Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

# FINGERPRINT BASED AES ENCRYPTION

**FOR**

**ITE4001- NETWORK INFORMATION SECURITY**

**BY**

**18BIT0126- NARU ROHITH REDDY**

**18BIT0150- YAGNA SAI SURYA**

**UNDER THE GUIDANCE OF**

**DR. ASWANI KUMAR CHERUKURI**

## CONTENTS OF THE DOCUMENT

| S. NO | CONTENTS | PG. NO |
|---|---|---|
| 1 | ABSTRACT | 3 |
| 2 | INTRODUCTION | 3-4 |
| 3 | LITERATURE SURVEY | 4-5 |
| 4 | FINGERPRINT ADDITION STEPS<br>• Steps in proposed methodology<br>• Key store importance<br>• Key store functions that uses fingerprint<br>• Terminal connection to emulator<br>• Sdk platform tool using<br>• Adding fingerprint via terminal | 5-17 |
| 5 | KEY STORE INTRODUCTION AND FUNCTION | 17-19 |
| 6 | ENCRYPTIONA AND DECRYPTION EXPLAINED | 19-23 |
| 7 | CODES | 23-33 |
| 8 | CONCLUSION | 33 |
| 9 | REFERENCES | 34 |

## 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 it’s not easy for the user to authenticate himself, but in the case of fingerprint it’s 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. 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.

## INTRODUCTION

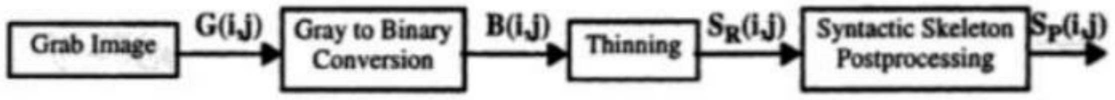
Biometrics recognition or also called biometric the most unique way to represent and to identify a person and one of the easiest ways to identify the intruder and to prevent data breaches and the main reason for biometrics is due to large number of populations It is the most convenient way for the person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

In many algorithms one of the most secured are DES,AES,3D_DES,RSA,Twofish,R C4 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 same 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.

## LITERATURE SURVEY

For this we need to enroll 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 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. 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. The numbers allotted to every print are 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 pair of, and zero to the last fraction. Arches and loops are allotted values of zero. Lastly, the numbers within the dividend and divisor are additional up, victimization the scheme:



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

The pattern needs to be recognized 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 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, short, lake, 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  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 enroll our finger print using android device bridge and we are using platform tools to do it and we are going to show this in the form of an app firstly We displayed the key in the log and displayed the status of authentication in the form of toast we need to understand the interface of studio.

## PROPOSED METHODOLOGY

### Why the android KeyStore?

Android KeyStore system protects key material from unauthorized use. Firstly, Android KeyStore mitigates unauthorized use of key material outside of the Android device by preventing extraction of the key material from application processes and from the Android device as a whole. Secondly, Android KeyStore mitigates unauthorized use of key material on the Android device by making apps specify authorized uses of their keys and then enforcing these restrictions outside of the apps' processes.

## Extraction prevention

Key material of Android KeyStore keys is protected from extraction using two security measures:

Key material never enters the application process. When an application performs cryptographic operations using an Android KeyStore key, behind the scenes plaintext, ciphertext, and messages to be signed or verified are fed to a system process which carries out the cryptographic operations. If the app's process is compromised, the attacker may be able to use the app's keys but cannot extract their key material (for example, to be used outside of the Android device).

Key material may be bound to the secure hardware (e.g., Trusted Execution Environment (TEE), Secure Element (SE)) of the Android device. When this feature is enabled for a key, its key material is never exposed outside of secure hardware. If the Android OS is compromised or an attacker can read the device's internal storage, the attacker may be able to use any app's Android KeyStore keys on the Android device, but not extract them from the device. This feature is enabled only if the device's secure hardware supports the particular combination of key algorithm, block modes, padding schemes, and digests with which the key is authorized to be used. To check whether the feature is enabled for a key, obtain a Key Info for the key and inspect the return value of KeyInfo.isInsideSecurityHardware().

## Hardware security module

Supported devices running Android 9 (API level 28) or higher installed can have a Strongbox Key master, an implementation of the Key master HAL that resides in a hardware security module. The module contains the following:

- Its own CPU.
- Secure storage.
- A true random-number generator.

- Additional mechanisms to resist package tampering and unauthorized sideloading of apps.

When checking keys stored in the Strongbox Key master, the system corroborates a key's integrity with the Trusted Execution Environment (TEE). To support low-power Strongbox implementations, a subset of algorithms and key sizes are supported:

- RSA 2048
- AES 128 and 256
- ECDSA P-256
- HMAC-SHA256 (supports key sizes between 8 bytes and 64 bytes, inclusive)
- Triple DES 168

https://developer.android.com/

the official android developer blog where we can find all the information needed.

## HOW TO STORE MY FINGERPRINT TO KEYSTORE?

There are practical issues with using something like a thumbprint for system access. Every device would have to be equipped with a physical thumbprint reader. Biometric authentication is a user identity verification process that involves biological input, or the scanning or analysis of some part of the body. Biometric authentication methods are used to protect many different kinds of systems - from logical systems facilitated through hardware access points to physical systems protected by physical barriers, such as secure facilities and protected research sites.

Here we use our own fingerprint for biometric authentication. Biometric authentication is widely known as the most effective type of authentication because it is extremely difficult to transfer biological material or features from one user to another. However, the traditional costs of biometric authentication have made it an impossible option for many projects. New technologies are making biometric authentication more realistically feasible for a range of different implementations.

The most common and evolving types of biometric authentication involve facial scanning. Facial scanning tools now have the ability to identify people and can be used for different types of security and authentication. Fingerprint-based authentication is also common. Some types of biometric authentication focus on particular features, such as eyes, whereas others use more comprehensive body scanning models.

Fingerprint manager have been used it belongs to java.lang.Object

```
public class FingerprintManager
extends Object
```

java.lang.Object
  ↳ android.hardware.fingerprint.FingerprintManager

We all know that everyone now a days have the smart phone which re coming with the fingerprint sensors that are able to read our fingerprints and store them as we know that our fingerprint will be stored as the string value where and which the string identifies the particular fingerprint.

Here also in the same way the key store will have this string value stored when we want to use it we convert into binary and add the padding to use it.

Adding my fingerprint to the emulator we will be using the sdk platform tools that help us to give the fingerprint to the android emulator device.

```
<?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.nisproj.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:icon="@mipmap/ic_launcher"
        android:label="ITE4001"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
tools:ignore="GoogleAppIndexingWarning"
    android:allowBackup="true"
    >
    <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>
```

From the above we will ask the user to give access to the fingerprint sensor on the phone in order to use the fingerprint

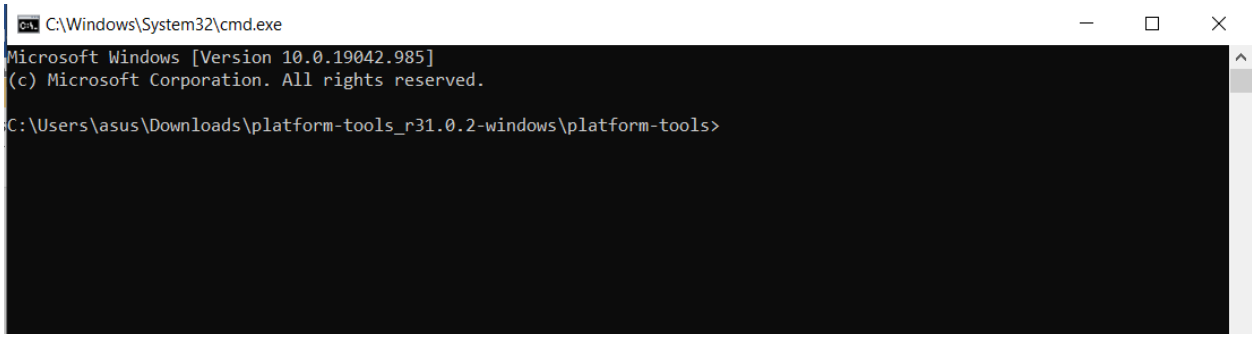
Step-1:

And now the sdk tools to scan fingerprint or give fingerprint input to emulator

Download the sdk from

https://developer.android.com/studio/releases/platform-tools

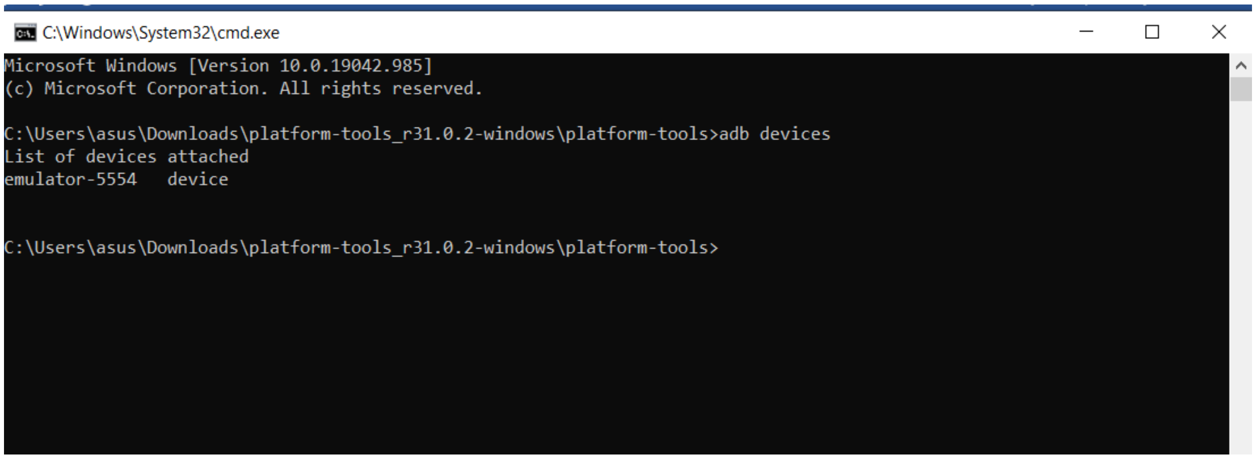
then unzip it to get the files then open cmd in the location then you get



