COMP3004B - Deliverable 3

Metadata

Team Name: Hyggelig Security
Application: hyggelig secure
Team Members:

- Kavan Salehi 101046945
- Connor Stewart 101046945
- Kevin Sullivan 100896774
- Gabriel Valachi 101068875

System Architecture

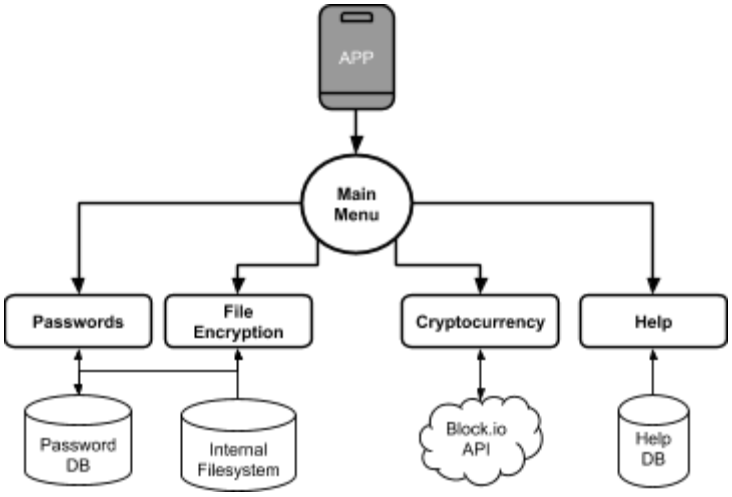

Figure 1: hyggelig secure Microservice Architecture

Hyggelig secure's architecture is a mixture of microservice and pipe-and-filter. The system provides four services: Passwords, File Encryption, Cryptocurrency, and Help. The application main menu acts as a gateway to the four services. Each service is a subsystem that parses input, may make changes to the input, then either stores the modified input, or updates the GUI with the result of the input passing through the subsystem. Given the flow of data through the subsystems, hyggelig secure's subsystems lends themselves to a pipe-and-filter architecture - data is pumped into filters by users, filters process the data, and place the data in appropriate sinks. Data stored in sinks can be pumped back through a different filter to be placed in a sink visible to the user. Applying a pipe-and-filter architecture to the subsystems also helps minimize coupling, as each subsystem makes use of its own filters and pumps. Decoupled subsystems help support a secure system, as data sharing between subsystems is minimized, and data flow is easier to follow.

Password Manager Architecture

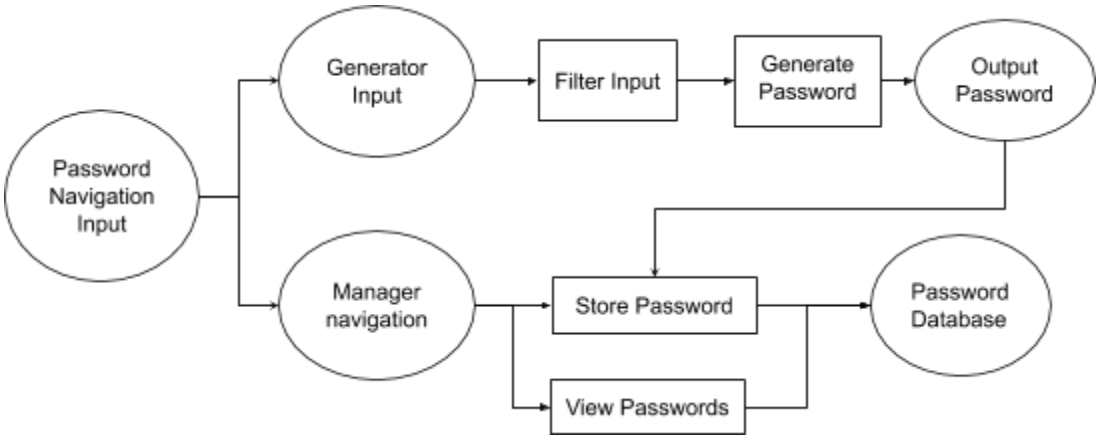

Figure 2: Passwords Subsystem Architecture Diagram

The Password navigation section includes two features, Password Manager and Password Generator, both of these being independent of each other and taking in some input and assigning that data

either to a data store or output. This demonstrates the use of the Pipe and Filter architecture style. When a user uses the password generator, they give a series of inputs which the system then transforms into a randomly generated password as output which is independent of all other subsystems. The password manager is the same, the user can choose to either store a password or view currently stored passwords, this is also independent and takes the users input and accesses a datastore. The password manager and generator are designed in this way so that a user does not need to interact with any other subsystems in order to use this feature and can use the system purely for this feature alone.

