Unlock PC with NFC Enabled Android Device

Edgar Villarreal
California Polytechnic State University
Advisor: Dr. Zachary Peterson

June 5, 2014

Abstract

Passwords have a fundamental flaw, ironically, its people that create the biggest issue. With a growing interest in NFC technology, my project aims to use NFC technology to facilitate the user login process to a computer. The project is composed of two approaches that attempt to solve the issue. The first method uses an Arduino device with an NFC reader shield and an Android device to login to any OS. The second approach uses a USB NFC reader and a smart card to login to a computer. This approach makes use of Linux PAM, a service available only on Linux systems.

Table of Contents

List of Figures and Tables 3
1 Introduction 4
2 Background 4
3 Specifications 5
    3.1 NFC Specifications 5
    3.2 Smart Card Specifications 6
4 Arduino Approach 6
    4.1 Hardware 6
        4.1.1 Part List 7
        4.1.2 Setup 7
    4.2 Software 8
        4.2.1 Requirements 8
        4.2.2 Sequence Diagram 8
    4.3 Implementation 9
        4.3.1 Arduino Side 9
        4.3.2 Android Side 9
5 NFC Reader Approach 9
    5.1 Hardware 10
        5.1.1 Part List 10
    5.2 Software 10
        5.2.1 Requirements 10
        5.2.2 Sequence Diagram 11
    5.3 Implementation 11
        5.3.1 Configure the Smart Card Reader 11
        5.3.2 Create Private Key 12
        5.3.3 Create Public Key 12
        5.3.4 Store Public Certificate on Smart Card 12
        5.3.5 Configure Smart Card for Login 13
        5.3.6 Test with Sudo 13
        5.3.7 Login with Smart Card 14
6 Conclusion 14
References 16
Source Code 17

List of Figures and Tables

Figure 1: Cutting Trace Between IRQ Pin and Pin 2 7
Figure 2: Connecting IRQ Pin with Pin 6 7
Figure 3: Sequence Diagram for Arduino Leonardo 8
Figure 4: ACR122U NFC Reader 10
Figure 5: Sequence Diagram for NFC Shield 11

Table 1: NFC Technical Details 5
Table 2: APDU Command Structure 6
Table 3: APDU Response Structure 6

1 Introduction

Passwords have been around for a few decades and have significantly improved since their creation. Unfortunately, even in modern techniques and advances in secret key and public key cryptosystems, strong password authentication is still a problem today. Social engineering poses a large threat since it negates all cryptosystems. Even more detrimental is the fact that people will generally try to use passwords that are easy to remember and as a result will create weak passwords that can be prone to a brute force attack. This is where my first approach fits in; by storing the password on an Android device, the complexity of a password can be significantly increase. This will eliminate the need for a user to remember complex and long passwords. More importantly, it's faster and always accurate, which is always an important factor for a user.

In modern security we are starting to lean towards multi factor authentication through security tokens or other techniques. We also make use of smart card technology to handle authentication and it's even becoming a point of interest for use with debit and credit cards. My second approach for this project implements a system that uses a smart card to login to a Linux system using public key cryptography. For this project I used Ubuntu 14.04 LTS and Linux PAM 1.0.1-6. The project originally was meant to also use Android and emulate a smart card, but due to limited time, the project was simplified to make use of a regular smart card instead.

2 Background

Smart cards are basically cards with an embedded integrated circuit. The main advantage of a smart card is that they can provide authentication, identification, process and store data. Smart cards can also provide strong security authentication for single sign-on. They are popular as hotel keys, SIM cards, debit cards and even as a form of identification [1].

Near field communication (NFC) technology enables simple and safe two-way interactions between electronic devices, allowing consumers to perform contactless transactions, access digital content, and connect electronic devices with a single touch [2]. NFC is a set of standards for smartphones and similar devices to establish radio communication with each other by touching them together or bringing them into close proximity, usually no more than a few centimeters [3]. Current applications include contactless transactions, data exchange and simplified setup of more complex communications such as Wi-Fi.

Many Android-powered devices that have NFC functionality support NFC card emulation. In most cases, the card is emulated by a separate chip in the device, called a secure element. The SIM

card is the secure element and can only be modified by wireless carriers. The problem with this is that developers can't modify the secure element making it useless for apps and negatively impacting the growth of NFC adoption. Android 4.4 introduces an additional method of card emulation that tdoes not involve a secure element, called host-based card emulation [4].