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>
```

Step-2 how to connect the emulator to the terminal cmd



```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb devices
List of devices attached
emulator-5554   device


C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>
```

Emulator-5554 is the emulator id that the android studio was using now we have the emulator connected to the cmd

Step-3 fingerprint enrollement

Open settings in the emulator and go to fingerprint add option

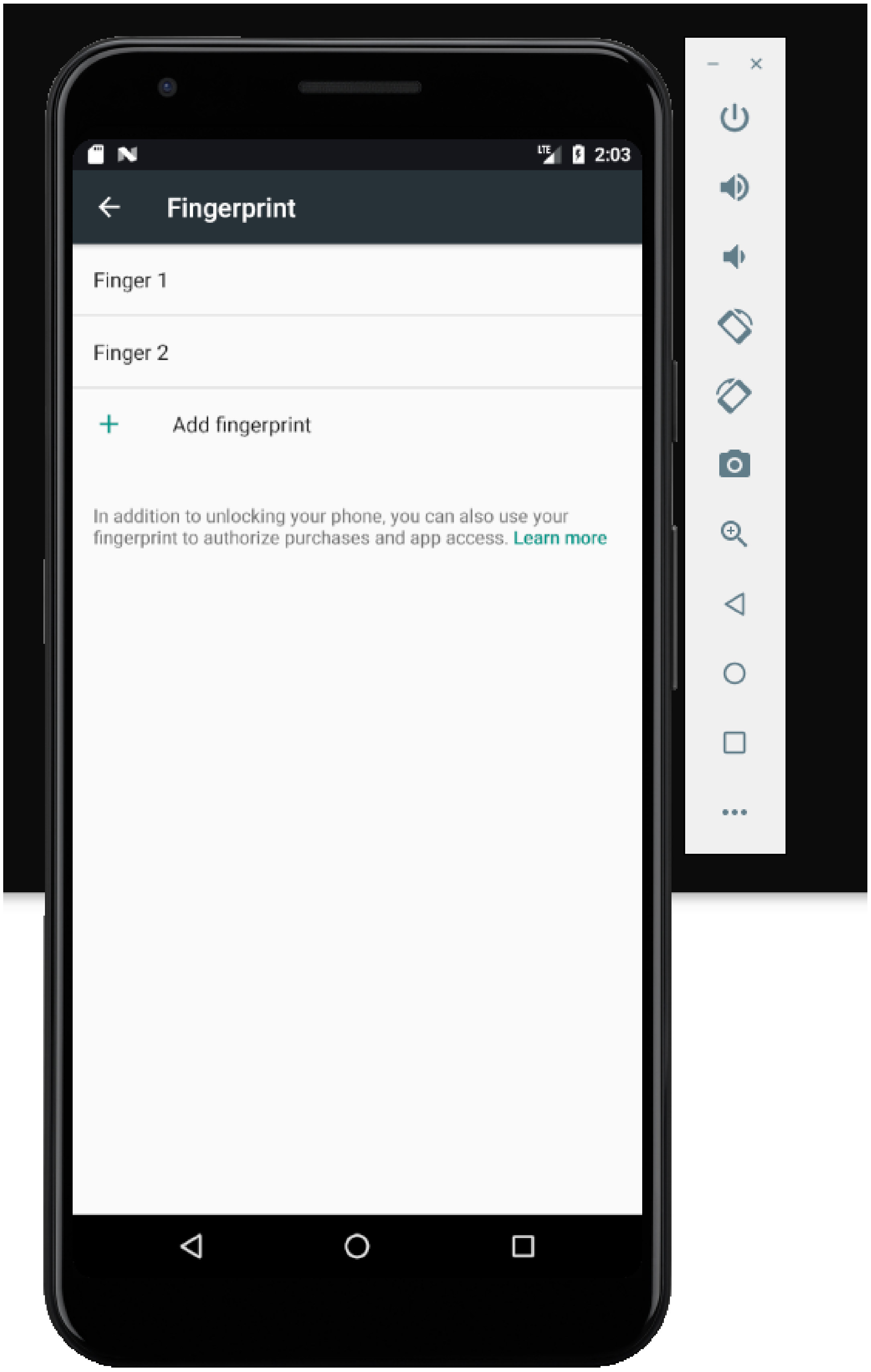
When we hit add fingerprint we have to give the fingerprint input from the terminal

Now initially we had two fingerprints that are added to the emulator now lets just try and add other fingerprint

Via the cmd

Using command

adb -e emu finger touch 333a333b333c



2:03

Fingerprint

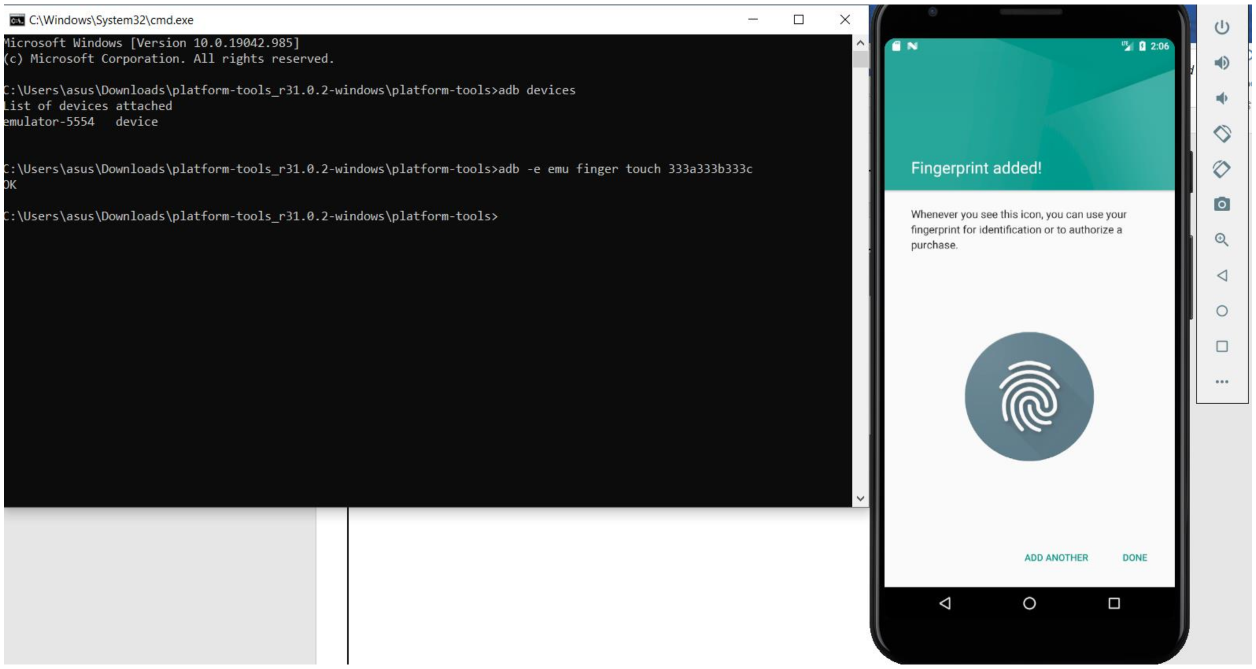
Finger 1

Finger 2

+ Add fingerprint

In addition to unlocking your phone, you can also use your fingerprint to authorize purchases and app access. Learn more

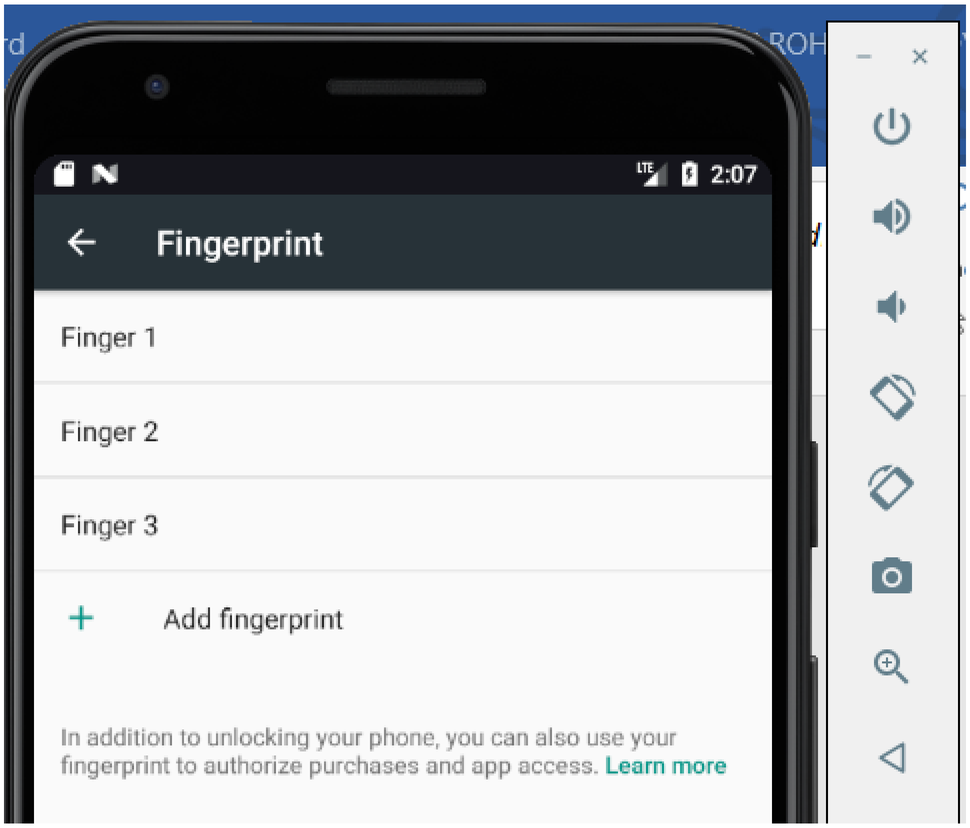
Step-4: finger print has been added successfully to the emulator.


C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb devices
List of devices attached
emulator-5554   device


C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>
```

Fingerprint added!