Cryptocurrency Subsystem Architecture

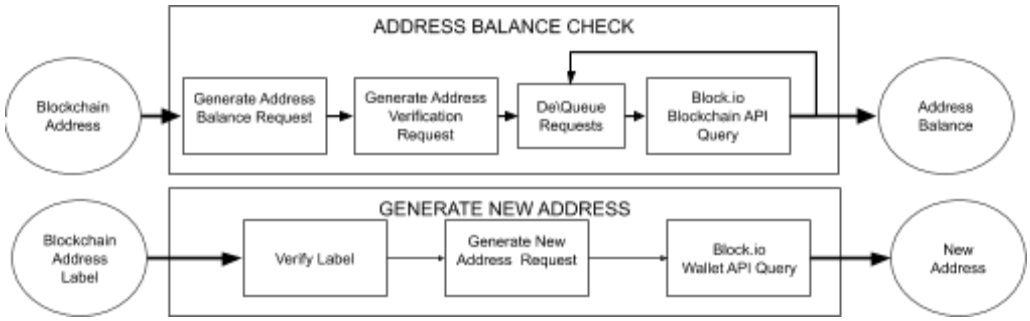

Figure 3: Cryptocurrency Subsystem Architecture Diagram

The Cryptocurrency Subsystem consists of two linear data flows: the address input pump to the "Address Balance Check" filter which pipes to the address balance sink, and the address label pump to the "Generate New Address" filter, which outputs to the new address sink. The "Generate New Address" filter is linear, simply verifying the label, generating an HTTP request, and querying Block.io before outputting the response to the sink. The "Address Balance Check" filter pipes to the "Generate Address Balance Request"filter, which pipes a request object to the "Generate Verification Request" filter, which then pipes the requests to be queued. The "De\Queue Requests" filter receives new requests, as well as feedback from the "Block.io Blockchain API Query" filter in the case of an invalid address. The "Block.io Blockchain API Query" filter feeds the sink as well, as the HTTP response in every case is displayed from the "Address Balance" sink.

Help Subsystem Architecture

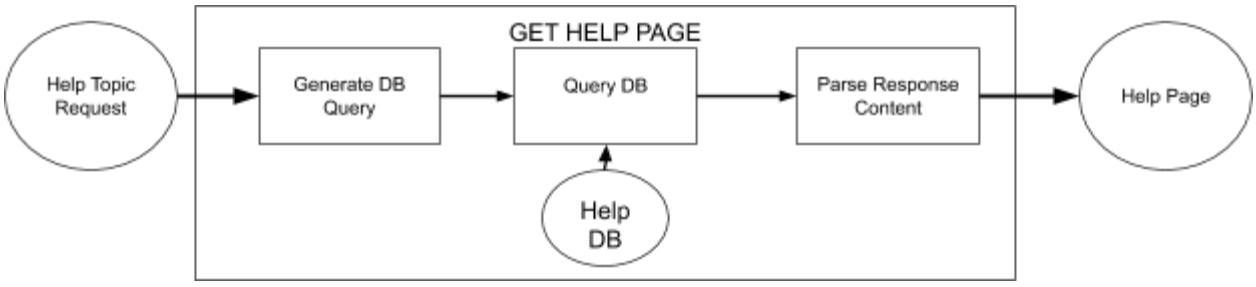

Figure 4: Help Subsystem Architecture Diagram

The help subsystem follows a linear data flow as well. Requests are made for certain help categories or help topics, which are made into queries, run against the database of help topics, then output to the "Help Page" sink. This sink is drawn on by the UI, and is planned to be drawn on by other subsystems.

Encryption Tools Architecture

The encryption tools consist of four main functional components: the encryption and decryption tools, signing and verification tools, and the private folder. The encryption and decryption tools use two systems: the symmetric key and the asymmetric key algorithms. The private folder uses the symmetric encryption system to protect files in the directory. The asymmetric key system is used for verifying that files were sent from the person who claims to have sent them. Overall, the system runs with a pipe-and-filter architecture, with non-linear flows depending on the request.