Host-based card emulation (HCE) is essentially an app on an Android device that emulates and NFC card. The main advantage of this is that the data is routed to the host CPU, instead of the secure element. Another helpful feature of HCE is that it is based around an Android Service which allows the card emulation to occur in the background, without interrupting user activities. The minimum requirement for the service to run is to have the screen on, even if the screen is locked. Unfortunately, there is a glaring issue with HCE and it's that the developer is in charge of handling security. Another concern are rooted devices, which essentially remove most of the protections for keeping the data secure, but this is a whole other issue.

3 Specifications

This section will cover technical details about NFC and smart card message structures.

3.1 NFC Specifications

NFC operates at relatively low speeds, especially when compared to other communication technologies, such as Bluetooth. One important fact is that, as seen in **Table 1**, the range of NFC is noticeably short, in fact some devices will only trigger when bumped together. However, battery consumption is also significantly smaller and doesn't require any sort of pairing like other communication standards. In Android, when using HCE mode, the NFC device consumes almost no additional energy; the service that performs the NFC task usually terminates once it completes its task.

	NFC [6]
Network Standard	ISO 13157 etc.
Network Type	Point-to-point
Cryptography	none
Range	< 0.2 m
Frequency	13.56 MHz
Bit rate	424 kbit/s
Set-up time	< 0.1 s
Power consumption	< 15mA (read)

Table 1: NFC Technical Details

3.2 Smart Card Specifications

In Android 4.4, an API was added to support host-based card emulation. The messages required for HCE as called APDU and the structure of an APDU is defined by ISO/IEC 7816-4. A "Command APDU" is sent by the NFC reader to the card and it contains the mandatory 4-bytes (CLA, INS, P1, P2) and an optional 0-255 byte payload (Command data, L_e) as seen in **Table 2** [7].

Field name	Length (bytes)	Description
CLA	1	Indicates the type of command
INS	1	Indicates the specific command
P1, P2	2	Instruction parameters for the command
L_c	0, 1 or 3	Number of bytes in the payload
Command Data	N_C	Command data
L_e	0, 1, 2 or 3	Encodes N_e bytes in response message

Table 2: APDU Command Structure

The smart card will receive the message and if it matches what it is expecting, it will respond with a "Response APDU" to the reader. A response APDU contains a mandatory 2-byte header (SW1, SW2) and a optional 0-65,636 payload (N_r) as seen in **Table 3**.

Field name	Length (bytes)	Description
Response data	N_r (at most N_e)	Response data
SW1-SW2	2	Command processing status

Table 3: APDU Response Structure

4 Arduino Approach

This section will discuss the approach that uses the Arduino device along with an NFC shield to read a smart card. It will cover the hardware and software used, as well as the implementation.

4.1 Hardware

This section will offer details about the hardware required and some issues/solutions that might arise when using this specific hardware.

4.1.1 Part List

1. Arduino Leonardo
2. Adafruit PN532 NFC/RFID Controller Shield
3. NFC enabled Android device (with API 4.4+)
4. Small wire

4.1.2 Setup

The Adafruit NFC shield is designed to be used with I2C and is drop-in compatible with any Classic Arduino. The NFC shield has the IRQ pin tied to pin 2 by default, unfortunately the Leonardo board uses pin 2 for I2C [8]. This kind of communication mismatch will not allow both devices to communicate with each other. In order to remedy this, you have to physically cut the trace on the NFC shield between the IRQ pin and pin 2, as seen in **Figure 1**.

Figure 1: Cutting Trace Between IRQ Pin and Pin 2

Then a wire needs to be soldered from the IRQ pin to pin 4 or any other pin after 4, as seen in **Figure 2**. Cutting the trace will cause permanent damage to the board and in order to revert the damage, the only option will be to connect the IRQ pin and pin 2, rather than pin 4.