Whenever you see this icon, you can use your fingerprint for identification or to authorize a purchase.



Fingerprint

Finger 1

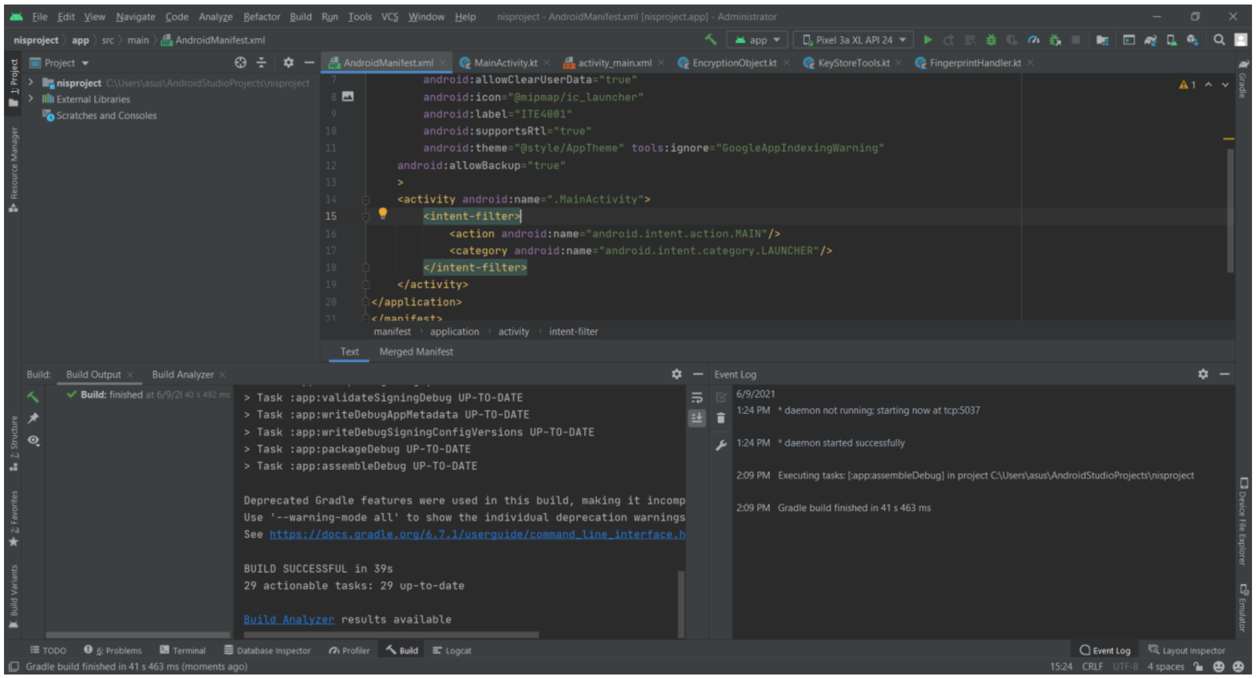
Finger 2

Finger 3

+ Add fingerprint

In addition to unlocking your phone, you can also use your fingerprint to authorize purchases and app access. Learn more

Step-5: finally, we had added the finger print to the emulator via cmd using the platform tools and now let us go further and build our project.



```
            android:allowClearUserData="true"
            android:icon="@mipmap/ic_launcher"
            android:label="ITE4001"
            android:supportsRtl="true"
            android:theme="@style/AppTheme" tools:ignore="GoogleAppIndexingWarning"
        android:allowBackup="true"
        >
        <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>
```

```
> Task :app:validateSigningDebug UP-TO-DATE
> Task :app:writeDebugAppMetadata UP-TO-DATE
> Task :app:writeDebugSigningConfigVersions UP-TO-DATE
> Task :app:packageDebug UP-TO-DATE
> Task :app:assembleDebug UP-TO-DATE

Deprecated Gradle features were used in this build, making it incomp
Use '--warning-mode all' to show the individual deprecation warnings
See https://docs.gradle.org/6.7.1/userguide/command_line_interface.h

BUILD SUCCESSFUL in 39s
29 actionable tasks: 29 up-to-date

Build Analyzer results available
```

```
6/9/2021
1:24 PM   * daemon not running; starting now at tcp:5037

1:24 PM   * daemon started successfully

2:09 PM   Executing tasks: [:app:assembleDebug] in project C:\Users\asus\AndroidStudioProjects\nisproject

2:09 PM   Gradle build finished in 41 s 463 ms
```

```
Gradle build finished in 41 s 463 ms (moments ago)
```

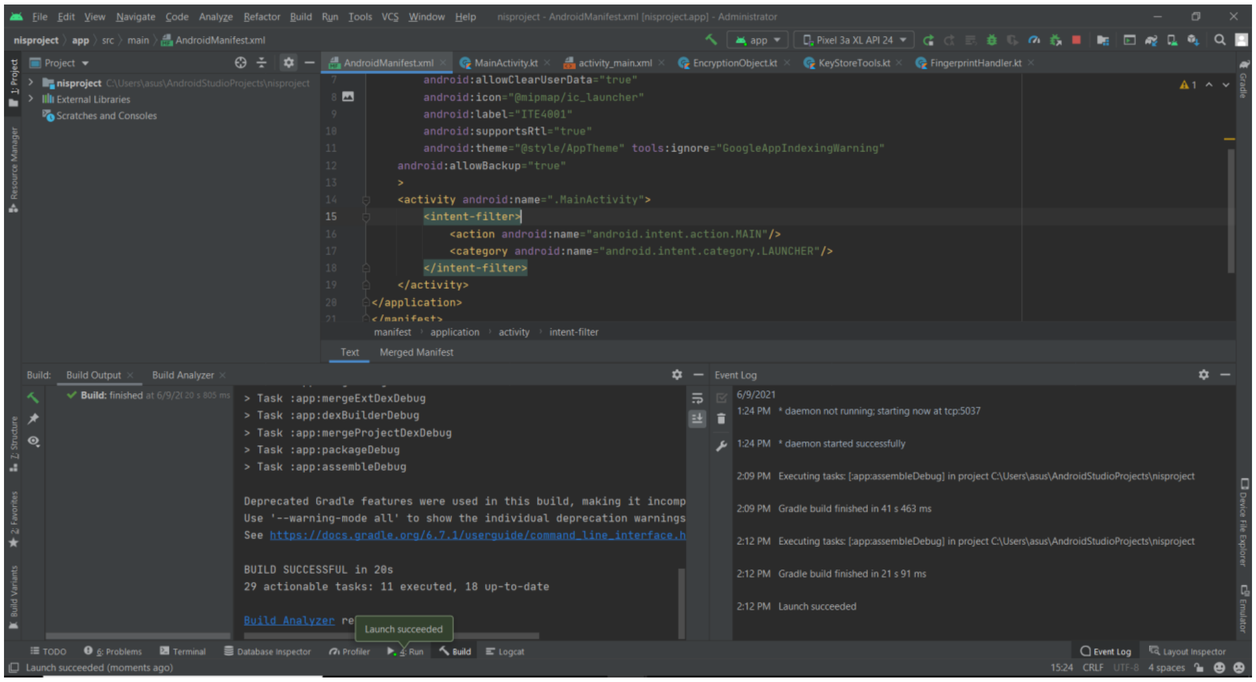
The build was successful and the application was ready to deploy now we can go and run to see the application running

When I want to encrypt, I will use my fingerprint first then the output will be given as per the fingerprint accepted or not

Step-6 we now run the application and get the output in the emulator

As and then the application has been installed we will be able to see the launch succeeded message in the bottom of the screen

And the emulator device will come online and now we can use the application to encrypt and decrypt the message



```
        android:allowClearUserData="true"
        android:icon="@mipmap/ic_launcher"
        android:label="ITE4001"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" tools:ignore="GoogleAppIndexingWarning"
    android:allowBackup="true"
    >
    <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>
```

```
Build: finished at 6/9/2( 20 s 805 ms
> Task :app:mergeExtDexDebug
> Task :app:dexBuilderDebug
> Task :app:mergeProjectDexDebug
> Task :app:packageDebug
> Task :app:assembleDebug

Deprecated Gradle features were used in this build, making it incomp
Use '--warning-mode all' to show the individual deprecation warnings
See https://docs.gradle.org/6.7.1/userguide/command_line_interface.h

BUILD SUCCESSFUL in 20s
29 actionable tasks: 11 executed, 18 up-to-date

Build Analyzer re
```

