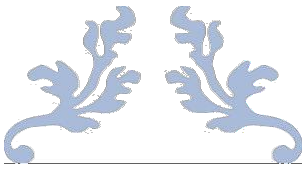
CSE1011-CRYPTOGRAPGHY FUNDAMENTALS



Fingerprint Encryption using AES algorithm

Project Supervisor: Prof. Govinda K

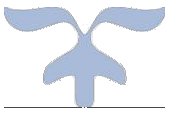


Table of Contents:

Abstract:

Fingerprint in hash cryptography is small key that helps to identify long public key. Fingerprints are generally used for key authentication and other elements of cryptography security, providing efficiency with the smaller data sets.

The keys used for authentication are very long and its not easy for the user to authenticate himself, but in the case of fingerprint its is the most easiest way to prove himself, so that it becomes the toughest job for the intruder to hack and stole the message but the tougher job is to decrypt the cipher text where he needs to authenticate himself

When fingerprints are used for the key authentication, systems can check these smaller data sets which are more easily in order to make sure that they are accessing the correct public key. Fingerprints are also more efficient for the storage. And the proposed method is DES we are going to implement the following project in AES and going to compare the m and we are going to find the best

Security certificate systems may manually perform key authentication to promote the best security practices. The fingerprints are important in public key cryptography and other modern types of digital security. They help us to refine the ways that cryptography works in modern systems, where the streamlining data sets is essential

Scope and Objectives:

The main scope of this project is to encrypt the message using a set of fingerprints and to decrypt with the same set of fingerprints.

In this way, we can safely convey the message. For this we are going to use one of the most popular secure and fastest algorithms named as AES

Introduction:

Biometrics recognition or also called biometric the most unique way to represent and to identify a person and one of the easiest ways to identify the intruder and to prevent data breaches and the main reason for biometrics is due to large number of populations

It is the most convenient way for the person to authenticate his identity, although iris face voice etc are used but fingerprints and palm geometry are mostly used but every method has its own positive's and negative points about it

Fingerprints are globally used by many official government bodies for authentication and in our old days we used to give our fingerprints in the form of ink and we used to keep them on some property documents and but things changed we are in a digital world we made many inventions and many methods and proposals and machine's came in to our life

we are mostly depending on them and they made our lives easy and now there are many techniques came in to check to record our fingerprints and even we are using to identify ourselves and we developed in such a way that we are converting the lines present on our fingerprint

Many came into our life and one of the most important things in our day to day life is communication and many messages chats and secret messages are transmitted and there has to be channel in between to share and that channel needed to be protected from intruders and we should not send the message directly

We need to send our message in special format in such a way that the intruder cannot able to decode our message and but there has to be something in common between user for that many inventions came and many algorithms were proposed

In that one of the most secured are DES,AES,3D_DES,RSA,Twofish,RC4 etc this here in this project we are going to use your fingerprint as our key and we are going to encrypt a message and decrypt the same with the receiver's key here on we for this we are using AES algorithm and we are using android emulator to show the mechanism in the form of an application based on android architecture

We are using the android key store to store the fingerprint and use it according to requirement of authentication, So already existing method is in DES we are going to propose for AES and we are going to compare the efficiency

Time taken using MATLAB although we learnt kotlin although th proposed is in kotlin and we also have shown key in the form of log.

So Here we are encrypting a message using the senders fingerprint and decrypt the message application so that it can the access the fingerprint store

- **DISADVANTAGES OF EXISTING METHOD:**

1.)The main disadvantage of DES is we can hack the key by trail an error and method also known as brute force method where we can get the partial key out

2.)Although,we can also hack the key by linear cryptoanalysis and there are amny ways to get the partial key out and decrypt our secret mesage

- **ADVANTAGES OF PROPOSED METHOD:**

1.) As per many factors,AES tops the best of all in software and hardware

2.)We will be using higher length keys like 128,193,256 so that hacking and partial hacking key is very difficult to decrypt the message
3it’s the most world wide used algorithm in many of the transmission ways and its widely found to be the best

4.)And one more factor is its an open source and free for all to use

5.)Although we are using long bit keys the decryption part by intruding and knowing the encrypted message and trying to decrypt is a tough thing

6.) To encrypt a 128 bit length key we almost need 2100+ attempts by trail and error method so as we increase the bit length the number of attempts to hack will increase exponentially

Pseudocode:

AES:

```
KeyExpansion(byte key[4 *
Nk], word w[Nb * (Nr + 1)], Nk)
begin
   i=0
   while (i < Nk)
      w[i] = word[key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]]
i=i+1 end while
   i = Nk
   while (i < Nb * (Nr + 1))
      word temp = w[i - 1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
      else if (Nk = 8 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
w[i] = w[i - Nk] xor temp i=i+1
end while
end
```

Bio-auth-process (Person P)

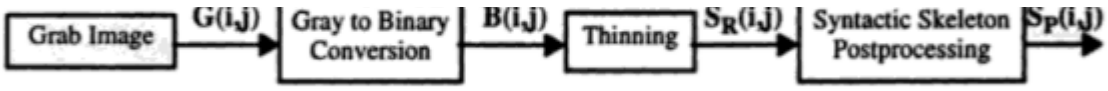
```
{

P.IDFP ← getFingerPrint(P)

P.IDIR ← getIris(P)

P.IDFA ← getFace(P)

P.IDFP ← feature_extract (P.IDFP)

P.IDIR ← feature_extract (P.IDIR)

P.IDFA ← feature_extract (P.IDFA)

for (i-0;i< dbtemplateFinger.length;i++)
```

```
{
Check(dbtemplateFinger(i))
If (dbtemplateFinger(i)) ← P.IDFP)
{
Q.feat ← feature_extract(dbtemplateFinger(i))
If (AFSASVM matcher(Q) ← P.IDFP)
P.IDFP belongs to user Q
Set P.IDFP_status ← 1}
else
set P.IDFP_status ← 0 // P has failed fingerprint authentication
}
for (j=0;j< dbtemplateIris.length;j++)
{
Check(dbtemplateIris(j))
If (dbtemplateIris(j)) ← P.IDIR)
{
Q.feat ← feature_extract(dbtemplateIris(j))
If (AFSASVM matcher(Q) ← P.IDIR)
set P.IDIR_status ← 1
P.IDIR belongs to user Q
}
else
set P.IDIR_status ← 0 // P has failed iris authentication
}
If ((P.IDFP_status ← 1) AND (P.IDIR_status ← 1))
{
P has passed multimodal authentication
return}
else If ((P.IDFP_status ← 0) AND (P.IDIR_status ← 0))
{
```

```
P has failed multimodal authentication
exit}
else
If (P.IDFP_status ← 1) OR (P.IDIR_status ← 1)
{
for (k=0;k< dbtemplateFace.length;k++)
{
Check(dbtemplateIFace(k))
If (dbtemplateFace(k)) ← P.IDFA)
{
Q.feat ← feature_extract(dbtemplateFace(k))
If (AFSASVM matcher(Q) ← P.IDFA)
set P.IDFA_status ← 1
P.IDFA belongs to user Q
P has passed multimodal authentication
return
}
else {
set P.IDFA_status ← 0
exit // P has failed face and multimodal authentication }
}
}
```