The encryption and decryption tools work with both symmetric and asymmetric techniques. The asymmetric tool requires that a public and private key pair is first created or imported. This is done by using a password from the password generator and importing a hash, cipher, and compression algorithm. The password and the algorithms are then inputted into the OpenPGP library with a request to make a pair of a public key and a private key. The public key can be used to encrypt information, whereas the private key can decrypt information when used with the key pair's password. The private key system makes use of AES to encrypt a file with a password which is hashed. This password can later be used to decrypt the encrypted file.

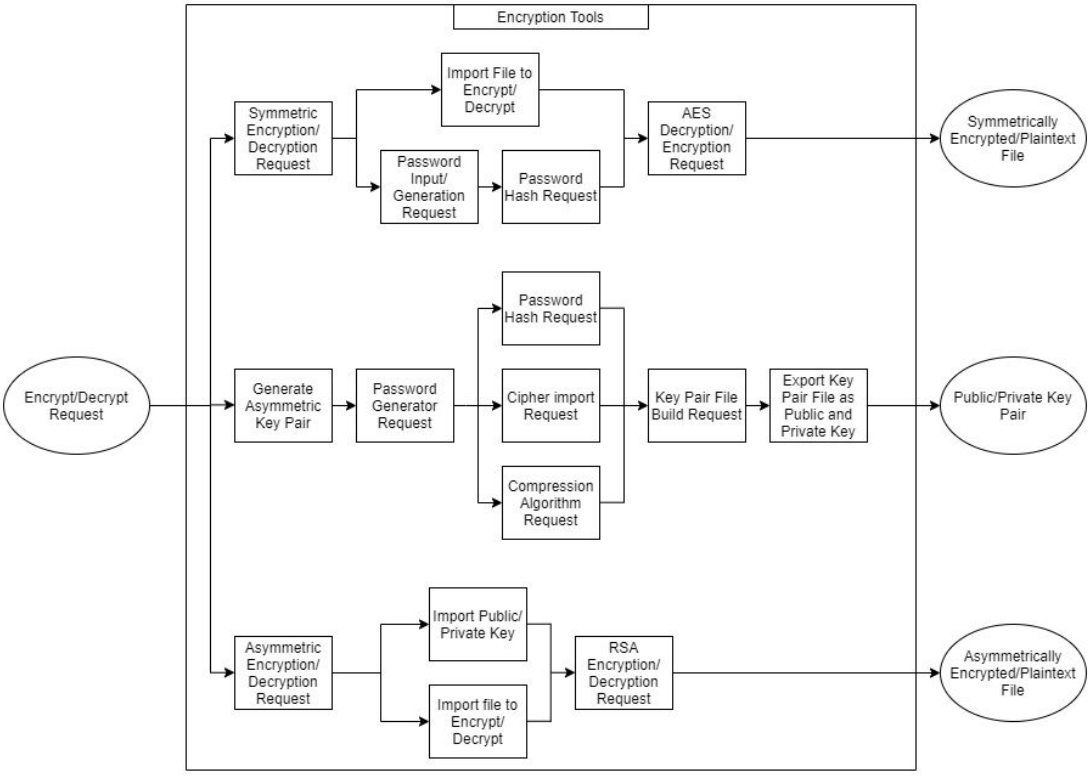

Figure 5: Encryption/Decryption System Architecture Design

The authentication system implements a public/private key verification scheme. To verify that a legitimate user is contacting the client (the application), the sender (from a high level) encrypts a piece of information using their private key, and the client will use the sender's public key to decrypt the

information. If the user can report the decrypted information, the file is correctly signed and therefore comes from a legitimate user.