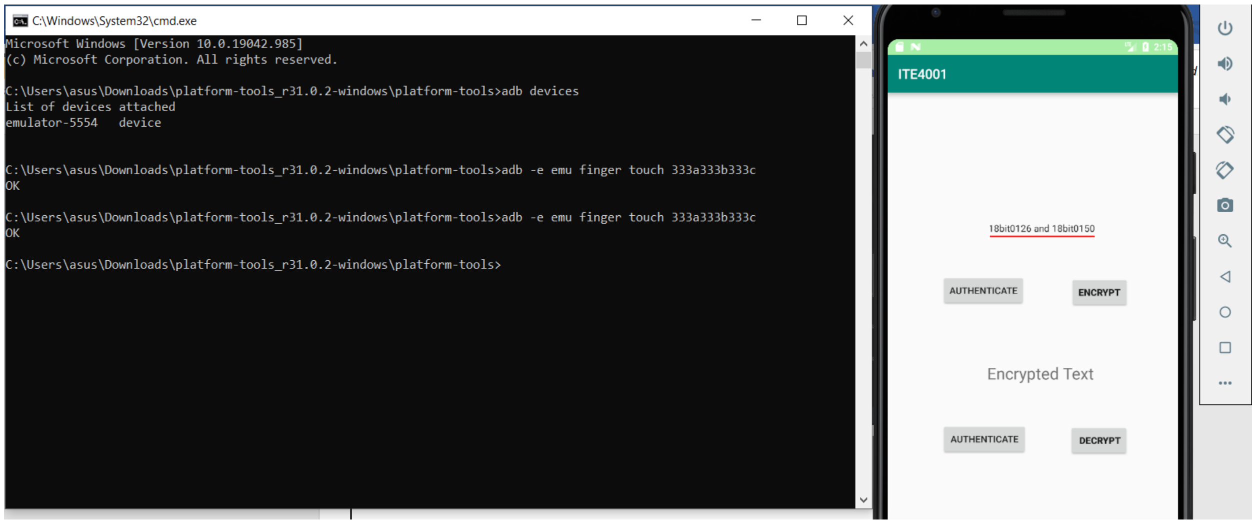
```
6/9/2021
1:24 PM  * daemon not running; starting now at tcp:5037

1:24 PM  * daemon started successfully

2:09 PM  Executing tasks: [:app:assembleDebug] in project C:\Users\asus\AndroidStudioProjects\nisproject

2:09 PM  Gradle build finished in 41 s 463 ms

2:12 PM  Executing tasks: [:app:assembleDebug] in project C:\Users\asus\AndroidStudioProjects\nisproject

2:12 PM  Gradle build finished in 21 s 91 ms

2:12 PM  Launch succeeded
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
nisproject - AndroidManifest.xml [nisproject.app] - Administrator
nisproject › app › src › main › AndroidManifest.xml
app
Pixel 3a XL API 24
AndroidManifest.xml
MainActivity.kt
activity_main.xml
EncryptionObject.kt
KeyStoreTools.kt
FingerprintHandler.kt
Project
nisproject C:\Users\asus\AndroidStudioProjects\nisproject
External Libraries
Scratches and Consoles
manifest › application › activity › intent-filter
Text
Merged Manifest
Build: Build Output Build Analyzer
Event Log
Launch succeeded
TODO 6: Problems Terminal Database Inspector Profiler 4: Run Build Logcat
Event Log Layout Inspector
Launch succeeded (moments ago)
15:24 CRLF UTF-8 4 spaces

Step-7: encrypting the message

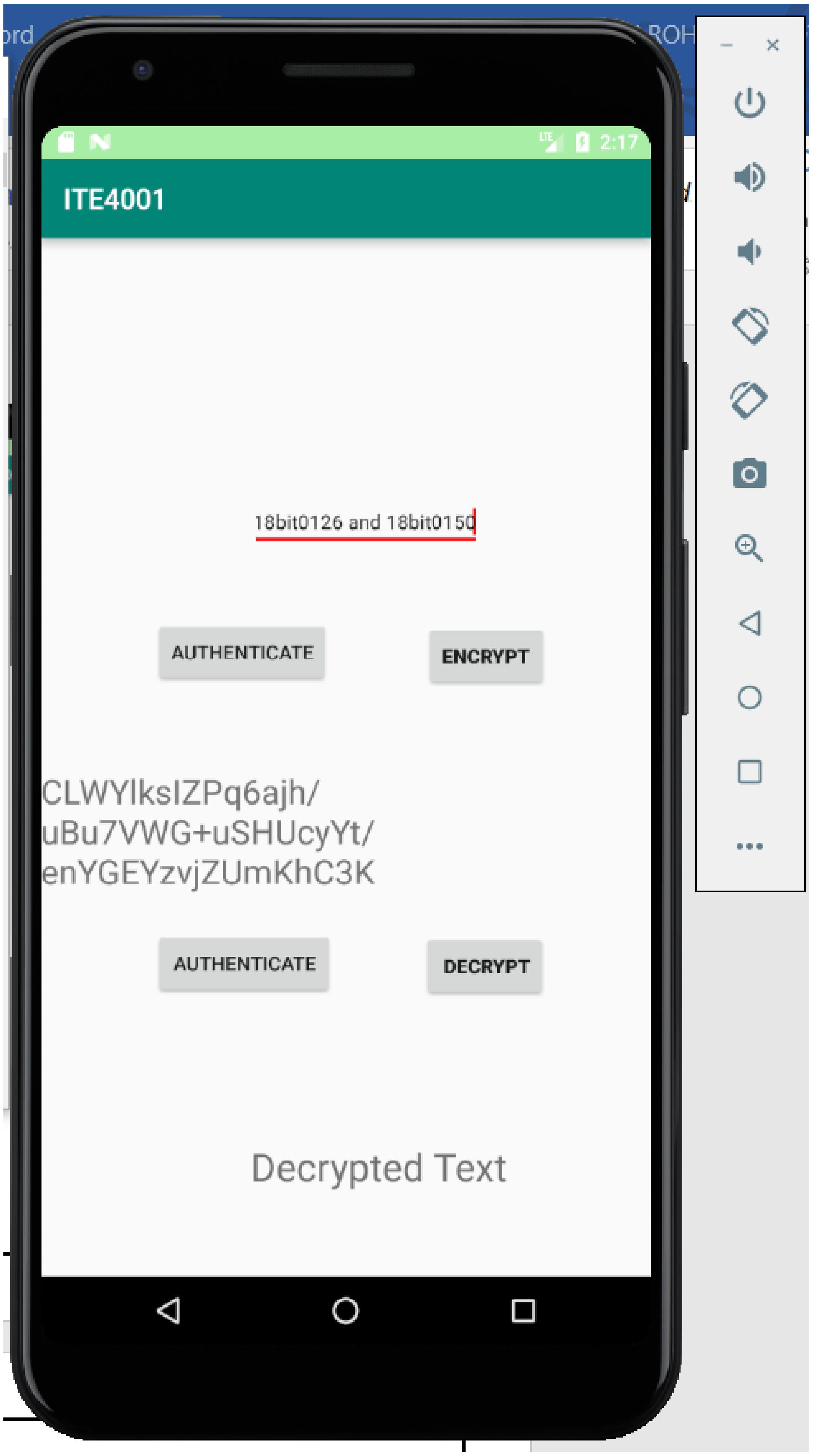
When I click authenticate the app will wait for fingerprint, we use same command as the fingerprint addition

adb -e emu finger touch 333a333b333c



```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb devices
List of devices attached
emulator-5554   device


C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>
```

Now click the encrypt button to see the encrypted text

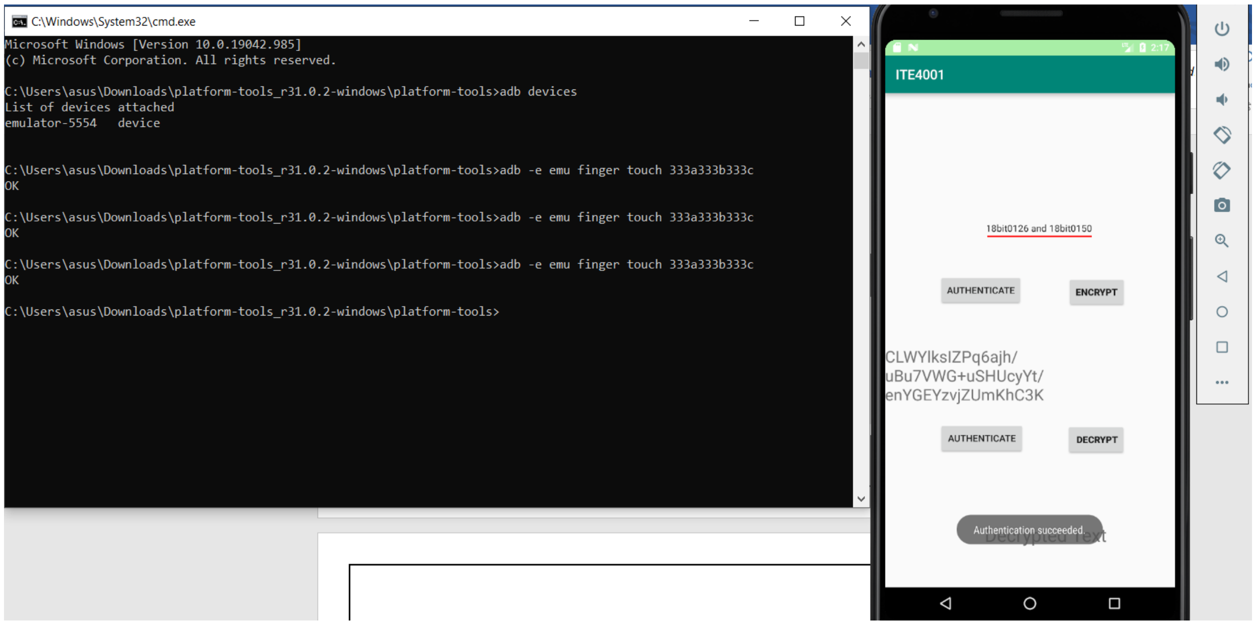


18bit0126 and 18bit0150

CLWYlksIZPq6ajh/uBu7VWG+uSHUcyYt/enYGEYzvjZUmKhC3K

Decrypted Text

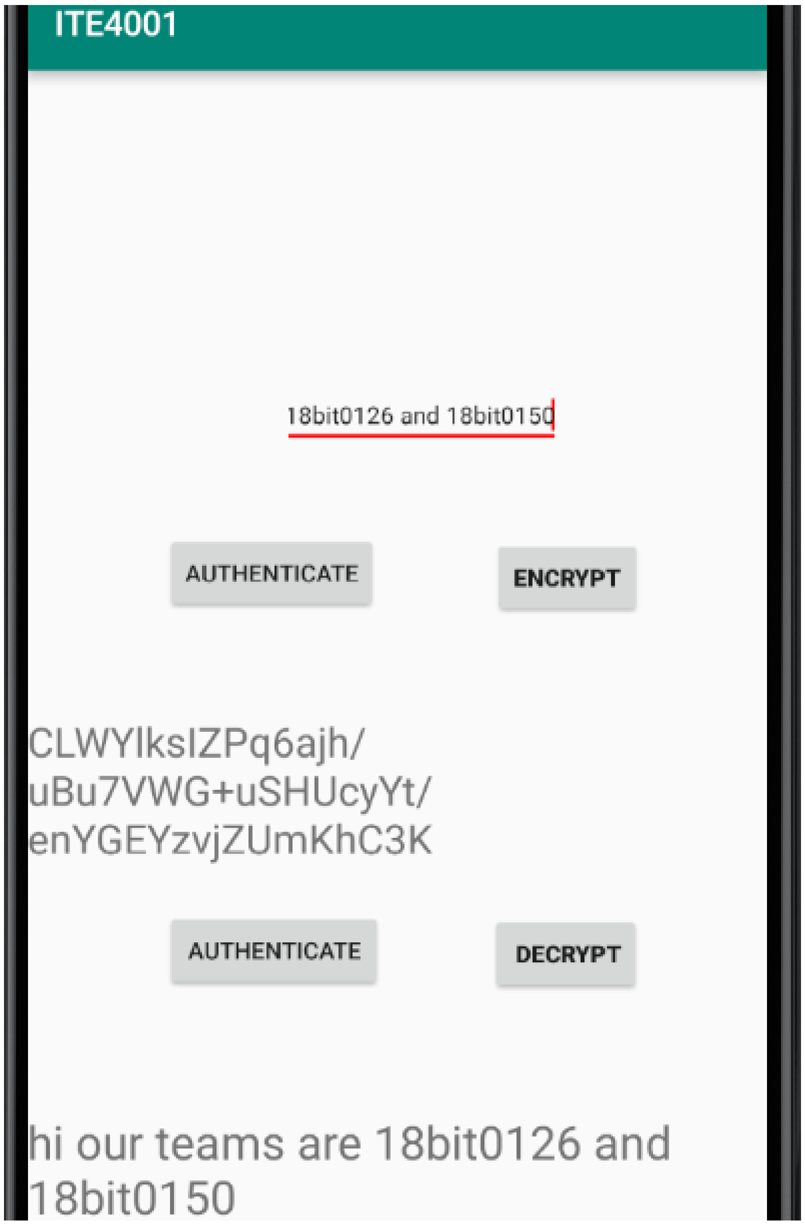
Now let’s try to decrypt the message now we have to give the fingerprint again



