MIT World Peace University

Wireless Devices and Mobile Security
Third Year B. Tech, Semester 5

Encryption and Decryption of Files and Text in an Android App

---

Lab Assignment 5

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 10

November 26, 2023

Contents

1 Aim

Write an android program to encrypt and decrypt text file. Use bouncy castle library API or java cryptography API.

2 Objectives

1. To understand the working of encryption and decryption of files and text in an android app.
2. To understand the working of bouncy castle library API or java cryptography API.
3. To understand the working of android app development.
4. To understand the working of android studio.
5. To understand the working of android emulator.

3 Theory

3.1 Android Studio

Figure 1: Android Studio Logo

1. **Overview:** Android Studio is the official integrated development environment (IDE) for Android app development. It is based on IntelliJ IDEA and provides a comprehensive set of tools for designing, building, testing, and debugging Android applications.
2. **Features:**
   - Android Studio includes a visual layout editor for designing user interfaces.
   - It supports multiple languages, including Java and Kotlin.
   - The built-in emulator allows developers to test their apps on various Android devices.

- Integration with version control systems like Git simplifies collaborative development.

3. **Advantages:**
   - Rich set of templates for common Android app components.
   - Seamless integration with Google services and libraries.
   - Robust debugging tools for identifying and fixing issues.

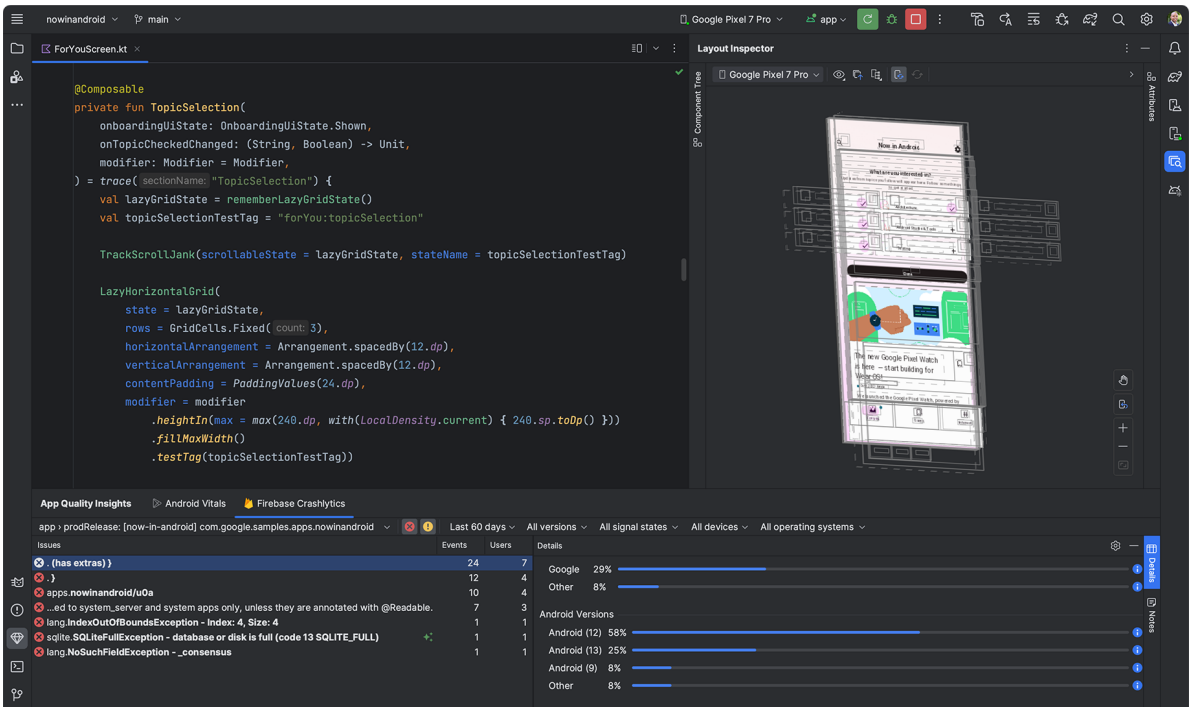

Figure 2: Android Studio Interface

3.2 Cryptography Libraries

3.2.1 Bouncy Castle Library API

1. **Introduction:** Bouncy Castle is a cryptography library that provides APIs for various cryptographic operations. It is written in Java and supports a wide range of algorithms and protocols.

