Goals

The goal of this project is to implement a secure channel for communications between a mobile device and a server hosted in a cloud. For this task, the client side communications are implemented on an Android device emulated on a windows 7 PC. The Cloud is hosted by the Amazon Web Services, which is simply a small linux virtual qmachine. The goal is to use Diffie-Hellman Key Exchange to create a session key between the cloud and the android to encrypt communications. Additionally, RSA asymmetric keys are used by the cloud and the android to verify the identity of each entity. Finally, a picture password system is created which will allow a user to authenticate himself / herself from the android device. Thus, we achieve secure communications between the android and the cloud, and all three entities are authenticated (cloud, android, user).

Environment Setup

Initial cloud setup is completed according to the instructions listed in [2]. The Android emulator and Eclipse environment can be set up according to [3]. Note that the Linux Virtual Machine set up in the cloud must be running the same version of java as the system you will be doing development on. (1.7 is used for this project).

Cloud

As stated the directions listed in [2] describe how to set up an Amazon Web Services (AWS) account, and how to start a linux virtual machine in the cloud. You can sign up and create an account at [4]. Once you have an account, you will want to use the EC2 service to host your linux image. You can find images at [5]. You will create an RSA key to login to this linux account (rather than using a username/password combination). This is a secure way to login as long as you don't lose your key, or let someone else gain access to it. The web address for your server can be found in the AWS gui (web page found once you log in to your AWS account). You can login using a java based ssh client which is provided at the AWS Console. You can also use cygwin to login and interact with your Linux instance. (You will have to download the appropriate packages for cygwin to work).

To login:

```
ssh -i YOURKEY.pem ubuntu@ec2-107-21-134-148.compute-1.amazonaws.com
```

Note that everything after the "@" should be the address for your instance.

I created a bin directory on the server in the home folder to store .class files for my project. To transfer the files from my PC to the cloud I used the following command:

```
scp -i ../YOURKEY.pem *.class ubuntu@ec2-107-21-134-148.compute-1.amazonaws.com:bin
```

Again everything after the "@" should be the address for your server. The :bin indicates to place it in the bin directory. (I would transfer files from a local "bin" directory, that did not have my key. The "../" indicates that the key file can be found one directory up from my current location). Once files are uploaded to the server, you can run the server by typing `java DHServer`. I personally used two separate cygwin windows open; one for using ssh and server commands, and another window for uploading files as I compiled new .class files.

You will need to set up some of the files needed for cryptographic support on the server. Details for this setup can be found in [1]. They will be very similar to the setup required for the windows

system as well (or whichever environment you choose to do development on). The cloud in our case is only used to host the java based server, no direct development is done in the cloud.

Android

Follow the instructions listed at [3] for how to setup Eclipse and the Android emulator. This is fairly straight forward. Once you have the emulator setup, you will need to follow the directions outlined in [1] to set up the Bouncy Castle provider. (Same as on the cloud). This will allow you to use the bouncy castle provider for encryption and decryption. There are several test files given in [1] that will allow you to verify that you have set up the provider correctly. (You can also run these test files on the cloud). Additionally, there are several sample projects and tutorials at [3] that you can use to verify the emulator is working properly. As a framework for this project, I used one of the projects listed at [6] as a starting point to begin the Android development for this project. (There is currently a lot of extra code in my project from [6] that is not being used, but just hasn't been deleted yet).

Source Code Information

There are two main areas of code development, android and cloud. Within each of these there are several classes and files that are important.

Android Code

The Android code is contained in two source files: Utils.java and DHClientAliceActivity.java. The Utils.java file is shared in common with the server, and it simply gives some utility functions for converting strings to hex format and hex arrays to a string. Also, there is a method which will create the same “random” number each time for testing purposes. The DHClientAliceActivity.java file is where all the UI manipulation takes place, in addition to cryptography, and network communications.

Currently, when the android code runs initially, the user is presented with several selections (main.xml). The only one with meaning for this project is the bottom choice “Create Channel”. This brings up the “dhexchange.xml” page. On this page the top button (“Create Secure Channel”) will trigger communications with the cloud. The server must be running in the cloud before this button is clicked. This will trigger the creation of a secure session key between the client and the server, and populate the text fields on this page with important values. Once the channel has been created, you can then press the “Enter Password” button at the bottom to continue to the picture password page (password.xml).

On the picture password page, 7 images need to be clicked in order to submit your password. (Currently there is no error checking to make sure that 7 images have been selected). Once 7 pictures have been chosen, the “Submit Password” button is clicked in order to encrypt the selection, and transmit it to the server. At this point client interaction is finished. However, if you want to reset the process without restarting the emulator, simply click on “Reset”, and you will be taken back to the “dhexchange.xml” page, where you can again create a secure channel, and then enter a password. (Note that in reality the client would need to receive from the server if its been authenticated, and then follow up with access to server resources).