```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb devices
List of devices attached
emulator-5554   device


C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>adb -e emu finger touch 333a333b333c
OK

C:\Users\asus\Downloads\platform-tools_r31.0.2-windows\platform-tools>
```

Then you can get the authentication succeeded the now click the decrypt button to get the original message

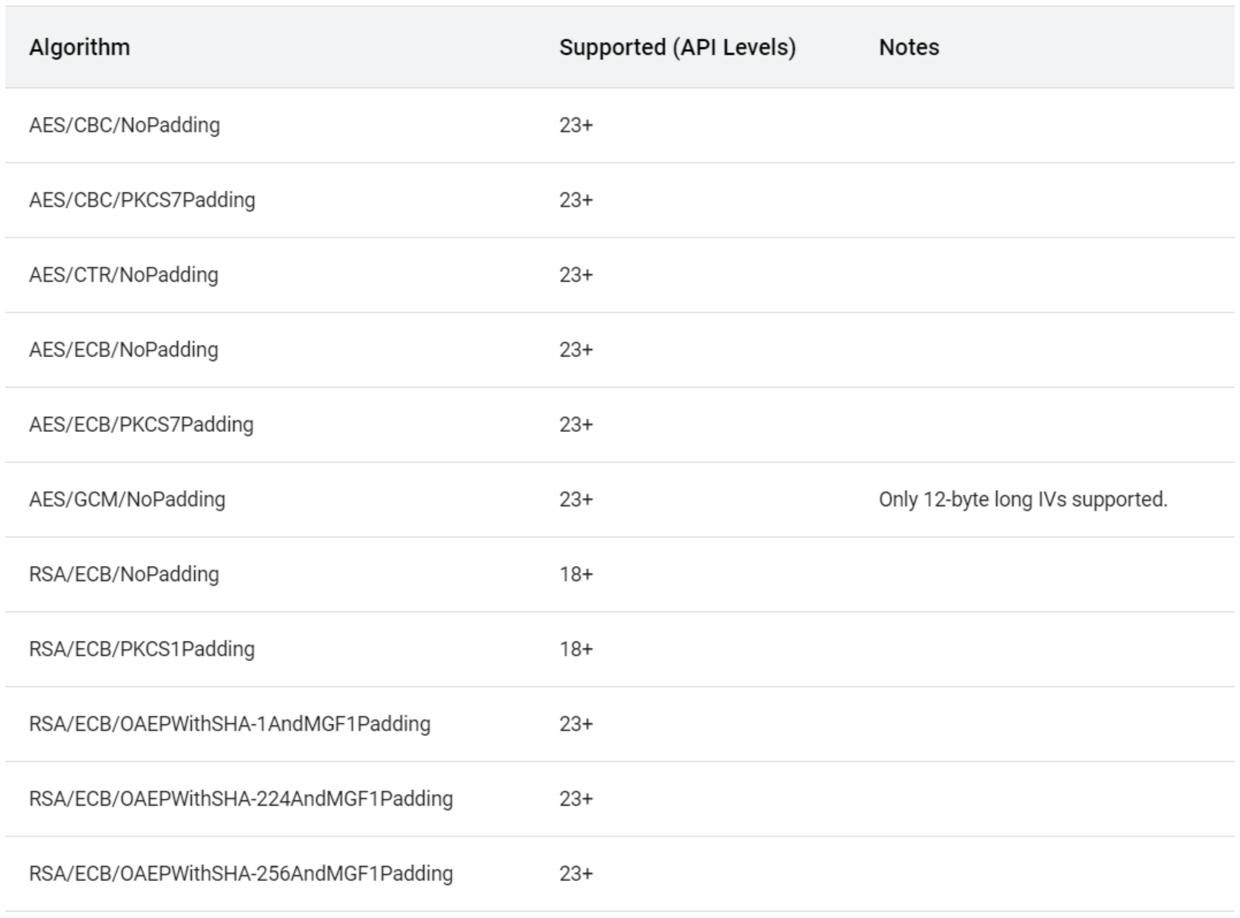
Here we are using the crypto object that will be shared between the encryption and decryption process where and which so that the data can be shared between both the process.



CLWYIksIZPq6ajh/
uBu7VWG+uSHUcyYt/
enYGEYzvjZUmKhC3K

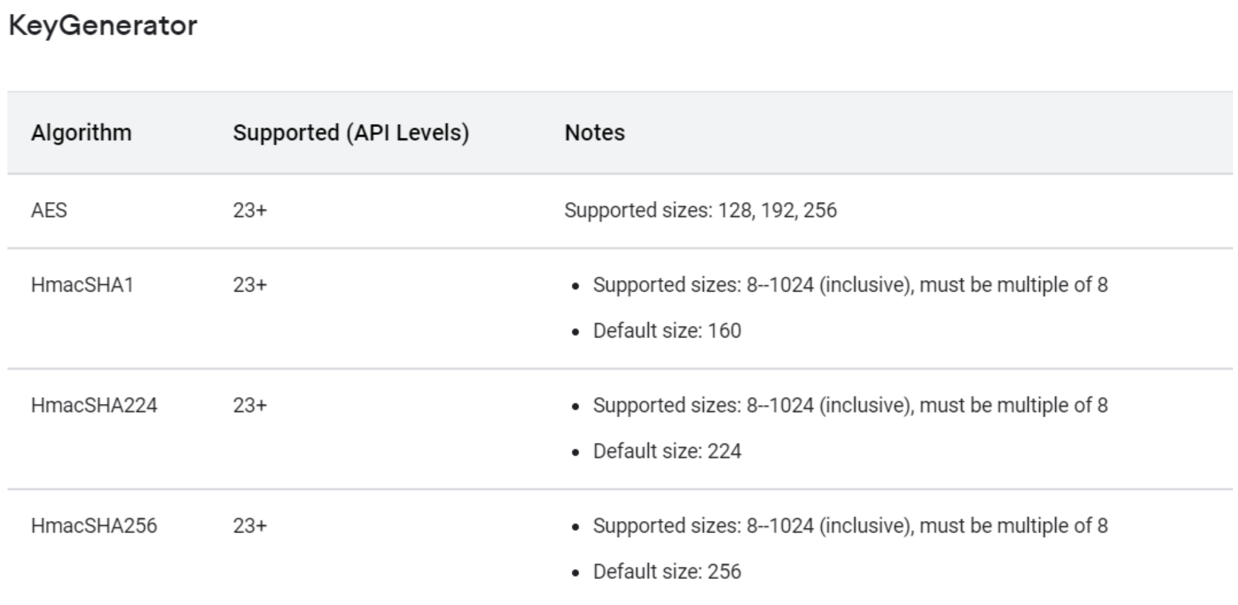
hi our teams are 18bit0126 and
18bit0150

## HOW THE KEY STORE WILL ENCRYPT THE MESSAGE?



| Algorithm | Supported (API Levels) | Notes |
|---|---|---|
| AES/CBC/NoPadding | 23+ | |
| AES/CBC/PKCS7Padding | 23+ | |
| AES/CTR/NoPadding | 23+ | |
| AES/ECB/NoPadding | 23+ | |
| AES/ECB/PKCS7Padding | 23+ | |
| AES/GCM/NoPadding | 23+ | Only 12-byte long IVs supported. |
| RSA/ECB/NoPadding | 18+ | |
| RSA/ECB/PKCS1Padding | 18+ | |
| RSA/ECB/OAEPWithSHA-1AndMGF1Padding | 23+ | |
| RSA/ECB/OAEPWithSHA-224AndMGF1Padding | 23+ | |
| RSA/ECB/OAEPWithSHA-256AndMGF1Padding | 23+ | |

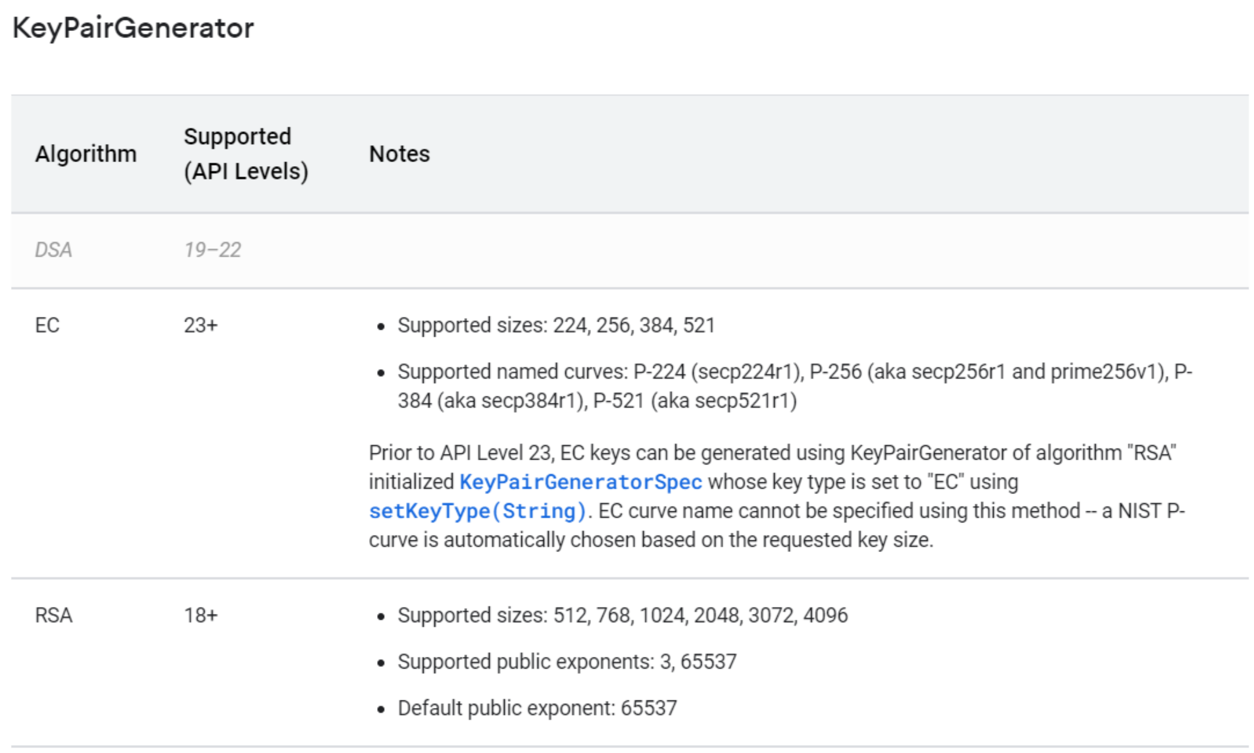
The above are the supported ciphers that we can make use of to encrypt and decrypt



### KeyGenerator

| Algorithm | Supported (API Levels) | Notes |
|---|---|---|
| AES | 23+ | Supported sizes: 128, 192, 256 |
| HmacSHA1 | 23+ | • Supported sizes: 8--1024 (inclusive), must be multiple of 8<br>• Default size: 160 |
| HmacSHA224 | 23+ | • Supported sizes: 8--1024 (inclusive), must be multiple of 8<br>• Default size: 224 |
| HmacSHA256 | 23+ | • Supported sizes: 8--1024 (inclusive), must be multiple of 8<br>• Default size: 256 |

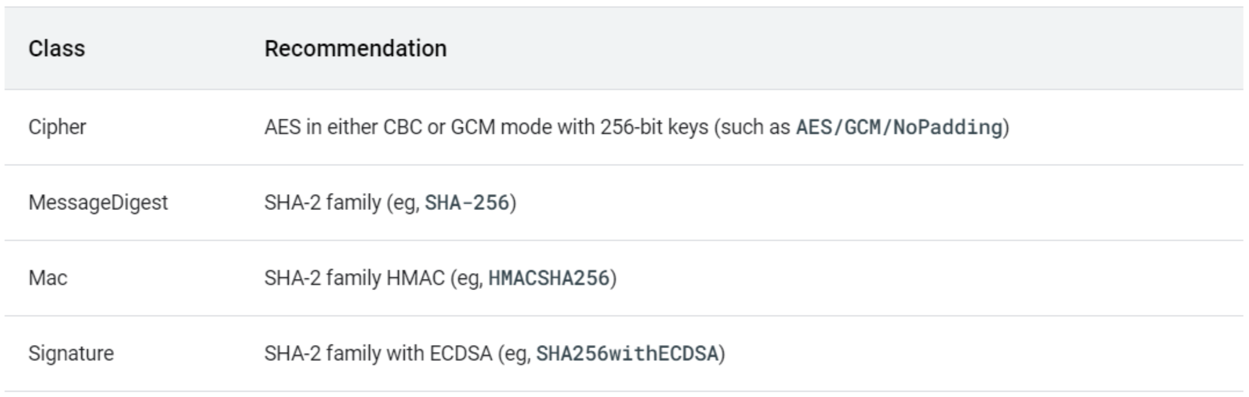
Key pair generator for asymmetric



### KeyPairGenerator

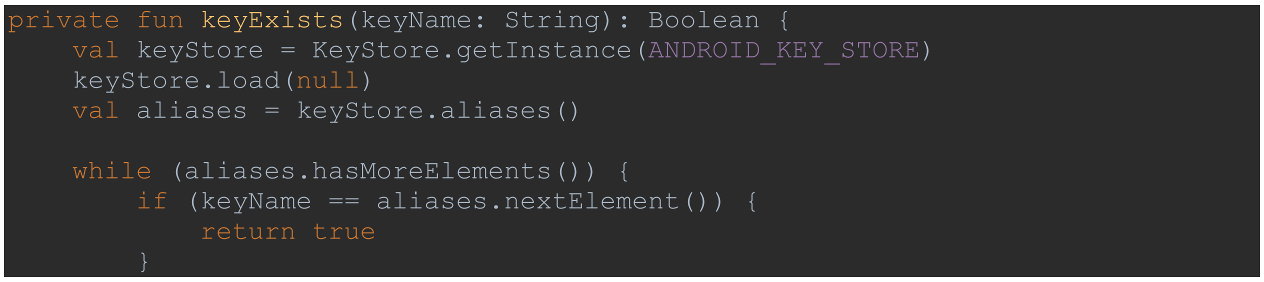
| Algorithm | Supported (API Levels) | Notes |
|---|---|---|
| *DSA* | *19–22* | |
| EC | 23+ | • Supported sizes: 224, 256, 384, 521<br>• Supported named curves: P-224 (secp224r1), P-256 (aka secp256r1 and prime256v1), P-384 (aka secp384r1), P-521 (aka secp521r1)<br>Prior to API Level 23, EC keys can be generated using KeyPairGenerator of algorithm "RSA" initialized `KeyPairGeneratorSpec` whose key type is set to "EC" using `setKeyType(String)`. EC curve name cannot be specified using this method -- a NIST P-curve is automatically chosen based on the requested key size. |
| RSA | 18+ | • Supported sizes: 512, 768, 1024, 2048, 3072, 4096<br>• Supported public exponents: 3, 65537<br>• Default public exponent: 65537 |

Algorithms we can select



| Class | Recommendation |
|---|---|
| Cipher | AES in either CBC or GCM mode with 256-bit keys (such as `AES/GCM/NoPadding`) |
| MessageDigest | SHA-2 family (eg, `SHA-256`) |
| Mac | SHA-2 family HMAC (eg, `HMACSHA256`) |
| Signature | SHA-2 family with ECDSA (eg, `SHA256withECDSA`) |

Initially we have to go for the checking any keys are there function to identify the key



