Hybrid Encipher to Protect Data on Shared Preferences

Anukul Dixit [1], Krishanu Das [2], Ronak Harish Patil [3], Brinda.S [4]
Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, India.

Abstract – In this present world different Android applications develop data on internal memory with Shared Preference interface. Due to this many loop holes can access the important information like passwords. Therefore, we propose an encryption system that would secure the confidential data through source code. We create an application and supply some data using Shared Preference. Then we apply the Encryption approach with Android Keystore system and an encryption method to hide important data.

Index Terms – Shared Preference, Android, Keystore, Encryption.

1. INTRODUCTION

The working structure of an android is developed by Google which is open source and is based on a modified version of Linux Kernel. It is designed for devices like tablets and touch screen mobiles. One of the popular operating systems is android. The android OS comes with a software development kit that provides the necessary APIs and tools to create new applications in JAVA platform. The Android Shared Preference interface a general structure that allows us to access and change key value pairs. By default, Android stores this information in an unencrypted XML file. In present world the data which are generated by the applications like passwords, files, etc. are not secured.

If device is lost, it leads to severe attacks. Attackers can access the data using ADB (Android Debug Bridge). ADB is a command line tool line tool that helps you communicate with the device. Another thing an attacker can use is Cheatdroid that can read and modify Shared Preference data. In order to minimize the security drawback Android Keystore system can be used. Android keystore system allows us store unique keys and values in cryptographic form which makes it very difficult to extract from the device. It is used by the Keychain API as well as Android keystore provider feature introduced in Android 4.3(API level 18). It does not let the key material to get extracted from the application process or android device. Also, it makes the apps to use their keys with authorization and restricted outside the app process. To protect the data on the device we propose the Hybrid Encipher system.

2. RELATED WORK

In this various encryption are used by different papers are discussed.

- Data security using authenticated encryption and decryption algorithm for android Phones
- Data secrecy using 3C with mobile application and packet encryption
- File encryption, decryption using AES algorithm in android Phone
- Secure storage of data on android based devices

2.1. Data security using authenticated encryption and decryption algorithm for android phones

It presents Conventional encryption schemes have very high computation costs. And also, there is a probability that the data is lost after this process. Hence, it is mandatory to implement a "Data Encryption and Decryption" scheme with "Cloud Backup" feature which is more efficient in order to provide high level of data security for android phones. Decrypt it to recover the encrypted data. In case of brute force attack, user need not worry about losing the data as "Automatic encryption" is triggered. Another mechanism to prevent the data from being known by an adversary is "Cloud Backup".

This paper concludes successful implementation of encryption/decryption, automatic encryption of the hidden private files and cloud backup of the file present in mobile phones. Using this we have tried to make it hard for the attackers to gain access to your private files or images saved in android smart-phones. The outcome shows the data security in Android Phones by using AES – 256 bit encryption algorithm [7].

2.2 Data secrecy using 3C with mobile application and packet encryption.

It presents the 3C concept that consists of 3 terms Customized, Cascade and Cryptography. It secures information from being seen or change and provides a protective means of communication over insecure channels by encrypting information several times using various algorithms choose by users as per their need. Thus, cipher text size and key size will grow which results in more security. This paper concludes the 3C concept is more secure than the traditional concept [8].

2.3 File encryption, decryption using AES algorithm in android phone

They have used Advanced Encryption Standard algorithm to overcome problems such as key size is increase but the process is slower than other methods. AES algorithm is good for both security and speed. AES algorithm is used for encryption and decryption. Thus, they conclude successful execution of file and image encryption as well as decryption. The user observes quicker file encryption and decryption. This shows that the AES encryption and decryption algorithm run quicker in android phone [9].

2.4 Secure storage of data on android based Devices

Here, they presented few mobile device security threats, the importance of protecting data and brief the protective storage component. A secure means of storage for Android platform is developed namely protective Storage. They introduced a flexible, simple to use and protective mechanism to protect the mobile phone user's information. Also, they have examined and presented the characteristic of performance analysis for the developed result [10].