Literature Review:

For **this** we need to enrol our fingerprint and we need to understand the process within fingerprint enrolment initially the sensor captures the image and converts the captured image into binary[X] format using some length techniques and it is done on the basis of the skeleton of the fingerprint and although the length of the binary will be so long and we need to reduce thin the format



Fig(1): Pre processing of an fingerprint in an over view

The ridges present over our fingerprints are unique and the raised position is considered and recorded in the form of digits also called epidermis the gap between the ridges is recorded and recorded on the basis of certain 5 fractions[XI] The numbers allotted to every print ar supported whether or not or not they're whorls. A whorl within the initial fraction is given a sixteen, the second Associate in Nursing eight, the third a four, the fourth a a pair of, and zero[VII] to the last fraction. Arches and loops ar allotted values of zero. Lastly, the numbers within the dividend and divisor ar additional up, victimization the scheme:

$$Ri/Rt + Rr/Rm + Lt/Rp + Lm/Li + Lp/Lr$$

Fig(2)-The fractions are as follows where L for left, m for middle,t for thumb, i for index, p for little finger

The pattern needs to be recognised and needed to measure physical difference between the ridges and valleys and many things make the fingerprint unclear such as noise

dust and many more and this result in inconsistent and non-uniform images of the finger print[XII] and the pattern are of generally 3 types arch,loop,whorl and they were recorded and checked again when tried to authenticate and the minutiae types are of ridge ending,bifurication,short,iland,lake,spur,bridge,delta,core, these types are recorded and checked

While zooming the fingerprint helps us to understand better and we can see the miniature's and lines in it [VI] and fingerprint s convert the 3d image into binary code and saves it in the form of alpha-binary form in the case of android

For this we are going to use android studio and its emulator to display our implementation firstly we are going to enrol our finger print using android device bridge[I] and we are using platform tools to do it and we are going to show this in the form of an app firstly

MainActivity.XML and for showing the key we need to add a line in KeyStore tools and imported log module to do so and we changed add the interface and changed the code in the main activity.xml and declared the attribute's using some id and while construction the application

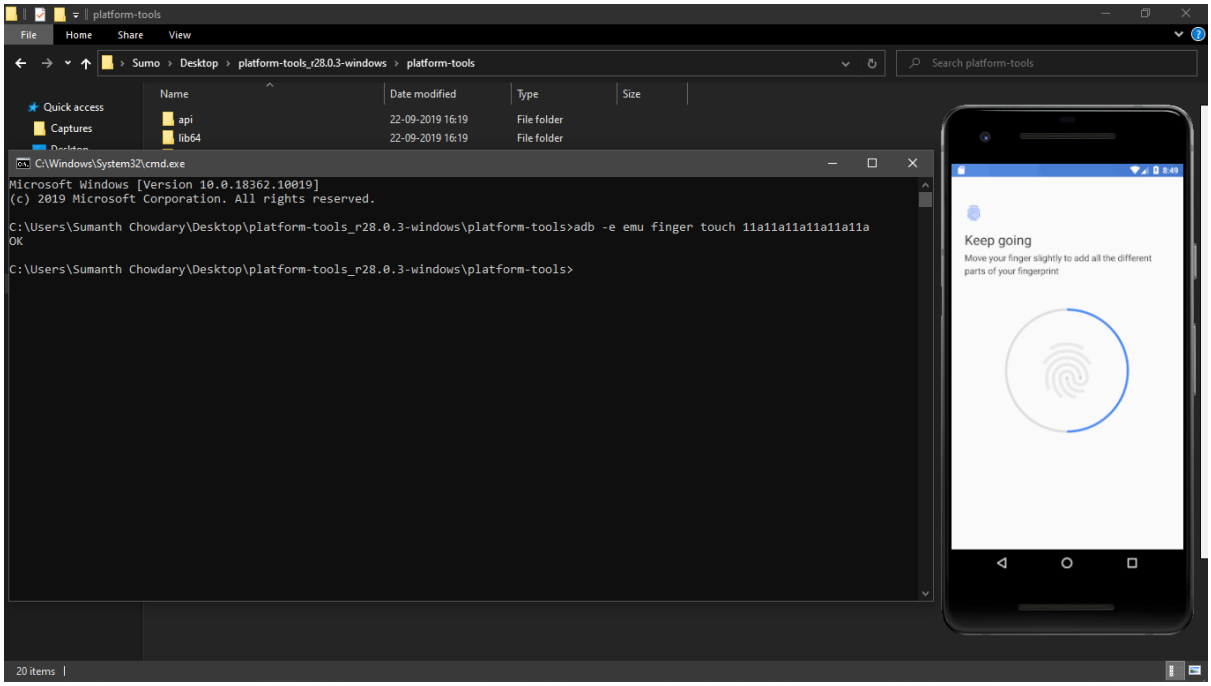
We displayed the key in the log and displayed the status of authentication in the form of toast[III]we need to understand the interface of studio although we exported the project which is in DES and we imported the predefined the AES module in

We declared the attributes on click and so that on cling the button certain code that declared in the background runs accordingly and changed the encryption type and executed it

