A MOBILE APPLICATION DEVELOPMENT PROJECT REPORT
ON
" ENCRYPTION AND DECRYPTION "

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

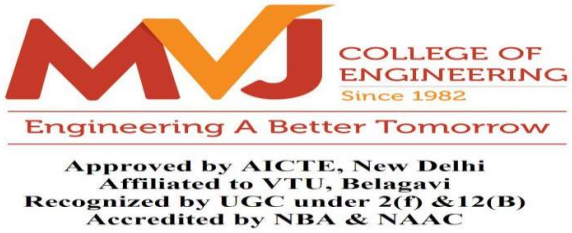
Submitted By:

TEAM MEMBERS NAME	USN
Jeanette Krizelda K	1MJ19CS069
Jennifer R	1MJ19CS070
Kavya ML	1MJ19CS077
Sowmya HS	1MJ19CS162

Under the Guidance of

Dr H Shaheen

Associate Professor, Department of Computer Science & Engineering



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MVJ COLLEGE OF ENGINEERING
BANGALORE-67

Academic Year-2021-22

ABSTRACT

The title of our project is “ENCRYPTION AND DECRYPTION”. Our aim is to develop an application named ENCRYPTION AND DECRYPTION that encrypts and decrypts the text by using different encryption and hash algorithms available. If you want to send sensitive information via email, simply paste the encrypted text into your email or attach the encrypted file, all the recipient has to do is to decrypt your text or file. Encryption and Decryption works with text information and files. Just select what you want to encrypt, and using the Encryption and Decryption app keep your documents, private information and files in a confidential way. The different encryption algorithms used in this app are as follows: Caesar Cipher, Vigenère Cipher, Advanced Encryption Standard (AES), Triple Data Encryption Standard , Play Fair Cipher. The different kinds of hash algorithms used in this app are as follows: MD5, SHA-256 and SHA-512.

TABLE OF CONTENTS

INTRODUCTION

Encryption is changing the way of information is displayed so that it is masked and the only way its true form can be viewed with a clear set of instruction. In its 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.

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 from one 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.

Hashing is a cryptographic technique that transforms any form of data into a special text string. For any given input, there is a deterministic output. When you put a plaintext into a hashing algorithm in simpler terms, you get the same outcome. Suppose you change anything about the input or the plaintext to the hashing algorithm. The hashing output also becomes different.

Hashing works by converting a readable text into an unreadable text of secure data. Hashing is efficiently executed but extremely difficult to reverse. Like I stated earlier, hashing and encryption are often mistaken. Encryption is a two-way function. The plaintext can be encrypted into ciphertext and decrypted back into plaintext using a unique key. The difference between encryption and hashing is that encryption is reversible while hashing is irreversible.

Hashing takes the password a user enters and randomly generates a hash using many variables (text and numbers). When you input your password to log in, it is matched to the hash password. This is because the input is the same as the output.

Now, let us discuss in brief about the various encryption algorithms used in this app.

1) **Caesar Cipher**

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter

of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

2) Vigenère Cipher

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the *Vigenère square or Vigenère table*. The table consists of the alphabets 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.

3) Advanced Encryption Standard

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

- **Points to remember:**

AES is a block cipher.

The key size can be 128/192/256 bits.

Encrypts data in blocks of 128 bits each.

That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

4) Triple Data Encryption Standard

Triple DES is a encryption technique which uses three instance of DES on same plain text. It uses three different types of key choosing technique in first all used keys are different and in second two keys are same and one is different and in third all keys are same.

Triple DES is also vulnerable to meet-in-the middle attack because of which it gives total security level of 2^112 instead of using 168 bit of key. The block collision attack can also be done because of short block size and using same key to encrypt large size of text. It is also vulnerable to sweet32 attack.

5) Play Fair

The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after

Lord Playfair who promoted the use of the cipher. In play fair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II. This was because Play fair is reasonably fast to use and requires no special equipment.

Now, lets discuss in brief about some hash functions used in this app.

1) MD5

The MD5 (message-digest algorithm) hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.

The MD5 hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures. But MD5 has been deprecated for uses other than as a noncryptographic checksum to verify data integrity and detect unintentional data corruption.

2) SHA-256

SHA 256 is a part of the SHA 2 family of algorithms, where SHA stands for Secure Hash Algorithm. Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against brute force attacks.

The significance of the 256 in the name stands for the final hash digest value, i.e. irrespective of the size of plaintext/cleartext, the hash value will always be 256 bits.

3) SHA -512

SHA-512, or Secure Hash Algorithm 512, is a hashing algorithm used to convert text of any length into a fixed-size string. Each output produces a SHA-512 length of 512 bits (64 bytes).

This algorithm is commonly used for email addresses hashing, password hashing, and digital record verification. SHA-512 is also used in blockchain technology, with the most notable example being the Bit Shares network.

Literature Review

Network is a collection of nodes. The basic objective of network is to transfer information from one place to another. Obviously, this information is required to be secured from third party access. The concept of cryptography was born from the need to secure critical information exchanged over insecure network. The sender crypts or encodes the information with a secret key while practicing cryptography so that it can only be understood by the tender recipient. On the other hand, cryptanalysis means unwanted access without the secret key to information. The cryptography makes use of different techniques like Diffie Hellman, AES, RSA, DES, Play Fair, BLOWFISH, based infrastructure and digital signatures for converting plain text in to corresponding chipper text. All these algorithms have their importance in different circumstances.

In 2015, a method used for increasing the security of sending text messages using public text communication services like email and SMS was explained. 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 LabVIEW and 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, utilizing private 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 with working frameworks like Android, iOS, Windows.

Problem Analysis

The aim of our project is to develop an app for encrypting and decrypting given input data from the user using various encryption/decryption and hashing algorithms.

The user can choose the type of encryption, decryption or hashing algorithm they want to use and provide the input data along with the required inputs for a particular algorithm to perform the desired function.

The user can then copy the encrypted/decrypted/hashed text and use it as they require.

The different encryption algorithms used in this app are as follows: Caesar Cipher, Vigenère Cipher, Advanced Encryption Standard, Triple Data Encryption Standard Play Fair Cipher.

The different kinds of hash algorithms used in this app are as follows: MD5, Sha-256 and SHA-512.