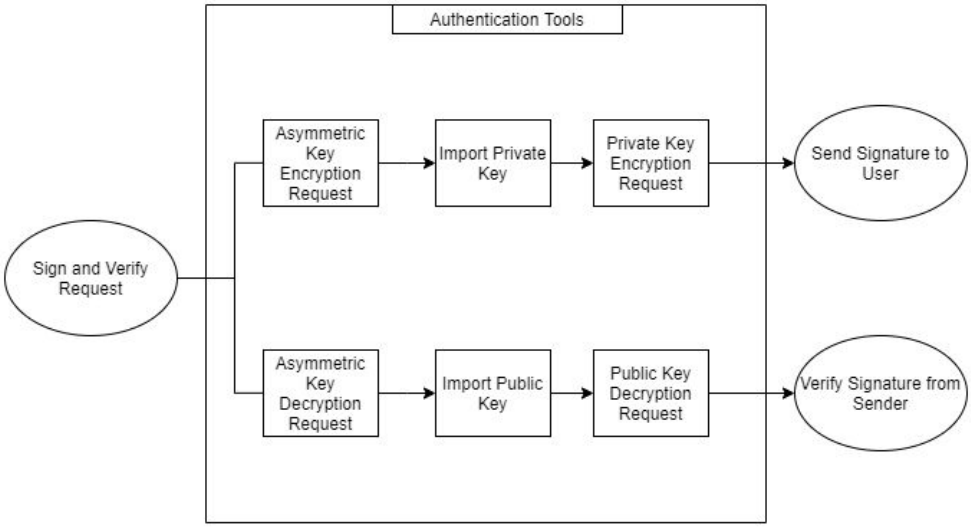

Figure 6: Authentication System Architecture Design

The private access tools takes advantage of the symmetric key encryption system to allow the encryption and decryption of individual files within a folder. The files themselves are indexed in a database, which can be viewed within the application. If a request to add a file to the private folder is made, the file is encrypted with symmetric AES and moved into the private folder. If a file is to be removed from the private folder, the encrypted file is deleted from within the private folder.

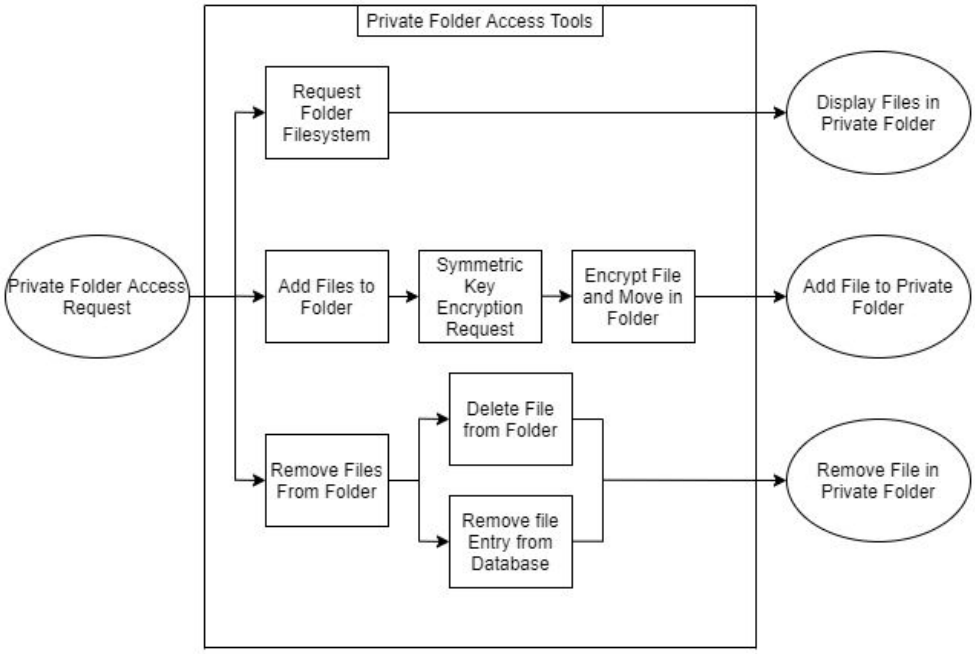

Figure 7: Private Folder System Architecture Design

System Design

Design Analysis

Each of hyggelig secure's subsystems is implemented as an Android Activity. Hyggelig secure is designed so that it provides a set of independent services with unique functionality. This design choice results in the system being loosely coupled and allowing all subsystems to run independently. This also allows additional subsystems to easily be added to the main system as additional microservices encapsulated in an Activity. Alternately, each subsystem could have been implemented as Fragments that are swapped in and out of a main activity to reduce the resources needed to run the application. However, this adds significant overhead in managing the code, as the Fragments require a number of supporting classes. Given that the system's functionalities are not resource-heavy, simplifying code management between subsystems was prioritized.

