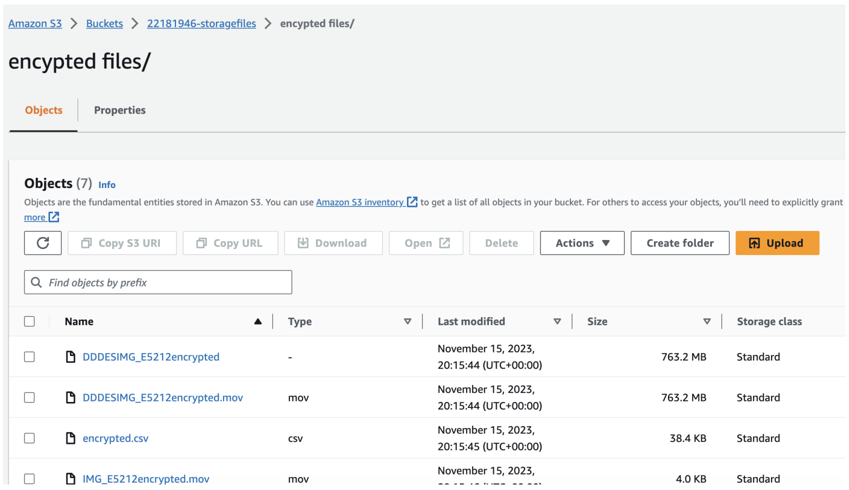
8 Final Result **8**

1 Introduction

In the past few years, more and more people have been saving and using data on multi-cloud storage platforms. It has many benefits, such as making sure that data is safe, keeping information from getting messed up, and stopping illegal problems from sellers. This project came up with a mixed method to make cloud data more secure and private. A multi-cloud server setup is used for the mixed method. There are two different parts to this combination method. An encryption layer that can handle security holes and keep the service running without stopping. When you use encoding and decoding methods through multi-cloud design, the data you store in the cloud is safer and more private. Using encryption methods makes it safer and more private to store data without slowing things down. For several types of medical information, different encryption methods are used and compared. The privacy and security problems with the mixed method are also looked at. In terms of speed, the mixed approach is better than multi-level encryption methods like Twofish, ChaCha2, Serpent, Camellia, and TripleDES. This is true no matter how much memory is used, how long it takes to secure and decode, or how long it takes to authenticate everything. The average precision measures, the amount of memory used, and the time needed for encryption and decoding were all better with the mixed method.

2 Requirements

1. Setup API environment in IntelliJ for SpringBoot(de Oliveira et al.; 2018) application.
2. Setup java jdk 8 version environment in the system.
3. Setup AWS account with S3 Bucket
4. Setup Postman Tool to access the service


Amazon S3 > Buckets > 22181946-storagefiles > encypted files/
encypted files/
Objects
Properties
Objects (7) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant
more

Name	Type	Last modified	Size	Storage class
DDDESIMG_E5212encrypted	-	November 15, 2023, 20:15:44 (UTC+00:00)	763.2 MB	Standard
DDDESIMG_E5212encrypted.mov	mov	November 15, 2023, 20:15:44 (UTC+00:00)	763.2 MB	Standard
encrypted.csv	csv	November 15, 2023, 20:15:45 (UTC+00:00)	38.4 KB	Standard
IMG_E5212encrypted.mov	mov	November 15, 2023,	4.0 KB	Standard

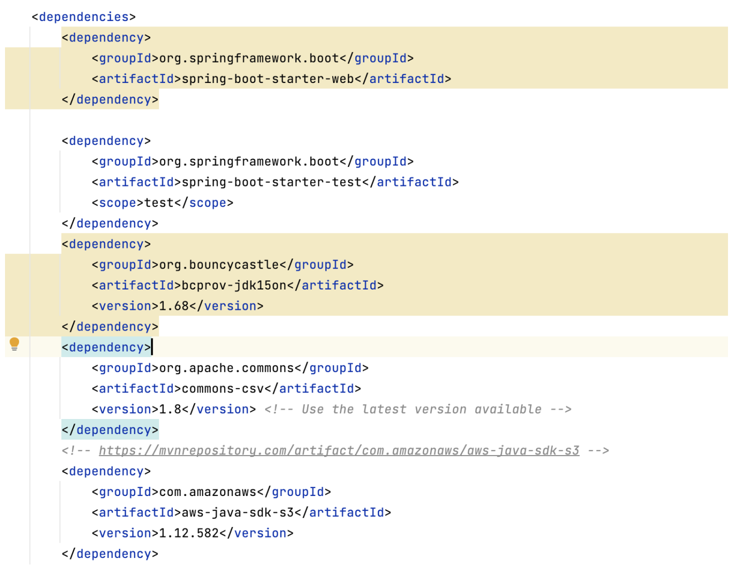
Figure 1: AWS S3 bucket service

.