IMPLEMENTATION

Software Requirements:

- Windows XP, Windows 7(ultimate, enterprise)
- Android Studio

Hardware Components:

- Processor – i3
- Hard Disk – 5 GB
- Memory – 1GB RAM
- Android Phone with kitkat and higher.

Step 1 - Setting up Android Studio and creating a new project by setting minimum SDK to API 32: Android 5.0 (Tiramisu).

Step 2 – Create Main Activity and design.

```
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
_out, 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
_out, 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();
    }
 }
}
```

Step 3 – Main Fragment.

```
public class MainFragment extends Fragment {
  private View view;
    @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
       view = inflater.inflate(R.layout.fragment_main, container, false);
       ((AppCompatActivity) getActivity()).getSupportActionBar().hide();
getActivity().getWindow().setFlags(WindowManager.LayoutParams.FLAG_F
ULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
 return view;
   }
}
```

Step 4 – Creating Launching Splash Screen activity by setting up the layout and assigning animations to Image View and Text View

```
public class SplashActivity extends AppCompatActivity {
  private int Splash_Screen = 3000;
   private ImageView LogoImg;
   private TextView Appname;
   private Animation topAnimation, leftAnimation;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        getSupportActionBar().hide();
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN
, WindowManager.LayoutParams.FLAG_FULLSCREEN);
      LogoImg = findViewById(R.id.SplashPC);
        //Appname = findViewById(R.id.SplashKeyboard);
  topAnimation = AnimationUtils.loadAnimation(this, R.anim.top_animation);
        //leftAnimation = AnimationUtils.loadAnimation(this,
R.anim.left_aimation);
 LogoImg.setAnimation(topAnimation);
        //Appname.setAnimation(leftAnimation);
 new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                startActivity(new Intent(SplashActivity.this, MainActivity.class));
                finish();
            }
```

```
        },Splash_Screen);
    }
}
```

Step 5 – Creating and launching the drawable for the buttons, functions and design of the application as per the requirements.

Step 6 – Adding the various Encryption and hashing algorithms.

AES.java

```
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

```
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

```
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

```
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;
    }
```

Vigenerecipher.java

```
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;
  }
}
```

HashMain.java

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.hash_main, container, false);
        ((AppCompatActivity) getActivity()).getSupportActionBar().hide();
getActivity().getWindow().setFlags(WindowManager.LayoutParams.FLAG_F
ULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
        Switch = view.findViewById(R.id.Swtich);
        Hash_Buuton = view.findViewById(R.id.hash_Buuton);
```

```
        Answer = view.findViewById(R.id.Answer);
        Textfield_Text = view.findViewById(R.id.TextArea);
        Textfield_salt = view.findViewById(R.id.salt);
    return view;
     }
    public void hash(View view) throws Exception {
        if (Textfield_Text.length() == 0) {
            Toast.makeText(view.getContext(), "Enter a message to Hash",
Toast.LENGTH_SHORT).show();
            return;
        }
        message = String.valueOf(Textfield_Text.getText());
        salt = String.valueOf(Textfield_salt.getText());
        String Algorithm = String.valueOf(Switch.getText());
        String answer="";
        switch (Algorithm) {
            case "MD5":
                answer=hashText("MD5",salt,message);
                Answer.setText(answer);
                break;
```

```
            case "SHA-256":
                answer=hashText("SHA-256",salt,message);
                Answer.setText(answer);
                break;
            case "SHA-512":
                answer=hashText("SHA-512",salt,message);
                Answer.setText(answer);
                break;
        }
    }
public void switchAlgho(View  view) {
        reset(null);
        String SwitchValue = Switch.getText().toString();
        switch (SwitchValue) {
            case "MD5":
                Switch.setText("SHA-256");
                break;
            case "SHA-256":
                Switch.setText("SHA-512");
                break;
```

```
            case "SHA-512":
                Switch.setText("MD5");
                break;
        }
    }
    public String hashText(String algo,String salt, String plainText)
            throws NoSuchAlgorithmException {
        MessageDigest m = MessageDigest.getInstance(algo);
        m.reset();
        if (salt.length() != 0) {
            m.update(salt.getBytes());
        }
        m.update(plainText.getBytes());
        byte[] digest = m.digest();
        BigInteger bigInt = new BigInteger(1,digest);
        String hashtext = bigInt.toString(16);
        // Now we need to zero pad it if you actually want the full 32 chars.
        while(hashtext.length() < 32 ){
            hashtext = "0"+hashtext;
        }
```

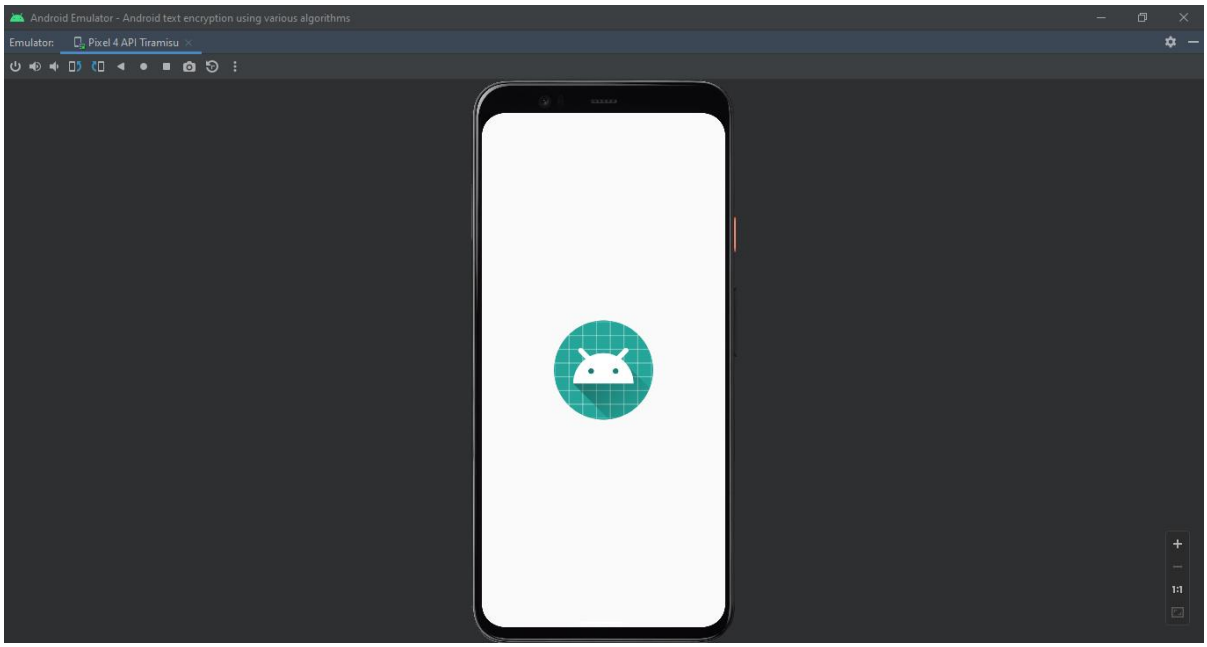
```
        return hashtext;
    }
 private  byte[] getRandomSalt() throws NoSuchAlgorithmException,
NoSuchProviderException
    {
        //Always use a SecureRandom generator
        SecureRandom sr = SecureRandom.getInstance("SHA1PRNG", "SUN");
        //Create array for salt
        byte[] salt = new byte[16];
        //Get a random salt
        sr.nextBytes(salt);
        //return salt
        return salt;
    }
public void reset(View view) {
        Textfield_Text.setText("");
        Textfield_salt.setText("");
        Answer.setText("");
        if(view!=null)
```

