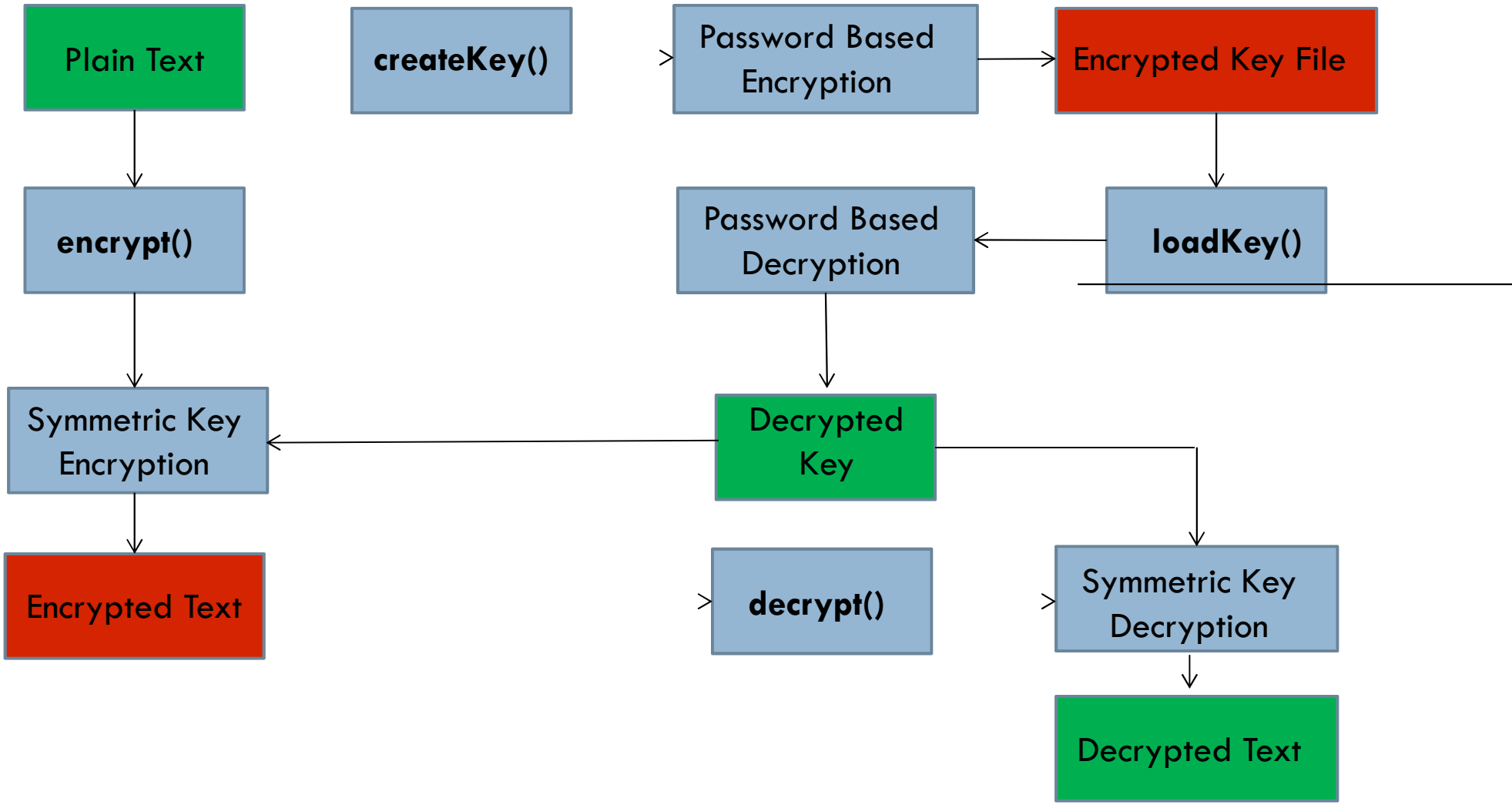
CRYPTOGRAPHY 456

FILE ENCRYPTION FOR ANDROID DEVICES

Lee Hudson

Contents

File Encryption Diagram



Utilizations

- AES Algorithm
- CBC Mode
- PKCS5Padding
- Password Based Encryption

AES Algorithm

- 128-bit block size
- Supported key lengths 128, 192, 256 bit
- Security relative to their submitted algorithm
- Efficiency in software and hardware

