School of Computing and Information
University of Pittsburgh

Lab Project 2
Secure Mobile Application Development
Integration with the Google Cloud Service

Version 1.1

This lab is based on mHealth project by *Haobing Huang* and *Runhua Xu*.

School of Computing and Information
University of Pittsburgh

Read the following guidelines before working in the project

Lab Project Goal:

The goal of this project is to learn the basic skills for developing a secure mobile health app. This project will involve the following tasks.

- Secure the outsourced medical record
  - Learn to use common widgets of the app
  - Learn to use cloud key management service provided by Google Cloud
  - Secure the medical record before outsourcing to the Firebase cloud
- Proximity based access control
  - Learn to use *nearby service* as the proximity -based access control approach

General Guidelines:

For this project, you will first need to review the developer's guides for **Android**, **Google Firebase** and **Cloud Key Management Service** if you don't have the relevant background and experiences. Check the following materials:

- Documentation for app developers [https://developer.android.com/docs/]
- Documentation for Google Firebase [https://firebase.google.com/docs/]
- Documentation for Google Cloud Key Management Service [https://cloud.google.com/kms/docs/]

NOTE:
Please note that we have integrated the Firebase by providing our LERSAIS's ***google-services.json*** file in the project. It may not allow you to manage the backend server, namely, the Firebase console. If you want to debug/test your project using your own Firebase console, please follow the instructions provided by Firebase and replace the ***google-services.json*** file that have been provided to you in the project skeleton with your own.

Moreover, we have also integrated the Google Cloud Key Management Service and provided the ***LERSAIS-mHealth-KMS-bdd9f7acef42.json*** file in the assets folder in the skeleton. You do not need to create your own.

Our LERSAIS lab has real android phones for testing in case you need. Please schedule a time slot with TA.

School of Computing and Information
University of Pittsburgh

Secure Outsourcing Medical Record and Proximity-based Access Approaches

A Healthcare Scenario [extended version]

In your lab project 1, we have the following scenario:

Suppose that a patient needs to go to different hospitals/clinics for different healthcare services/checkup/treatment. It is often tedious to fill the medical forms to provide his/her medical history information such as allergy history, family genetic history, etc. Furthermore, the staff/information systems of the hospital or health clinics/units may be not fully trustworthy to store and manage his/her medical history record.

Thus, the goal here is to design a mobile healthcare application to help a user manage the medical history record. It is a user-centric medical record management application where users have full and complete control of their healthcare data.

Now, we add more features based on the above scenario:

The medical history record in your lab project 1 is stored in cloud database in plaintext. However, the medical history record is sensitive data for patients due to privacy concerns. Thus, it is necessary to encrypt the medical history record before outsourcing it to the cloud.

Then, we consider the medical history record usage scenario. Suppose that a patient is receiving treatment in the treatment room and he/she needs to present his/her medical history record to a group of people including attending physician, assistant physician, trainee physicians and nurses in the treating room. Thus, it is necessary to design a proximity-based data access control approaches with consideration of privacy concerns.

School of Computing and Information
University of Pittsburgh

Task 1: Secure Outsourcing of Medical Record

This task adopts Google Cloud Key Management Service to generate a key for each user. The generated key will be used to encrypt/decrypt the medical record that will be stored in the Firebase Database. Generally speaking, we treat the Cloud KMS as the trusted-third-party to help manage users private key and the Firebase Database as the *honest-but-curious* cloud; this means the cloud service executes the commands as per user's requests, but it will try to collect/infer the sensitive information from the data it manages, and the commands that users execute.

For documentation of Google Cloud KMS, please visit:

- https://cloud.google.com/kms/docs/

The procedure is described as follows:

1. Initialize a private key in the Cloud KMS when a new user registers in the app.
2. Different from lab project 1, before a user tries to save his/her medical in the cloud database, the application first retrieves his/her private key and uses the encryption service of the Cloud KMS to encrypt the medical record. Then the encrypted data will be saved in the cloud database.
3. When the user wants to review his/her medical record, the app first downloads the encrypted medical record from the cloud database, and then decrypts it using decryption service of the Cloud KMS.