Password Subsystem

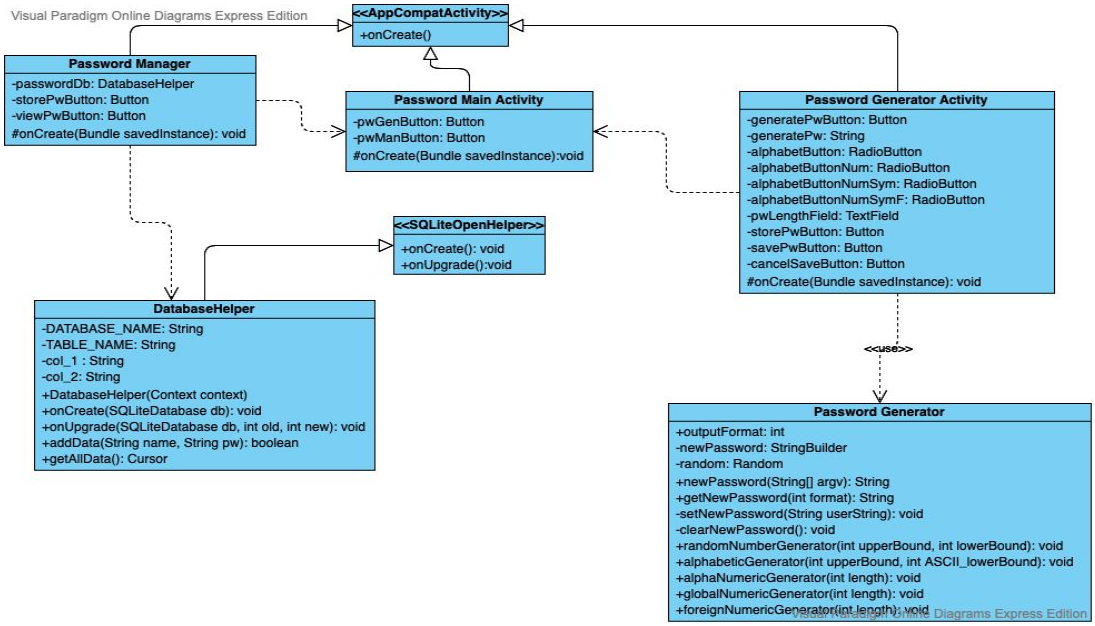

Figure 8: Password Subsystem UML Class Diagram

The password subsystem is designed in a way that there are two easy to use features that will provide a service to the user and improve their security. The password manager activity and password generator activity are designed to have no dependencies on any other subsystems making them loosely coupled. The password generator has added functionality to assign a generated password a name to be able to store it in the password manager for a better user experience but this does not make the generator dependant on password manager in any way because it can still be used as a stand alone feature.

The Password manager is designed so that it depends on a database which stores passwords locally on the device. A drawback to this design choice is that if a user was to delete the app or were to lose their device all of their stored passwords would be gone. A solution to this would be to have a server which contains all of their stored passwords and require some type of verification process to access them again.

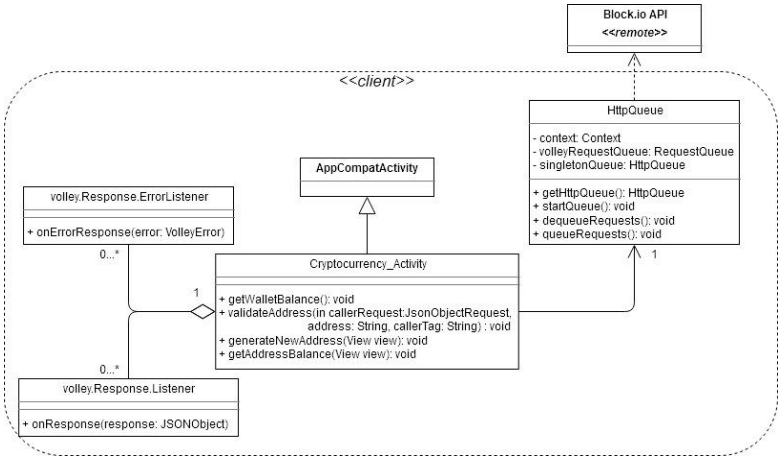

Figure 9: Cryptocurrency Subsystem UML Class Diagram

Cryptocurrency Subsystem

The Cryptocurrency subsystem was designed using the Android Volley library to action HTTP requests against the Block.io API. Volley allows for definition of response callbacks by request object, which is useful for the Crypto subsystem as each functionality has different needs. This allows the implementation of the feedback loop shown in **Figure 3** by cancelling any queued requests tied to an invalid address in the response callback for validateAddress(). Though the subsystem is externally coupled to the Block.io API, it only relies on HTTP request helper objects and the HttpQueue to function. Even if the HTTP request library is changed, the API for queueing requests from the main activity can remain the same.