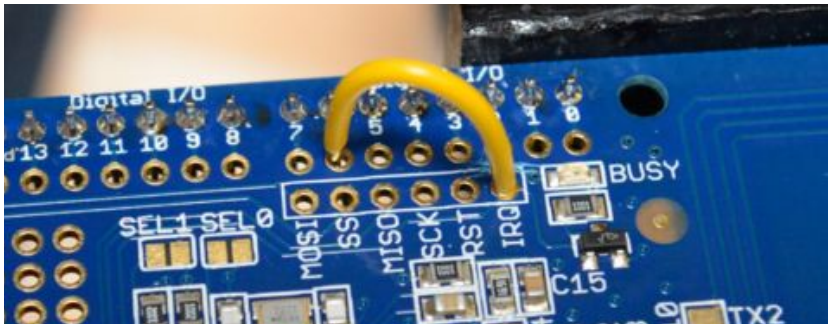

Figure 2: Connecting IRQ Pin with Pin 6

There are other methods that can be used to avoid damage to the shield. One method is by removing the header for pin 2, that connects the NFC shield to the Arduino board. Then connecting the IRQ pin with any pin after 4. The reason this method is better, is because it might be important to preserve the physical connection if the shield will be used with other Arduino devices.

4.2 Software

This section will offer details about the software required, including libraries and software used to develop. The implementation method is explained in detail and a sequence diagram is provided to illustrate the implementation.

4.2.1 Requirements

1. Adafruit NFC Shield I2C library [9]
2. Arduino IDE [10]
3. Android development IDE + Android SDK

4.2.2 Sequence Diagram

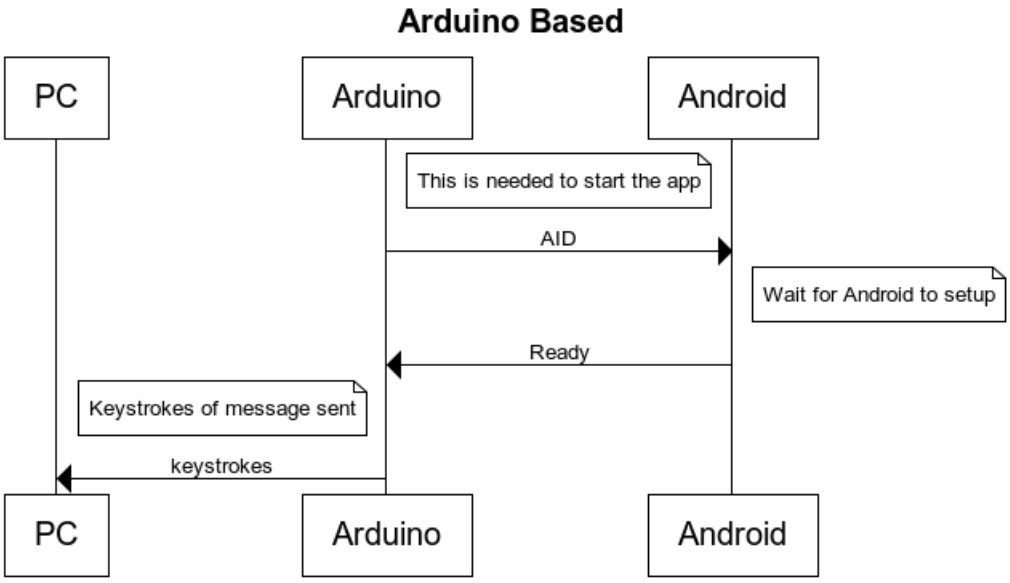

Figure 3: Sequence Diagram for Arduino Leonardo

4.3 Implementation

This section largely focuses on the content of the messages used to communicate between devices. The actual process used to login to the system is also discussed.

4.3.1 Arduino Side

The Arduino Leonardo device will start by sending a specially formatted message, Command APDU. The first byte, 0x00, indicates that this message is a Command message with no security. The second byte, 0xA4, indicates that the message is a Select instruction. The next two bytes, 0x04, 0x00, don't serve any purpose, they can be anything. The next byte is the length of the payload, which in this case is 0x05. The payload has to be the Application ID (AID) which can be up to 16 bytes [11].

message structure - 0x00, 0xA4, 0x04, 0x00, 0x05, 0xF2, 0x22, 0x22, 0x22, 0x22

When the Arduino receives the response from the Android device it parses out the password. The Arduino Leonardo allows for serial communication over USB and appears as a virtual com port. This feature makes the Leonardo appear as a generic keyboard, and it can send the password through virtual keystrokes.

