Spring 2023 - ICSI 526
Homework 2

Jacob Clouse

March 17, 2023

1 Running CBC and OFB modes:

There are two separate AES files included: One contains my code for CBC mode and the other contains my code for OFB mode. I couldn't get them to work together inside of the same file so I had to split them out into different programs. NOTE: This program was mainly coded on a Linux Mint machine using BASH to compile and Git/Github to host my code (using a private git repo). Running CBC and OFB modes are almost identical, but I have included instructions for both here for the sake of completeness.

How to Compile and Run CBC Mode:

1. Make sure you have a working copy of the Java JDK and a text editor on your machine (I use VS Code because of the built in terminal).
2. Make sure you have x3 test files of 2000 characters each (with 0%, 25% and 50% duplication respectively).
3. Make a copy of the **CBC** folder I have provided and cd into it with your terminal. The ONLY two items you should have inside are the **AES.java** and **AES_Demo.java** files (the demo is required to run this program).
4. You can compile the program by using the syntax:

   ```
   javac AES_Demo.java AES.java
   ```

   This should create a several class files within the **CBC** folder, including a AES_Demo.class file.
5. Now you can run this file using the syntax:

   ```
   java AES_Demo
   ```

   If successful, a window should pop up titled **Jacob Clouse CBC Demo:** that will allow you to select your sample data files.
6. You can use the *Browse Files* button and navigate to the first test file on your computer. **DO NOT** select either 'Preserve Image Header' or 'Reduced AES - 4 rounds'.
7. You can select where you want the encrypted output file to go using the **Choose Save Directory** button. **NOTE:** On linux, I have noticed that the output file sometimes will be stored in the parent directory of the folder you initially selected.

8. Finally, you can click Begin AES and it will encrypt your file (there is no decryption in this file, so the 'Encryption Time' and 'Decryption Time' fields may be blank). You now have your encrypted output and you repeat the process to encrypt other files as you please.

How to Compile and Run OFB Mode:

1. Make sure you have a working copy of the Java JDK and a text editor on your machine (I use VS Code because of the built in terminal). If you have already run CBC mode, then you should be all set.
2. Make sure you have x3 test files of 2000 characters each (with 0%, 25% and 50% duplication respectively). Again, you can just reuse the same input files from CBC mode.
3. Make a copy of the **OFB** folder I have provided and cd into it with your terminal. The ONLY two items you should have inside are the **AES.java** and **AES_Demo.java** files (the demo is required to run this program).
4. You can compile the program by using the syntax:

   ```
   javac AES_Demo.java AES.java
   ```

   This should create a several class files within the **OFB** folder, including a AES_Demo.class file.
5. Now you can run this file using the syntax:

   ```
   java AES_Demo
   ```

   If successful, a window should pop up titled **Jacob Clouse OFB Demo:** that will allow you to select your sample data files.
6. You can use the *Browse Files* button and navigate to the first test file on your computer. **DO NOT** select either 'Preserve Image Header' or 'Reduced AES - 4 rounds'.
7. You can select where you want the encrypted output file to go using the **Choose Save Directory** button. **NOTE:** On linux, I have noticed that the output file sometimes will be stored in the parent directory of the folder you initially selected.
8. Finally, you can click Begin AES and it will encrypt your file (there is no decryption in this file, so the 'Encryption Time' and 'Decryption Time' fields may be blank). You now have your encrypted output and you repeat the process to encrypt other files as you please.

2 My Index of Coincidence Table:

I created this table by feeding the encrypted files into a python IoC Calculator. With ECB, you can clearly see that that the IoC is much higher in the 50% shared character file vs the 0% shared file. Even the 25% shared file is much higher than baseline and it demonstrates that ECB just is not secure enough to warrant use.
Within my CBC implementation, there is very little difference between the results of the 0%, 25% and 50% shared character files. This does an adequate job of diffusing our output and making it look much more like random noise. The same can be said for my OFB implementation too, where we see practically identical levels of IoC across the files.

Mode	% Dup	IOC
ECB	0%	0.01995247623811906
	25%	0.02444322161080540 4
	50%	0.03179889944972486
CBC	0%	0.003902951475737869
	25%	0.003873936968484242
	50%	0.003944972486243122
OFB	0%	0.003928964482241 12
	25%	0.003864432216108054
	50%	0.003921960980490245

3 Running Extended Image Hiding Program:

When I modified this code, I again wanted to reuse the framework that was provided to me in the original application. I made a copy of the original file and added a few touches to the GUI and choices for MSB and LSB. Basically, I added two new radio button groups: hostButtonGroup and secretButtonGroup. These two groups affect the two new boolean variables I created: HostIsLSBVar and SecretIsLSBVar (both are initialized as true). What happens is that we are using these two groups to update the true false values of our two variables.
If hostButtonGroup equals 'HostLSB', then we update HostIsLSBVar to equal true. If it equals HostMSB, then we update HostIsLSBVar to equal false. Same thing with secretButtonGroup, where if it equals SecretLSB then SecretIsLSBVar is true and if it equals SecretMSB then SecretIsLSBVar is false.

How to Compile and Run Image Hiding:

1. Make sure you have a working copy of the Java JDK and a text editor on your machine (I use VS Code because of the built in terminal). Again, you should have this if you did the CBC and OFB portions.

2. Make a copy of the **Image Hiding Extended** folder I have provided and cd into it with your terminal. The ONLY items you should have inside are the **ImageHiding.java** file and the original picture files that were provided (host and secret).

3. You can compile the program by using the syntax:

```
javac ImageHiding.java
```

This should create a several class files within the **Image Hiding Extended** folder, including a ImageHiding.class file.

4. Now you can run this file using the syntax:

   ```
   java ImageHiding
   ```

   If successful, a window should pop up titled **Jacob Clouse Image Hiding:** that will allow you adjust the MSB and LSB of the pictures.

5. The window that pops up will have all the buttons that the original demo did (ie: the Window that shows you current encoded bits, a + button to add more, a - button to remove more, and two displays showing the host and secret images).
   HOWEVER, there are now x4 radio buttons present. You can select If you want to use the MSB or LSB of the Host image to hide your values (one or the other, not both). In combination with that, we can chose to encode either the MSB or LSB of the Secret image. Both default to LSB, so you can adjust them to the two choices you need.

6. After you have made your selection, just hit either the + or - button and it will start encoding using the settings that you have selected.

Results of Image Hiding Extension: I found that the results from my adjustment to the Image Hiding Program seem