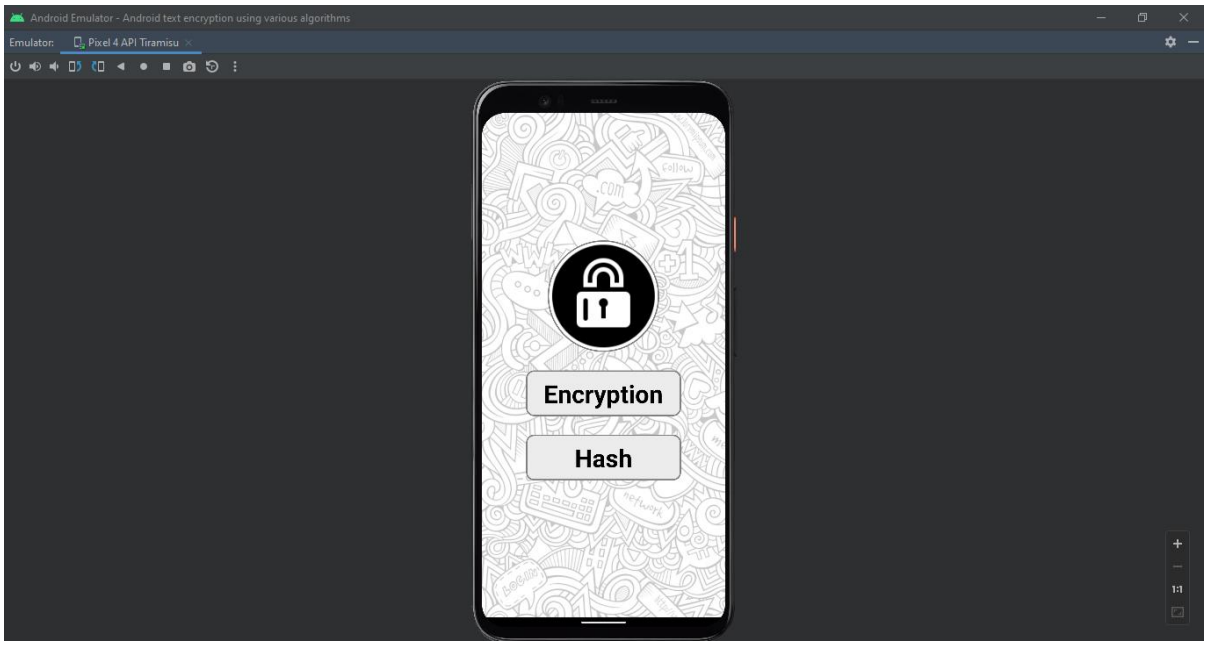
```
        Toast.makeText(view.getContext(), "All data has been deleted",
Toast.LENGTH_SHORT).show();
    }
  public void copyToClipboard(View view) {
          String copyText = String.valueOf(Answer.getText());
          if (Answer.length() == 0) {
              Toast.makeText(view.getContext(), "There is no message to copy",
Toast.LENGTH_SHORT).show();
              return;
          }
         int sdk = android.os.Build.VERSION.SDK_INT;
          if (sdk < android.os.Build.VERSION_CODES.HONEYCOMB) {
              android.text.ClipboardManager clipboard =
(android.text.ClipboardManager)

view.getContext().getSystemService(Context.CLIPBOARD_SERVICE);
              clipboard.setText(copyText);
          } else {
              android.content.ClipboardManager clipboard =
(android.content.ClipboardManager)
```

