A Project Report

on

Android Encryption Using Various Algorithms

Submitted in partial fulfillment of the
requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Mr. Ravindra Kumar Chahar
Associate Professor

Submitted By

18SCSE1010089 – Saket Joshi
18SCSE1010672 – Shivank Verma

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING / DEPARTMENT OF COMPUTERAPPLICATION
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER - 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"Android Encryption Using Various Algorithms"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY 2021 TO DECEMBER 2021**, under the supervision of MR. RAVINDRA KUMAR CHAHAR, ASSOCIATE PROFESSOR, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

18SCSE1010089 – SAKET JOSHI

18SCSE1010672 – SHIVANK VERMA

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

MR. Ravindra Kumar Chahar

Associate Professor

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of 18SCSE1010089 – SAKET JOSHI, 18SCSE1010672 – SHIVANK VERMA has been held on _____ and his/her work is recommended for the award of BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Place:

Abstract

The rise of internet has created a major challenge for securing sensitive information sent over the network. Several solutions have been proposed in the past to protect user data. Encryption algorithm is one of them. At present, various types of encryption algorithms like AES, DES, RSA and others are available. In this report, we have proposed an algorithm for ourselves which is 128-bit text size and 128-bit key size. We have compared the proposed algorithm with different algorithm comparisons. The result is showing our proposed algorithm gives better performance. By the difference we see that our recommended encryption process's average runtime is 0.006 sec which is less than other algorithms (AES, DES). It also protects various cryptanalytic attacks - key attack, brute force attack, man inthe middle attack and it is also suitable for the implementation of software and hardware.

Keywords*: encryption, decryption, symmetric, 128-bit, secret key, brute force attacks*

Table of Contents

Title		Page No.
Abstract		I
Chapter 1	Introduction	1
	1.1 Introduction	1
	1.1.1 Decryption	1
	1.1.2 AES	1
	1.1.3 DES	
	1.1.4 Vigenere Cipher	
	1.2 Formulation of Problem	6
	1.2.1 Tool and Technology Used	7
Chapter 2	Literature Survey/Project Design	8

CHAPTER-1
Introduction

1.1Introduction

Encryption is changing the way of information is displayed so that it ismasked and the only way its true form can be viewed with a clear set of instruction. Inits simple sense, Encryption is the process of encoding all user data or information using symmetric or asymmetric key in such a way that even if an unauthorized party tries to access the data without keys, they won't be able to read it.

1.1.1 Decryption

Decryption is the reverse process of encryption. It is the process of decoding all encrypted data using symmetric or asymmetric key in such a way that only authorized parties can access the data and can read it.

The main purpose of developing a cryptosystem is to protect the user's data privacy. Information's are transferred fromone user to another. In that case, possibilities of data losing or stealing data still remain there. So it is necessary for us to inspect the overall access of data from any unauthorized public or interceptor. So, to protect the information's from being lost or stolen, encryption is used/cryptosystem is necessary.

1.1.2. Data Encryption Standard

Data Encryption Standard (DES) is asymmetric key block cipher. In DES the key length is 112 bits or 168 bits and block size is 64-bits length. Now a day the increasing computational power is available which makes DES weak. For this reason, it can be attack by Brute ForceAttacks other cryptanalytic attacks. Intriple DES algorithm the size of block and key increased.

Without any doubt, AES is the strongest algorithm ever because it supports any combination of data and key length. In AES the number of round is variable and depends on length of the key. It uses 10 rounds for 128 bit keys, 12 rounds for 192 bit keys, 14 rounds for 256 bit keys thatcan be divide into 4 basic operational blocks. These blocks are considered as array of bytes and organized as a matrix ofthe order of 4*4, which is called as state and subject to rounds where various transformations are done.

The XOR logical functions can be applied to binary bits and also considered as an encryption cipher. In the encryption context the strength of XOR cipher depends on the length and the nature of the key. The XOR cipher with a lengthy random key can ensure better security performance. But large XOR key increases unpredictability and can confront brute force attack. In our algorithm, we have used 128-bit block size and 128-bit key size and also use a XOR in our permutation function.

1.2.Formulation of Problem

- To develop an algorithm for data security in an android platform.

- To maintain the security of the system by preventing cryptanalytic attack like key attacks and bruteforce attacks.

1.2.1. Tool and Technology Used

Front End – Android Studio 3.0

Android Studio is the official[7] integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.[8] It is available for download on Windows, macOS and Linux based operating systems or as a subscription-based service in 2020.[9][10] It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013 at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014.[11] The first stable build was released in December 2014, starting from version 1.0.[12]

On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development.[13] Java is still supported, as is C++.

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support

The following is a list of Android Studio's major releases:

Version	Release date
Arctic Fox (2020.3.1)	July 2021
4.2	May 2021
4.1	Oct 2020
4.0	May 2020
3.6	February 2020

Version	Release date
3.5	August 2019
3.4	April 2019[29]
3.3	January 2019
3.2	September 2018
3.1	March 2018
3.0	October 2017
2.3	March 2017
2.2	September 2016
2.1	April 2016
2.0	April 2016
1.5	November 2015
1.4	September 2015
1.3	July 2015
1.2	April 2015
1.1	February 2015
1.0	December 2014

System requirement

Basic system requirements for Android Studio[7]			
	Microsoft Windows	**Mac**	**Linux**
Operating System Version	Microsoft Windows 7/8/10 (32- or 64-bit) *The Android Emulator only supports 64-bit Windows.*	Mac OS X 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave)	GNOME or KDE desktop *Tested on gLinux based on Debian (4.19.67-2rodete2).*
Random Access Memory (RAM)	4 GB RAM minimum; 8 GB RAM recommended.		
Free digital storage	2 GB of available digital storage minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image).		
Minimum required JDK version	Java Development Kit 8		
Minimum screen resolution	1280 x 800		

.