2. **Key Features:**
   - Bouncy Castle supports both symmetric and asymmetric encryption algorithms.
   - It includes implementations for various cryptographic standards like PKCS, OpenPGP, and S/MIME.
   - The library provides a flexible and extensible architecture for cryptographic operations.

3. **Use Cases:**
   - Commonly used in Java applications for secure communication.
   - Integration with other security protocols and frameworks.

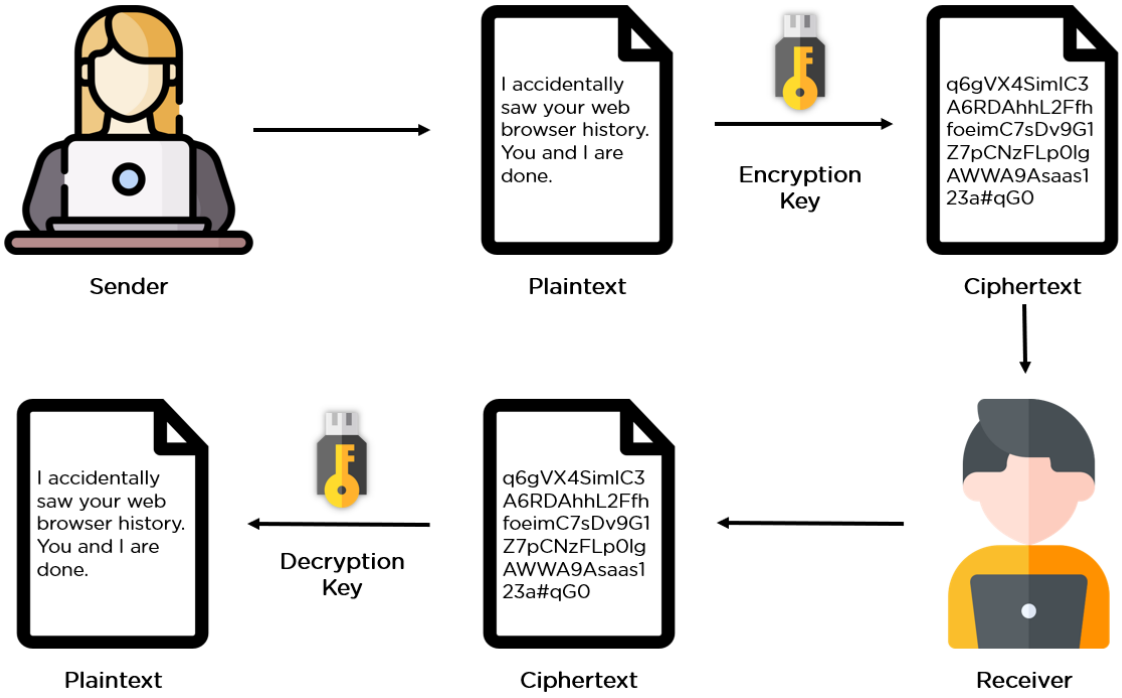

Figure 3: Encryption and Decryption

3.2.2 Java Cryptography API

1. **Overview:** The Java Cryptography Architecture (JCA) is a framework for handling cryptographic operations in Java applications. It includes the Java Cryptography Extension (JCE), which provides implementations for cryptographic algorithms.
2. **Key Components:**
   - **Message Digests and Digital Signatures:** JCA supports various algorithms for creating message digests and digital signatures.
   - **Key Management:** Provides classes for key generation, key storage, and key exchange.
   - **Secure Random Number Generation:** Ensures the generation of secure random numbers.
3. **Integration with Bouncy Castle:** Java Cryptography API can be integrated with the Bouncy Castle library for extended cryptographic functionalities.