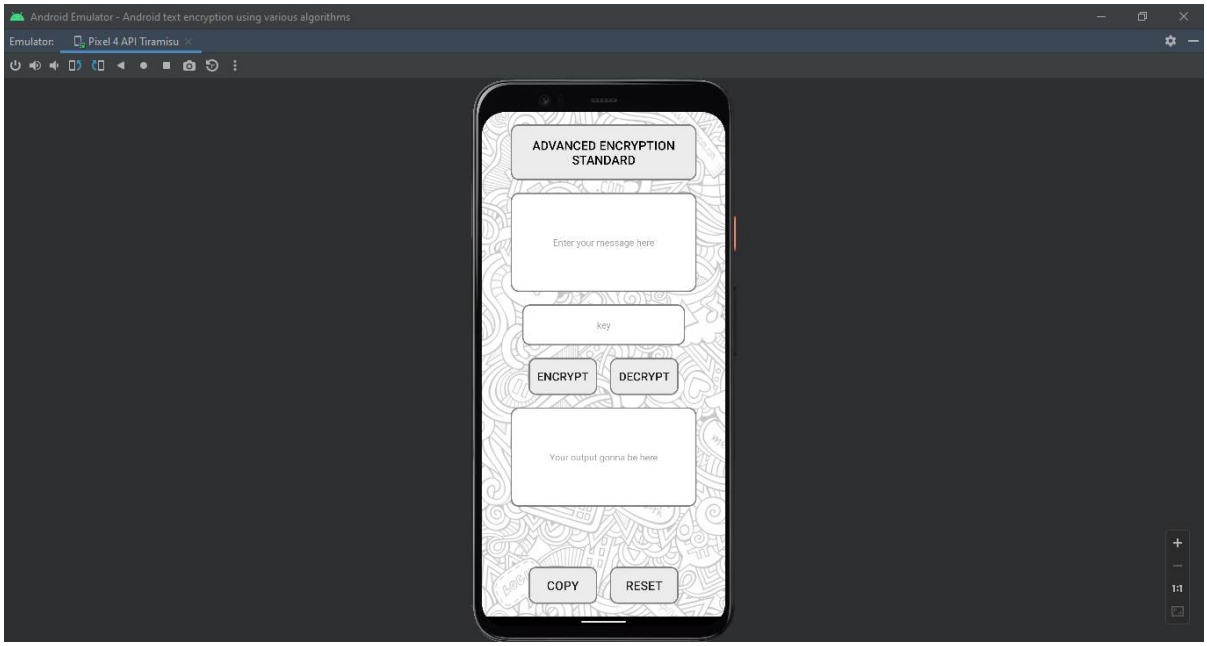
```
view.getContext().getSystemService(Context.CLIPBOARD_SERVICE);
            android.content.ClipData clip = android.content.ClipData
                    .newPlainText("Your message :", copyText);
            clipboard.setPrimaryClip(clip);
        }
        Toast.makeText(view.getContext(),
                "Your message has be copied", Toast.LENGTH_SHORT).show();
    }
}
```

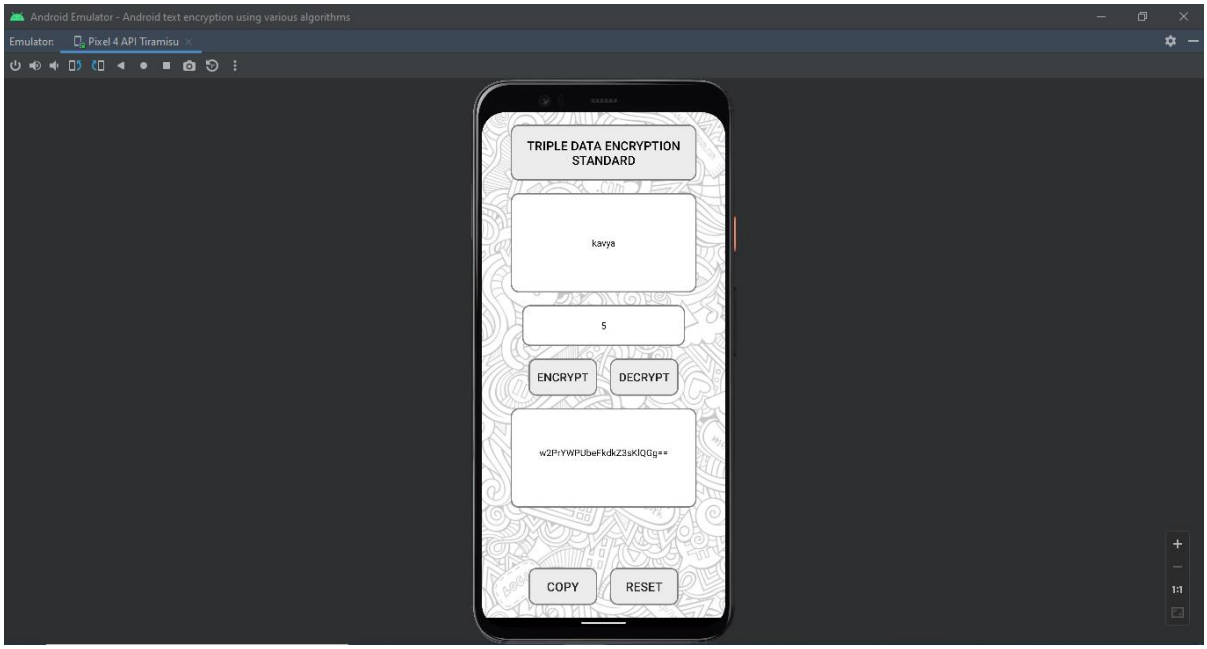
Step 7- Run the application.