Back End – Java

Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Environment – JDK and JRE

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps them to code and run Java programs. It is possible to install more than one JDK version on the same computer.

JRE is a piece of a software which is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don’t need to install JDK, but just JRE to run Java programs. Though, all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.

Operating System – Android, Windows

Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

AES

AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity and electronic data protection.

The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks.

NIST stated that the newer, advanced encryption algorithm would be unclassified and must be "capable of protecting sensitive government information well into the [21st] century." It was intended to be easy to implement in hardware and software, as well as in restricted environments -- such as a smart card -- and offer decent defenses against various attack techniques.

AES was created for the U.S. government with additional voluntary, free use in public or private, commercial or noncommercial programs that provide encryption services. However, nongovernmental organizations choosing to use AES are subject to limitations created by U.S. export control.

How AES encryption works
AES includes three block ciphers:

AES-128 uses a 128-bit key length to encrypt and decrypt a block of messages.
AES-192 uses a 192-bit key length to encrypt and decrypt a block of messages.
AES-256 uses a 256-bit key length to encrypt and decrypt a block of messages.
Each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic keys of 128, 192 and 256 bits, respectively.

Symmetric, also known as secret key, ciphers use the same key for encrypting and decrypting. The sender and the receiver must both know -- and use -- the same

secret key.

The government classifies information in three categories: Confidential, Secret or Top Secret. All key lengths can be used to protect the Confidential and Secret level. Top Secret information requires either 192- or 256-bit key lengths.

There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.

Image displaying the relationships between keys, ciphers and ciphertext in AES
AES uses 128-, 192- or 256-bit keys to encrypt and decrypt data.
The AES encryption algorithm defines numerous transformations that are to be performed on data stored in an array. The first step of the cipher is to put the data into an array, after which the cipher transformations are repeated over multiple encryption rounds.

The first transformation in the AES encryption cipher is substitution of data using a substitution table. The second transformation shifts data rows. The third mixes columns. The last transformation is performed on each column using a different part of the encryption key. Longer keys need more rounds to complete.

Features of AES :-
NIST specified the new AES algorithm must be a block cipher capable of handling 128-bit blocks, using keys sized at 128, 192 and 256 bits.

Other criteria for being chosen as the next AES algorithm included the following:

Security. Competing algorithms were to be judged on their ability to resist attack as compared to other submitted ciphers. Security strength was to be considered the most important factor in the competition.
Cost. Intended to be released on a global, nonexclusive and royalty-free basis, the candidate algorithms were to be evaluated on computational and memory efficiency.
Implementation. Factors to be considered included the algorithm's flexibility, suitability for hardware or software implementation, and overall simplicity.

Difference between AES-128 and AES-256 :-

Security experts consider AES safe against brute-force attacks. A brute-force attack is when a threat actor checks all possible key combinations until the correct key is found. The key size employed for AES encryption therefore needs to be

large enough so that it cannot be cracked by modern computers, even considering advancements in processor speeds based on Moore's law.

A 256-bit encryption key is significantly more difficult for brute-force attacks to guess than a 128-bit key; however, because the latter takes so long to guess, even with a huge amount of computing power, it is unlikely to be an issue for the foreseeable future, as a malicious actor would need to use quantum computing to generate the necessary brute force.

Still, 256-bit keys also require more processing power and can take longer to execute. When power is an issue -- particularly on small devices -- or where latency is likely to be a concern, 128-bit keys are likely to be a better option.

When hackers want to access a system, they will aim for the weakest point. This is typically not the encryption of a system, regardless of whether it's a 128-bit key or a 256-bit key. Users should make sure the software under consideration does what they want it to do, that it protects user data in the way it's expected to and that the overall process has no weak points.

Additionally, there should be no gray areas or uncertainty about data storage and handling. For example, if data resides in the cloud, users should know the location of the cloud. Most importantly, the security software that has been selected should be easy to use to ensure that users do not need to perform unsecure workarounds to do their jobs.

Is AES secure?
Security experts maintain that AES is secure when implemented properly. However, AES encryption keys need to be protected. Even the most extensive cryptographic systems can be vulnerable if a hacker gains access to the encryption key.

To ensure the security of AES keys:

- Use strong passwords.
- Use password managers.
- Implement and require multifactor authentication (MFA).
- Deploy firewalls and antimalware software.
- Conduct security awareness training to prevent employees from falling victim to social engineering and phishing attacks.

DES

The DES (Data Encryption Standard) algorithm is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology (NIST). The algorithm takes the plain text in 64-bit blocks and converts them into ciphertext using 48-bit keys.

Since it's a symmetric-key algorithm, it employs the same key in both encrypting and decrypting the data. If it were an asymmetrical algorithm, it would use different keys for encryption and decryption.