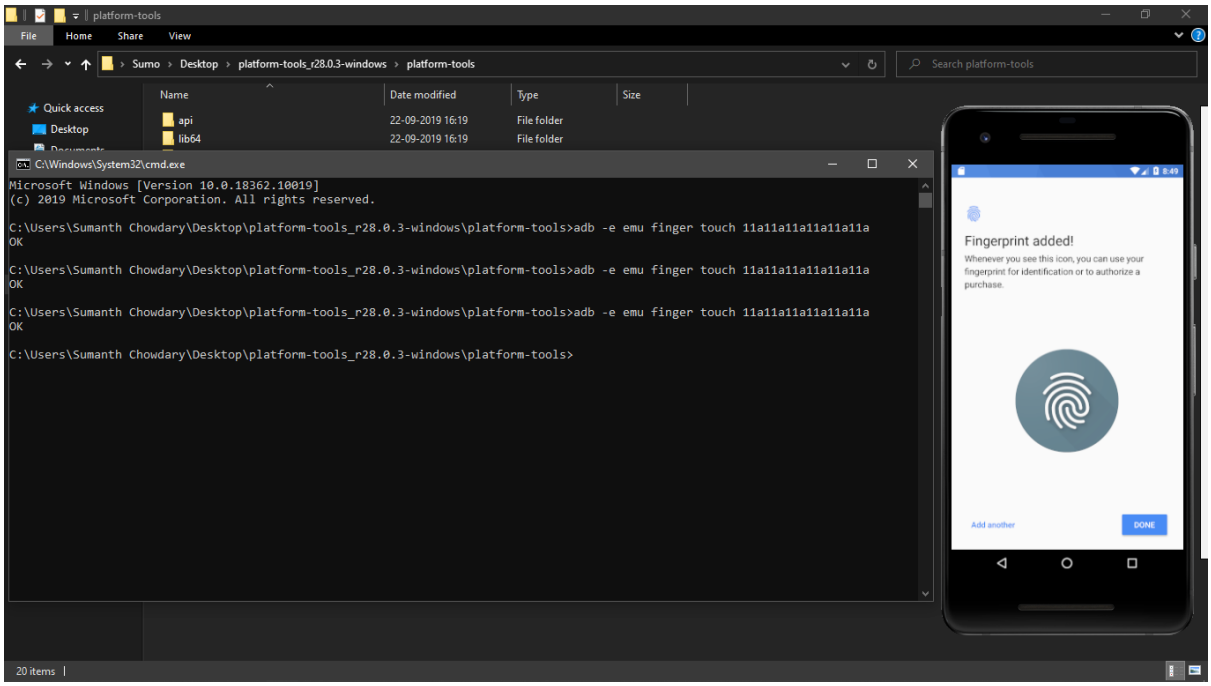
Enrolling Fingerprint Manually:

Use will use Android Device Bridge(adb) to enroll finger print using platform tools in emulator we will be using command in adb terminal and enter fingerprint details manually to add our fingerprint in emulator

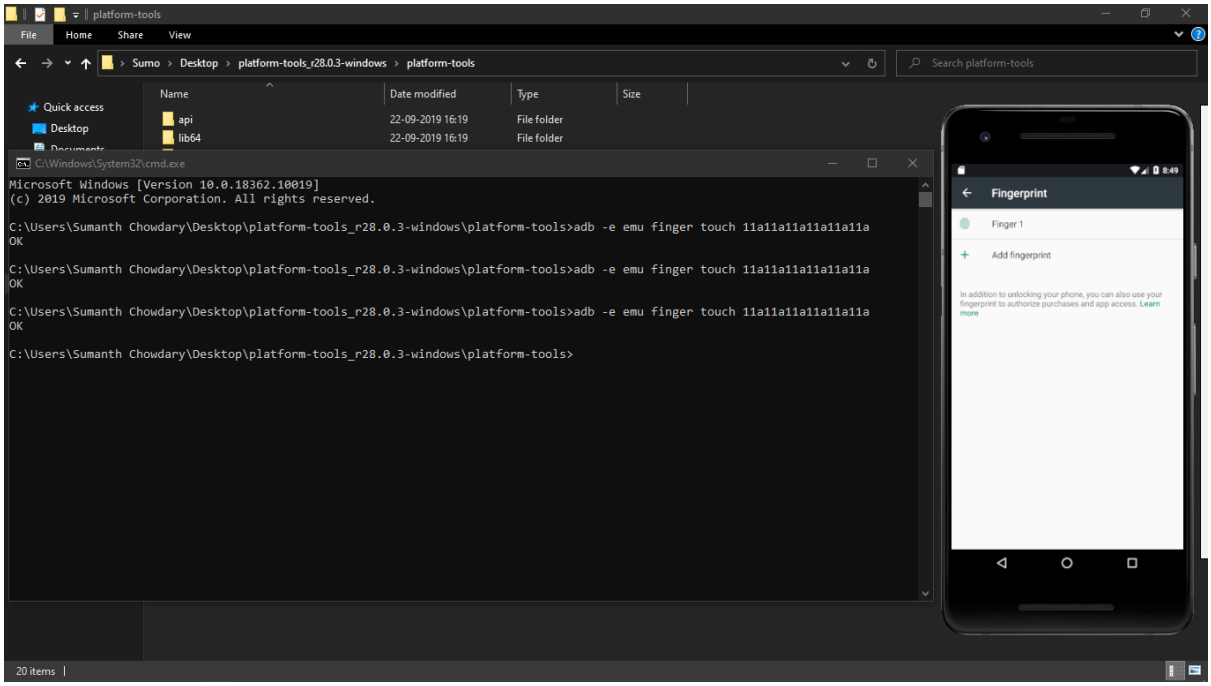
```
adb -e emu finger touch [finger_id]
```