4.3.2 Android Side

When the Android device receives the message, the OS (Android) will search for the app that has the specified AID. If the OS finds the app, then it starts the service and it runs it in the background. If Android doesn't find the app with the corresponding AID, nothing will happen, it won't even send a message to the reader indicating that it is not found. The service is free to do whatever it was programmed to do. For my project the service will create a Response APDU with the password in its payload. My app allows the password to be changed and is protected by a pin that the user can set. The pin itself can also be changed. Both values are stored in SharedPreferences in private mode. Which means only the app can see these values; assuming a non-rooted device.

5 NFC Reader Approach

This section will discuss the approach that uses the NFC Reader to read a smart card. It will cover the hardware and software used, as well as the implementation.

5.1 Hardware

This section will cover the hardware used for this method.

5.1.1 Part List

1. ACR122U USB NFC Reader
2. Rewritable smart card

Figure 4: ACR122U NFC Reader

5.2 Software

This section will offer details about the software required, including libraries and software used to develop. The implementation method is explained in detail and a sequence diagram is provided to illustrate the implementation.

5.2.1 Requirements

1. Smart card
2. Linux side applications/middleware:
    a. pcscd and pcsc-tools
    b. libssl-dev
    c. libpcsclite-dev
    d. opensc
    e. libpam-pkcs11
    f. pkg-config
    g. libengine-pkcs11-openssl

5.2.2 Sequence Diagram

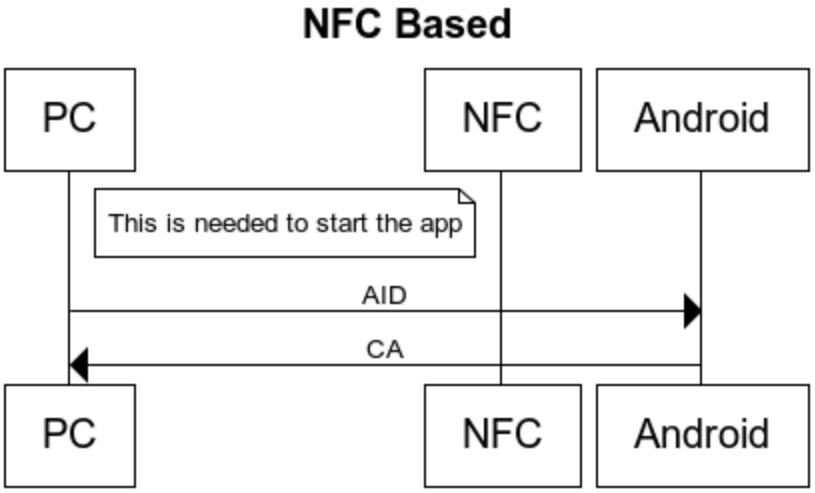

Figure 5: Sequence Diagram for NFC Shield

5.3 Implementation

This section explains the steps that I took to implement the NFC reader approach that I took in my project.

5.3.1 Configure the Smart Card Reader

Implementing this part can be significantly harder depending on your luck. Also I used a smart card for this, not HCE. Since there are so many steps and so many different readers, even clones, Once all the software is installed you need to confirm that the smart card reader is detected. If a smart card is presented to the reader, the card should be detected and the information should be displayed.

```
pcsc_scan
```

The next step is to have opensc detect the card reader.

```
opensc-tool --list-readers
```

Now you need to make sure that opensc can read the card. If anything fails at this point then either the device drivers or one of the applications from the list is missing.

```
opensc-explorer
```

5.3.2 Create Private Key

Assuming everything went well, the next step is to create a private key.

```
pkcs15-init --create-pkcs15
```

Now before you are able to install private keys you need to have at least one pin to protect the objects.

```
pkcs15-init --store-pin --auth-id 01 --label "Your Label"
```

Now generate a key, I used a 1024 bit RSA key.

```
pkcs15-init --generate-key rsa/1024 --auth-id 01
```

Check everything went well by checking the key.

```
pkcs15-tool --list-keys
```

5.3.3 Create Public Key

No create a public key using openssl.

```
openssl

engine dynamic -pre SO_PATH:/usr/lib/engines/engine_pkcs11.so
-pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD -pre
MODULE_PATH:opensc-pkcs11.so

req -engine pkcs11 -new -key id_45 -keyform engine -x509 -out
cert.pem -text
```