3 Installation

1. Add Bouncy Castle dependency to the Pom.XML

2. Add AWS s3 cloud dependency in pom.XML to connect AWS cloud.(Saeed et al.; 2019)

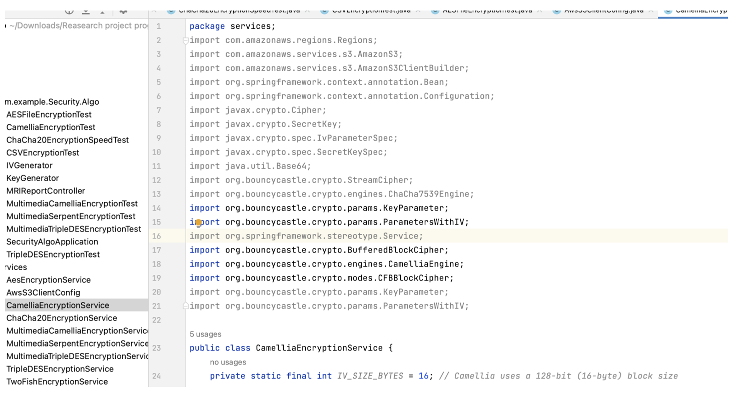


```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.bouncycastle</groupId>
        <artifactId>bcprov-jdk15on</artifactId>
        <version>1.68</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-csv</artifactId>
        <version>1.8</version> <!-- Use the latest version available -->
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-s3 -->
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-s3</artifactId>
        <version>1.12.582</version>
    </dependency>
```

Figure 2: MVN dependency list

1. Import all the required packages as per code implementation snippet 3 classes (de Oliveira et al.; 2018).

2. The Bouncy Castle Jar was used to secure and decrypt the files. With Bouncy Castle, you can quickly secure and decrypt data, no matter what it is. The BouncyCastle JCE service is free and works with Java Cryptographic Extensions. It was written code for the spring boot program for this exercise. It is possible to use safe methods in Java with the Bouncy Castle Crypto package. The package is made so that it has a simple API that can be used anywhere, even with the brand-new J2ME. It also has extra features that let the algorithms work with the JCE framework.



~/Downloads/Reasearch project pro

m.example.Security.Algo
AESFileEncryptionTest
CamelliaEncryptionTest
ChaCha20EncryptionSpeedTest
CSVEncryptionTest
IVGenerator
KeyGenerator
MRIReportController
MultimediaCamelliaEncryptionTest
MultimediaSerpentEncryptionTest
MultimediaTripleDESEncryptionTest
SecurityAlgoApplication
TripleDESEncryptionTest
vices
AesEncryptionService
AwsS3ClientConfig
CamelliaEncryptionService
ChaCha20EncryptionService
MultimediaCamelliaEncryptionServic
MultimediaSerpentEncryptionService
MultimediaTripleDESEncryptionServic
TripleDESEncryptionService
TwoFishEncryptionService

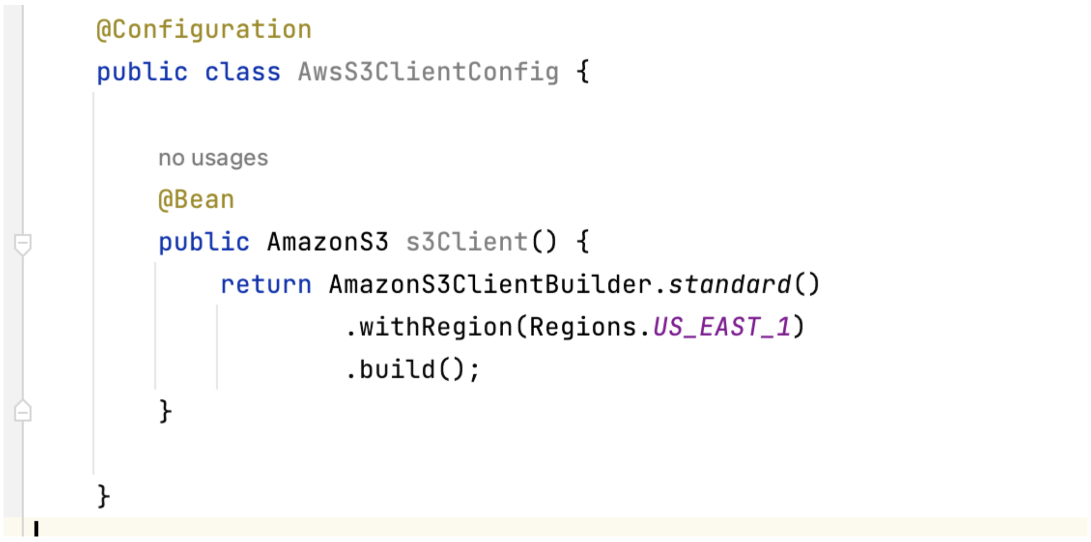
```
package services;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import org.bouncycastle.crypto.StreamCipher;
import org.bouncycastle.crypto.engines.ChaCha7539Engine;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.crypto.params.ParametersWithIV;
import org.springframework.stereotype.Service;
import org.bouncycastle.crypto.BufferedBlockCipher;
import org.bouncycastle.crypto.engines.CamelliaEngine;
import org.bouncycastle.crypto.modes.CFBBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.crypto.params.ParametersWithIV;

5 usages
public class CamelliaEncryptionService {
    no usages
    private static final int IV_SIZE_BYTES = 16; // Camellia uses a 128-bit (16-byte) block size
```