The main thread on the android can not block (i.e. connect to sockets, etc). So network communications are done using the AsyncTask class. We extend this class in two different ways to create the “DHExchangeTask”, and the “PasswordVerifyTask”. The DHExchange task is triggered by the “Create Secure Channel” button referenced above. The task creates all the needed cryptographic support, and network support, and performs communications with the server.

The “PasswordVerifyTask” is triggered by the “Submit Password” button referenced above. This task takes the password supplied by the user, encrypts it, and submits it to the server. Ideally, we would then check to see if access has been granted, and then access the server resources. However, this is where the code stops, and the server will simply display a message if the user has been authenticated or not.

Server Code

The server has several files that are important for implementation. The following files were downloaded from [7] in order to implement ReedSolomon encoding and decoding:

GenericGF.java – a file that implements a generic finite field (or Gallois Field). It seems this only implements QR (256) fields at this time.

GenericGFPoly.java – a file that implements polynomials and operations over a given finite field.

ReedSolomonException.java – a file that defines a custom exception for this package of files.

ReedSolomonEncoder.java – a file that inputs a set of coefficients, and adds error correcting coefficients to the original input.

ReedSolomonDecoder.java – a file that inputs a set of coefficients, and a degree parameter, that attempts to reconstruct the original set of coefficients that would have been output by the Encoder.

The files listed above were used “as is”, with modifications done only to print test messages.

The Utils.java file is the same file as used by the Android, provided in [1].

The FuzzyVault.java file is a start of an implementation for a fuzzy vault scheme. It is not complete, as it does not have a ReedSolomonDecoder that is of the correct type to work with the fuzzy vault scheme proposed in [8]. This class also implements our fuzzy password. The server currently is just set to use one password, which is hard coded for testing. This is done by a call to the EnrollUser(int []) method. This stores the password information in the member variables of the class. Our password is 23, 37, 12, 17, 88, 67, 5. The method LoginUser inputs an array of integers, and tests to see if it matches the password that was enrolled. (the input is first concatenated with error correcting codes computed in the enrollment process, and then these values are input to the ReedSolom Decoder). If there is a match the server will print that access is granted. Otherwise, access will be denied. (This answer should be encrypted and sent to the client, but this is not yet implemented). The Lock() and Unlock() methods match with the algorithms described in [8], but they do not quite work correctly, due to the lack of a special purpose ReedSolomon Decoder at this time.

The file DHServer.java handles all the logic and network communications for the server. Initially, cryptographic objects are created and initialized using the InitRSAKeys() method and InitDHValues() method. (note that the RSA keys are hard coded into both server and client). The public values g and p are also hard coded into both the client and server. Finally, the DroidComm() method is called in order to wait for a connection from a client, and perform the necessary handshake to create a session key between the android and the cloud.

Reference Material

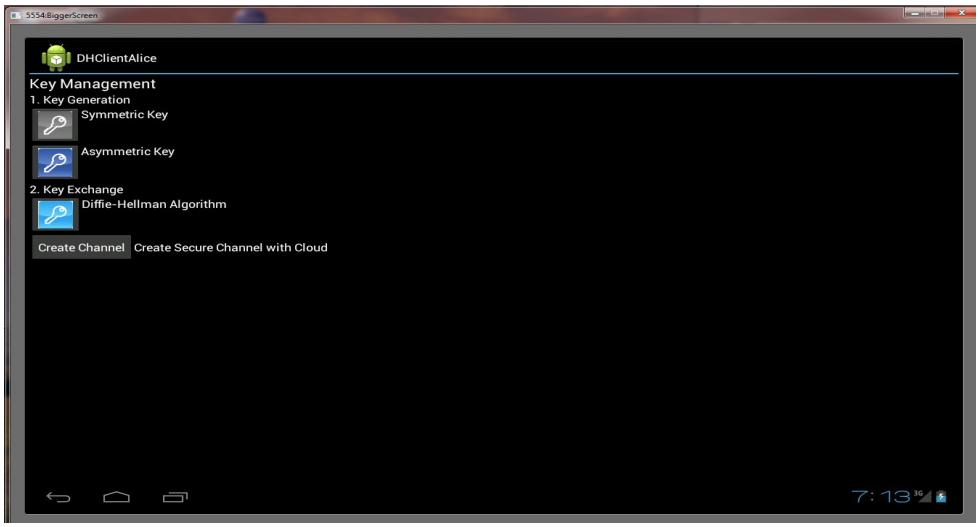
1) http://www.wrox.com/WileyCDA/WroxTitle/Beginning-Cryptography-with-Java.productCd-0764596330.html

2) http://www.wrox.com/WileyCDA/WroxTitle/Beginning-Mobile-Application-Development-in-the-Cloud.productCd-1118034694.html