CBC Mode

- Encryption of all blocks are chained together
- Encryption is randomized by using an initialization vector (IV)

PKCS5Padding

- Password based encryption standard
- Number of bytes padded equals to 8
- All padded bytes have the same value (Number of bytes added)

Password Based Encryption

- Password based encryption uses a password as the key
- The security reacts with the user rather than in the physical medium
- We use the password based encryption to encrypt the AES key to allow the user more versatility
  - This gives the user full control over who can use the AES key

Code Overview

- Two separate Encrypts and Decrypts
  - Key Encryption
    - Password Based Encryption (Done in createKey())
  - File Encryption
    - Symmetric Key Encryption (Done in encrypt())
  - Key Decryption
    - Password Based Decryption (Done in loadKey())
  - File Decryption
    - Symmetric Key Encryption (Done in decrypt())
- Things to remember…
  - The password must match
  - The key must be the same

createKey()

- Generate an AES session key
  - KeyGenerator generator = KeyGenerator.getInstance("AES", "BC");
- Get the bytes of the key
  - Byte[] keyBytes = encryptionKey.getEncoded();
- Create a salt array
  - Random.nextBytes(salt);
- Create a cipher to encrypt the data
  - Cipher cdec = Cipher.getInstance("PBEWithSHAAnd3KeyTripleDES");
- Create a password-based encryption keySpec
  - PBEKeySpec pbeSpec = new PBEKeySpec(password, salt, ITERATIONS);
- Encrypt using a password-based encryption
  - Byte[] encryptedBytes = cdec.doFinal(keyBytes);
- Write the encrypted keyBytes and salt to a file
  - FileOutputStream fos = new FileOutputStream(KEY_FILENAME);

loadKey()

- Read in the keyBytes and salt to a byte array
  - FileInputStream fis = new FileInputSteam(KEY_FILENAME);
  - ByteArrayOutputStream baos = new ByteArrayOutputStream();
- Separate keyBytes from salt
  - Byte[] salt = new byte[8];
  - Byte[] keyBytes = new byte[length];
- Create a cipher to encrypt the data
  - Cipher cdec = Cipher.getInstance("PBEWithSHAAnd3KeyTripleDES");
- Create a PBEKeySpec
  - PBEKeySpec pbeSpec = new PBEKeySpec(password, salt, ITERATIONS);
- Create a key using the PBEKeySpec
  - Key sKey = keyFact.generateSecret(pbeSpec);
- Decrypt the key using the created cipher
  - Byte[] plainKey = cdec.doFinal(keyBytes);
- Return the decrypted key

encrypt()

- Using a password passed into loadKey, grab the encryption key from the key file
  - Key encryptionKey = loadKey(password);
- Create the SecretKeySpec that will be used in the encryption process
  - SecretKeySpec key = new SecretKeySpec(encryptionkey.getEncoded(), “AES”);
- Generate a cipher using the key to initialize it
  - Cipher cipher = Cipher.getInstance(“AES/CBC/PKCS5Padding”, “BC”);
- Generate the IvSpec that is used in the cipher initialization, used for CBC mode
  - Byte[] ivBytes = new byte[16];
  - IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
- Initialize the cipher
  - Cipher.init(cipher.DECRYPT_MODE, key, ivSpec);
- Create a FileOutputStream in order to write out the ivBytes and the encrypted text
  - FileOutputStream fos = new FileOutputStream(“/sdcard/Assignment 3/” + fileOutput);
- Create a FileInputStream to read in the file to be encrypted
  - FileInputStream fis = new FlleInputStream((“/sdcard/Assignment 3/” + fileInput);

encrypt() Continued…

- Wrap the FileOutputStream with a CipherOutputStream
  - CipherOutputStream cos = new CipherOutputStream(fos, cipher);
  - Cos.write(theByte);
  - This code is performed within a loop, iterating byte by byte.