Go through the code provided in the skeleton. The interface/view is already provided to you, so you do need to worry about designing it. Specifically, you will accomplish the following tasks:

Task 1.1 Initialize the crypto key. (SignupActivity)

Note that the Google Cloud KMS information has been provided in the skeleton. The service account **json** *file is provided in the assets folder. The project id, location, and key ring information has been provided in* **Constant** *class.*

Task 1.2 Implement the encryption/decryption function. (CloudKMSUtil)

Task 1.3 Familiarize yourself with the view of medical record and acquire/encapsulate the information. (**MedicalRecordEditActivity**)

Note that model class **MedicalHistoryRecord** *has been provided. What you need to do is acquire the information from the view and encapsulate that into the* **MedicalHistoryRecord** *object/class, and vice versa.*

Task 1.4 Implement the EHR encryption. (**EncryptMedicalRecordThread**)

Note that the basic encryption function has been provided in CloudKMSUtil (Task 1.2). What you need to do is adopting the provided encryption function to encrypt an original **MedicalHistoryRecord** *into an encrypted* **MedicalHistoryRecord***.*

Task 1.5 Implement the EHR decryption (**MedicalRecordViewActivity**)

Note that we only provide **MedicalRecordViewActivity** *in the skeleton. Thus, this is an open task. You can refer to the the encryption procedure in Task 1.3/ 1.4 and related files as example to implement the decryption algorithm and present the record in the view.*

Task 2: Proximity-based Access Control

Recall the requirement in the scenario. A patient needs to share his/her medical record to a group of doctors/nurses in the same room. Considering that the feature is based on location proximity, we adopt the **Google Nearby** service to build simple interactions between nearby devices and people. **Nearby** helps you find and interact with services and devices close to you (within about 30 m or 100 ft).

Specifically, this task adopts **Nearby Messages** and **Nearby Connections** service to implement the proximity-based medical record access control, respectively.

- **Nearby Messages** *exposes simple publish and subscribe methods that rely on proximity. Your app publishes a payload that can be received by nearby subscribers.*
    - *Discover and exchange information with other devices, without having to be on the same local network. Nearby Messages enables seamless nearby interactions such as multiplayer gaming, realtime collaboration, forming a group, broadcasting a resource, or sharing content.*
    - *The* **Nearby Messages** *API is available for Android and iOS, and enables communication between the two platforms.*
- **Nearby Connections** *is a peer-to-peer networking API that allows apps to easily discover, connect to, and exchange data with nearby devices in real-time, regardless of network connectivity.*
    - *Discover other devices nearby and create high-bandwidth peer-to-peer connections for real-time cross-device experiences like gaming and file sharing.*