History of DES Algorithm
DES is based on the Feistel block cipher, called LUCIFER, developed in 1971 by IBM cryptography researcher Horst Feistel. DES uses 16 rounds of the Feistel structure, using a different key for each round.

DES became the approved federal encryption standard in November 1976 and was subsequently reaffirmed as the standard in 1983, 1988, and 1999.

DES's dominance came to an end in 2002, when the Advanced Encryption Standard (AES) replaced the DES encryption algorithm as the accepted standard, following a public competition to find a replacement. The NIST officially withdrew FIPS 46-3 (the 1999 reaffirmation) in May 2005, although Triple DES (3DES), remains approved for sensitive government information through 2030.

Triple DES Algorithm
Triple DES is a symmetric key-block cipher which applies the DES cipher in triplicate. It encrypts with the first key (k1), decrypts using the second key (k2), then encrypts with the third key (k3). There is also a two-key variant, where k1 and k3 are the same keys.

Key Takeaways

- The NIST had to replace the DES algorithm because its 56-bit key lengths were too small, considering the increased processing power of newer computers. Encryption strength is related to the key size, and DES found itself a victim of the ongoing technological advances in computing. It reached a point where 56-bit was no longer good enough to handle the new challenges to encryption.
- Note that just because DES is no longer the NIST federal standard, it doesn't mean that it's no longer in use. Triple DES is still used today, but it's considered a legacy encryption algorithm. Note that NIST plans to disallow all forms of Triple-DES from 2024 onward.

DES Algorithm Steps

To put it in simple terms, DES takes 64-bit plain text and turns it into a 64-bit ciphertext. And since we're talking about asymmetric algorithms, the same key is used when it's time to decrypt the text.

The algorithm process breaks down into the following steps:

- The process begins with the 64-bit plain text block getting handed over to an initial permutation (IP) function.
- The initial permutation (IP) is then performed on the plain text.
- Next, the initial permutation (IP) creates two halves of the permuted block, referred to as Left Plain Text (LPT) and Right Plain Text (RPT).
- Each LPT and RPT goes through 16 rounds of the encryption process.
- Finally, the LPT and RPT are rejoined, and a Final Permutation (FP) is performed on the newly combined block.
- The result of this process produces the desired 64-bit ciphertext.

The encryption process step (step 4, above) is further broken down into five stages:

- Key transformation
- Expansion permutation
- S-Box permutation
- P-Box permutation
- XOR and swap

For decryption, we use the same algorithm, and we reverse the order of the 16 round keys.

DES Modes of Operation

Experts using DES have five different modes of operation to choose from.

- Electronic Codebook (ECB). Each 64-bit block is encrypted and decrypted independently
- Cipher Block Chaining (CBC). Each 64-bit block depends on the previous one and uses an Initialization Vector (IV)
- Cipher Feedback (CFB). The preceding ciphertext becomes the input for the encryption algorithm, producing pseudorandom output, which in turn is XORed with plaintext, building the next ciphertext unit
- Output Feedback (OFB). Much like CFB, except that the encryption algorithm input is the output from the preceding DES

- Counter (CTR). Each plaintext block is XORed with an encrypted counter. The counter is then incremented for each subsequent block

DES Implementation and Testing

DES implementation requires a security provider. However, there are many available providers to choose from, but selecting one is the essential initial step in implementation. Your selection may depend on the language you are using, such as Java, Python, C, or MATLAB.

Once you decide on a provider, you must choose whether to have a random secret key generated by the Key Generator or create a key yourself, using a plaintext or byte array.

Vigenere Cipher

The Vigenère cipher is a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers, based on the letters of a keyword. It employs a form of polyalphabetic substitution.

First described by Giovan Battista Bellaso in 1553, the cipher is easy to understand and implement, but it resisted all attempts to break it until 1863, three centuries later. Many people have tried to implement encryption schemes that are essentially Vigenère ciphers. In 1863, Friedrich Kasiski was the first to publish a general method of deciphering Vigenère ciphers.

In the 19th century the scheme was misattributed to Blaise de Vigenère (1523–1596), and so acquired its present name.

Description :-

In a Caesar cipher, each letter of the alphabet is shifted along some number of places. For example, in a Caesar cipher of shift 3, a would become D, b would become E, y would become B and so on. The Vigenère cipher has several Caesar ciphers in sequence with different shift values.

To encrypt, a table of alphabets can be used, termed a tabula recta, Vigenère square or Vigenère table. It has the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.[citation needed]

For example, suppose that the plaintext to be encrypted is

attackatdawn.
The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":

LEMONLEMONLE
Each row starts with a key letter. The rest of the row holds the letters A to Z (in shifted order). Although there are 26 key rows shown, a code will use only as many keys (different alphabets) as there are unique letters in the key string, here just 5 keys: {L, E, M, O, N}. For successive letters of the message, successive letters of the key string will be taken and each message letter enciphered by using its

corresponding key row. The next letter of the key is chosen, and that row is gone along to find the column heading that matches the message character. The letter at the intersection of [key-row, msg-col] is the enciphered letter.