- Use a while loop to run through the text file, encrypting the text and writing it out to a new file

```
int theByte = 0;
While((theBytes = fis.read()) != -1)
{
    cos.write(theByte);
}
```

decrypt()

- Using a password passed into loadKey, grab the encryption keyfrom the key file
  - Key encryptionKey = loadKey(password);
- Create the SecretKeySpec that will be used in the encryption process
  - SecretKeySpec key = new SecretKeySpec(encryptionkey.getEncoded(), “AES”);
- Generate a cipher using the key to initialize it
  - Cipher cipher = Cipher.getInstance(“AES/CBC/PKCS5Padding”, “BC”);
- Create file input/output streams to read in the ivBytes and file to be decrypted
  - FileInputStream fis = new FIleInputStream((“/sdcard/Assignment 3/” + fileInput);
  - FileOutputStream fos = new FileOutputStream(“/sdcard/Assignment 3/” + fileOutput);
- Create your ivBytes and load them using the FileInputStream
  - byte[] ivBytes = new byte[16];
  - Fis.read(ivBytes);
- Generate the IvSpec that is used in the cipher initialization
  - IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
- Initialize the cipher
  - Cipher.init(cipher.DECRYPT_MODE, key, ivSpec);

decrypt() Continued…

- Create a CipherInputStream in order to decrypt the file
  - CipherInputStream cis = new CipherInputStream(fis, cipher);
- Use a while loop that reads through the encrypted file with the CipherInputStream, decrypting the file and outputting it to the decrypted file

```
int theByte = 0;
While((theByte = cis.read()) !=1)
{
    fos.write(theByte);
}
```

Android

- Android requires a GUI for the user to interact with
  - Buttons and Edit Texts can be created by using XML Code
    - Button can be created as follows:
      - <Button
        android:id="@+id/execute"
        android:layout_width="fille_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Execute" />
    - Edit Text can be created as follows:
      - <Edit Text
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="What is your password?" />

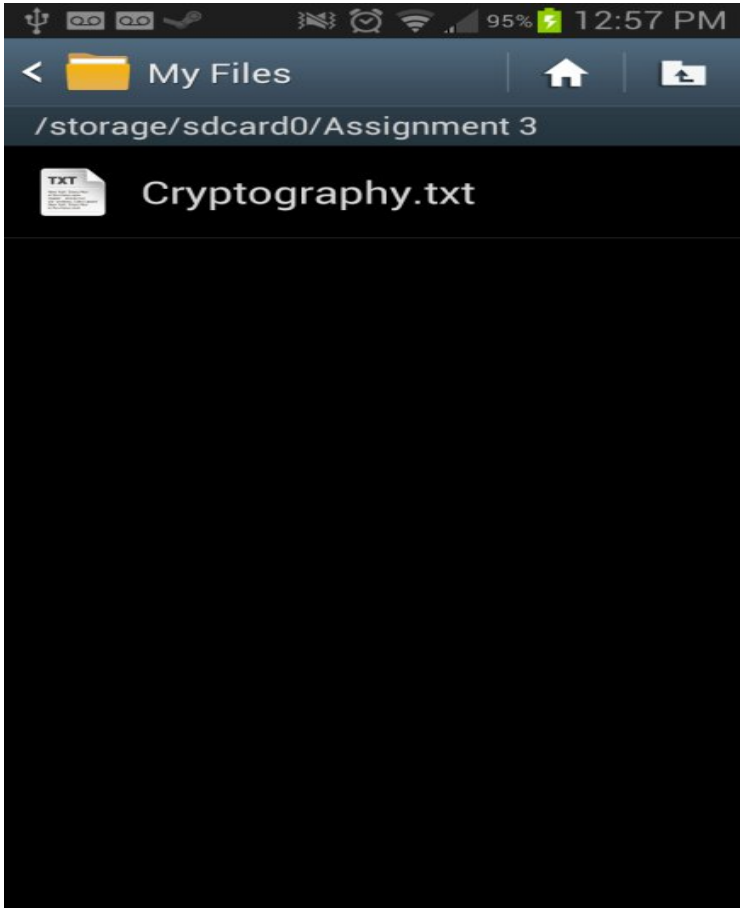
Android Continued…

- Android requires permission to be set in the Manifest file
  - File Encryption contains:
    - *android.permission.READ_EXTERNAL_STORAGE*
    - *android.permission.WRITE_EXTERNAL_STORAGE*