Now check the public key was created normally.

```
openssl verify -CAfile cert.pem cert.pem
```

5.3.4 Store Public Certificate on Smart Card

The final step is to store the public certificate on the card.

```
pkcs15-init --store-certificate cert.pem --auth-id 01 --id 45
--format pem
```

5.3.5 Configure Smart Card for Login

Now you need to configure the smart card for login. Present the smart card to the reader and check the certificates installed in the smart card.

```
pkcs11_listcerts
```

Now configure libpam-pkcs11.

```
mkdir /etc/pam_pkcs11 /etc/pam_pkcs11/cacerts
/etc/pam_pkcs11/crls

zcat
/usr/share/doc/libpam-pkcs11/examples/pam_pkcs11.conf.example.gz
| sudo tee /etc/pam_pkcs11/pam_pkcs11.conf
```

Now modify the file "/etc/pam_pkcs11/pam_pkcs11.conf"

```
use_mappers = digest, cn, pwent, uid, mail, subject, null;
     →   use_mappers = pwent;
```

Now add the signed certificate to "/etc/pan_pkcs11/cacerts/" and rehash the list of CA certificates.

```
cd /etc/pam_pkcs11/cacerts

sudo pkcs11_make_hash_link
```

5.3.6 Test with Sudo

Finally test that it all works by modifying the file that handles the sudo authentication, "/etc/pam.d/sudo".

```
#%PAM-1.0

auth sufficient pam_pkcs11.so

@include common-auth
@include common-account
```

```
session required pam_permit.so
session required pam_limits.so
```

Test to make sure that it works.

```
sudo -i
```

If it fails revert the "/etc/pam.d/sudo" file and check all the steps. Even if it works it would be best to revert sudo to its original authentication.

5.3.7 Login with Smart Card

Finally, add the following line right after the block of comments in the file "/etc/pam.d/common-auth". This line will allow login with a smart card, but if not present then it will fall back to the default password process.

```
auth       [success=2 default=ignore]      pam_pkcs11.so
```

6 Conclusion

The Arduino approach did make entering the password much easier, but it has a glaring security vulnerability; the password is sent in cleartext. An alternative would be to encrypt the message, unfortunately Arduino struggles performing complex algorithms. The more complex encryption algorithms can take a few minutes to decrypt a message, which make them unreasonable to use. Even with simpler algorithms, another issue is that Arduino memory can easily be dumped, revealing the encryption key. There are methods to prevent an Arduino from being read, but the extent that the Arduino device needs to be modified, will permanently leave the device inoperable for any other use. This means that once the key becomes compromised it will make the Arduino device useless.

Public Key Infrastructure (PKI) is obviously a better choice and an Arduino device should be able to handle the functionality with reasonable speeds. Unfortunately, I was advised that implementing a PKI algorithm would be too complex for the scope of this project; even worse I would probably implement it poorly. In the end I decided that their was no reasonable method to securely transmit the password. As a result, this method turned into more of a convenience method that will help use large complex passwords, but not very securely.

My second approach makes use of 512 RSA encryption and it's very fast. Unfortunately, this method was originally meant to be used with HCE, but I had trouble getting it to work. The biggest

issue is that HCE doesn't allow direct card reads like a typical smart card. There is a way around this, but it would require manually sending the authentication certificate, which is too tedious and it is something that I did not want to pursue. Linux PAM can be modified to support HCE, but again the amount of work to implement that is out of the scope of this project.

References

[1] Hendry, Mike. Multi-application Smart Cards. 1st ed. *Cambridge: Cambridge University Press*, 2007. Cambridge Books Online. Web. 06 June 2014. http://dx.doi.org/10.1017/CBO9780511536694

[2] "About the Technology." | *NFC Forum*. N.p., n.d. Web. 05 June 2014.

[3] "About NFC." *CISTEMS*. N.p., n.d. Web. 05 June 2014.

[4] "Host-based Card Emulation." *Android Developers*. N.p., n.d. Web. 05 June 2014.

[5] "Near Field Communication." *Wikipedia*. Wikimedia Foundation, 06 Apr. 2014. Web. 05 June 2014.

[6] "Near Field Communication Technology Standards." – *NearFieldCommunication.org*. N.p., n.d. Web. 05 June 2014.