For example, the first letter of the plaintext, a, is paired with L, the first letter of the key. Therefore, row L and column A of the Vigenère square are used, namely L. Similarly, for the second letter of the plaintext, the second letter of the key is used. The letter at row E and column T is X. The rest of the plaintext is enciphered in a similar fashion:

Plaintext: attackatdawn
Key: LEMONLEMONLE
Ciphertext: LXFOPVEFRNHR
Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in that row and then using the column's label as the plaintext. For example, in row L (from LEMON), the ciphertext L appears in column A, so a is the first plaintext letter. Next, in row E (from LEMON), the ciphertext X is located in column T. Thus t is the second plaintext letter.

Source Code :-

MainActivity.java

```
package Main;

import android.os.Bundle;
import android.view.View;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import com.example.Algorithms.R;

import Encryption.EncryptionMain;
import Hash.HashMain;



public class MainActivity extends AppCompatActivity {
   EncryptionMain encryptionMain;
   HashMain hashMain;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Fragment fragment = new MainFragment();
      FragmentManager fragmentManager = getSupportFragmentManager();
      fragmentManager.beginTransaction().replace(R.id.container,
fragment).commit();
   }

   public void goToEncryption(View view) {
      encryptionMain = new EncryptionMain();
      FragmentManager manager = getSupportFragmentManager();
      FragmentTransaction transaction = manager.beginTransaction();

transaction.setCustomAnimations(android.R.anim.fade_in,android.R.anim.fade_o
ut, android.R.anim.fade_in, android.R.anim.fade_out);
```

```
        transaction.replace(R.id.container, encryptionMain);
        transaction.addToBackStack(null);
        transaction.commit();
    }

    public void goToHash(View view) {
        hashMain = new HashMain();
        FragmentManager manager = getSupportFragmentManager();
        FragmentTransaction transaction = manager.beginTransaction();

transaction.setCustomAnimations(android.R.anim.fade_in,android.R.anim.fade_o
ut, android.R.anim.fade_in, android.R.anim.fade_out);
        transaction.replace(R.id.container, hashMain);
        transaction.addToBackStack(null);
        transaction.commit();
    }

    public void encryptionButtonClick(View view) {
        try {
            switch (view.getId()) {
                case R.id.Swtich:
                    encryptionMain.switchAlgho(view);
                    break;
                case R.id.Encrypt_Buuton:
                    encryptionMain.encrypt(view);
                    break;
                case R.id.Decrypt_Buuton:
                    encryptionMain.decrypt(view);
                    break;
                case R.id.copy_button:
                    encryptionMain.copyToClipboard(view);
                    break;
                case R.id.reset_button:
                    encryptionMain.reset(view);
                    break;
            }
        }
        catch (Exception e){
            e.printStackTrace();
    }

    }
```

```
    public void HashButtonClick(View view) {
        try {
            switch (view.getId()) {
                case R.id.Swtich:
                    hashMain.switchAlgho(view);
                    break;
                case R.id.hash_Buuton:
                    hashMain.hash(view);
                    break;
                case R.id.copy_button:
                    hashMain.copyToClipboard(view);
                    break;
                case R.id.reset_button:
                    hashMain.reset(view);
                    break;
            }
        }
        catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

AES.java

```
package Encryption.Algorithms;

import android.util.Base64;
import android.util.Log;

import java.security.MessageDigest;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES {
   public String AESencrypt ( byte[] key, byte[] clear) throws Exception {

      MessageDigest md = MessageDigest.getInstance("md5");
      byte[] digestOfPassword = md.digest(key);

      SecretKeySpec skeySpec = new SecretKeySpec(digestOfPassword, "AES");
      Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding");
      cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
      byte[] encrypted = cipher.doFinal(clear);
      return Base64.encodeToString(encrypted, Base64.DEFAULT);

   }
   public String AESdecrypt (String key,byte[] encrypted) throws Exception {
      MessageDigest md = MessageDigest.getInstance("md5");
      byte[] digestOfPassword = md.digest(key.getBytes("UTF-16LE"));

      SecretKeySpec skeySpec = new SecretKeySpec(digestOfPassword, "AES");
      Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding");
      cipher.init(Cipher.DECRYPT_MODE, skeySpec);
      byte[] decrypted = cipher.doFinal(encrypted);
      return new String(decrypted, "UTF-16LE");
   }
}
```

Caesarcipher.java

```
package Encryption.Algorithms;

public class Caesarcipher {
   String message;
   char ch;
   char NumbTest[] = {'0','1', '2', '3', '4', '5', '6', '7', '8', '9'};
   public String caesarcipherEnc (String message,int key){
      String encryptedMessage = "";
      int n = 1;
      for (int i = 0; i < message.length(); i++) {
         n = 1;
         ch = message.charAt(i);
         for (int j = 0; j < NumbTest.length; j++)
         {
            if (ch == NumbTest[j])
            {

               if((char)key+ch>'9')
                  break;
               ch = (char) (ch + key);
               encryptedMessage += ch;
               n = 0;
               break;
            }
         }
         if (n == 0)
         {
            continue;
         } else
            if (ch >= 'a' && ch <= 'z')
            {
            ch = (char) (ch + key);

            if (ch > 'z') {
               ch = (char) (ch - 'z' + 'a' - 1);
            }

            encryptedMessage += ch;
         } else if (ch >= 'A' && ch <= 'Z') {
            ch = (char) (ch + key);
```