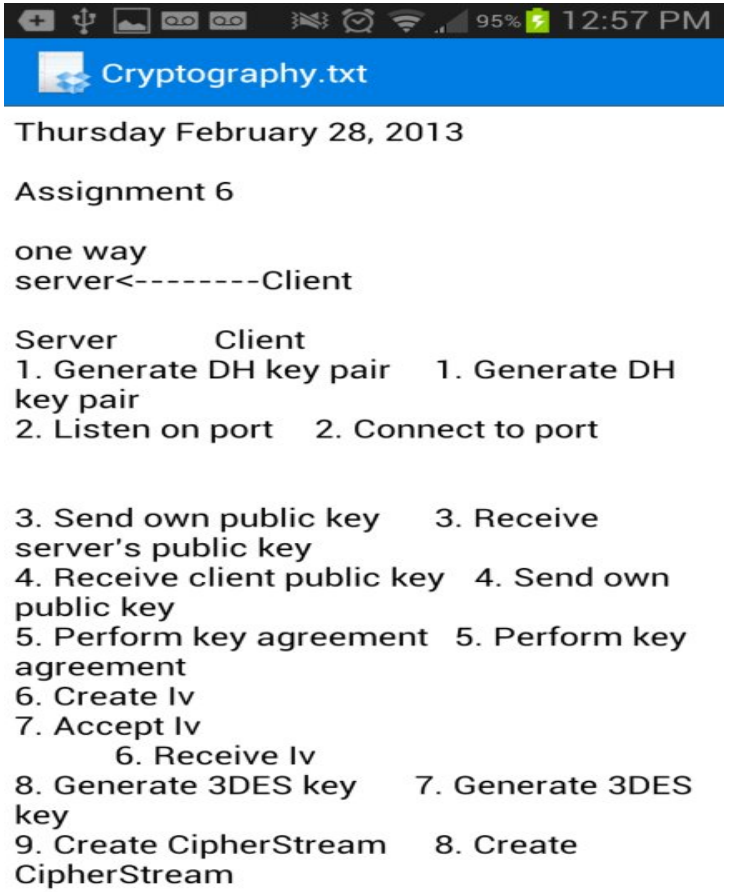
Android Continued…

- Inside the main activity, in the onCreate() method we initialize the objects with the resource id
  - execute = (Button) findViewById(R.id.*execute*);
  - password = (EditText) findViewById(R.id.*passWord*);
- In this instance the file location was hardcoded into a variable as follows:
  - "/sdcard/Assignment 3/AESkey.txt";
    - However, in future implementations a file explorer would be more beneficial to use for searching for the file.