4 Android App Permissions

4.1 Permissions Required

1. **Android Media Store API:**
   - The `READ_EXTERNAL_STORAGE` permission is required to read from external storage, including media files.
   - For writing media files, the `WRITE_EXTERNAL_STORAGE` permission is necessary.

- To capture photos or videos using the device's camera, the `CAMERA` permission is required.

2. **Usage in AndroidManifest.xml:**

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
```

3. **Best Practices:**

- Request these permissions at runtime on devices running Android 6.0 (API level 23) and higher.
- Handle permission responses gracefully to ensure a smooth user experience.

4.1.1 Android Manifest.xml

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.FileSealer"
        tools:targetApi="34">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Listing 1: Manifest of Filesealer

4.2 App - Filesealer

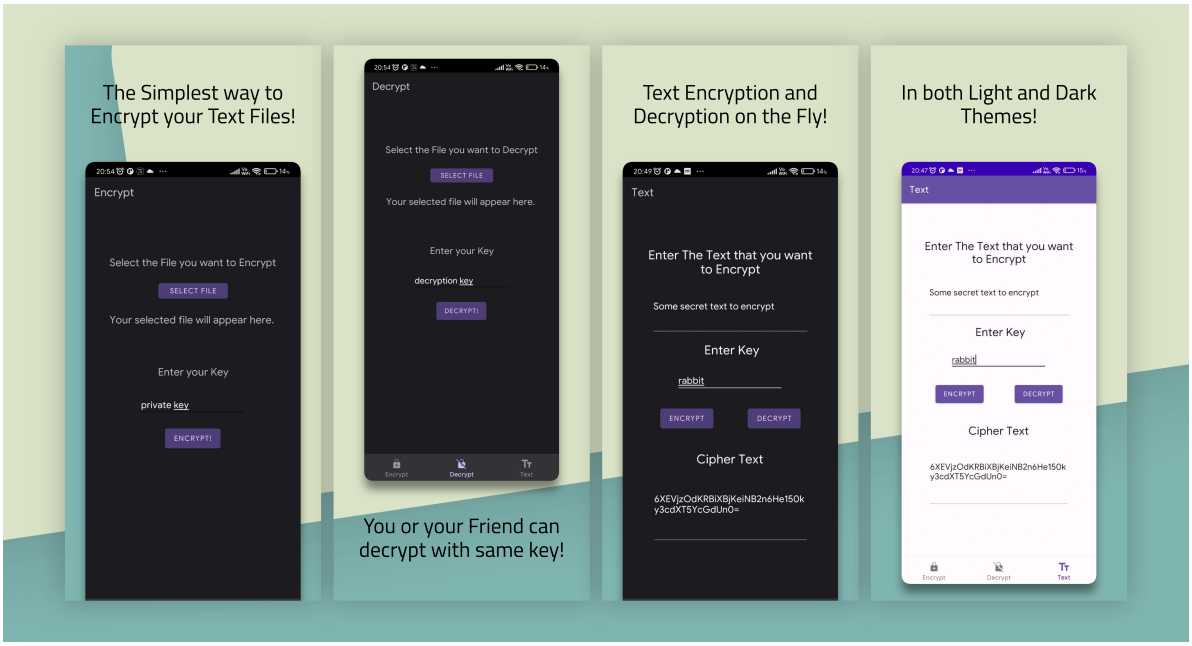

Figure 4: Filesealer App Presentation

Link of the app

https://play.google.com/store/apps/details?id=com.krishnaraj.filesealer

5 Platform

Operating System: Arch Linux x86 64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Python 3.10.1

6 Screenshots

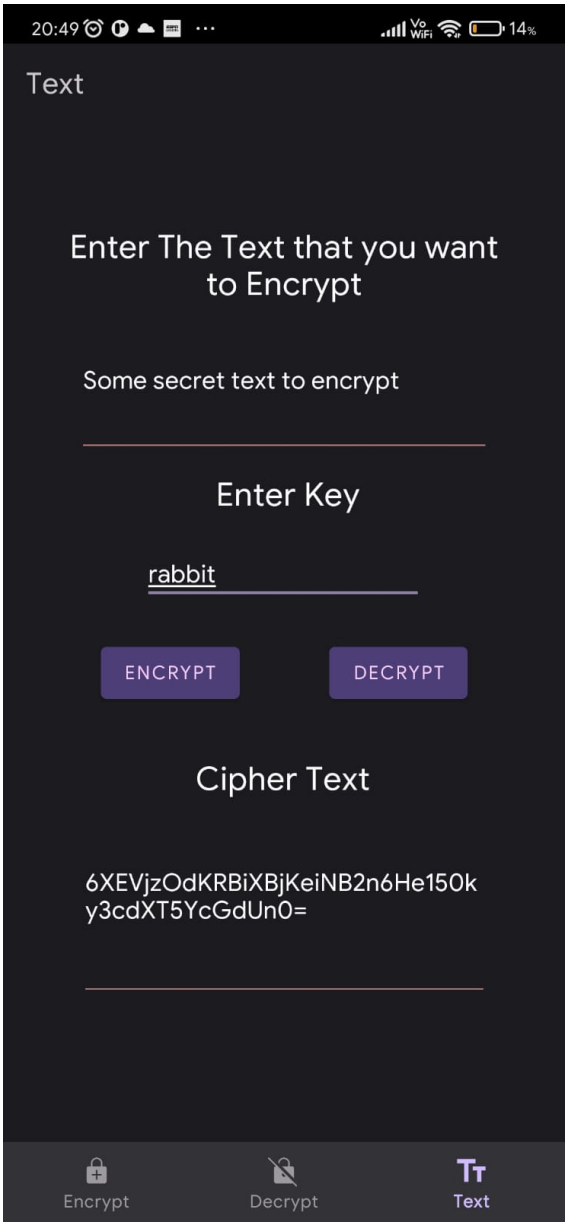

Figure 5: Text Encryption and Decryption

7 Code