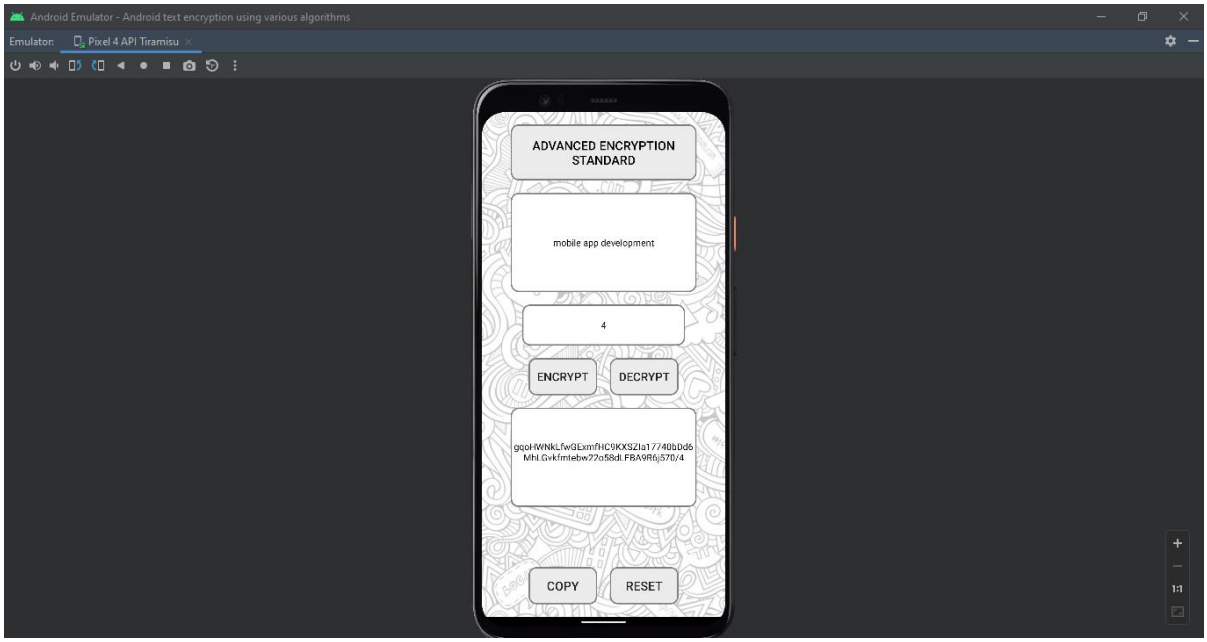
RESULTS

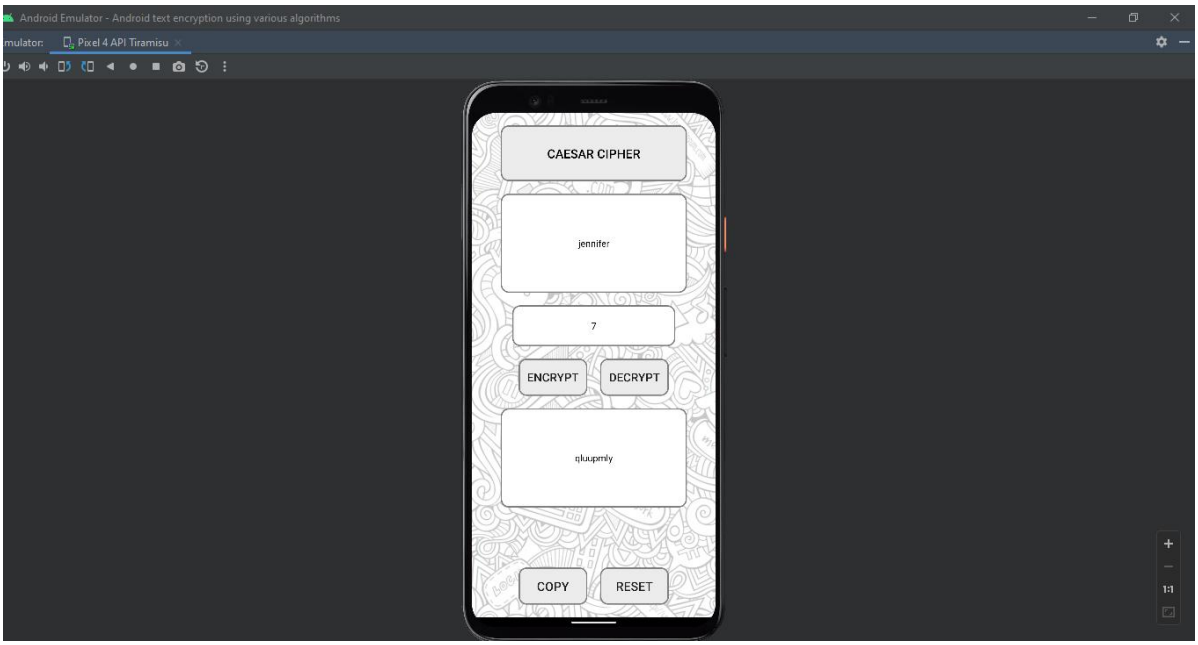

Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu

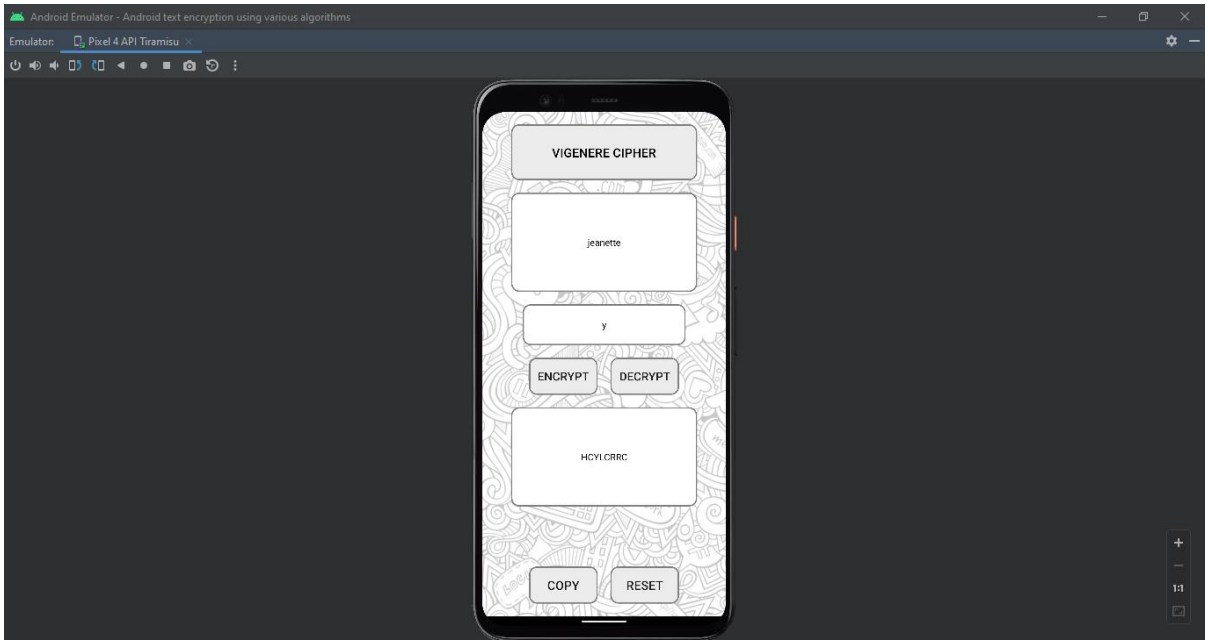

Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
Encryption
Hash

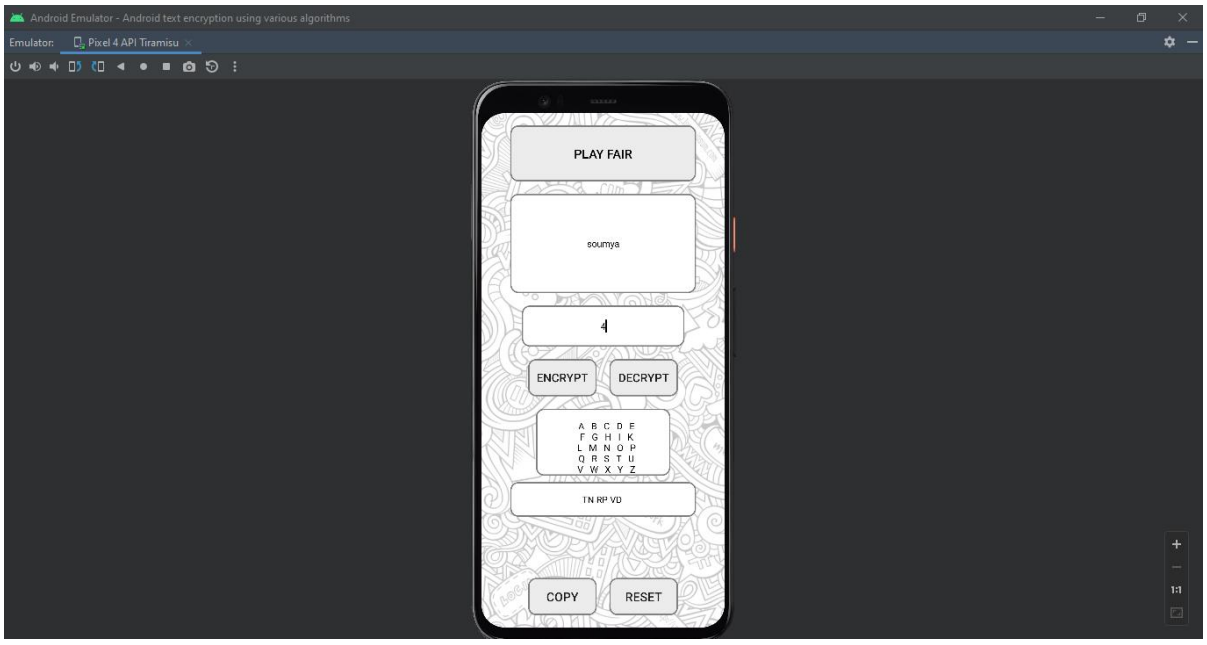

Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
ADVANCED ENCRYPTION STANDARD
Enter your message here
key
ENCRYPT
DECRYPT
Your output gonna be here
COPY
RESET

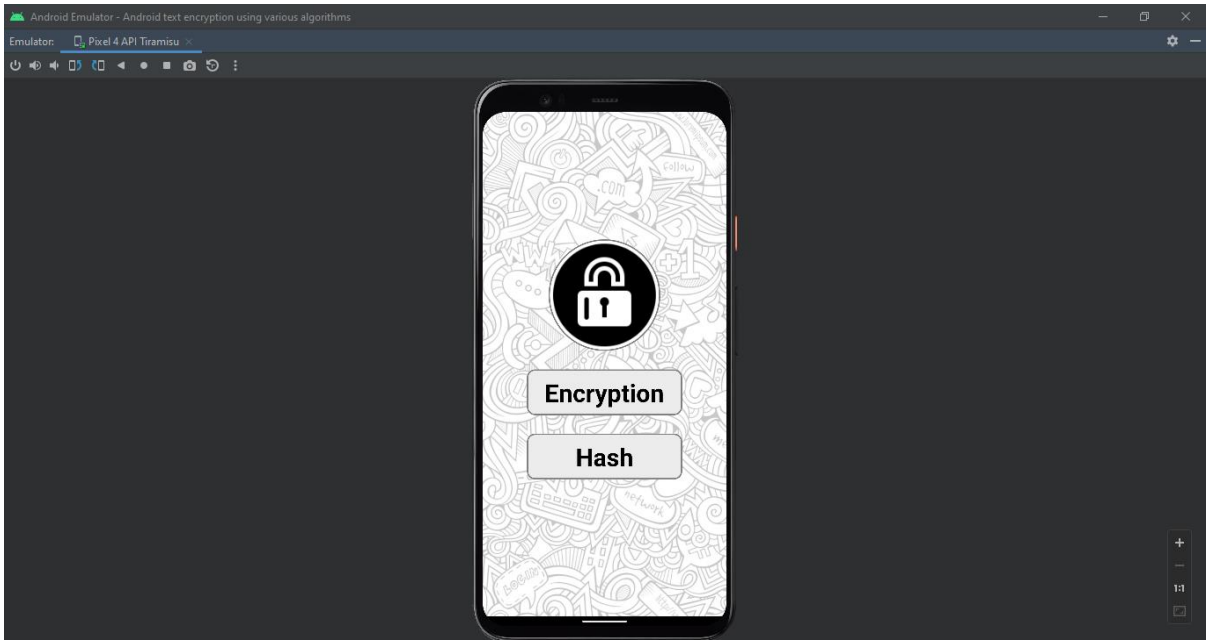

Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
TRIPLE DATA ENCRYPTION STANDARD
kavya
5
ENCRYPT
DECRYPT
w2PrYWPUbeFkdkZ3sKlQGg==
COPY
RESET