3. PORPOSED MODELLING

In our hybrid encipher method, we first encrypt the data with certain encryption algorithm before moving to the Shared Preference storage. The key which is used to encrypt the data is secured using Android Keystore system. The Android keystore system is a container in which passwords and keys are stored in cryptographic manner. Firstly, the Android Keystore checks unauthorized use of password outside the device by preventing extraction of password from the application process as well as the whole device. Secondly, it checks unauthorized use of key on the device by providing specific authorization to the apps to use their keys and then enforcing these restrictions outside the applications process. Once the keys are stored in the Keystore system it is almost impossible to reverse engineer to extract the keys.

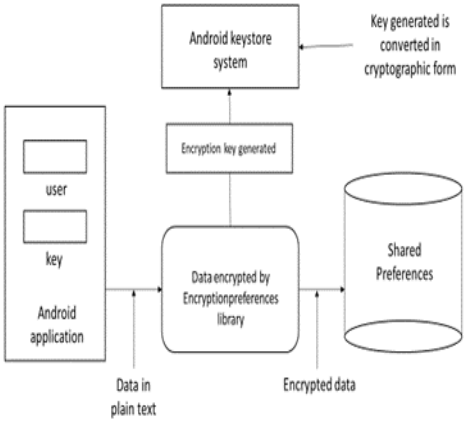

Fig – Flow diagram of hybrid encipher

4. RESULTS AND DISCUSSIONS

This is the source code that uses Encrypted Preference library to develop encrypted Shared Preference.

```
String name = username.getText().toString();
String pwd = password.getText().toString();
String key = "secret_key";
final EncryptedPreferences
encryptedPreferences = new
EncryptedPreferences.Builder(this).withEncryptionPassword(key).build();
encryptedPreferences.edit()
        .putString(name,"username");
        .putString(pwd,"password");
        .apply();
```

Fig - Code snippet using EncryptedPreferences library

As we can see in the above figure the string value “secret_key” is the password which is used for encryption work, which can be retrieved using reverse engineering. So, we combine the encryption method with Android keystore system. The next code shows how the password is stored in Android keystore system.

```
String name = username.getText().toString();
String pwd = password.getText().toString();
String encryptedString = null;

final KeyGenerator keyGenerator;
String TRANSFORMATION = "AES/GCM/NoPadding";
byte() iv;
try{
    keyGenerator = KeyGenerator
            .getInstance(KeyProperties.KEY_ALGORITHM_AES,
            provider:"AndroidKeyStore");
    final KeyGenParameterSpec keyGenParameterSpec =
          new KeyGenParameterSpec.Builder(keystoreAlias:"sample_alice",
          purposes:KeyProperties.PURPOSE_ECRYPT | KeyProperties.PURPOSE_DECRYPT)
          .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
          .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
          .build();
    keyGenerator.init(keyGenParameterSpec);
    final SecretKey secretKey = keyGenerator.generatekey();
    final Cipher cipher = Cipher.getinstance(TRANSFORMATION);
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    iv = cipher.getIV();
    encryption = cipher.doFinal("secret_key".getBytes(charsetName:"UTF-8"));
    encryptedString = new String(encryption, charsetName:"UTF-8");
```

Fig – Code snippet generating encrypted string by Android keystore system

The password is stored in a nonexportable vault as the keystore generation happens at runtime. This makes it difficult for attackers to extract the password. We used the encrypted byte array for decryption as given in the figure below.

```
KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(param:null);
final KeyStore.SecretkeyEntry secretKeyEntry = (KeyStore.SecretKeyEntry)
 keystore.getEntry(alias:"sample_alice",protParam:null);
final SecretKey secretKeyy = secretKeyEntry.getSecretKey();
final Cipher cipherr = Cipher.getInstance("AES/GCM/NoPadding");
final GCMParameterSpec spec = new GCMParameterSpec(tLen:128,iv);
cipher.init(Cipher.DECRYPT_MODE,secretKey,spec);
final byte[] decodedData = cipher.doFinal(encryption);
final String unencryptedString = new String(decodeData,charsetName:"UTP-8");
```

Fig – Code snippet generating decrypted string using Android keystore system

For verification purpose we can use the ADB shell or the cheat droid application to see what values are developed with Shared preferences.

This is the test application we created for the demonstration.