```
package com.krishnaraj.filesealer;

import android.os.Bundle;

import com.google.android.material.bottomnavigation.BottomNavigationView;

import androidx.appcompat.app.AppCompatActivity;
import androidx.navigation.NavController;
```

```
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;

import com.krishnaraj.filesealer.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        BottomNavigationView navView = findViewById(R.id.nav_view);
        // Passing each menu ID as a set of Ids because each
        // menu should be considered as top level destinations.
        AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(
                R.id.navigation_home, R.id.navigation_dashboard, R.id.
    navigation_notifications)
                .build();
        NavController navController = Navigation.findNavController(this, R.id.
    nav_host_fragment_activity_main);
        NavigationUI.setupActionBarWithNavController(this, navController,
    appBarConfiguration);
        NavigationUI.setupWithNavController(binding.navView, navController);
    }

}
```

Listing 2: MainActivity.java

7.1 Encryption using Bouncy Castle Library API

```
private String encryptBouncyCastle(String strToEncrypt, String secretKey, Context
    context) {
    // make sure nothing is empty
    if (strToEncrypt.isEmpty()) {
        showToast(context, "Please enter a string to encrypt.");
        return strToEncrypt;
    }

    String encryptionKey = secretKey;

    if (encryptionKey.isEmpty()) {
        showToast(context, "Please enter a key.");
        return encryptionKey;
    }

    if (encryptionKey.length() < 32) {
        int keyLength = encryptionKey.length();
        int repeatKey = 32 / keyLength;
        encryptionKey = new String(new char[repeatKey]).replace("\0",
    encryptionKey);
        int newKeyLength = encryptionKey.length();
        int addKey = 32 - newKeyLength;
```