Screen Shots



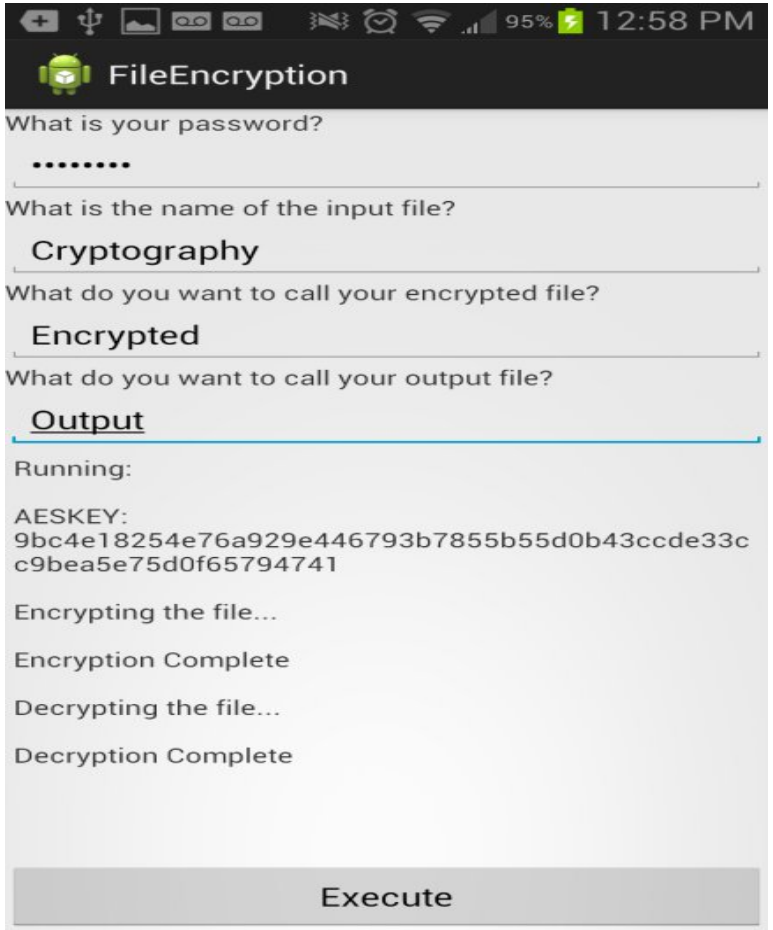
File to be encrypted



Text that has not been Encrypted

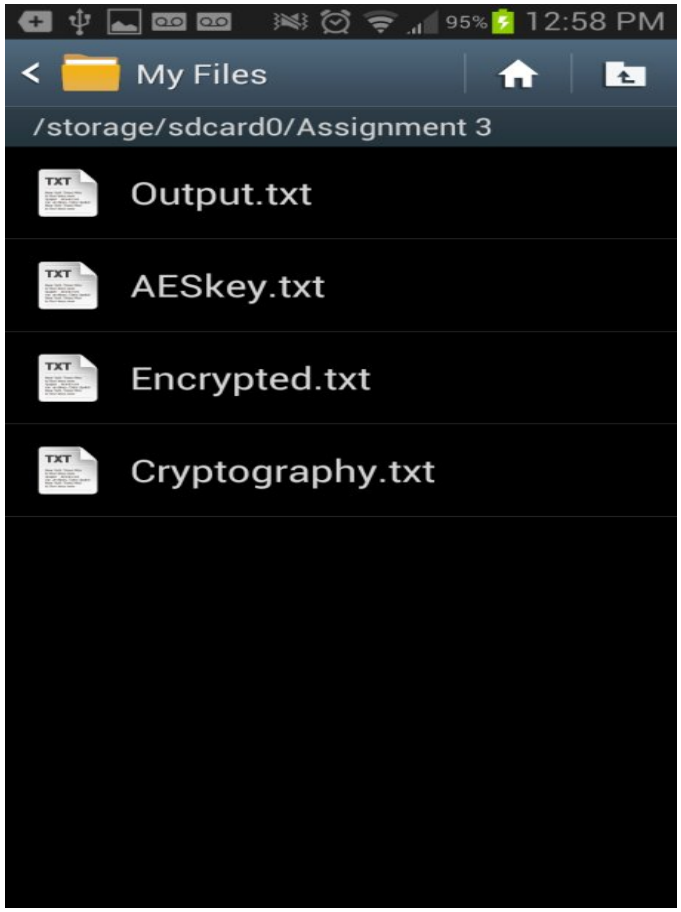
Screen Shots Continued..

