23/10 Install Java Card JDK and its dependencies, read specification.

24/10 Install and get accustomed to Eclipse, set up project, configure it with Java Card JDK. Created a new GitHub repository and initialised it with documents, JDK specifications, and code.

27/10 Install OpenCard framework and read documentation

28/10 Started working on skeleton code for host-side app using code samples and OpenCard api docs.

29/10 Wrote most of both host-side and card-side app, haven’t yet worked out how to install onto card because I don’t have my card reader.

31/10 Reader arrived. Read more of the documentation, built a project. Installed gpshell, installed reader driver, successfully loaded gpshell sample applet onto card, listed contents, and removed it.

2/11 Attempted to load project onto card, discovered that my java cards are not JCDK 3.0.5 compatible. Choice: Get more up-to-date java cards, or use the older build process with the older JCDK? Reverting to older process means no handy Eclipse plugin. Documentation is also much better in newer version. Decided to buy new cards.

4/11 Attempted to run sample project by running OCF host app and connecting to card. OCF wouldn’t recognise the card. I think I’ve identified that the reader needs an OCF driver, but it only has a PC/SC driver. Sources are few and far between. May have to redo application in C++ to use PCSC API. Could also be incompatibility with Windows 10, perhaps try in Ubuntu first.

Data sheet for reader only specifies PCSC whereas for some other readers their data sheets also mention OCF.

From OCF programmer’s guide: “The OpenCard Reference implementation comes with a lockable CardTerminal implementation for PCSC card readers”. Downloaded reference implementation, there’s source code for PCSC-related stuff. Pcsc-wrapper-src.jar contains class files. Should look into usage. Very little information online about them. In particular, contains Pcsc10CardTerminal which emulates OCF stuff on PCSC.

New cards arrived. Tried installing apps, got Unknown ISO7816 error: 0x6438 for apps compiled via eclipse. Possibly because card is jcdk 3.0.4 compatible, compiled using jcdk 3.0.5. Appears to be card-defined execution error.

5/11 Decided to give the Python library pyscard a try instead of OCF. Spent the day running into various problems to do with dependencies it couldn’t find. Turns out plugin only for python 2, but I was trying python3. Working now, but unable to establish context. Not sure what the problem was, but it appears to be that particular version (1.7.0), because it worked when I tried a different version. With it, I was able to successfully test a small program that sends a SELECT APDU to the ISD and prints the sw1-sw2 output.

The pyscard Python extension is likely to be my tool of choice going forward. It’s compatible with both Microsoft PC/SC and Linux PC/SC lite, so the code will be portable and relatively easy to write.

7/11 Followed through applet compiling process for JCDK 2.2.2. Problem running first demo with jcwde, gives message “card was unexpected at this time”. No idea what it means. TODO: Try another demo. Not a serious problem though, it’s the simulation test suite so it’s optional.

Compiled sample code to class file, attempted to convert to CAP file. Converter has message “card was unexpected at this time”. Wtf?

Identified. It doesn’t like spaces in the JC\_HOME environment variable. Had to change directory structure.

New problem: Script also can’t deal with space in path of JAVA\_HOME. Scripts involve something like %JAVA\_HOME%\bin\java -classpath %\_CLASSES% com.sun.javacard.converter.Converter %\*, where they should have quotes around the environment variable. Error on distributor’s part. Have to reinstall jdk into a different directory tomorrow.

08/11 Was able to solve the problems and compile and convert an applet, and store/remove it from the card. Used JCDK HelloWorld sample applet source file.

09/11 Looked into different asymmetric cryptography protocols.

11/11 Configured Atom with Python and Java IDEs so I could develop both ends side-by-side. Adapted the build process to work with my project structure so I could upload my own apps, not just sample apps. Was able to successfully select my applet on the card and send a message, the applet checked its CLA and INS values, and returned them. Now have a better idea of how applet selection works.

Wrote a small script to automate the process of compiling source code, converting to a .cap file, uploading it onto the card, and running it with a test host application.

12/11 Wrote a test application that takes a byte string via an apdu reading “Hello World”, storing it, and returning the string upon a later apdu request.

14/11 Read some standards on protocols, decided to implement OPACITY protocol. Researched various components e.g. EC cryptography, Diffie-Hellman etc. and read parts of NIST 800-56A which was referenced by OPACITY protocol.

15/11 More reading of standards. Started writing host-side skeleton code to plan the structure of the app, researching Python crypto plugins (using ecdsa plugin). Researched Basic Encoding Rules defined in ASN.1, rules for encoding data types for serialisation, necessary because opacity CVC values sent between card and host are in this format.

18/11 Implemented various helper functions for host app, including cv\_extract, hash, truncate8, verify\_mac, extract\_fields. Researched Basic Encoding Rules (BER) to understand the structure of CVC values send in the protocol. Learned more about Elliptic Curve Diffie-Hellman, experimented with python ‘cryptography’ library, much more extensive than ecdsa and other crypto libraries.

22/11 Implemented various auxiliary functions for card applet including hash, truncate, get\_secret, kdf, and wrote mostly-complete code up to the ‘Zeroize Z’ stage of the card protocol.

23/11 Discovered no CMAC function exists on Java Card 2.2.2 platform. Only implemented on Java Card 3.x. Had to write my own implementation of CMAC in order to stick to the protocol. Wrote about 100 lines to this end, AESCMAC128.java.

25/11 Finished writing CMAC function. Got a little confused on a couple of points so referenced another implementation <https://pastebin.com/JQ9xQ5vK>. Used this to write the rest of the code in the authenticate function on the card, covering all points in the protocol. Still things to be done, including how to initialise with a CVC, properly parsing the format of EC keys.

26/11 Made modifications to CMAC class. Fixed some errors in main opacity implementation, and cleaned up so it uses fewer temporary buffer arrays. Wrote basic host-side code for key issuing system. Decided the card will generate its keypair and send the public key to the issuing system, which formats CVC and sends it to the card in a separate APDU. Created new compile/install scripts.

Attempted to compile. Various errors relating to uses of, and indirect casts to, int instead of short. Fixed those. Successfully compiles and converts. GPShell install command returns 6A80 (wrong data / incorrect values in command data) Will fix tomorrow.

30/11 Fixed. Problem was likely due to install method not calling register. Kp.genKeyPair() not working, throws (6F, 00). Turns out it was because KeyPair constructor can’t take arbitrary key lengths, only the constants in KeyBuilder. In JC 2.2.2 they only go to 192b, but protocol requires 256b. Instead, have to separately initialise public and private keys using NIST EC parameters and use the other KeyPair constructor. After adding it in, found it didn’t accept the compressed G, so had to enter the full uncompressed version. Fixed this, now generated keypair without error. Had trouble with getW() command to obtain public key. After a while figured out 64B buffer not big enough. Need 65B to accommodate extra 0x04 at the beginning.

Am now reasonably confident key issuing process works. Will move on to debugging authentication process.

3/12 Fixed usage of hash function and other errors on the authentication code. Removed unnecessary array conversion functions and cleaned up code. Implemented various uncompleted functions on host side including ec\_dh, kdf.

Realised that assumption behind signature was wrong. Should be calculated by the card and sent to host. Adapted issuing code to account for this.