```
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
}
```

now we have the function to check the key exists or not now we try creating key store if not present

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

now we give a function to get the key from the keystore

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

now we see how the key is prepared

```
        private fun generateAesKey(keyName: String) {
            try {
                /*HERE THE KEY GENERATOR FUNCTION WHERE WE WILL BE ABLE TO
PROVIDE THE ALGORITHM */
                val keyGenerator = KeyGenerator.getInstance(
                    KeyProperties.KEY_ALGORITHM_AES,
                    ANDROID_KEY_STORE

                )
                /*HERE WE ARE BUILDING THE KEY AND WE HAVE TO SPECIFY OUR
USAGE */
                val builder = KeyGenParameterSpec.Builder(
                    keyName,
                    KeyProperties.PURPOSE_ENCRYPT or
KeyProperties.PURPOSE_DECRYPT
                )
                /*HERE WE SPECIFY THE MODE OF AES AND KEY-SIZE
                * LIKE WE SPECIFY THE BLOCK MODE HERE WE USE CBC
                * ENCRYPTION PADDING HERE WE USE THE PKCS7 PADDING */
                builder.setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                    .setKeySize(256)

.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
/*HERE WE ARE GOING TO SET THE USER AUTHENTICATION AS TRUE SO THAT
* THE KEY WHEN IT WAS CREATED IT WILL BE ENCRYPTED USING OUR FINGERPRINT
* HERE THE FINGERPRINT WILL BE STORED AS 128-BIT AND THE KEY IS OF 256-BIT
* WHICH MAKES IT TWO BLOCK INPUT WHEN WE GO FOR THE ENCRYPTION USING THE
* FINGERPRINT */
                builder.setUserAuthenticationRequired(true)
                keyGenerator.init(builder.build())
                val keys:SecretKey=keyGenerator.generateKey()
                Log.d("secret_key", keys.toString())

/* HERE WE ARE ADDING THE EXCEPTIONS WHEN EVER ANYTHING GOES WRONG IN THE
ABOVE
* THE ERROR MESSAGES WERE LOGGED*/
            } 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)
            }}
```
how do we be able to encrypt the message

```
    /* WE ARE PREPARING THE CIPHER OBJECT FOR ENCRYPTION
    * CIPHER MODE SPECIFIES WHAT THE CIPHER DOES WITH THE KEY
    * HERE FOR THE CIPHER OBJECT WE GIVE THE MODE AND THE KEY AS THE
PARAMETERS*/
    fun cipherForEncryption(): Cipher {
        try {
            cipherEnc.init(Cipher.ENCRYPT_MODE, key)
            Log.i("Info",key.toString())
/*HERE WE HAD SUCCESSFULLY CREATED THE CIPHER FOR ENCRYPTION AND NOW WE
RETURN
* THE CIPHER OBJECT */
            return cipherEnc
        /* IF THERE IS ANY DISTURBANCE ABOVE WE WILL THE LOG THE EXCEPTION*/
        } catch (e: KeyPermanentlyInvalidatedException) {
            throw RuntimeException("Key Permanently Invalidated", e)
        } catch (e: Exception) {
            throw RuntimeException("Failed to init Cipher", e)
        }