School of Computing and Information
University of Pittsburgh

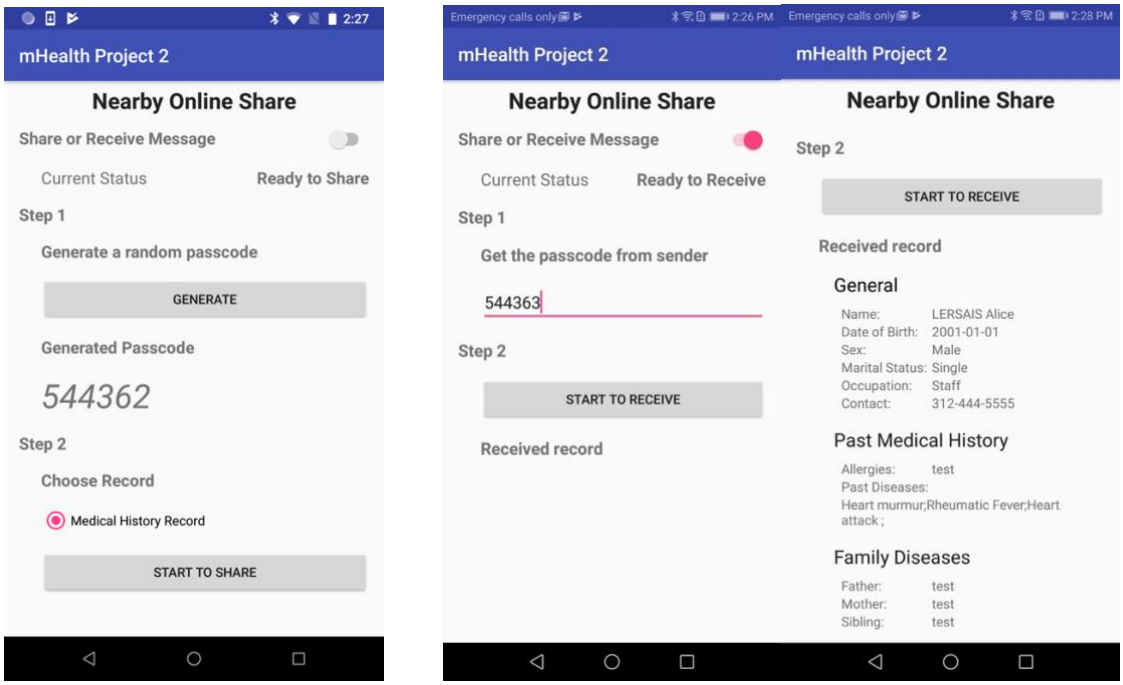
Task 2.1 Implement the nearby record sharing using Nearby Message API.

Note that we have provided a sample view and activity code in the skeleton. Go through **NearbyRecordOnlineShareActivity** *and* **activity_share** *views and add the missing parts. Specifically, the procedures are as follows:*

- *The Medical Record Sender*
  - *Task 2.1.1 Generate a random alphanumeric passcode and display*
  - *Send the EHR using nearby message*
    - *Task 2.1.2 Get the encrypted medical record from the cloud database*
    - *Task 2.1.2 Decrypt the encrypted medical record*
    - *Task 2.1.2 Form the message from the medical record using the passcode for confidentiality and Share the medical record using Nearby Message*
    - *Reference*
      - *Nearby https://developers.google.com/nearby/messages/overview*
      - *RNCryptorNative https://github.com/TGIO/RNCryptorNative*
- *The Medical Record Receiver*
  - *Receive the EHR using nearby message*
    - *Get the passcode from users' input*
    - *Task 2.1.3 Receive the medical record via nearby message*
    - *Task 2.1.3 Decrypt the message and form the medical record using the passcode, and display the medical record*

Note that it is recommended to use two real cell phones to test such nearby functions.

Sample views



School of Computing and Information
University of Pittsburgh

Task 2.2 (BONUS) Implement the nearby record sharing using Nearby Connection approach.

Note that we have provided a sample in the skeleton. Go through and ***activity_send*** *view and try to create* **NearbyRecordOfflineShareActivity** *to implement this task. We have integrated* **NearbyConnectionsActivity** *from google samples which provide the basic functions for* **Nearby Connection**. *Your* **NearbyRecordOfflineShareActivity** *can extend from the* **Activity** *and add the features you like.*

A sample procedure is as follows:

- *The Medical Record Sender*
  - *Trigger the button to register the Nearby Connections Service*
  - *Shake the device to start advertising and wait the connection*
  - *Establish the connection by using an authentication method*
  - *Send the data*
- *The Medical Record Receiver*
  - *Trigger the button to register the Nearby Connections Service*
  - *Shake the device to start advertising and wait for the connection*
  - *Establish the connection by using an authentication method*
  - *Receive the data*

Reference from google samples to help you understand the Nearby Connections
https://github.com/googlesamples/android-nearby/tree/master/connections

Sample views