Fig(3)-Enrolling a fingerprint using adb shell



Fig(4)-Succesfully added fingerprint to emulator



Fig(5)-Enrolled fingerprint in fingerprint's manager

Code:

```
package com.michalrola.fingerprintauth

import android.Manifest
import android.app.KeyguardManager
import android.content.Context
import android.content.SharedPreferences
import android.content.pm.PackageManager
import android.hardware.fingerprint.FingerprintManager
import android.os.Bundle
import android.support.v4.app.ActivityCompat
import android.support.v7.app.AppCompatActivity
import android.widget.EditText
import android.widget.Toast
import com.michalrola.fingerprintauth.authentication.EncryptionObject
import kotlinx.android.synthetic.main.activity_main.*


class MainActivity : AppCompatActivity() {


    companion object {
        val TAG = MainActivity::class.java.simpleName
        private const val SECURE_KEY = "data.source.prefs.SECURE_KEY"
    }

    private lateinit var fingerprintManager: FingerprintManager
    private lateinit var keyguardManager: KeyguardManager
    private lateinit var cryptoObjectEncrypt: FingerprintManager.CryptoObject
    private lateinit var cryptoObjectDecrypt: FingerprintManager.CryptoObject
    private var encryptedMessage: String = "" //should have: message + separator +
IV from first cipher
    private val separator = "-"
```

```
    private lateinit var pref: SharedPreferences
    private lateinit var editor: SharedPreferences.Editor


        //    private lateinit var encryptedTextView: TextView
//    private lateinit var decryptedTextView: TextView
//    private lateinit var encryptButton: Button
//    private lateinit var decryptButton: Button
//    private lateinit var listenerButton: Button
//    private lateinit var listenerButtonEnc: Button
    private lateinit var pinEditText: EditText

    private val encryptionObject = EncryptionObject.newInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

//        encryptedTextView = findViewById(R.id.encryptedTextView) as TextView
//        decryptedTextView = findViewById(R.id.decryptedTextView) as TextView
//        encryptButton = findViewById(R.id.encryptButton) as Button
//        decryptButton = findViewById(R.id.decryptButton) as Button
//        listenerButton = findViewById(R.id.listenerButton) as Button
//        listenerButtonEnc = findViewById(R.id.listenerButtonEnc) as Button
        pinEditText = findViewById(R.id.pinEditText) as EditText

        keyguardManager = getSystemService(Context.KEYGUARD_SERVICE) as
KeyguardManager
        fingerprintManager = getSystemService(Context.FINGERPRINT_SERVICE) as
FingerprintManager


        pref = this.getSharedPreferences(
            "com.michalrola.secure.pref",
            Context.MODE_PRIVATE
        )

        editor = pref.edit()
        checkFingerprint()

        val encryptedTextFromSharedPref =
            if (pref.getString(SECURE_KEY, null) != null)
pref.getString(SECURE_KEY, null) else ""

        encryptedTextView.text = encryptedTextFromSharedPref
        encryptedMessage = encryptedTextFromSharedPref


        listenerButtonEnc.setOnClickListener {
            createFingerprintHandlerEnc()
        }

        encryptButton.setOnClickListener {
            encryptMessage()


        }
        listenerButton.setOnClickListener {
            createFingerprintHandlerDec()
        }

        decryptButton.setOnClickListener {
            decryptMessage()
        }
    }

    private fun decryptMessage() {
        val mess = pref.getString(SECURE_KEY, null).split(separator)[0]
        val decryptedData = encryptionObject.decrypt(
            encryptionObject.cipherDec,
            mess
        )
```

```
            decryptedTextView.text = decryptedData
        }

        private fun encryptMessage() {
            try {
                encryptedMessage = encryptionObject.encrypt(
                    encryptionObject.cipherEnc,
                    pinEditText.text.toString().toByteArray(Charsets.UTF_8),
                    separator
                )
                editor.putString(SECURE_KEY, encryptedMessage)
                editor.apply()

                encryptedTextView.text = encryptedMessage
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }

        private fun createFingerprintHandlerEnc() {
            try {
                cryptoObjectEncrypt =
    FingerprintManager.CryptoObject(encryptionObject.cipherForEncryption())
                val fingerprintHandler = FingerprintHandler(this)
                fingerprintHandler.startAuth(fingerprintManager, cryptoObjectEncrypt)
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }

        private fun createFingerprintHandlerDec() {
            try {
                cryptoObjectDecrypt = FingerprintManager.CryptoObject(
                    encryptionObject.cipherForDecryption(
                        pref.getString(SECURE_KEY, null).split(separator)[1].replace("\
    n", "")
                    )
                )
                val fingerprintHandler = FingerprintHandler(this)
                fingerprintHandler.startAuth(fingerprintManager, cryptoObjectDecrypt)
            } catch (e: java.lang.Exception) {
                e.printStackTrace()
            }
        }

        private fun checkFingerprint() {
            if (!keyguardManager.isKeyguardSecure) {
                Toast.makeText(this,
    getString(R.string.fingerpint_not_entabled_message), Toast.LENGTH_SHORT).show()
                return
            }

            if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.USE_FINGERPRINT)
                    != PackageManager.PERMISSION_GRANTED
            ) {
                Toast.makeText(
                    this,
                    getString(R.string.fingerpint_permissions_not_entabled_message),
                    Toast.LENGTH_LONG
                ).show()
                return
            }
            if (!fingerprintManager.hasEnrolledFingerprints()) {
                Toast.makeText(
                    this,
                    getString(R.string.fingerpint_not_registered_message),
                    Toast.LENGTH_LONG
                ).show()
                return
            }
        }
```

```
}
```

Keystore Tools:

```
package com.michalrola.fingerprintauth.authentication.tools

import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import java.security.InvalidAlgorithmParameterException
import java.security.KeyStore
import java.security.NoSuchAlgorithmException
import java.security.NoSuchProviderException
import javax.crypto.KeyGenerator
import javax.crypto.SecretKey
import android.util.Log

class KeyStoreTools {
    companion object {
        private const val ANDROID_KEY_STORE = "AndroidKeyStore"

        private fun keyExists(keyName: String): Boolean {
            val keyStore = KeyStore.getInstance(ANDROID_KEY_STORE)
            keyStore.load(null)
            val aliases = keyStore.aliases()

            while (aliases.hasMoreElements()) {
                if (keyName == aliases.nextElement()) {
                    return true
                }
            }

            return false
        }

        fun getKeyFromKeyStore(keyname: String): SecretKey {
            val keyStore = createKeyStore()
            if (!keyExists(keyname)) {
                generateAesKey(keyname)
            }
            return keyStore.getKey(keyname, null) as SecretKey
        }

        private fun createKeyStore(): KeyStore {
            val keyStore = KeyStore.getInstance(ANDROID_KEY_STORE)
            keyStore.load(null)
            return keyStore
        }

        private fun generateAesKey(keyName: String) {
            try {
                val keyGenerator = KeyGenerator.getInstance(
                    KeyProperties.KEY_ALGORITHM_AES,
                    ANDROID_KEY_STORE

                )
                val builder = KeyGenParameterSpec.Builder(
                    keyName,
                    KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
                )
                builder.setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                    .setKeySize(256)
                    .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)

                builder.setUserAuthenticationRequired(true)
                keyGenerator.init(builder.build())
                keyGenerator.generateKey()

            } catch (e: NoSuchAlgorithmException) {
                throw RuntimeException("Failed to create a symmetric key", e)
            } catch (e: NoSuchProviderException) {
```

```
                throw RuntimeException("Failed to create a symmetric key", e)
            } catch (e: InvalidAlgorithmParameterException) {
                throw RuntimeException("Failed to create a symmetric key", e)
            }

        }
    }

}
```

Encryption Object:

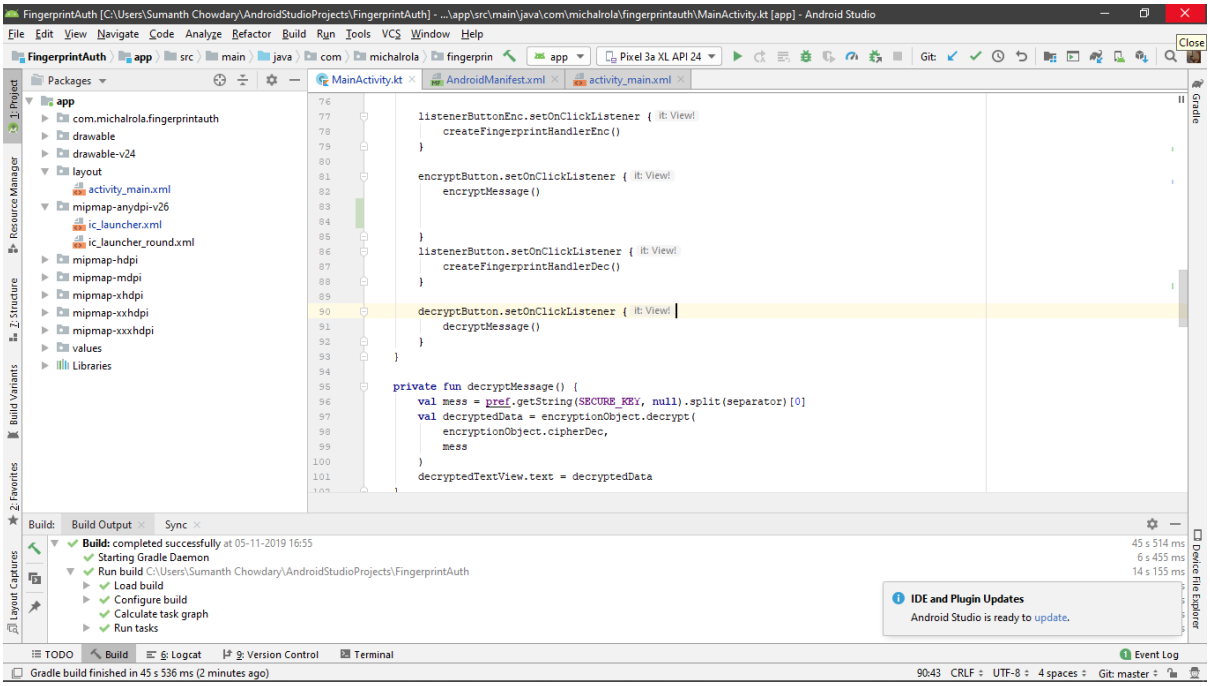
```
package com.michalrola.fingerprintauth.authentication

import android.security.keystore.KeyPermanentlyInvalidatedException
import android.security.keystore.KeyProperties
import android.util.Base64
import android.util.Log
import com.michalrola.fingerprintauth.authentication.tools.KeyStoreTools
import javax.crypto.Cipher
import javax.crypto.SecretKey
import javax.crypto.spec.IvParameterSpec


class EncryptionObject {
    companion object {
        private const val KEY_NAME = "FingerprintKey"
        fun newInstance(): EncryptionObject {
            return EncryptionObject()
        }
    }


    private val key: SecretKey =
        KeyStoreTools.getKeyFromKeyStore(KEY_NAME)

    val cipherEnc: Cipher = createCipher()
    val cipherDec: Cipher = createCipher()

    fun encrypt(cipher: Cipher, plainText: ByteArray, separator: String): String {
        val enc = cipher.doFinal(plainText)
        return Base64.encodeToString(
            enc,
            Base64.DEFAULT
        ) + separator + Base64.encodeToString(
            cipher.iv,
            Base64.DEFAULT
        )

        return ""
    }

    fun decrypt(cipher: Cipher, encrypted: String): String {

        return cipher.doFinal(
            Base64.decode(
                encrypted,
                Base64.DEFAULT
            )
        ).toString(Charsets.UTF_8)
    }

    fun cipherForEncryption(): Cipher {
        try {
            cipherEnc.init(Cipher.ENCRYPT_MODE, key)
            Log.i("Info",key.toString());

            return cipherEnc
        } catch (e: KeyPermanentlyInvalidatedException) {
            throw RuntimeException("Key Permanently Invalidated", e)
        } catch (e: Exception) {
```

```
                throw RuntimeException("Failed to init Cipher", e)
            }

        }

        fun cipherForDecryption(IV: String): Cipher {
            try {
                cipherDec.init(
                    Cipher.DECRYPT_MODE, key, IvParameterSpec(
                        Base64.decode(
                            IV.toByteArray(Charsets.UTF_8),
                            Base64.DEFAULT
                        )
                    )
                )
                return cipherDec
            } catch (e: KeyPermanentlyInvalidatedException) {
                throw RuntimeException("Key Permanently Invalidated", e)
            } catch (e: Exception) {
                throw RuntimeException("Failed to init Cipher", e)
            }

        }


        private fun createCipher(): Cipher {
            return Cipher.getInstance(
                KeyProperties.KEY_ALGORITHM_AES + "/"
                        + KeyProperties.BLOCK_MODE_CBC + "/"
                        + KeyProperties.ENCRYPTION_PADDING_PKCS7
            )
        }
    }
```

Android Manifest:

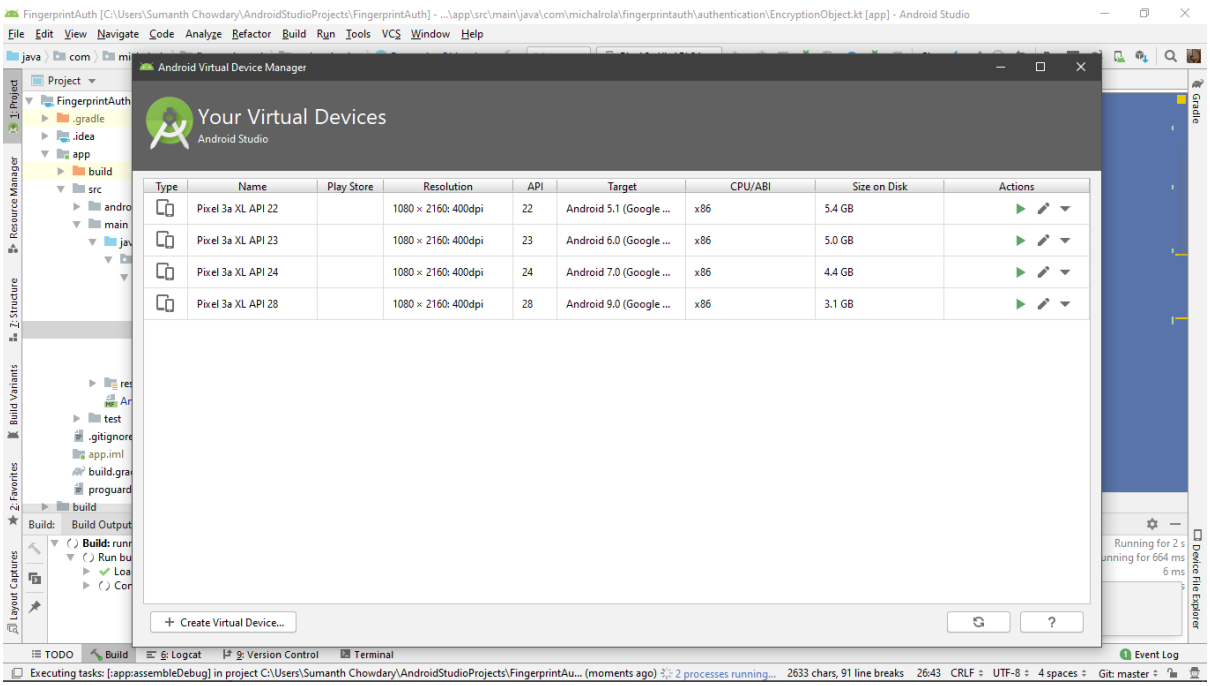
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
package="com.michalrola.fingerprintauth">