[7] "Home." *ISO/IEC 7816-4:2005*. N.p., n.d. Web. 05 June 2014.

[8] "Adafruit PN532 RFID/NFC Breakout and Shield." *Shield Wiring*. N.p., n.d. Web. 05 June 2014.

[9] "Adafruit/Adafruit_NFCShield_I2C." *GitHub*. N.p., n.d. Web. 05 June 2014.

[10] "Arduino - Software." *Arduino - Software*. N.p., n.d. Web. 05 June 2014.

[11] "5. Basic Organizations." *ISO 7816-4 (ISO7816 Part 4 Section 5) Smart Card Standard, Basic Organizations*. N.p., n.d. Web. 05 June 2014.

Source Code

```
#include <Wire.h>
#include <Adafruit_NFCShield_I2C.h>

#define IRQ 8 // this trace must be cut and rewired!
#define RESET 3

Adafruit_NFCShield_I2C nfc(IRQ, RESET);
uint32_t oldCard = 0;

// This program will connect the NFC shield to the Arduino device.
// It also sends the appropriate AID to the Android device in order
// to start an NFC communication.

void setup() {
   // set up Serial library at 115200 bps
   Serial.begin(115200);
   nfc.begin();

   while (!Serial) {
      ; // wait for serial port to connect. Needed for Leonardo only
   }

   // find Adafruit RFID/NFC shield
   printChipInfo(nfc.getFirmwareVersion());

   // configure board to read RFID tags
   nfc.SAMConfig();

   //initiate the Keyboard
   Keyboard.begin();
}

void printChipInfo(uint32_t versionData) {
   Serial.println("Looking for PN532...");

   if (!versionData) {
      Serial.println("Didn't find PN53x board");
      while (1)
         ; // halt
   }

   Serial.print("Found PN5");
   Serial.println((versionData >> 24) & 0xFF, HEX);
   Serial.print("Firmware ver. ");
```

```
   Serial.print((versionData >> 16) & 0xFF, DEC);
   Serial.print('.');
   Serial.println((versionData >> 8) & 0xFF, DEC);
}

void loop() {
   bool success;
   uint8_t responseLength = 32;

   // set shield to inListPassiveTarget
   success = nfc.inListPassiveTarget();

   if (success) {

      Serial.println("Found something!");

      uint8_t selectApdu[] = {0x00, /* CLA */
      0xA4, /* INS */
      0x04, /* P1  */
      0x00, /* P2  */
      0x05, /* Length of AID  */
      0xF2, 0x22, 0x22, 0x22, 0x22 /* AID defined on Android App */};
      uint8_t response[32];

      success = nfc.inDataExchange(selectApdu, sizeof(selectApdu),
response,
            &responseLength);

      if (success && responseLength > 2) {
         Keyboard.println(hexToString(response, responseLength));
      }
      else {
         Serial.println("Failed sending SELECT AID");
      }
   }

   delay(1000);
}

// Convert the response from hex to a string message
String hexToString(uint8_t *response, uint8_t responseLength) {

   String respBuffer;

   for (int i = 0; i < responseLength; i++) {

      if (response[i] >= 0x20 && response[i] <= 0x7E)
         respBuffer = respBuffer + String((char) response[i]);
```

```
    }

    return respBuffer;
}
```

```
package com.eviljack.nfcunlock;

import java.util.Arrays;

import android.content.Context;
import android.content.SharedPreferences;
import android.nfc.cardemulation.HostApduService;
import android.os.Bundle;

/**
 * This is the APDU Service which supports card emulation. Card
emulation only
 * provides a byte-array based communication channel.
 */

public class CardService extends HostApduService {
   // AID for the card service.
   private static final String CARD_AID = "F222222222";
   // Format: [Class | Instruction | Parameter 1 | Parameter 2]
   private static final String COMMAND_APDU_HEADER = "00A40400";
   // "OK" status sent in response (0x9000)
   private static final byte[] OK_SW = HexStringToByteArray("9000");
   // "UNKNOWN" status sent in response (0x0000)
   private static final byte[] BAD_SW = HexStringToByteArray("0000");
   private static final byte[] SELECT_APDU =
BuildSelectApdu(CARD_AID);
   final static char[] hexArray = {'0', '1', '2', '3', '4', '5', '6',
'7', '8',
         '9', 'A', 'B', 'C', 'D', 'E', 'F'};

   /**
    * This method will be called when a command APDU has been
received from a
    * remote device.
    *
    * @param commandApdu
    *           The APDU received from the remote device
    * @param extras
    *           A bundle containing extra data. May be null.
    * @return a byte-array containing the response APDU. May be null.
    */
   @Override
   public byte[] processCommandApdu(byte[] commandApdu, Bundle
extras) {
      if (Arrays.equals(SELECT_APDU, commandApdu)) {
         SharedPreferences secure = getApplicationContext()
               .getSharedPreferences("SECURE", Context.MODE_PRIVATE);
         String account = secure.getString("password", "pass");
```

```
            byte[] accountBytes = account.getBytes();

            return ConcatArrays(accountBytes, OK_SW);
        }
        else {
            return BAD_SW;
        }
    }

    public static byte[] BuildSelectApdu(String aid) {
        return HexStringToByteArray(COMMAND_APDU_HEADER