A drawback to the current design is that adding additional functionalities (e.g. archiving addresses, or changing cryptocurrency networks) would likely lead to crowding the screen with too many options. Instead, refactoring to implement an Adapter pattern to the cryptocurrency activity where each functionality "plugs in" to the main activity one at a time would provide a more modular interface, with easy access to more functionality.

Help Subsystem

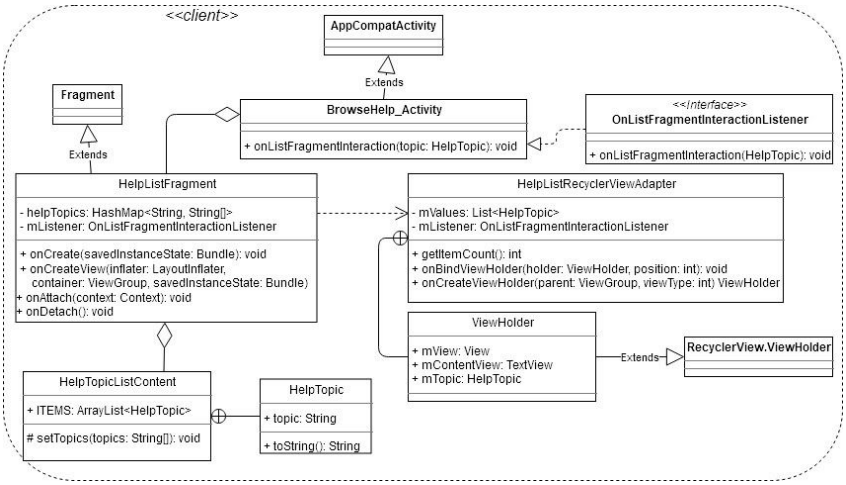

Figure 10: Help Subsystem UML Class Diagram

The Help subsystem was designed using Android's Fragment classes. Though this adds complexity to the subsystem design, it allows for easy changing between lists of help topics and pages with help information. It also allows for easy extension of the help pages, as help topics are created as classes instead of simply Strings. For example, this could accommodate additional state that is used when searching for help topics. As part of the Android ListFragments, the Adapter pattern is built-in to adapt classes with a RecyclerView, providing a scrollable interface.