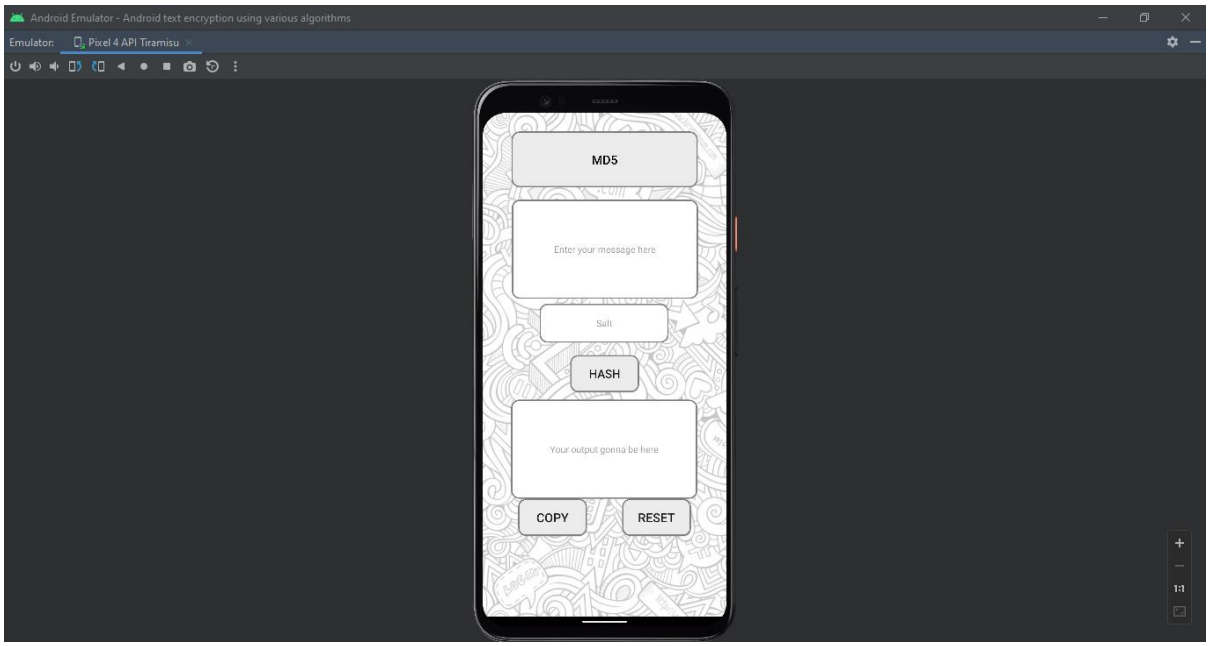

Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
ADVANCED ENCRYPTION STANDARD
mobile app development
4
ENCRYPT
DECRYPT
gqoHWNkLfwGExmfHC9KXSZIa17740bDd6
MhLGvkfmtebw72o58dl.FBA9R6j570/4
COPY
RESET

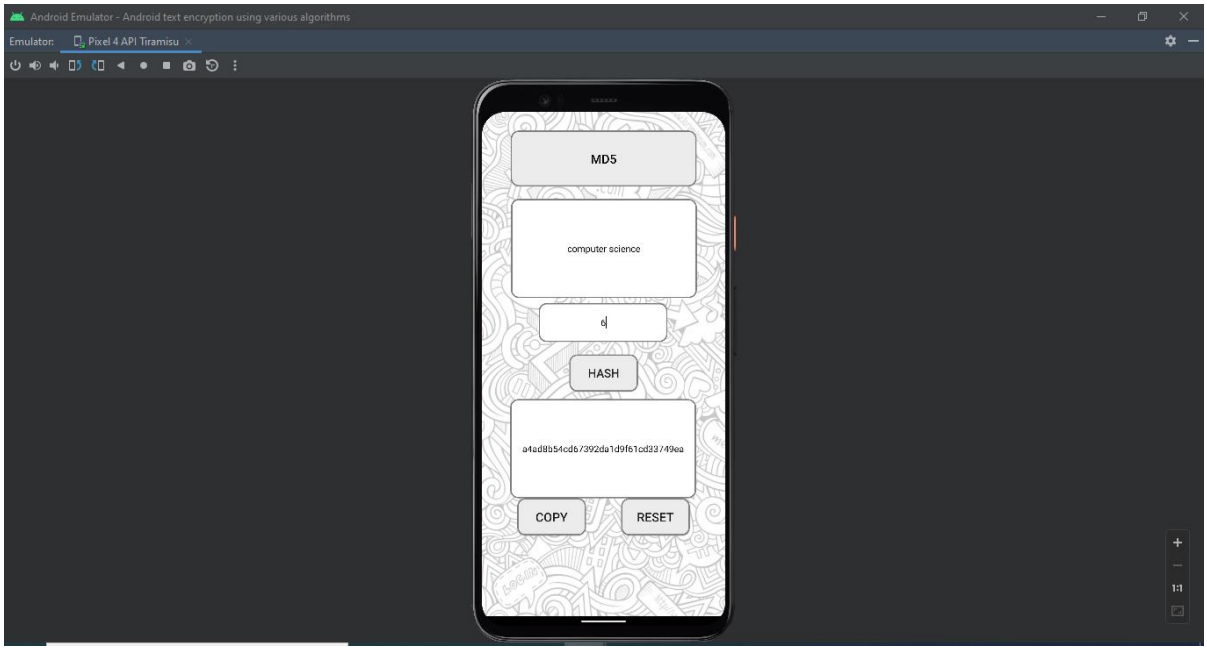

Android Emulator - Android text encryption using various algorithms
Pixel 4 API Tiramisu
CAESAR CIPHER
jennifer
7
ENCRYPT
DECRYPT
qluupmly
COPY
RESET