<uses-permission android:name="android.permission.USE_BIOMETRIC"/>
<uses-permission android:name="android.permission.USE_FINGERPRINT"/>

<application android:enabled="true"
        android:allowClearUserData="true"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="CSE1011"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" tools:ignore="GoogleAppIndexingWarning">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>

            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>

</manifest>
```

Compilation:

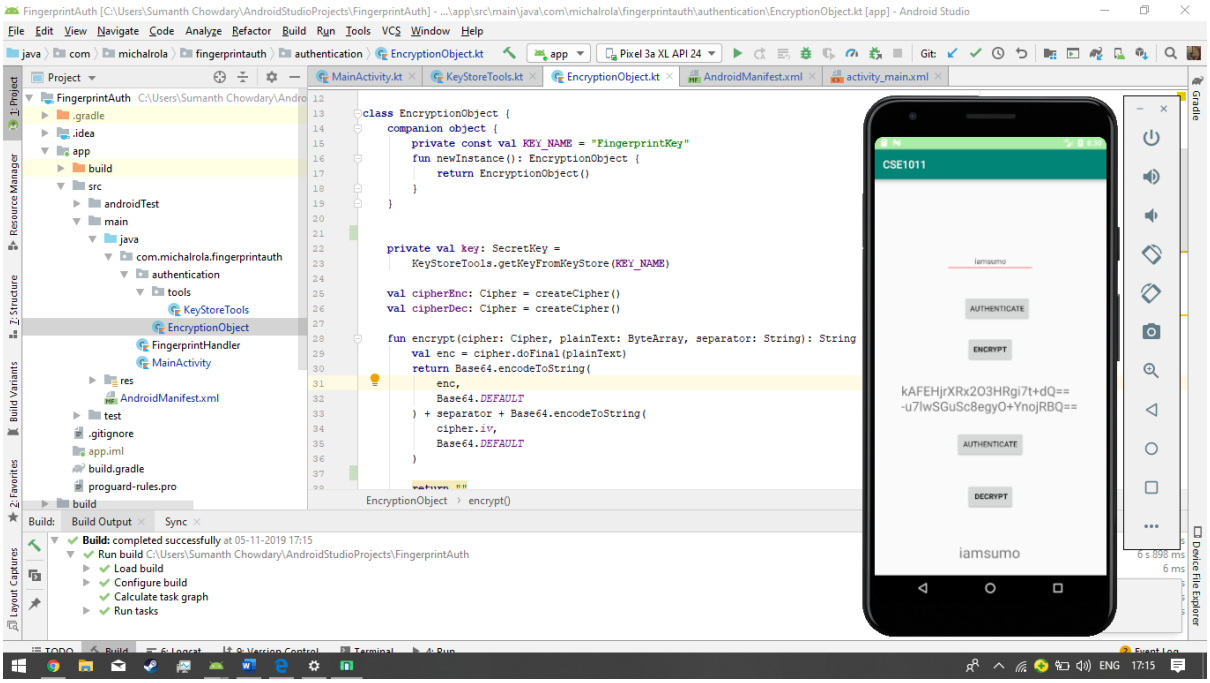


Fig(6)-Successful compilation of build

Emulator:

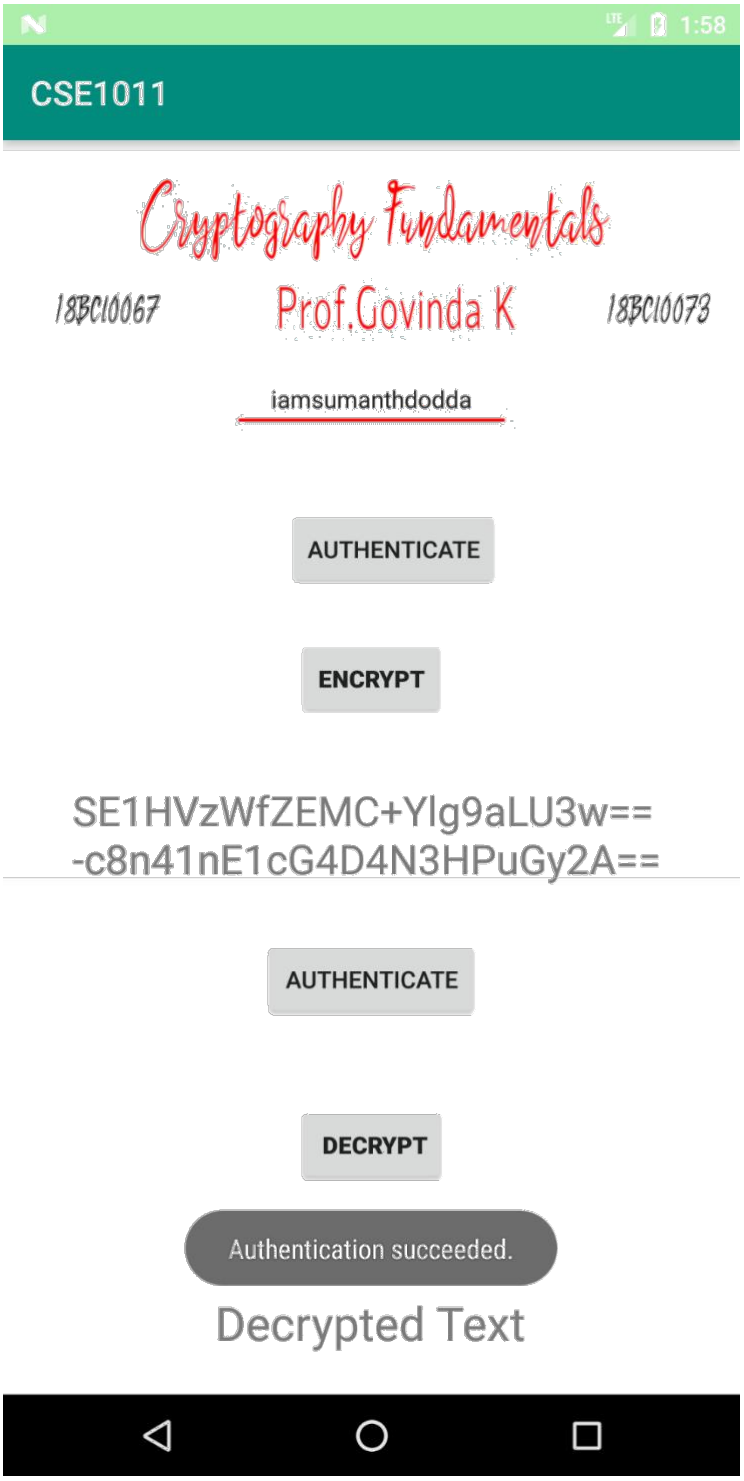


Fig(7)-Shows the Android Virtual Device Manager(AVD)



Fig(8)-Shows the active virtual device

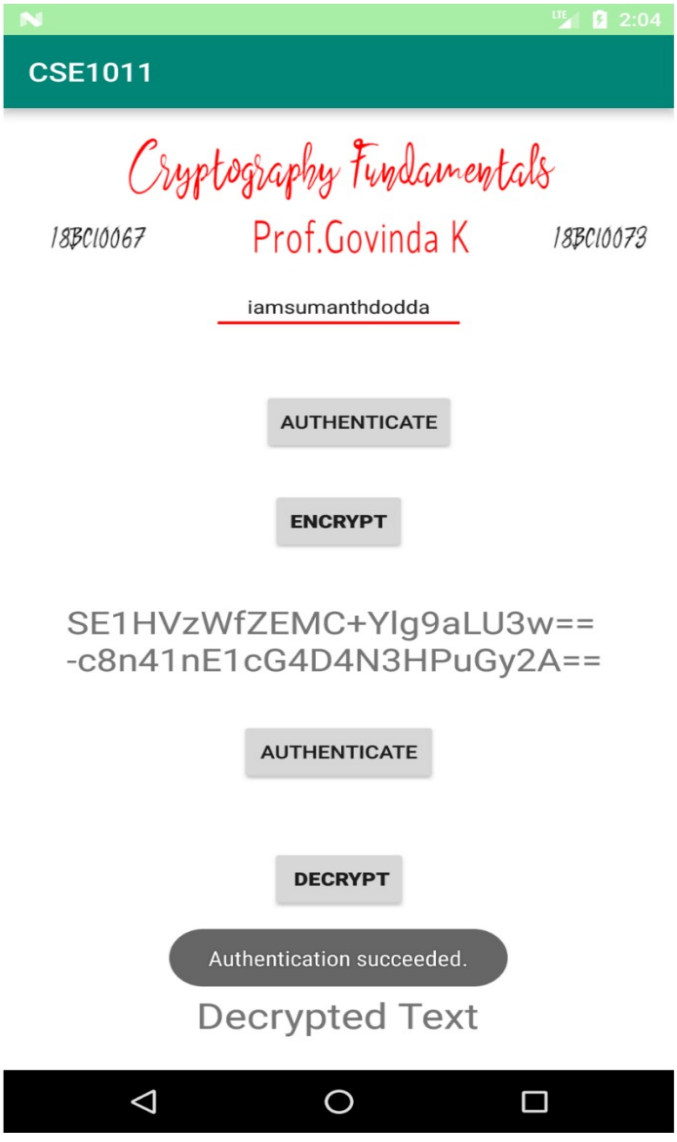
Testing:



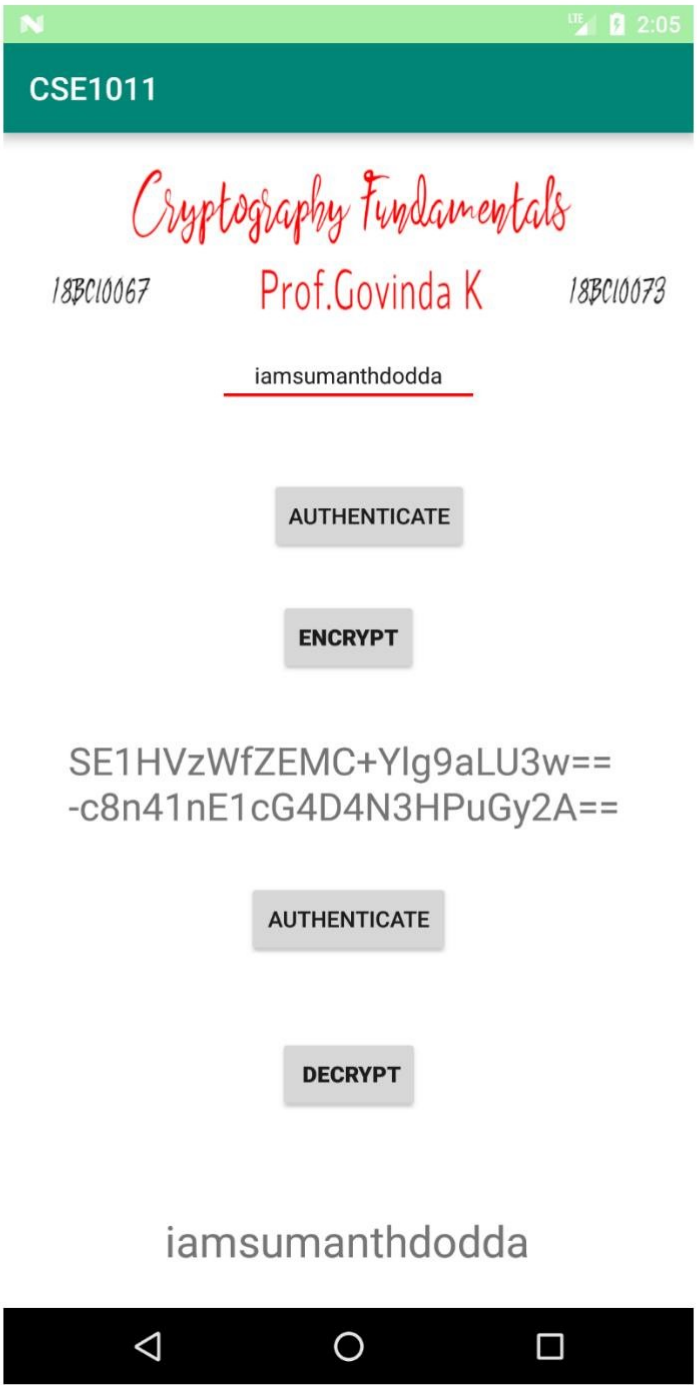
Fig(9)-Shows the status of authentication after verification of sender's fingerprint

Authencating 2nd person's fingerprint to decrypt:

Fig(10)-Authenticating receiver's Fingerprint



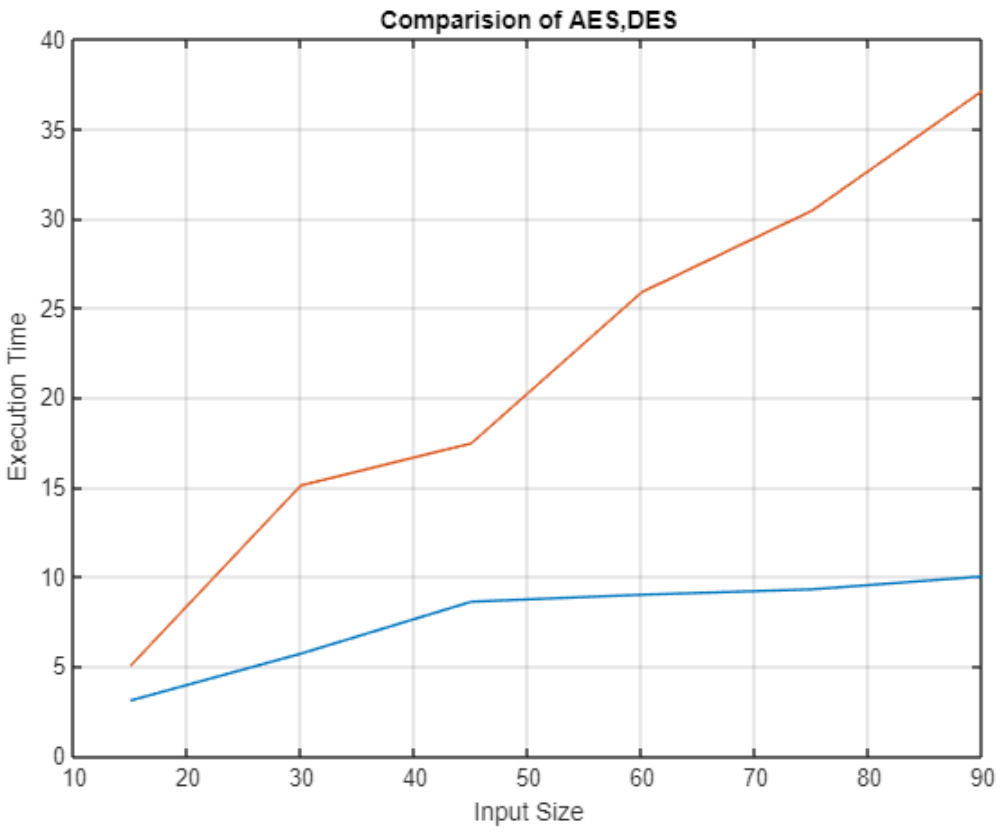