Figure 3: import packages

4 Connect to Multi-cloud Environment

The approach behind this study is that a multi-cloud system will be set up that uses both Amazon S3 structures and Azure Blob storage to store and handle secure files well. A multi-cloud method using both Amazon S3 and Azure Blob storage was chosen to have a safe way to view files that work across multiple cloud sources. Also, Amazon S3 and Azure Blob storage are known for being safe and flexible places to store private data, which keeps personal files safe. That's because they have their own security built right in, so dual encryption is going to happen.

The code snippet 4 below is about AWS S3 connection via API service



```
@Configuration
public class AwsS3ClientConfig {

    no usages
    @Bean
    public AmazonS3 s3Client() {
        return AmazonS3ClientBuilder.standard()
                .withRegion(Regions.US_EAST_1)
                .build();
    }

}
```

Figure 4: connect to aws s3 bucket

5 Proposed Model Flow and Implementation

5.1 Steps Involved in Proposed Model

1. The system recognizes PDF, multimedia, and text files before encryption. Content analysis or file extensions may do this.
2. File type-specific encryption. Camellia is using PDF encryption. Multimedia using TripleDES encryption. Encrypt text using ChaCha20.
3. Encrypted data across clouds. Azure Blob Service and AWS store data. The public Amazon S3 bucket allows encrypted PDFs and text. Azure Blob containers allow encryption of media.
4. Safe Key Management: Encryption and decoding need keys. Secure and use keys. Patients' IDs are used for multiple file identification, whereas the central repository uses symmetric keys for uploads.
5. File Security: Limit file access. A module decrypts files by type when they are retrieved.
6. Healthcare App Integration: Its loose connection simplifies healthcare app integration and cloud data storage.
7. Track access, security, and changes for compliance and forensic.

5.2 Cipher Process

When the encryption procedure is first started, the initialization vector—a random or pseudo-random value—is utilized to secure the data. For the encryption procedure, it is an extra input. The key is utilized by the encryption algorithm to both encrypt and decrypt plaintext. It appears that the key in your code is 128 bits (16 bytes). Extended keys offer enhanced security but incur greater computational costs.This study examines the TwoFish Algorithm with a key length of sixteen bytes. ChaCha20 and Camellia both contain 16 bytes. Below is a small code snippet of TripleDES encryption

```
 public class MultimediaTripleDESEncryptionService {
 private byte[] key;
 private byte[] iv;
 private static final String ALGORITHM = "DESede";
 private static final String TRANSFORMATION = "DESede/CBC/
PKCS5Padding";

 public MultimediaTripleDESEncryptionService(byte[] key, byte[] iv)
{
      this.key = key;
      this.iv = iv;
 }

 public byte[] encrypt(byte[] plaintext) throws Exception {
      byte[] encryptedBytes = performCipherOperation(Cipher.
ENCRYPT_MODE, plaintext);
      return Base64.getEncoder().encode(encryptedBytes);
 }
```

```
    public byte[] decrypt(byte[] encryptedText) throws Exception {
        byte[] decryptedBytes = performCipherOperation(Cipher.
    DECRYPT_MODE, Base64.getDecoder().decode(encryptedText));
        return decryptedBytes;
    }

    private byte[] performCipherOperation(int cipherMode, byte[] data)
    throws Exception {
        SecretKey secretKey = new SecretKeySpec(key, ALGORITHM);
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(cipherMode, secretKey, ivParameterSpec);

        return cipher.doFinal(data);
    }
}
```
Listing 1: Multimedia TripleDESEncryptionService.java

