Decision Log

Requirements Decisions:

1. AWS Elastic Beanstalk makes the deployment process very easy, handling the web server setup as well as providing easy integration with other AWS services which we can potentially use. We plan on using the 12-month Free Tier and staying within those limits to avoid any costs for this project.

2. While there are other frameworks and languages for creating REST APIs, the Django REST Framework simplifies this process a lot, while still allowing flexibility in creating APIs. We both also want to build on our previous Django experience.

3. The most implementations and documentation of the OBEX protocol working on a Windows-based machine are programmed in Java, using the Java APIs for Bluetooth (JABWT) and following JSR-82, the standard to develop Bluetooth applications in Java. JSR-82 has a library called BlueCove that acts as an interface with the Microsoft Bluetooth stack. Thus, we will be using Java for creating the PC background process.

Other Decisions

4. We have decided to store our user device info, as well as our top level directory (TLD) to serve files in encrypted files to allow strict communication and file transfer with the chosen device, and from the directory that the user chooses. The symmetric key used for encryption/decryption will be an environment variable so that it is separated from our code.

   UPDATE: The symmetric key is stored as plaintext in the project files. It cannot be accessed through the program, even if the file lives under the TLD.

5. We use Firebase Cloud Messaging (FCM) because it is the primary cloud messaging service used with Android devices. FCM will help our cloud server communicate with our Android Server.

   UPDATE: For the interest of time, we have decided to rely on TeamViewer instead of creating our own cloud server that uses FCM to communicate with our Android server to automate the communication. Using TeamViewer allows us to meet our end goal of accessing files from our PC at home from a front-facing client (Android, iOS, Web) remotely. The end-user will now remote into the Android server and interact with that app to initiate the file transfer. One drawback of this approach is that it requires us to control the phone rather than allowing us to send data back and forth in the background. This would interfere with someone else using that Android device. However, the approach also removes potential dangers of automating everything, which is difficult to do so gracefully.

6. For our PC server, we encrypt information (see 4) using a symmetric key. We require the user to provide his/her own key at the setup of the program to maintain user privacy. We also require that the password/key length is 16 bytes so that brute-forcing any cipher (i.e., registered mac address, registered TLD) is computationally infeasible.

   UPDATE: Because the symmetric key is only used once at the beginning of starting the PCServer program to encrypt and decrypt the TLD and UDA files, it does not need to known to the user. If a symmetric key is needed, one is computed randomly and stored in the project files. From there, it will be used to encrypt a missing TLD or UDA file and to encrypt their contents.

7. Our PC server now has 2 threads: one is for checking whether the PC Server's Bluetooth is turned off and checking which state PC Server is in, and the other is for accepting file transfer requests. This allows us to asynchronously accept file transfers (up to a limit of 5, to not overload our server).

8. We created our own protocol class (CloudToothMessage), compiled it as a JAR file, and included it in our Android and PC server libs/ folder. This object wraps the message type, and the message content, which allow us to easily communicate with the client and server using objects rather than bytes (and then assembling them back up into objects).

9. Our Android Server makes use of the Google Drive API to upload the requested file from the PC Server up to the drive of the Google account chosen as part of the Android Server setup. We chose to use Google Drive for this because it is the most popular cloud service and some storage comes free with all Google accounts.

10. PC Server will not send the TLD, UDA, or private key files to Android Server, even if they are under the TLD hierarchy. These are files that should not be tampered with and giving access to these files is a security risk to the user of the program.