```
            if (ch > 'Z') {
                ch = (char) (ch - 'Z' + 'A' - 1);
            }

            encryptedMessage += ch;
        } else
            encryptedMessage += ch;
    }

    return encryptedMessage;
}
public String caesarcipherDec (String message,int key)
{
    String decryptedMessage = "";
    int n = 1;
    for (int i = 0; i < message.length(); i++) {
        n = 1;
        ch = message.charAt(i);
        for (int j = 0; j < NumbTest.length; j++) {
            if (ch == NumbTest[j]) {

                if((char)key+ch>'9')
                    break;
                ch = (char) (ch - key);
                decryptedMessage += ch;
                n = 0;
                break;
            }
        }
        if (n == 0)
        {
            continue;
        } else if (ch >= 'a' && ch <= 'z') {
            ch = (char) (ch - key);

            if (ch < 'a') {
                ch = (char) (ch + 'z' - 'a' + 1);
            }

            decryptedMessage += ch;
        } else if (ch >= 'A' && ch <= 'Z') {
            ch = (char) (ch - key);
```

```
            if (ch < 'A') {
                ch = (char) (ch + 'Z' - 'A' + 1);
            }

            decryptedMessage += ch;

        } else
            decryptedMessage += ch;
    }
    return decryptedMessage;
  }
}
```

DES.java

```
package Encryption.Algorithms;

import android.util.Base64;
import android.util.Log;

import java.security.MessageDigest;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class DES {
    public String encrypt ( byte[] key, byte[] clear) throws Exception {

        MessageDigest md = MessageDigest.getInstance("md5");
        byte[] digestOfPassword = md.digest(key);

        SecretKeySpec skeySpec = new SecretKeySpec(digestOfPassword, "DESede");
        Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        byte[] encrypted = cipher.doFinal(clear);
        return Base64.encodeToString(encrypted, Base64.DEFAULT);

    }
    public String decrypt (String key,byte[] encrypted) throws Exception {
        MessageDigest md = MessageDigest.getInstance("md5");
        byte[] digestOfPassword = md.digest(key.getBytes("UTF-16LE"));

        SecretKeySpec skeySpec = new SecretKeySpec(digestOfPassword, "DESede");
        Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
        byte[] decrypted = cipher.doFinal(encrypted);
        return new String(decrypted, "UTF-16LE");
    }
}
```

PlayFair.java

```
package Encryption.Algorithms;

public class PlayFair {

private String t1="";



   public String getT1() {
      return t1;
   }

   public PlayFair(String t1) {
      this.t1 = t1;
   }

   public class Basic {
      String allChar = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

      boolean indexOfChar(char c) {
         for (int i = 0; i < allChar.length(); i++) {
            if (allChar.charAt(i) == c)
               return true;
         }
         return false;
      }
   }

   Basic b = new Basic();
   char keyMatrix[][] = new char[5][5];

   boolean repeat(char c) {
      if (!b.indexOfChar(c)) {
         return true;
      }
      for (int i = 0; i < keyMatrix.length; i++) {
         for (int j = 0; j < keyMatrix[i].length; j++) {
            if (keyMatrix[i][j] == c || c == 'J')
               return true;
         }
      }
```

```
    return false;
}

public void insertKey(String key) {
    key = key.toUpperCase();
    key = key.replaceAll("J", "I");
    key = key.replaceAll(" ", "");
    int a = 0, b = 0;

    for (int k = 0; k < key.length(); k++) {
        if (!repeat(key.charAt(k))) {
            keyMatrix[a][b++] = key.charAt(k);
            if (b > 4) {
                b = 0;
                a++;
            }
        }
    }

    char p = 'A';

    while (a < 5) {
        while (b < 5) {
            if (!repeat(p)) {
                keyMatrix[a][b++] = p;

            }
            p++;
        }
        b = 0;
        a++;
    }

    for (int i = 0; i < 5; i++) {

        for (int j = 0; j < 5; j++) {

        }
    }
    t1=t1.concat("\n");
    for(int i=0;i < 5;i++)
    {
        for(int j=0;j < 5;j++)
```

```
            t1 = t1 +"   "+ keyMatrix[i][j];

             t1=t1.concat("\n");

          }
       }



int rowPos(char c) {
   for (int i = 0; i < keyMatrix.length; i++) {
      for (int j = 0; j < keyMatrix[i].length; j++) {
         if (keyMatrix[i][j] == c)
            return i;
      }
   }
   return -1;
}

int columnPos(char c) {
   for (int i = 0; i < keyMatrix.length; i++) {
      for (int j = 0; j < keyMatrix[i].length; j++) {
         if (keyMatrix[i][j] == c)
            return j;
      }
   }
   return -1;
}

public String encryptChar(String plain) {
   plain = plain.toUpperCase();
   char a = plain.charAt(0), b = plain.charAt(1);
   String cipherChar = "";
   int r1, c1, r2, c2;
   r1 = rowPos(a);
   c1 = columnPos(a);
   r2 = rowPos(b);
   c2 = columnPos(b);