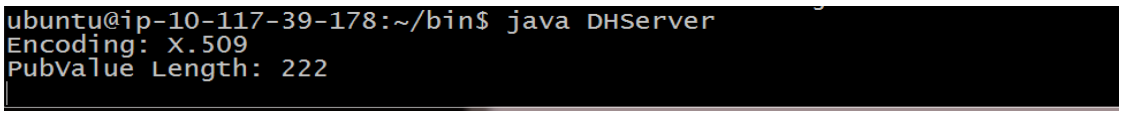
3) http://developer.android.com/sdk/index.html
4) http://aws.amazon.com/
5) http://alestic.com/
6) https://sites.google.com/site/sfsmobilesecuritycourseware/modules
7) http://code.google.com/p/zxing/source/browse/trunk/core/src/com/google/zxing/common/reedsolomon/?r=2281
8) Juels A., and Sudan, M. 2002. A Fuzzy Vault Scheme. In *International Symposium on Information Theory.*

Screen shots

Click on “Create Channel”



Start Server in cloud

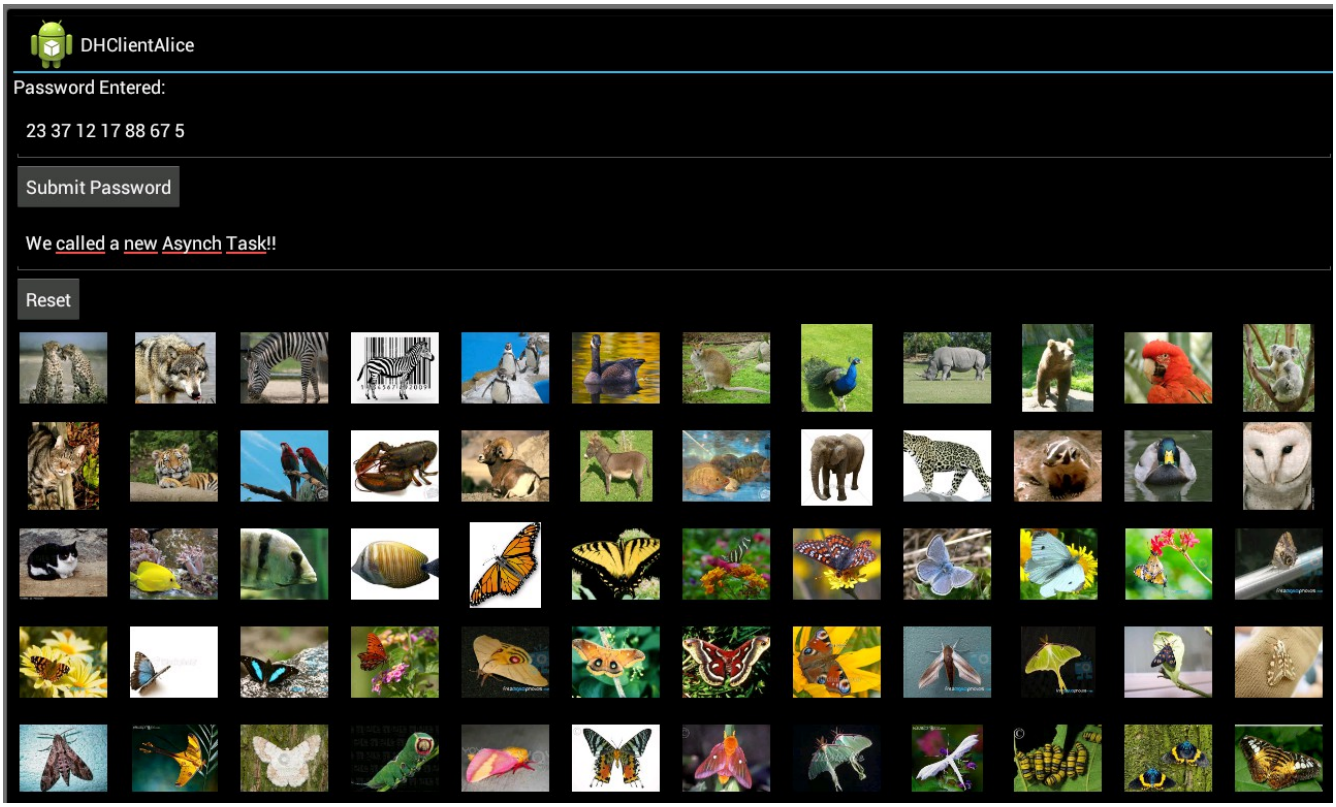


Click on “Create Secure Channel” … Then click “Enter Password” to advance

See the session key is the same as on the droid show in image above.

```
ubuntu@ip-10-117-39-178:~/bin$ java DHServer
Encoding: X.509
PubValue Length: 222
Client Connection Accepted
Calculated DH Session Key: cbd2c382aaae12809d84bec23c604f8d2ebef6f813b
de7f1d6a5b58632070cb0
```

Select images to make up your password (Selections displayed at top). Then click “Submit Password”. To start over you can click reset to go to the previous page.



Here the server shows verified, with our password.