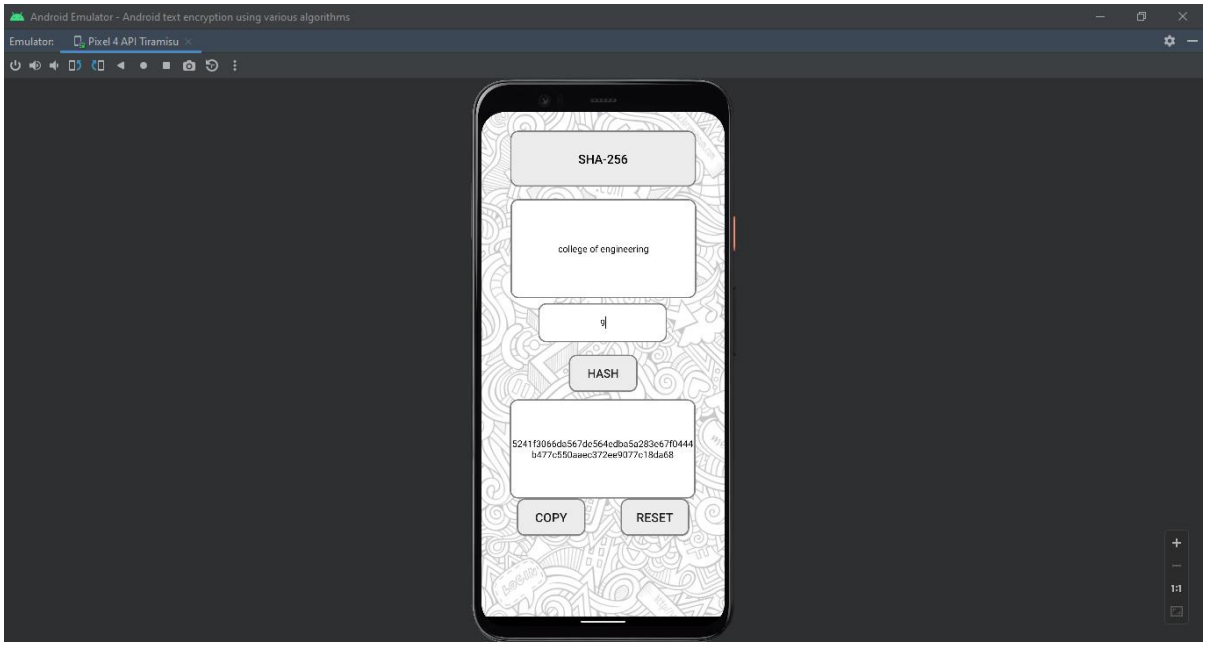

Android Emulator - Android text encryption using various algorithms
Emulator: Pixel 4 API Tiramisu
VIGENERE CIPHER
jeanette
y
ENCRYPT
DECRYPT
HCYLCRRC
COPY
RESET

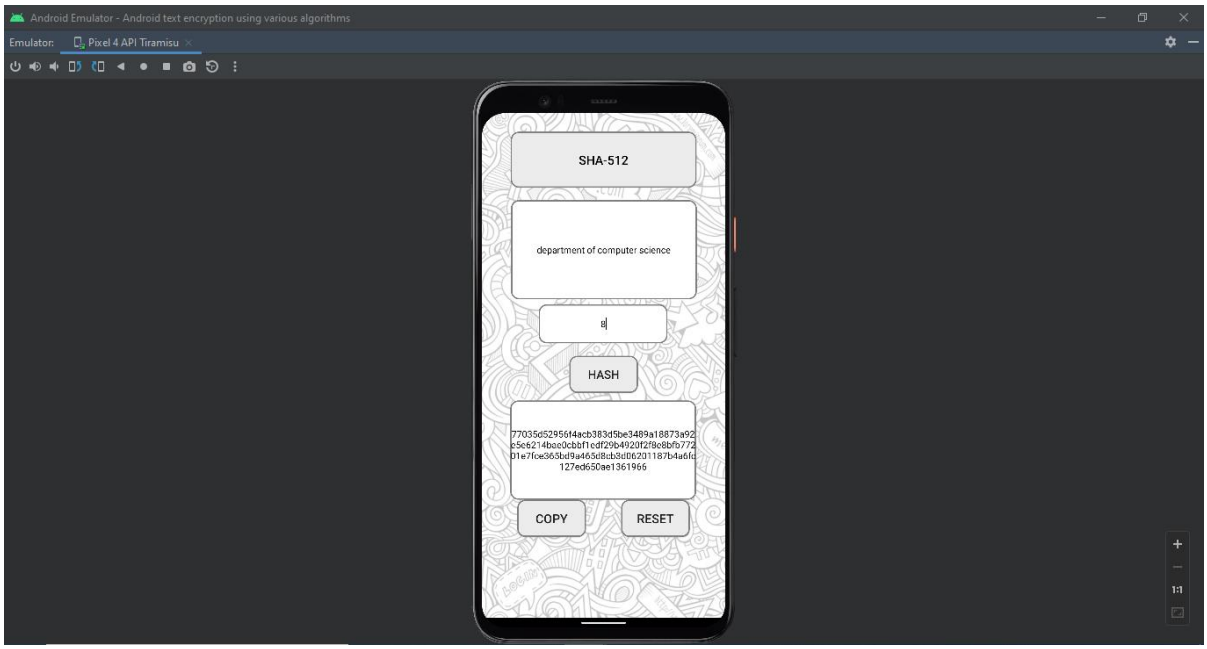

Android Emulator - Android text encryption using various algorithms
Emulator: Pixel 4 API Tiramisu
PLAY FAIR
soumya
4
ENCRYPT
DECRYPT
A B C D E
F G H I K
L M N O P
Q R S T U
V W X Y Z
TN RP VD
COPY
RESET


Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
Encryption
Hash


Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
MD5
Enter your message here
Salt
HASH
Your output gonna be here
COPY
RESET


Android Emulator - Android text encryption using various algorithms
Emulator: Pixel 4 API Tiramisu
MD5
computer science
HASH
a4ad8b54cd67392da1d9f61cd337 49ea
COPY
RESET


Android Emulator - Android text encryption using various algorithms
Emulator: Pixel 4 API Tiramisu
SHA-256
college of engineering
HASH
5241f3066da567dc564cdba5a283c67f0444
b477c5f0aaec372ee9077c18da68
COPY
RESET


Android Emulator - Android text encryption using various algorithms
Emulator:
Pixel 4 API Tiramisu
SHA-512
department of computer science
8
HASH
COPY
RESET

\

APPLICATIONS

If you want to send sensitive information via email, simply paste the encrypted text into your email or attach the encrypted file, all the recipient has to do is to decrypt your text or file. Encryption and Decryption works with text information and files. Just enter the data/text you want to encrypt, and using the Encryption and Decryption app keep and share your documents, private information and files in a confidential way.

ADVANTAGES AND DISADVANTAGES

Advantages:

• Fast and easy way of to send secure stuff.

• Easy process to encrypt text.

• Highly secure as type of algorithm and secret key is required while encryption and decryption.

Disadvantages:

• Password have to be shared which can be hacked and used.

• Only small length of text can be sent like hardly 2-3 lines.

• Requires active internet connection.

CONCLUSION

Hence, this application can be used to encrypt, decrypt ad hash data/text based on the user requirement using various algorithms.

There is always a room for improvement in any software application, however efficient the system may be. The application is flexible enough for future modifications. The application has been factored into different modules to make the system adapt to further changes. Every effort has been made to cover all the user requirements and make it user friendly. Appropriate messages and tool tips have been provided wherever necessary.

In conclusion, in the present world, where everything is linked to each other through networks, data is easily available at the hands of many and can be used by anyone for their own needs as and when required regardless of one's privacy. Hence, encryption is a very important step to ensure the safety and privacy of one's data as it can be easily acquired and misused for malicious purposes.

REFERENCES

- https://stackoverflow.com
- https://developer.android.com
- https://github.com
- https://www.geeksforgeeks.org