Secure Airport Tower
First Development Journal

frederic.jacobs@epfl.ch
hantao.zhao@epfl.ch

11 mars 2012

1 Setting up

During the set up phase, we chose Git over SVN. There are many reasons that explain this decision but we'll highlight here only a few. We really fancy the social features of GitHub for collaboration. GitHub Issues makes it easier to report bugs and to fix them. Git is also way simpler to set up than SVN. The branching of Git makes it way easier to work on different branches and then merge them with the master at a further stage. We set it up as a private repository according to the requirements of this course.

2 Project's Architecture

Our program is structured into different packages.

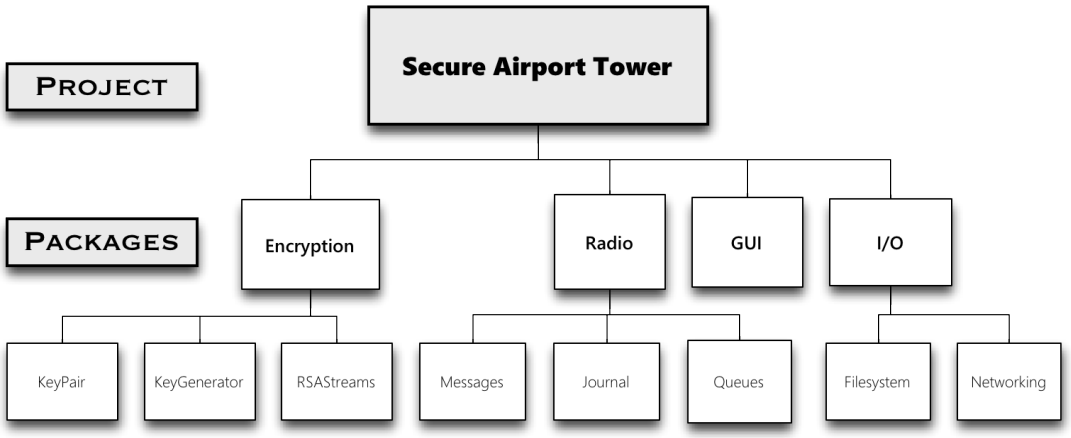

FIGURE 1 – The current architecture of our software. It is subject to change in later iterations for consistency.

2.1 Encryption

This package contains two main classes namely *KeyPair*, which can store both public and private keys, and *KeyGenerator*, which can generate both public and private keys with a given length during the initialization of the program. In addition of these two classes we coded a *RSAInputStream* and *RSAOutputStream* which can handle encryption and decryption of incoming and outgoing streams.

2.2 GUI

The Graphical User Interface hasn't been coded yet but the package is already there, waiting to be implemented.

2.3 I/O

At this current stage of development we haven't coded the network support yet. Thus, we built a custom class to write the messages in the filesystem instead of sending them straight to the network.

2.4 Radio

The Radio handles all messaging by sorting them into priority queues. The radio package also holds all of the Message classes which defines what the messages contains. We also provide support for data messages (using the I/O package to get some special hash treatment).

3 Q&A

In ITP-02, we were asked two questions.

1. Is it a good idea to use a superclass to define Messages ?
2. Why is it an abstract class ?

In a object-oriented universe like Java, we like to model things hierarchically. We want all possible types of messages to have a behavior or to have some instance variables therefore defining a super-class Messages makes a lot of sense. We were asked to have these instance variables in each Message Type so we implemented these in our super-class.

```
byte[] planeID;  //The unique identifier of the plane
int length; //Length of the message
int priority; //Priority in PriorityQueue
int posx; //Positioning data
int posy;
MessageType type; //Type of Message
```

The reason why we want this to be an abstract class is really straightforward. We defined a Message super-class because we expect messages to behave a given way but we also want these messages to have a type. All messages will inherit these attributes from the super-class but the super-class won't be instantiable to assure we call the right constructor when creating a message with a certain type. The keyword *abstract* does exactly this.