```
        encryptionKey += encryptionKey.substring(0, addKey);
    }

    Log.d("EncryptFragment", "Encryption Key: " + encryptionKey);
    Log.d("EncryptFragment", "String to Encrypt: " + strToEncrypt);

    Security.addProvider(new BouncyCastleProvider());
    byte[] keyBytes;

    try {
        keyBytes = encryptionKey.getBytes(StandardCharsets.UTF_8);
        SecretKeySpec skey = new SecretKeySpec(keyBytes, "AES");
        byte[] input = strToEncrypt.getBytes(StandardCharsets.UTF_8);

        synchronized (Cipher.class) {
            @SuppressLint("GetInstance") Cipher cipher = Cipher.getInstance("AES/
    ECB/PKCS7Padding", "BC");
            cipher.init(Cipher.ENCRYPT_MODE, skey);

            byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
            int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
            ctLength += cipher.doFinal(cipherText, ctLength);
            Log.d("EncryptFragment", "ctLength: " + ctLength);
            // log the encrypted string
            return Base64.encodeToString(cipherText, Base64.DEFAULT);
        }
    } catch (NoSuchAlgorithmException | NoSuchPaddingException |
    NoSuchProviderException |
                InvalidKeyException | BadPaddingException |
    IllegalBlockSizeException e) {
        e.printStackTrace();
        Log.d("EncryptFragment", "Exception: " + e.getMessage());
        showToast(context, "Error: Unable to Encode this Text");
    } catch (ShortBufferException e) {
        throw new RuntimeException(e);
    }
    return encryptionKey;
}
```

7.2 Decryption using Bouncy Castle Library API

```
private String decryptWithAES(String key, String strToDecrypt, Context context) {
    Security.addProvider(new BouncyCastleProvider());
    byte[] keyBytes;

    String encryptionKey = key;

    if (encryptionKey.isEmpty()) {
        showToast(context, "Please enter a key.");
        return encryptionKey;
    }

    if (encryptionKey.length() < 32) {
        int keyLength = encryptionKey.length();
        int repeatKey = 32 / keyLength;
        encryptionKey = new String(new char[repeatKey]).replace("\0",
    encryptionKey);
        int newKeyLength = encryptionKey.length();
```

```
        int addKey = 32 - newKeyLength;
        encryptionKey += encryptionKey.substring(0, addKey);
    }

    Log.d("DecryptFragment", "Encryption Key: " + encryptionKey);

    try {
        keyBytes = encryptionKey.getBytes(StandardCharsets.UTF_8);
        SecretKeySpec skey = new SecretKeySpec(keyBytes, "AES");
        byte[] input = android.util.Base64.decode(strToDecrypt.trim(), android.
    util.Base64.DEFAULT);

        synchronized (Cipher.class) {
            @SuppressLint("GetInstance") Cipher cipher = Cipher.getInstance("AES/
    ECB/PKCS7Padding", "BC");
            cipher.init(Cipher.DECRYPT_MODE, skey);

            byte[] plainText = new byte[cipher.getOutputSize(input.length)];
            int ptLength = cipher.update(input, 0, input.length, plainText, 0);
            Log.d("DecryptFragment", "ptLength: " + ptLength);
            Log.d("DecryptFragment", "plainText: " + Arrays.toString(plainText));
            ptLength += cipher.doFinal(plainText, ptLength);
            Log.d("DecryptFragment", "ptLength: " + ptLength);
            // make the plaintext based on the pt length
            String decryptedString = new String(plainText, 0, ptLength);
            // log the decrypted string
            Log.d("DecryptFragment", "Decrypted String: " + decryptedString);
            return decryptedString;
        }
    } catch (NoSuchAlgorithmException | NoSuchPaddingException |
    NoSuchProviderException |
                InvalidKeyException | BadPaddingException |
    IllegalBlockSizeException e) {
        e.printStackTrace();
        Log.d("DecryptFragment", "Exception: " + e.getMessage());
        showToast(context, "Error: Unable to Decode this Text");
    } catch (ShortBufferException e) {
        throw new RuntimeException(e);
    }