6 Obtained Results

ChaCha20 and Camellia exhibit rates that are comparable, especially when it comes to lesser file sizes. Figure 5 These algorithms may perform admirably in applications that have strict performance requirements. TwoFish consistently demonstrates superior performance compared to alternative algorithms, owing to its streamlined design and implementation, which results in minimal encryption times. Although Triple DES provides satisfactory performance, it demonstrates extended encryption times in comparison to the alternative algorithms. Whether or not it is a viable option is contingent upon the particular security demands of the application.

Performance Invariance Across File Sizes Figure 6, Camellia's consistent performance as PDF file sizes increase demonstrates its adaptability to various data quantities. Fig. 6 illustrates that the encryption durations of Camellia remain within a reasonable range, rendering it suitable for implementation in applications that handle PDF files of diverse sizes.

Triple DES always encrypts files quickly and securely for large files Figure 7. For 10MB files, it takes 2.5 milliseconds, and for 3GB files, it takes 5781 milliseconds. Because of how well it works, even with larger video files, it is a reliable choice for applications that deal with a wide range of data amounts. It takes about the same amount of time to protect different types of video files with Triple DES, which is one of the strongest encryption methods. It gives programs that work with multimedia files of different sizes the freedom and trust they need. The Camellia and Serpent methods may have trouble with larger datasets, even though they work well for encrypting smaller video files.

.