    }
/*HERE TO ENCRYPT THE MESSAGE WE WILL RUN THE DOFINAL AND THEN WE COMBINE
* WITH THE IV FROM CIPHER AND THEN WE RETURN THE CIPHERTEXT
* AND IV WITH BASE64 TO SECURE AND NOW WE STORE IT IN THE SHARED
* PREFERENCE WHICH HELPS US TO PASS DATA FROM ONE ACTIVITY
* TO THE OTHER ACTIVITY */

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
    )
}
```

Now how we decrypt the message

```
/*HERE IS THE PLACE WE USE THE DECRYPTION CIPHER OBJECT THAT HAVE BEEN
CREATED
* NOW HERE WE NEED THE IV FROM THE ENCRYPTED CIPHER WE CAN GET THE CIPHER
TEXT AND
*  IV WITH SEPARATOR NOW WE USE THIS IV TO CONVERT THE BASE64 AND THEN
* NOW WE HAVE THE ORIGINAL CIPHER TEXT THROUGH WHICH WE CAN DECRYPT AND
* GET THE ORIGINAL TEXT  */
fun cipherForDecryption(IV: String): Cipher {
    try {
        cipherDec.init(
            Cipher.DECRYPT_MODE, key, IvParameterSpec(
                Base64.decode(
                    IV.toByteArray(Charsets.UTF_8),
                    Base64.DEFAULT
                )
            )
        )
        return cipherDec
        /*ON SUCCESS WE WILL RETURN THE CIPHER-DEC
        * NOW HERE BELOW WE HAD GIVEN THE CATCH BLOCKS
        * TO CATCH ANY EXCEPTIONS  */
    } catch (e: KeyPermanentlyInvalidatedException) {
        throw RuntimeException("Key Permanently Invalidated", e)
    } catch (e: Exception) {
        throw RuntimeException("Failed to init Cipher", e)
    }

}
/* NOW WE ARE GOING TO DECRYPT THE CIPHER THAT WE HAVE GOT
* HERE WE DECRYPT THE BASE64 TO AN NORMAL TEXT*/
fun decrypt(cipher: Cipher, encrypted: String): String {

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

How the cipher objects are created and what it contains it contains the key with block mode of aes and type of padding

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

now we has seen how we are able to create the cipher object and how we can encrypt and decrypt the message

now we see how we use the authentication using fingerprint manager

```
/* HERE WE ARE CREATING THE FUNCTION THAT WAIT FOR THE FINGERPRINT LISTENING
* AND THE CALLBACKS AND THE ENCRYPTION CIPHER WILL BE PACKED INSIDE THE
* CRYPTO OBJECT FOR WHICH THE FINGERPRINT WILL BE THE KEY*/

fun startAuth(fingerprintManager: FingerprintManager, cryptoObject:
FingerprintManager.CryptoObject) {
    cancellationSignal = CancellationSignal()

    if (ActivityCompat.checkSelfPermission(
            context,
            Manifest.permission.USE_FINGERPRINT
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
    fingerprintManager.authenticate(cryptoObject, cancellationSignal, 0,
this, null)
}
```

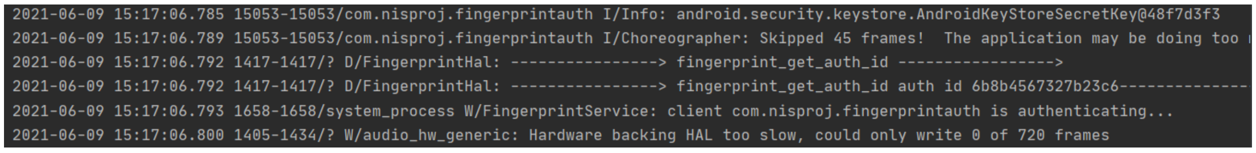
The above function is the one that waits for the user fingerprint to get scanned.

## Activitymain.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

    <EditText
        android:id="@+id/pinEditText"
        style="@style/Widget.AppCompat.EditText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="196dp"
        android:layout_marginBottom="628dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:textAlignment="center"
        android:textAllCaps="true"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.554"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.112"
        tools:hint="Enter Text To Encrypt" />
```

```
<Button
    android:id="@+id/listenerButtonEnc"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="280dp"
    android:layout_marginBottom="536dp"
    android:text="Authenticate"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.262"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.082" />

<Button
    android:id="@+id/encryptButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="356dp"
    android:layout_marginBottom="460dp"
    android:text="Encrypt"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.789"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.654" />

<TextView
    android:id="@+id/encryptedTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="455dp"
    android:layout_marginBottom="377dp"
    android:hint="Encrypted Text"
    android:maxLength="50"
    android:text="Encrypted Text"
    android:textSize="24sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.543"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.428" />

<TextView
    android:id="@+id/decryptedTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="711dp"
    android:layout_marginBottom="60dp"
    android:maxLength="50"
    android:text="Decrypted Text"
    android:textSize="27sp"
    app:layout_constraintBottom_toBottomOf="parent"
```

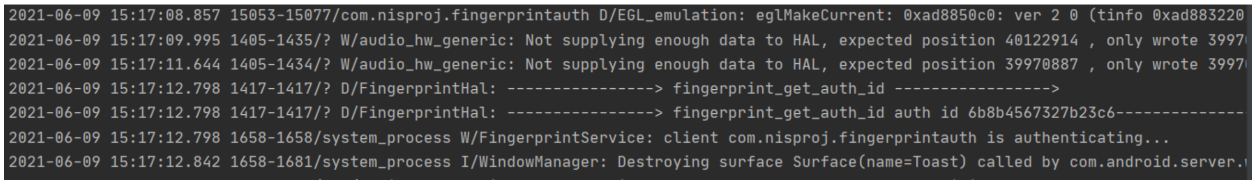
```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.595"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <Button
        android:id="@+id/listenerButton"
        android:layout_width="128dp"
        android:layout_height="48dp"
        android:layout_marginTop="504dp"
        android:layout_marginBottom="255dp"
        android:text="Authenticate"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.265"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.197" />

    <Button
        android:id="@+id/decryptButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" Decrypt"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.788"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.715" />

    <android.support.constraint.Guideline
            android:id="@+id/guideline"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            app:layout_constraintGuide_begin="20dp"
            app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```
Here now we had the build and run the code now lets just take a look through the logcat



```
2021-06-09 15:17:06.785 15053-15053/com.nisproj.fingerprintauth I/Info: android.security.keystore.AndroidKeyStoreSecretKey@48f7d3f3
2021-06-09 15:17:06.789 15053-15053/com.nisproj.fingerprintauth I/Choreographer: Skipped 45 frames!  The application may be doing too
2021-06-09 15:17:06.792 1417-1417/? D/FingerprintHal: ----------------> fingerprint_get_auth_id ----------------->
2021-06-09 15:17:06.792 1417-1417/? D/FingerprintHal: ----------------> fingerprint_get_auth_id auth id 6b8b4567327b23c6-------------
2021-06-09 15:17:06.793 1658-1658/system_process W/FingerprintService: client com.nisproj.fingerprintauth is authenticating...
2021-06-09 15:17:06.800 1405-1434/? W/audio_hw_generic: Hardware backing HAL too slow, could only write 0 of 720 frames
```

Here we can observe the logcat during the encryption where we have the secret key and the authid that was assigned to my finger print

During the decryption we get the same authid as I am the one going to decrypt it



```
2021-06-09 15:17:08.857 15053-15077/com.nisproj.fingerprintauth D/EGL_emulation: eglMakeCurrent: 0xad8850c0: ver 2 0 (tinfo 0xad883220
2021-06-09 15:17:09.995 1405-1435/? W/audio_hw_generic: Not supplying enough data to HAL, expected position 40122914 , only wrote 3997
2021-06-09 15:17:11.644 1405-1434/? W/audio_hw_generic: Not supplying enough data to HAL, expected position 39970887 , only wrote 3997
2021-06-09 15:17:12.798 1417-1417/? D/FingerprintHal: ----------------> fingerprint_get_auth_id ---------------->
2021-06-09 15:17:12.798 1417-1417/? D/FingerprintHal: ----------------> fingerprint_get_auth_id auth id 6b8b4567327b23c6-------------
2021-06-09 15:17:12.798 1658-1658/system_process W/FingerprintService: client com.nisproj.fingerprintauth is authenticating...
2021-06-09 15:17:12.842 1658-1681/system_process I/WindowManager: Destroying surface Surface(name=Toast) called by com.android.server.
```

In the above both images we can verify that the authid that was same which mean that iam the one who has encrypted and decrypted the message.

Mainactivity.kt

```
package com.nisproj.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.nisproj.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"
    }
    /*ADDING THE FINGERPRINT TO OUR CIPHER OBJECTS WE HERE WILL
    BE ABLE TO CREATE THE CRYPTO OBJECT*/
    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 pinEditText: EditText
    private val encryptionObject = EncryptionObject.newInstance()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        pinEditText = findViewById(R.id.pinEditText) as EditText
        keyguardManager = getSystemService(Context.KEYGUARD_SERVICE) as
KeyguardManager
        fingerprintManager = getSystemService(Context.FINGERPRINT_SERVICE) as
FingerprintManager
        /*SHARED PREFERENCE HAS BEEN USED INORDER TO
        * GET THE SECRET KEY */
        pref = this.getSharedPreferences(
            "com.nisproj.secure.pref",
            Context.MODE_PRIVATE
        )
        /*HERE NOW WE HAD THE SHARED PREFERENCES THAT STORE THE ENCRYPTED
OBJECT
        * THE ENCRYPTED MESSAGE AND THE IV
        * NOW DURING THE ENCRYPTION WE GIVE FINGERPRINT TO GET THE
        * ENCRYPTION OBJECT FROM CRYPTO OBJECT BY GIVING OUR FINGERPRINT
        * AND THEN WE WILL BE ABLE TO DECRYPT IT */
        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()
        }
    }
    /*CREATING THE CRYPTO OBJECT  */
    private fun createFingerprintHandlerEnc() {
        try {
            cryptoObjectEncrypt =
FingerprintManager.CryptoObject(encryptionObject.cipherForEncryption())
            val fingerprintHandler = FingerprintHandler(this)
            fingerprintHandler.startAuth(fingerprintManager,
cryptoObjectEncrypt)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    /* WE HERE HAVE THE DECRYPT OBJECT THEN WE START AUTHENTICATION CRYPTO
OBJECT*/
    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
        }
    }
}
```