Fig(11)-Shows the cipher text & status of authentication



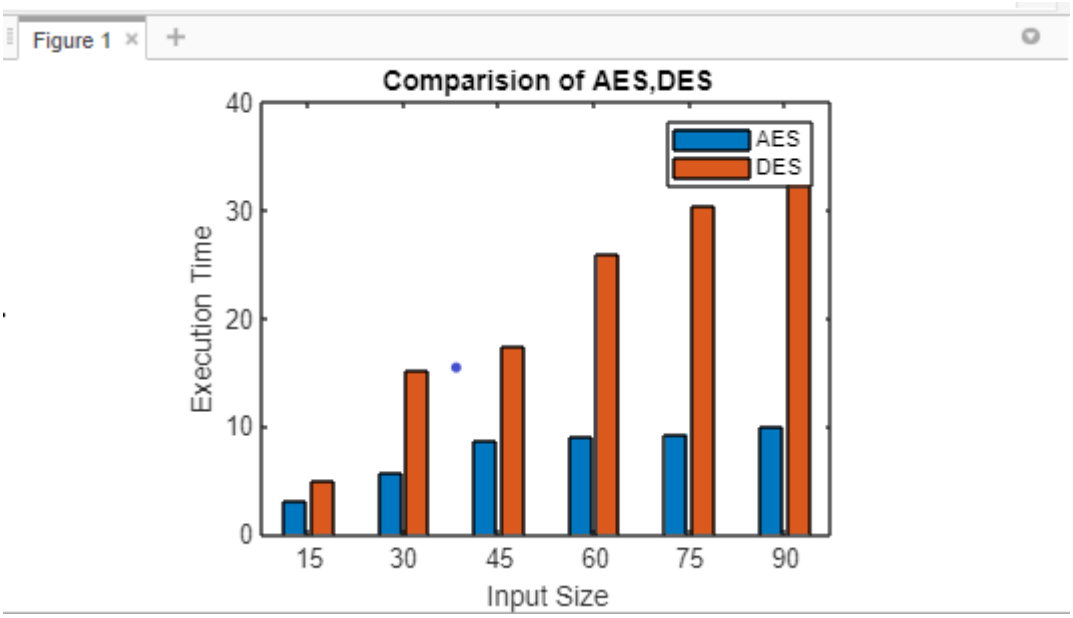
Fig(12)-Shows the message sent after receiver's verification

Comparison b/w proposed method and previous method:

We used MATLAB to plot graphs



Fig(13)-Comparison of execution times of AES & DES(X-Input Size Y-Time)



Fig(14)-Histogram view and shows the AES execution time lesser than DES

Conclusion:

As we are in the world where technology plays an important role in our day to day life, we were daily transforming towards digital life where security plays an important role so we need to enhance more security and the only thing unique for everyone and its very hard to hack is our finger print where we implemented fingerprint as our key and one more factor that is very essential and important is encryption algorithms there are many methods available around like AES ,DES,RSA and many more here the existing method is DES and proposed method is AES where it is more secure and compilation time for AES is less when compared with DES as shown in the above graph so that we can conclude that fingerprint encryption is more unique and more secure and faster.