    return "";
}
```

8 Conclusion

Thus, we have studied and implemented encryption using bouncy castle API.

9 FAQ

1. *1. What is Bouncy Castle used for?*

   - **Purpose:** Bouncy Castle is primarily used as a cryptography library in Java applications.
   - **Functionality:** It provides APIs for various cryptographic operations, including both symmetric and asymmetric encryption algorithms.
   - **Use Cases:** Bouncy Castle is commonly employed for ensuring the security of data during communication and storage in Java applications.

2. *2. What do you mean by message digest? List different algorithms.*

   - **Message Digest:** A message digest is a fixed-size hash value computed from the input data, commonly used for ensuring data integrity.
   - **Algorithms:**
     (a) **MD5 (Message Digest Algorithm 5):** Produces a 128-bit hash value.
     (b) **SHA-1 (Secure Hash Algorithm 1):** Generates a 160-bit hash value. Note: It's now considered insecure for cryptographic purposes.
     (c) **SHA-256, SHA-384, and SHA-512:** Part of the SHA-2 family, producing hash values of 256, 384, and 512 bits, respectively.

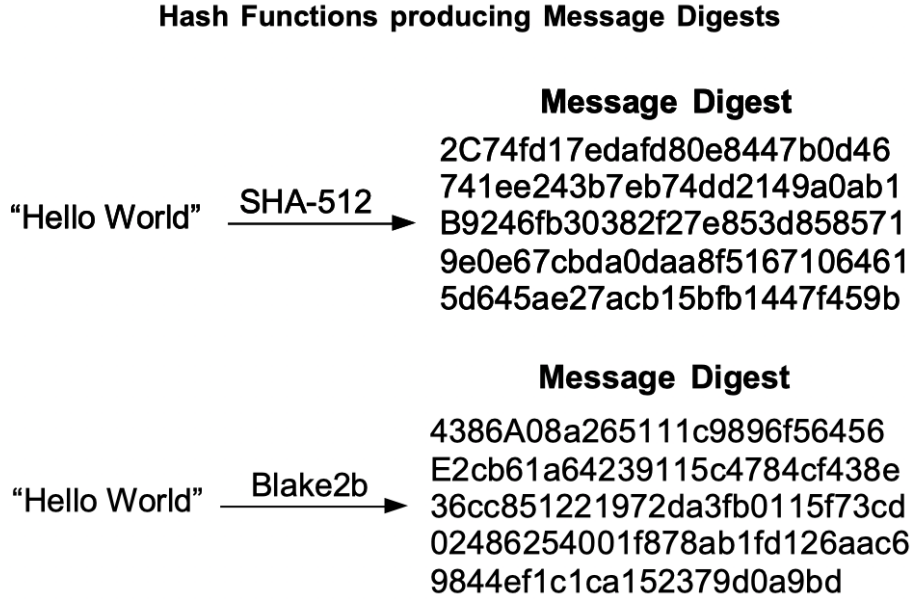

Figure 6: Example of a Message Digest

References

[1] Official Android Studio documentation.
https://developer.android.com/studio

[2] Official Bouncy Castle documentation.
https://www.bouncycastle.org/documentation.html

[3] Android Developer Guide on Permissions Overview.
https://developer.android.com/guide/topics/permissions/overview

[4] Android Developer Guide on Requesting Permissions at Run Time.
https://developer.android.com/training/permissions/requesting

[5] Java Cryptography Architecture (JCA) Reference Guide.
https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html