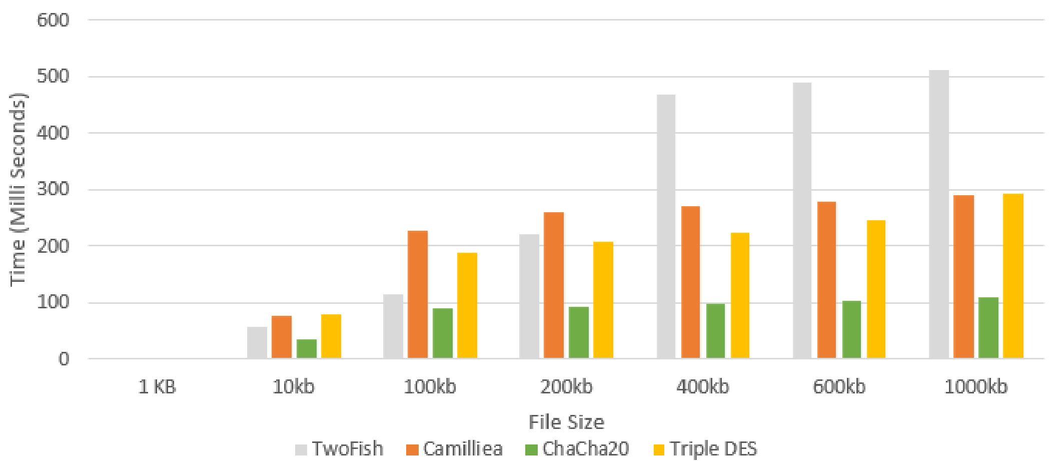


Figure 5: average computation speed of text files with 3 algorithms

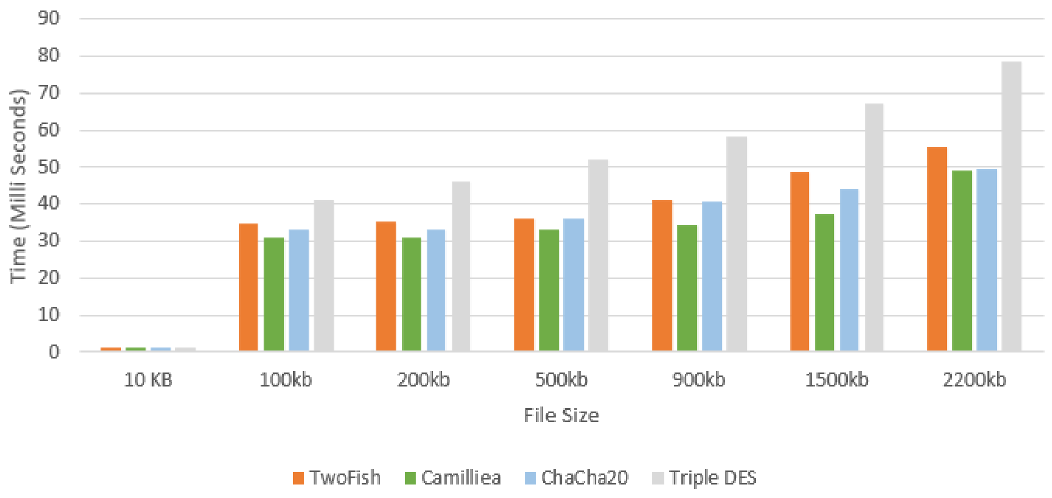


Figure 6: average computation speed of pdf files with 3 algorithms

7 Evaluating the result

Comparative tests of this strategy in Table 1 with encryption using a single cloud, hybrid cloud, or multi-cloud vendor will be done. In the second test, text, pdf, and multimedia files were treated the same. However, when the file size exceeded 1000 kilobytes, the encryption process performed poorly, resulting in a lower-quality decrypted file. The file size increased twice as well.

The encryption rates of the last three algorithms demonstrated are computed using the corresponding dataset to produce the ultimate outcomes. The parameters enumerated below were considered during the evaluation of encryption techniques for this study. The computation is performed by utilizing the file size and the key bytes employed by each method. The comparison between small block sizes (bytes) for encryption and the calculation time of the encryption algorithm for large files is also conducted.

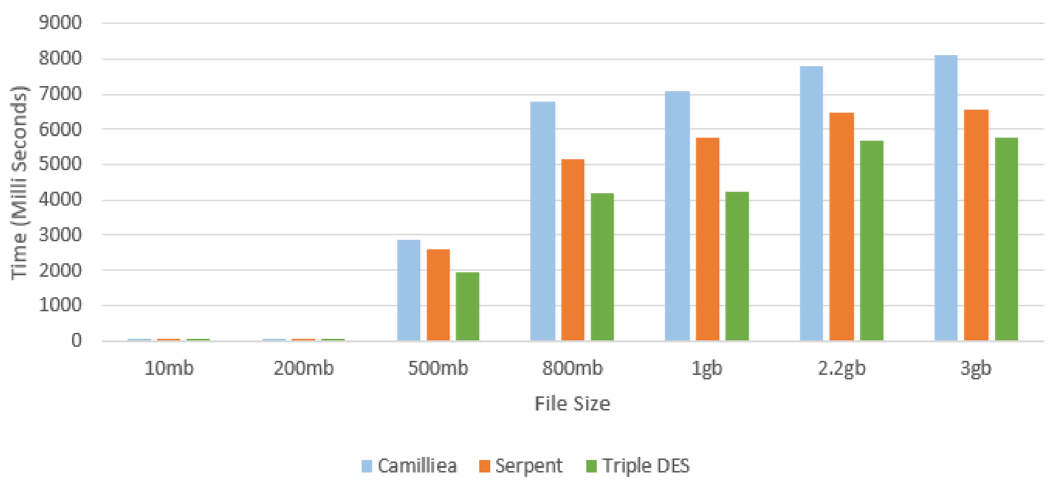


Figure 7: average computation speed of image files with 3 algorithms

Schemes	**Algorithm Speed**	**Data Overhead**	**Data Integrity**	**Key Security**
Paper1	×	✓	×	✓
Paper2	✓	×	✓	×
Paper3	×	×	×	✓
Proposed Model	✓	✓	✓	✓

Table 1: Comparison of papers with compare to performance indicator

8 Final Result

Cryptography and multi-cloud analysis demonstrate the uniform distribution of the study's key security, data latency, and encryption speed histograms in Table 2. The entropy value of the hybrid model is 7.99951, which is near the optimal value, i.e., 80% performance is increased. The encryption effect of the proposed cryptographic framework is effective, and the secret keyspace is sizable. Further investigation establishes that the proposed framework has the potential to bolster the security layer of healthcare applications in numerous ways. The execution duration of the proposed scheme is evaluated by executing several files of different sizes. In contrast to conventional security models, our methodology exhibits significantly reduced computation time.

Time (Milli seconds)	ChaCha20	Camellia	Triple DES
Text dataset[1000Kb]	109.7	288.3	292.3
Pdf dataset[2200Kb]	49.7	48.7	78.4
Images dataset [1Gb]	N/A	6795.7	4241

Table 2: Hybrid Model with Results of Medical Dataset

References

de Oliveira, C., Turnquist, G. and Antonov, A. (2018). *Developing Java Applications with Spring and Spring Boot*, Packt Publishing.
URL: *https://books.google.ie/books?id=EuLIvAEACAAJ*

Saeed, I., Baras, S. and Hajjdiab, H. (2019). Security and privacy of aws s3 and azure blob storage services, *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pp. 388–394.