                + String.format("%02X", aid.length() / 2) + aid);
    }

    public static String ByteArrayToHexString(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];

        for (int i = 0, v; i < bytes.length; i++) {
            v = bytes[i] & 0xFF;
            hexChars[i * 2] = hexArray[v >>> 4];
            hexChars[i * 2 + 1] = hexArray[v & 0x0F];
        }

        return new String(hexChars);
    }

    public static byte[] HexStringToByteArray(String s)
            throws IllegalArgumentException {
        int len = s.length();
        byte[] data;

        if (len % 2 == 1) {
            throw new IllegalArgumentException(
                    "Hex string must have even number of characters");
        }

        data = new byte[len / 2];

        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) <<
4) + Character
                    .digit(s.charAt(i + 1), 16));
        }

        return data;
    }

    public static byte[] ConcatArrays(byte[] first, byte[]... rest) {
```

```
        int totalLength = first.length;
        byte[] result;
        int offset;

        for (byte[] array : rest) {
            totalLength += array.length;
        }

        result = Arrays.copyOf(first, totalLength);
        offset = first.length;

        for (byte[] array : rest) {
            System.arraycopy(array, 0, result, offset, array.length);
            offset += array.length;
        }

        return result;
    }

    @Override
    public void onDeactivated(int reason) {
        // TODO Auto-generated method stub
    }
}
```

```
package com.eviljack.nfcunlock;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.Editable;
import android.text.InputFilter;
import android.text.method.DigitsKeyListener;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

/*
 * A simple launcher activity which displays a view. This class sets
up the
 * entire project. It also checks to make sure that the PIN is
correct. It also
 * opens the appropriate activity if the correct PIN is given.
 *
 * Note: There is no PIN if you don't provide one. Which means you
will be able
 * to change the password without a PIN so long as you don't provide
one.
 */
public class MainActivity extends Activity {
   private final static int MAX_LENGTH = 8;

   private Button changePass;
   private Button changePin;
   private Class<?> classHolder;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main_activity);
      initLayout();
      initListeners();
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.main, menu);
```

```
        return true;
    }

    private void initLayout() {
        changePass = (Button) findViewById(R.id.changePassword);
        changePin = (Button) findViewById(R.id.changePIN);
    }

    private void initListeners() {
        changePass.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                classHolder = PasswordHandler.class;
                checkPin();
            }
        });

        changePin.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                classHolder = PinHandler.class;
                checkPin();
            }
        });
    }

    private void checkPin() {
        AlertDialog.Builder alert = new AlertDialog.Builder(this);

        SharedPreferences secure =
getApplicationContext().getSharedPreferences(
                "SECURE", Context.MODE_PRIVATE);

        if (!secure.contains("pin")) {
            Intent newEvent = new Intent(getApplicationContext(),
classHolder);

            startActivity(newEvent);
            return;
        }

        alert.setTitle("Enter your PIN");
        alert.setMessage("PIN");

        // Set an EditText view to get user input
        final EditText input = new EditText(this);
```

```
      input.setFilters(new InputFilter[] {
            // Maximum 2 characters.
            new InputFilter.LengthFilter(MAX_LENGTH),
            // Digits only.
            DigitsKeyListener.getInstance(),});

      // Digits only & use numeric soft-keyboard.
      input.setKeyListener(DigitsKeyListener.getInstance());

      alert.setView(input);

      alert.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int whichButton)
{
            Editable value = input.getText();
            SharedPreferences secure = getApplicationContext()
                  .getSharedPreferences("SECURE",
Context.MODE_PRIVATE);

            if (value.toString().equals(secure.getString("pin",
"0"))) {
               Intent newEvent = new Intent(getApplicationContext(),
                     classHolder);

               startActivity(newEvent);
            }
         }
      });

      alert.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int whichButton)
{
            // Nothing to be done
         }
      });

      alert.show();
   }
}
```

```