## Fingerprinthandler.kt

```
package com.nisproj.fingerprintauth

import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.hardware.fingerprint.FingerprintManager
import android.os.CancellationSignal
import android.support.v4.app.ActivityCompat
import android.widget.Toast

class FingerprintHandler(private val context: Context) :
FingerprintManager.AuthenticationCallback() {

    lateinit var cancellationSignal: CancellationSignal
    /* HERE WE ARE CREATING THE FUNCTION THAT WAIT FOR THE FINGERPRINT
LISTENING
    * AND THE CALLBACKS AND THE ENCRYPTION CIPHER WILL BE PACKED INSIDE THE
    * CRYPTO OBJECT FOR WHICH THE FINGERPRINT WILL BE THE KEY*/

    fun startAuth(fingerprintManager: FingerprintManager, cryptoObject:
FingerprintManager.CryptoObject) {
        cancellationSignal = CancellationSignal()

        if (ActivityCompat.checkSelfPermission(
                context,
                Manifest.permission.USE_FINGERPRINT
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            return
        }
        fingerprintManager.authenticate(cryptoObject, cancellationSignal, 0,
this, null)
    }
    override fun onAuthenticationError(errorCode: Int, errString:
CharSequence?) {
        informationMessage(context.getString(R.string.auth_error_message))
    }
    override fun onAuthenticationSucceeded(result:
FingerprintManager.AuthenticationResult?) {
        informationMessage(context.getString(R.string.aith_success_message))
    }
    override fun onAuthenticationHelp(helpCode: Int, helpString:
CharSequence?) {
        informationMessage(context.getString(R.string.auth_help_message))
    }
    override fun onAuthenticationFailed() {
        informationMessage(context.getString(R.string.auth_failed_message))
    }
    private fun informationMessage(message: String) {
        Toast.makeText(context, message, Toast.LENGTH_LONG).show()
    }}
```

## keystoretools.kt

```
package com.nisproj.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 {
        /* HERE WE ARE GOING TO GET THE KEYSTORE OBJECT THROUGH THIS
COMMAND*/
        private const val ANDROID_KEY_STORE = "AndroidKeyStore"
/* CHECKING FOR ANY PREVIOUS KEY ARE PRESENT IN THE KEYSTORE */
        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
        }
/*FUNCTION TO GET THE KEY FROM THE KEYSTORE */

        fun getKeyFromKeyStore(keyname: String): SecretKey {
            val keyStore = createKeyStore()
            if (!keyExists(keyname)) {
                generateAesKey(keyname)
            }
            return keyStore.getKey(keyname, null) as SecretKey
        }
        /*FUNCTION TO CREATE THE KEYSTORE IF IT IS NOT FOUND INITIALLY */

        private fun createKeyStore(): KeyStore {
            val keyStore = KeyStore.getInstance(ANDROID_KEY_STORE)
            keyStore.load(null)
            return keyStore
        }
        /*HERE WE WILL BE GOING TO GENERATE THE AES KEY */

        private fun generateAesKey(keyName: String) {
            try {
                /*HERE THE KEY GENERATOR FUNCTION WHERE WE WILL BE ABLE TO
PROVIDE THE ALGORITHM */
                val keyGenerator = KeyGenerator.getInstance(
                    KeyProperties.KEY_ALGORITHM_AES,
                    ANDROID_KEY_STORE
                )
                /*HERE WE ARE BUILDING THE KEY AND WE HAVE TO SPECIFY OUR
USAGE */
                val builder = KeyGenParameterSpec.Builder(
                    keyName,
                    KeyProperties.PURPOSE_ENCRYPT or
KeyProperties.PURPOSE_DECRYPT
                )
                /*HERE WE SPECIFY THE MODE OF AES AND KEY-SIZE
                * LIKE WE SPECIFY THE BLOCK MODE HERE WE USE CBC
                * ENCRYPTION PADDING HERE WE USE THE PKCS7 PADDING */
                builder.setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                    .setKeySize(256)

.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
/*HERE WE ARE GOING TO SET THE USER AUTHENTICATION AS TRUE SO THAT
* THE KEY WHEN IT WAS CREATED IT WILL BE ENCRYPTED USING OUR FINGERPRINT
* HERE THE FINGERPRINT WILL BE STORED AS 128-BIT AND THE KEY IS OF 256-BIT
* WHICH MAKES IT TWO BLOCK INPUT WHEN WE GO FOR THE ENCRYPTION USING THE
* FINGERPRINT */
                builder.setUserAuthenticationRequired(true)
                keyGenerator.init(builder.build())
                val keys:SecretKey=keyGenerator.generateKey()
                Log.d("secret_key", keys.toString())

/* HERE WE ARE ADDING THE EXCEPTIONS WHEN EVER ANYTHING GOES WRONG IN THE
ABOVE
* THE ERROR MESSAGES WERE LOGGED*/
            } 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)
            }

        }
    }

}
```

**Encryptionobject.kt**

```
package com.nisproj.fingerprintauth.authentication

import android.security.keystore.KeyPermanentlyInvalidatedException
import android.security.keystore.KeyProperties
import android.util.Base64
import android.util.Log
import com.nisproj.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()
        }
    }

    /* GETTING THE KEY FROM THE KEYSTORE
    * HERE WE STORE THE KEY IN VAR KEY AS SECRET KEY*/
    private val key: SecretKey =
        KeyStoreTools.getKeyFromKeyStore(KEY_NAME)
    /*HERE NOW WE WILL BE CREATING TWO CIPHER OBJECTS ONE FOR ENCRYPTION AND
    * THE OTHER FOR DECRYPTION
    * CIPHER FORMAT WILL BE AS THE CONTAINS THE BLOCK MODE
    * PADDING PKCS7 AND THE ALGORITHM AES
    * THERE ARE NOW TWO CIPHER OBJECTS WHERE WE USE ONE FOR ENCRYPTION
    * AND OTHER FOR DECRYPTION */

    val cipherEnc: Cipher = createCipher()
    val cipherDec: Cipher = createCipher()
    /*HERE TO ENCRYPT THE MESSAGE WE WILL RUN THE DOFINAL AND THEN WE COMBINE
    * WITH THE IV FROM CIPHER AND THEN WE RETURN THE CIPHERTEXT
    * AND IV WITH BASE64 TO SECURE AND NOW WE STORE IT IN THE SHARED
    * PREFERENCE WHICH HELPS US TO PASS DATA FROM ONE ACTIVITY
    * TO THE OTHER ACTIVITY */

    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
        )

    }
    /* NOW WE ARE GOING TO DECRYPT THE CIPHER THAT WE HAVE GOT
    * HERE WE DECRYPT THE BASE64 TO AN NORMAL TEXT*/
    fun decrypt(cipher: Cipher, encrypted: String): String {

        return cipher.doFinal(
            Base64.decode(
                encrypted,
                Base64.DEFAULT )).toString(Charsets.UTF_8)}
    /* WE ARE PREPARING THE CIPHER OBJECT FOR ENCRYPTION
    * CIPHER MODE SPECIFIES WHAT THE CIPHER DOES WITH THE KEY
    * HERE FOR THE CIPHER OBJECT WE GIVE THE MODE AND THE KEY AS THE
PARAMETERS*/
    fun cipherForEncryption(): Cipher {
        try {
            cipherEnc.init(Cipher.ENCRYPT_MODE, key)
            Log.i("Info",key.toString())
/*HERE WE HAD SUCCESSFULLY CREATED THE CIPHER FOR ENCRYPTION AND NOW WE
RETURN
* THE CIPHER OBJECT */
```

```
            return cipherEnc
        /* IF THERE IS ANY DISTURBANCE ABOVE WE WILL THE LOG THE EXCEPTION*/
        } catch (e: KeyPermanentlyInvalidatedException) {
            throw RuntimeException("Key Permanently Invalidated", e)
        } catch (e: Exception) {
            throw RuntimeException("Failed to init Cipher", e)
        }}
    /*HERE IS THE PLACE WE USE THE DECRYPTION CIPHER OBJECT THAT HAVE BEEN
CREATED
    * NOW HERE WE NEED THE IV FROM THE ENCRYPTED CIPHER WE CAN GET THE CIPHER
TEXT AND
    *  IV WITH SEPARATOR NOW WE USE THIS IV TO CONVERT THE BASE64 AND THEN
    * NOW WE HAVE THE ORIGINAL CIPHER TEXT THROUGH WHICH WE CAN DECRYPT AND
    * GET THE ORIGINAL TEXT  */
    fun cipherForDecryption(IV: String): Cipher {
        try {
            cipherDec.init(
                Cipher.DECRYPT_MODE, key, IvParameterSpec(
                    Base64.decode(
                        IV.toByteArray(Charsets.UTF_8),
                        Base64.DEFAULT
                    ) ))
            return cipherDec
            /*ON SUCCESS WE WILL RETURN THE CIPHER-DEC
            * NOW HERE BELOW WE HAD GIVEN THE CATCH BLOCKS
            * TO CATCH ANY EXCEPTIONS  */
        } 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
        )}}
```

## 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 it's 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. So, by this we conclude that fingerprint encryption is more unique and more secure and faster.

## REFERENCES

I. https://stackoverflow.com/questions/35335892/android-mfingerprintscanner-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=xqWFUv3Kj&sig=5LGdObW8OggftCU2AGTMHnou3fg#v=onepage&q=bio metrics%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-work670934/

IX. https://www.sciencedirect.com/science/article/abs/pii/S003132030500144 5

X. https://books.google.co.in/books?hl=en&lr=&id=1Wpx25D8qOwC&oi=fnd&pg=PR11&dq=fingerprint+recognition&ots=9yR_0Rjtd_&sig=nC57AFfQSmqVZhnaXWQxrBoehM&redir_esc=y#v=onepage&q=fingerprint%20recognition &f=false

XI. https://en.wikipedia.org/wiki/Pattern_recognition