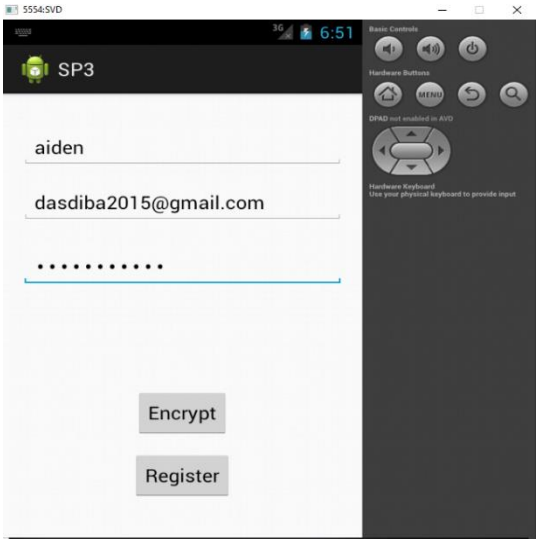

Fig – User interface for SharedPreference application

After filling the details, we click on to the to the encrypt button to encrypt the above user name and email address with the password the user provided. That password is then stored into the android keystore system to covert it in a cryptographic form. Then we click on to the register button so that SharedPreferences are created on the local storage of the system.

Below is the data created by the SharedPreference interface before applying the encryption method. This is achieved by navigating through the following path, /data/data<packagename>/shared_prefs/<filename.xml>.

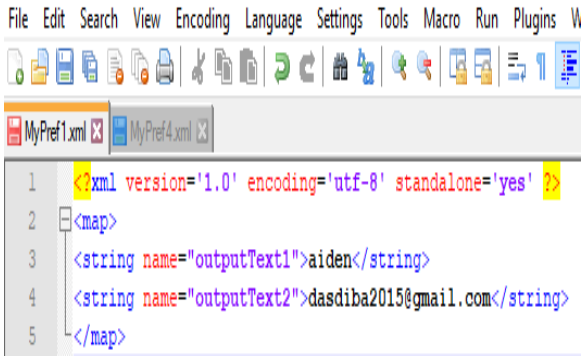

Fig – Contents of SharedPreferences XML file before encryption

After the encryption method was applied we can see that both the name and email are encrypted and not readable. As these values cannot be decrypted easily, it makes difficult for an attacker to extract the values.

Thus, we can say that Hybrid encipher gives the expected result.

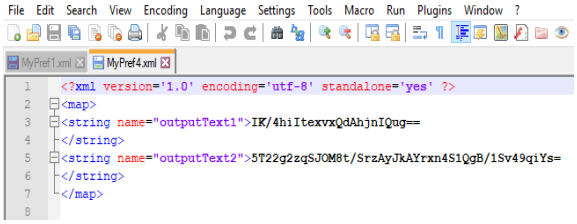

Fig – Contents of SharedPreferences XML file after encryption

5. CONCLUSION

This paper discussed of the data exposure of an application due to generation of XML files generated in the internal memory of the application. We proposed a different method to encrypt the data in shared preferences. We combined the android keystore system and the encryption approach to secure the leaking data from an application. The results indicated that the proposed encryption approach secures local data on the device.

REFERENCES

[1] "SharedPreference"[Online] Available: https://developer.android.com/reference/android/content/SharedPreferen ces.html
[2] "ADB" [Online] Available: https://developer.android.com/studio/command-line/adb.html
[3] "CheatDroid application" [Online] Available: https://play.google.com/store/apps/details?id=com.felixheller.sharedpref seditor&hl=en
[4] "Android Keystore System" [Online] Available: https://developer.android.com/training/articles/keystore.html#Security Features
[5] “EncryptedPreferences” [Online] Available: https://github.com/PDDStudio/EncryptedPreferences
[6] “Data security using authenticated encryption and decryption algorithm for Android phones” [online]Available: https://ieeexplore.ieee.org/document/8229903/
[7] “Data secrecy using 3c with mobile application and packet encryption” [Online]Available: https://ieeexplore.ieee.org/document/8076794/
[8] “File Encryption, Decryption Using AES Algorithm in Android Phone” [Online]Available: https://www.researchgate.net/publication/315669325_File_Encryption _Decryption_Using_AES_Algorithm_in_Android_Phone
[9] “Secure Storage of Data on Android Based” [Online]Available:Devices http:// www.ijetch.org/vol8/880-ST011.pdf