package com.eviljack.nfcunlock;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

/*
 * This activity allows the user to change the password. If no
password is
 * provided, nothing happens, even if you press the "Submit" button.
 */

public class PasswordHandler extends Activity {
   private Button cancelButton;
   private Button submitButton;
   private EditText passwordEditText;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_card_emulation);
      initLayout();
      initListener();
   }

   private void initLayout() {
      cancelButton = (Button) findViewById(R.id.passwordCancel);
      submitButton = (Button) findViewById(R.id.passwordSubmit);
      passwordEditText = (EditText) findViewById(R.id.passwordField);
   }

   private void initListener() {
      cancelButton.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View v) {
            finish();
         }
      });

      submitButton.setOnClickListener(new OnClickListener() {

         @Override
```

```
         public void onClick(View v) {
            if (passwordEditText.getText().length() > 0) {
               SharedPreferences secure = getApplicationContext()
                     .getSharedPreferences("SECURE",
Context.MODE_PRIVATE);
               secure.edit()
                     .putString("password",

passwordEditText.getText().toString()).commit();

               finish();
            }
         }
      });
   }
}
```

```
package com.eviljack.nfcunlock;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.InputFilter;
import android.text.method.DigitsKeyListener;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

/*
 * This activity allows the user to change the PIN. If no PIN is
provided,
 * nothing happens, even if you press the "Submit" button. If no PIN
is
 * provided, the password can be changed without a PIN.
 */
public class PinHandler extends Activity {
   private final static int MAX_LENGTH = 8;

   private Button cancelButton;
   private Button submitButton;
   private EditText passwordEditText;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_pin_handler);

      initLayout();
      initListener();
   }

   private void initLayout() {
      cancelButton = (Button) findViewById(R.id.pinCancel);
      submitButton = (Button) findViewById(R.id.pinSubmit);
      passwordEditText = (EditText) findViewById(R.id.pinText);

      passwordEditText.setFilters(new InputFilter[] {
            // Maximum 2 characters.
            new InputFilter.LengthFilter(MAX_LENGTH),
            // Digits only.
            DigitsKeyListener.getInstance(), // Not strictly needed,
IMHO.
```

```
        });

        // Digits only & use numeric soft-keyboard.

passwordEditText.setKeyListener(DigitsKeyListener.getInstance());
    }

    private void initListener() {

        cancelButton.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                finish();
            }
        });

        submitButton.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                if (passwordEditText.getText().length() > 0) {
                    SharedPreferences secure = getApplicationContext()
                            .getSharedPreferences("SECURE",
Context.MODE_PRIVATE);
                    secure.edit()
                            .putString("pin",
passwordEditText.getText().toString())
                            .commit();

                    finish();
                }
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.piin_handler, menu);
        return true;
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eviljack.nfcunlock"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <uses-feature
        android:name="android.hardware.nfc.hce"
        android:required="true" />

    <uses-permission android:name="android.permission.NFC" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.eviljack.nfcunlock.MainActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name="com.eviljack.nfcunlock.CardService"
            android:exported="true"
            android:permission="android.permission.BIND_NFC_SERVICE"
>
            <intent-filter>
                <action
android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE" />

                <category
android:name="android.intent.category.DEFAULT" />
            </intent-filter>

            <meta-data
```

```
android:name="android.nfc.cardemulation.host_apdu_service"
                android:resource="@xml/aid_list" />
        </service>

        <activity
            android:name="com.eviljack.nfcunlock.PasswordHandler"
            android:label="@string/title_activity_main_activity"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name="com.eviljack.nfcunlock.PinHandler"
            android:label="@string/title_activity_pin_handler"
            android:screenOrientation="portrait" >
        </activity>
    </application>

</manifest>
```