Encryption Tools Subsystem

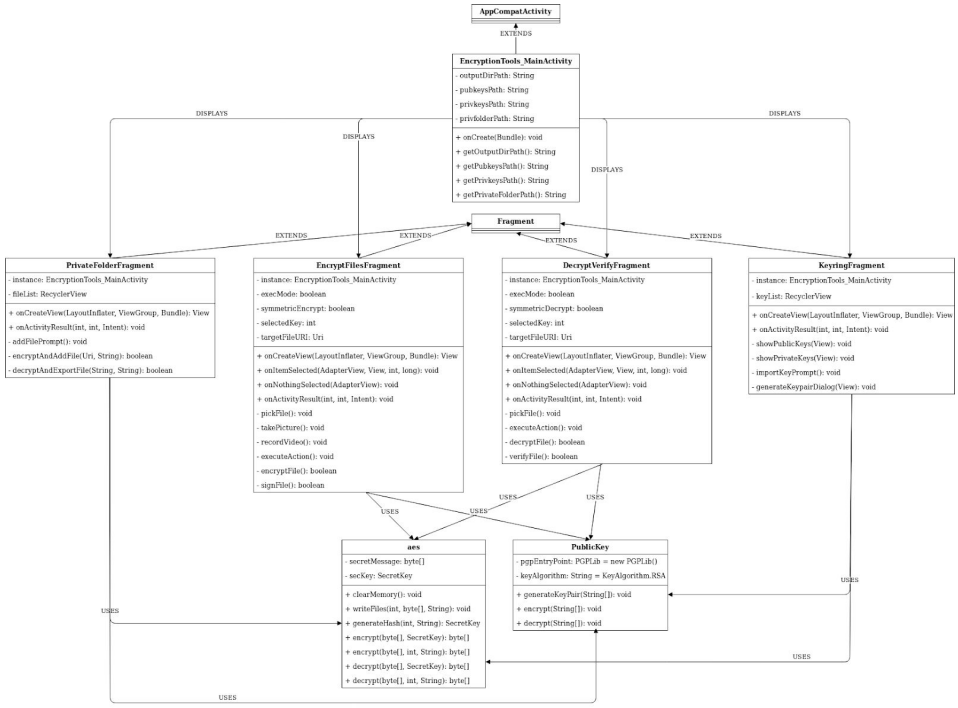

Figure 11: Encryption Tools Subsystem UML Class Diagram

Each activity in the Encryption Tools subsystem is handled in a single Fragment, which controls both the user interface and the program logic. This causes the UI and logic of each subroutine to be coupled, making the code slightly less comfortable to navigate, but also allows each subsystem to operate independently and privately from the other subsystems. The AES and public key encryption systems are decoupled from the Fragments, since they are depended on by all processes and since the encryption methods' usage is consistent across all subsystems; this allows their methods to be quickly modified to suit the program's needs, without having to change the specific usages by each Fragment/subroutine. The encryption system classes - aes and PublicKey - are used to access external libraries implementing OpenPGP functions.

Private files are stored, encrypted with AES given a user-chosen password, in an application-specific folder that cannot be directly accessed by any other applications or by the user. Public and private keys are also stored in an application-specific directory, public keys separate from private keys and vice versa, as ASCII-armored OpenPGP keys. This is primarily due to limitations with the OpenPGP library used that does not allow keys to be imported from memory. However, although it would be ideal to store keys in a database file for organizational purposes, storing them in the filesystem allows them to be easily accessible with default Java methods from any Fragment-extending subroutine, not requiring any special methods for database access.

Design Patterns

Singleton

The cryptocurrency activity makes use of Android's Volley library to queue and action HTTP requests against the Block.io API. When checking an address's balance, it is crucial that the request to validate the address and request to check that address's balance are in the same queue so that if the address is not valid, the balance check request can be cancelled. By creating the HttpQueue class as a singleton, the cryptocurrency activity can ensure that only one Volley RequestQueue exists, ensuring that all HTTP requests are submitted to the same queue.

Facade

The Password Generator makes use of the Facade Design pattern. The password generator class encapsulates the whole password generator subsystem interface into an entire class which the Password generator activity class can use to retrieve the generated passwords. This allows for generating passwords to become easier to use.

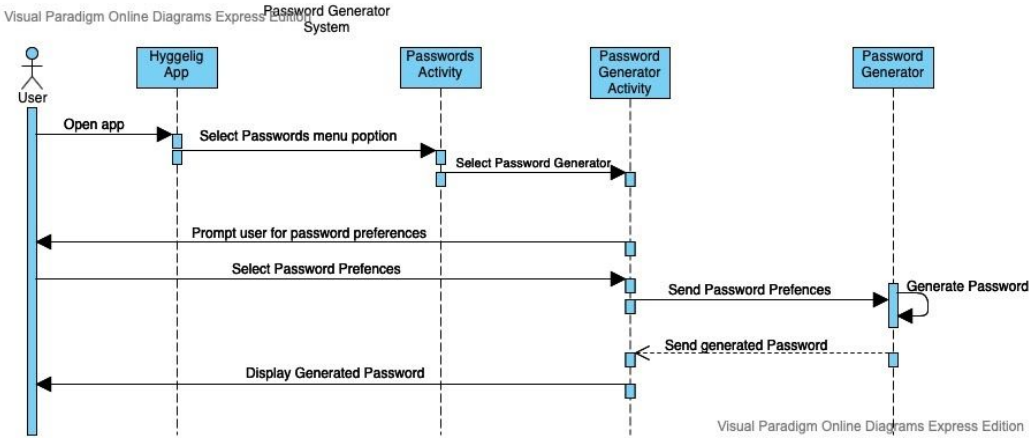

Figure 12: Sequence Diagram of the Password Generator

Task Assignments

Kavan Salehi	● Passwords Subsystem ○ Facade Design Pattern
Connor Stewart	● Encryption Tools Architecture: ○ Private Folder Architecture ○ Encryption/Decryption Architecture (symmetric and asymmetric) ○ Authentication (Sign and Verify) Architecture
Kevin Sullivan	● Cryptocurrency Subsystem ○ Singleton Design Pattern ● Help Subsystem
Gabriel Valachi	● Encryption Tools Subsystem