GUI for file encryption

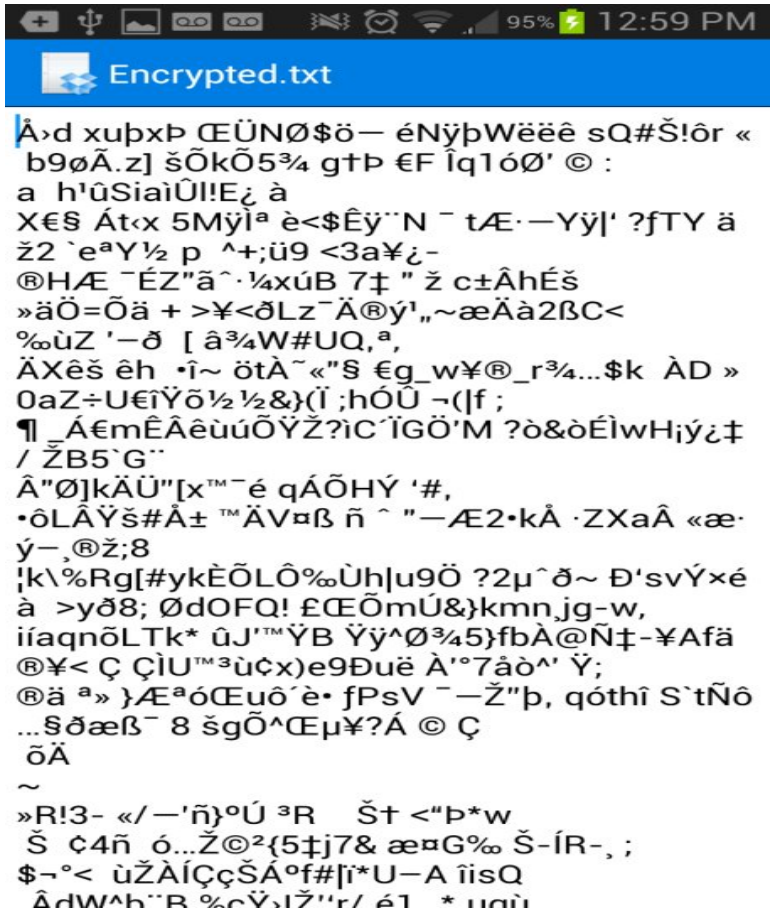


GUI during execution process

Screen Shots Continued

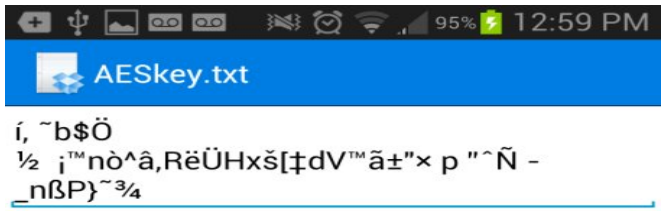


Files store after execution

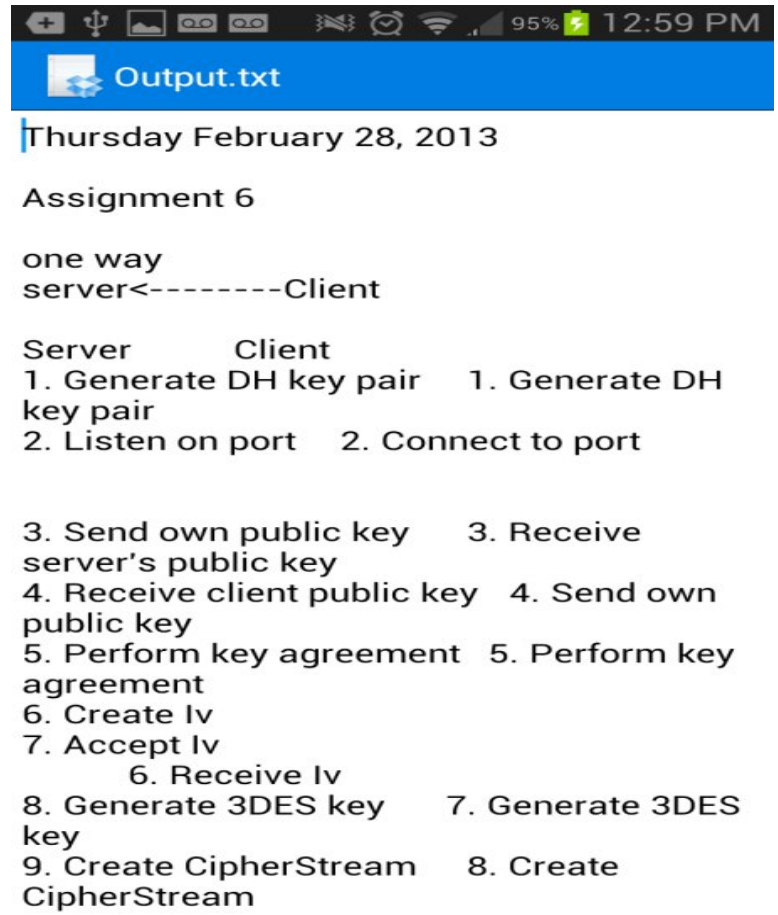


Encrypted Text

Screen Shots Continued



Encrypted Key



Decrypted Text