   if (c1 == c2) {
      ++r1;
      ++r2;
      if (r1 > 4)
```

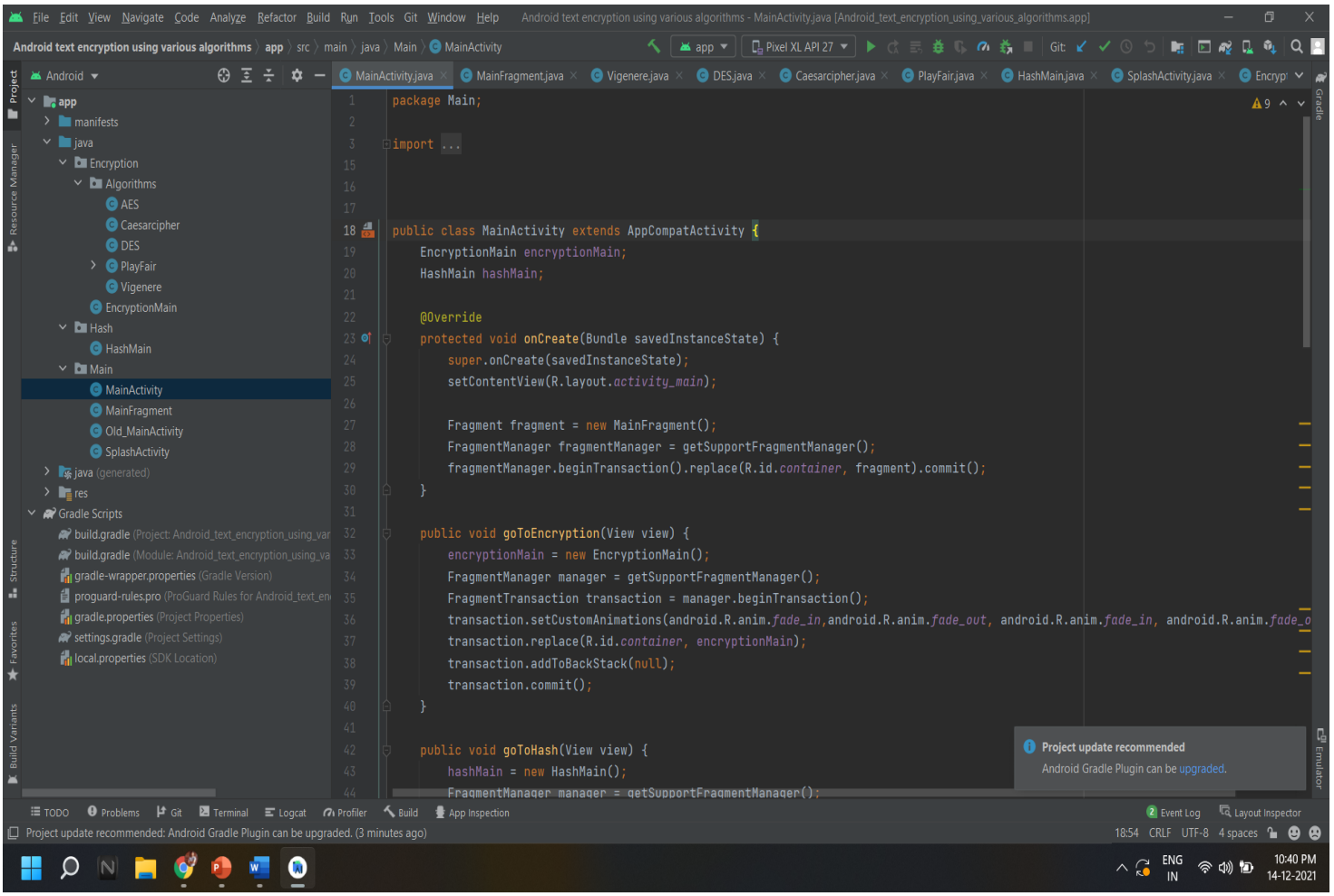
```
            r1 = 0;

        if (r2 > 4)
            r2 = 0;
        cipherChar += keyMatrix[r1][c2];
        cipherChar += keyMatrix[r2][c1];
    } else if (r1 == r2) {
        ++c1;
        ++c2;
        if (c1 > 4)
            c1 = 0;

        if (c2 > 4)
            c2 = 0;
        cipherChar += keyMatrix[r1][c1];
        cipherChar += keyMatrix[r2][c2];

    } else {
        cipherChar += keyMatrix[r1][c2];
        cipherChar += keyMatrix[r2][c1];
    }
    return cipherChar;
}


public String Encrypt(String plainText, String key) {
    insertKey(key);
    String cipherText = "";
    plainText = plainText.replaceAll("j", "i");
    plainText = plainText.replaceAll(" ", "");
    plainText = plainText.toUpperCase();
    int len = plainText.length();