References:

I. https://stackoverflow.com/questions/35335892/android-m-fingerprintscanner-on-android-emulator
II. https://github.com/rolls01/FingerprintAuth.git
III. https://www.javatpoint.com/kotli n-android-toast
IV. https://books.google.co.in/books?hl=en&lr=&id=WfCowMOvpioC&oi=fnd&pg=PA1&dq=biometrics+authentication&ots=xqWF-Uv3Kj&sig=5LGdObW8OggftCU2AGTMHnou3fg#v=onepage&q=biometrics%20authentication&f=false
V. https://www.explainthatstuff.com/fingerprintscanners.html
VI. https://www.youtube.com/watch?v=oCY5sPkkaTM
VII. https://en.wikipedia.org/wiki/Fingerprint
VIII. https://www.androidauthority.com/how-fingerprint-scanners-work-670934/
IX. https://www.sciencedirect.com/science/article/abs/pii/S0031320305001445
X. https://books.google.co.in/books?hl=en&lr=&id=1Wpx25D8qOwC&oi=fnd&pg=PR11&dq=fingerprint+recognition&ots=9yR_0Rjtd_&sig=nC57AFfQSmqVZhn-aXWQxrBoehM&redir_esc=y#v=onepage&q=fingerprint%20recognition&f=false
XI. https://en.wikipedia.org/wiki/Pattern_recognition