    if (len / 2 != 0) {
        plainText += "X";
        ++len;
    }

    for (int i = 0; i < len - 1; i = i + 2) {
        cipherText += encryptChar(plainText.substring(i, i + 2));
        cipherText += " ";
    }
    return cipherText;
```

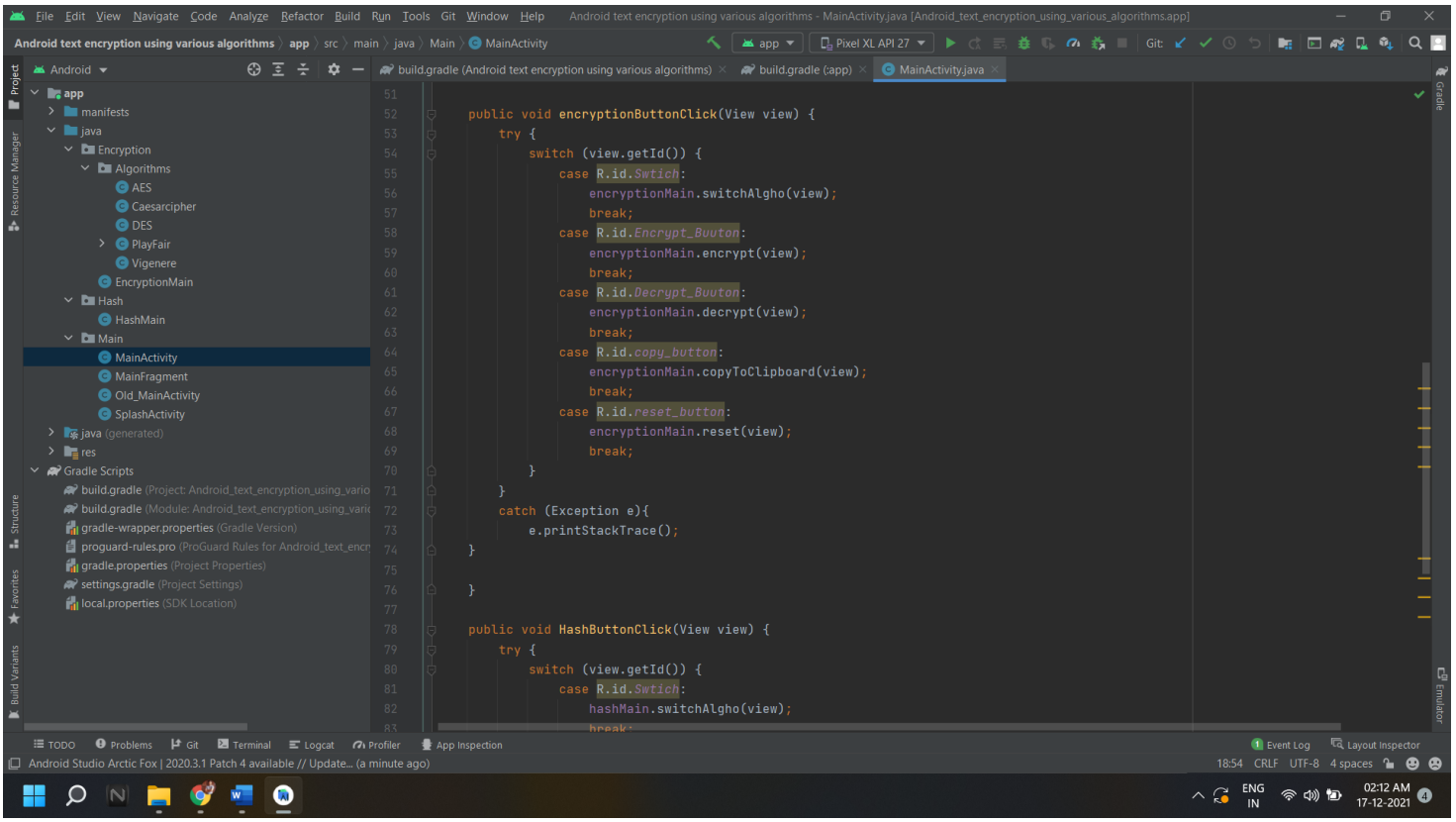
```
}

public String decryptChar(String cipher) {
    cipher = cipher.toUpperCase();
    char a = cipher.charAt(0), b = cipher.charAt(1);
    String plainChar = "";
    int r1, c1, r2, c2;
    r1 = rowPos(a);
    c1 = columnPos(a);
    r2 = rowPos(b);
    c2 = columnPos(b);

    if (c1 == c2) {
        --r1;
        --r2;
        if (r1 < 0)
            r1 = 4;

        if (r2 < 0)
            r2 = 4;
        plainChar += keyMatrix[r1][c2];
        plainChar += keyMatrix[r2][c1];
    } else if (r1 == r2) {
        --c1;
        --c2;
        if (c1 < 0)
            c1 = 4;

        if (c2 < 0)
            c2 = 4;
        plainChar += keyMatrix[r1][c1];
        plainChar += keyMatrix[r2][c2];

    } else {
        plainChar += keyMatrix[r1][c2];
        plainChar += keyMatrix[r2][c1];
    }
    return plainChar;
}

public String Decrypt(String cipherText, String key) {
```

```
        String plainText = "";
        cipherText = cipherText.replaceAll("j", "i");
        cipherText = cipherText.replaceAll(" ", "");
        cipherText = cipherText.toUpperCase();
        int len = cipherText.length();
        for (int i = 0; i < len - 1; i = i + 2) {
            plainText += decryptChar(cipherText.substring(i, i + 2));
            plainText += " ";

        }
        return plainText;
    }


}
```

Vigenere.java

```
package Encryption.Algorithms;

public class Vigenere {
   public String Vigenereencrypt (String text, String key)
   {

      String res = "";
      text = text.toUpperCase();
      key = key.toUpperCase();
      for (int i = 0, j = 0; i < text.length(); i++) {
         char c = text.charAt(i);
         if (c < 'A' || c > 'Z') continue;
         res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
         j = ++j % key.length();
      }
      return res;

   }
   public String Vigeneredecrypt (String text, String key)
   {
      String res = "";
      text = text.toUpperCase();
      key = key.toUpperCase();
      for (int i = 0, j = 0; i < text.length(); i++) {
         char c = text.charAt(i);
         if (c < 'A' || c > 'Z') continue;
         res += (char) ((c - key.charAt(j) + 26) % 26 + 'A');
         j = ++j % key.length();
      }

      return res;

   }
}
```


File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window Help
Android text encryption using various algorithms - MainActivity.java [Android_text_encryption_using_various_algorithms.app]
Android text encryption using various algorithms app src main java Main MainActivity
Pixel XL API 27
Android
MainActivity.java
MainFragment.java
Vigenere.java
DES.java
Caesarcipher.java
PlayFair.java
HashMain.java
SplashActivity.java
app
manifests
java
Encryption
Algorithms
AES
Caesarcipher
DES
PlayFair
Vigenere
EncryptionMain
Hash
HashMain
Main
MainActivity
MainFragment
Old_MainActivity
SplashActivity
java (generated)
res
Gradle Scripts
build.gradle (Project: Android_text_encryption_using_var
build.gradle (Module: Android_text_encryption_using_va
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Android_text_en
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
package Main;
import ...
public class MainActivity extends AppCompatActivity {
EncryptionMain encryptionMain;
HashMain hashMain;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Fragment fragment = new MainFragment();
FragmentManager fragmentManager = getSupportFragmentManager();
fragmentManager.beginTransaction().replace(R.id.container, fragment).commit();
}
public void goToEncryption(View view) {
encryptionMain = new EncryptionMain();
FragmentManager manager = getSupportFragmentManager();
FragmentTransaction transaction = manager.beginTransaction();
transaction.setCustomAnimations(android.R.anim.fade_in,android.R.anim.fade_out, android.R.anim.fade_in, android.R.anim.fade_o
transaction.replace(R.id.container, encryptionMain);
transaction.addToBackStack(null);
transaction.commit();
}
public void goToHash(View view) {
hashMain = new HashMain();
FragmentManager manager = getSupportFragmentManager();
Project update recommended
Android Gradle Plugin can be upgraded.
TODO Problems Git Terminal Logcat Profiler Build App Inspection
Event Log Layout Inspector
Project update recommended: Android Gradle Plugin can be upgraded. (3 minutes ago)
18:54 CRLF UTF-8 4 spaces
ENG IN
10:40 PM 14-12-2021


File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window Help
Android text encryption using various algorithms - MainActivity.java [Android_text_encryption_using_various_algorithms.app]
Android text encryption using various algorithms › app › src › main › java › Main › MainActivity
app
Pixel XL API 27
Git:
Android
build.gradle (Android text encryption using various algorithms)
build.gradle (:app)
MainActivity.java
app
manifests
java
Encryption
Algorithms
AES
Caesarcipher
DES
PlayFair
Vigenere
EncryptionMain
Hash
HashMain
Main
MainActivity
MainFragment
Old_MainActivity
SplashActivity
java (generated)
res
Gradle Scripts
build.gradle (Project: Android_text_encryption_using_vario
build.gradle (Module: Android_text_encryption_using_vari
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Android_text_encr
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
public void encryptionButtonClick(View view) {
try {
switch (view.getId()) {
case R.id.Swtich:
encryptionMain.switchAlgho(view);
break;
case R.id.Encrypt_Buuton:
encryptionMain.encrypt(view);
break;
case R.id.Decrypt_Buuton:
encryptionMain.decrypt(view);
break;
case R.id.copy_button:
encryptionMain.copyToClipboard(view);
break;
case R.id.reset_button:
encryptionMain.reset(view);
break;
}
}
catch (Exception e){
e.printStackTrace();
}
}
public void HashButtonClick(View view) {
try {
switch (view.getId()) {
case R.id.Swtich:
hashMain.switchAlgho(view);
TODO
Problems
Git
Terminal
Logcat
Profiler
App Inspection
Event Log
Layout Inspector
Android Studio Arctic Fox | 2020.3.1 Patch 4 available // Update... (a minute ago)
18:54 CRLF UTF-8 4 spaces
ENG
IN
02:12 AM
17-12-2021


ADVANCED ENCRYPTION STANDARD
FlW4awe3eBScplutRlTGd8iMIPeqI9saFE6mpc6Hufk=
6
ENCRYPT
DECRYPT
Hello World.
COPY
RESET


ADVANCED ENCRYPTION STANDARD
Hello World.
6
ENCRYPT
DECRYPT
FlW4awe3eBScplutRlTGd8iMIPeqI9saFE6mpc6Hufk=
COPY
RESET

CHAPTER-2
Literature Survey

Cioc.I.Bet .al in 2015 explained a method used for increasing the security of sending text messages using public text communication services like email and SMS. It utilizes content encryption before sending the message through email or cell phone (SMS), so, even the message is received and seen by another unapproved individual, it can't be comprehended. So, that application was executed in LabVIEWand can be used for sending encoded content email between at least two clients, utilizing open email administrations. For encryption, their proposed application utilizes content encryption strategy like balanced and deviated encryption, utilizingprivate encryption key or private or open encryption key. For sending encoded SMS using that application, the instant message must be recently scrambled, and after that the encoded message will be replicated to the content window of the application for sending SMS running on the cell phone. A comparable application can be additionallycreated for cell phones with workingframeworks like Android, iOS, Windows portable and so forth their application can be utilized likewise with any instant message administration, similar to yippee detachment, Facebook Messenger and so on.