> CompSci 351 Sections 01/02: Fall 2020; Instructor: Kartik Nayak, TAs: Chenghong Wang, David Pujol

Lab 1. Secret-Key Encryption

Instructions:

- Due Tuesday, September 15, 11:59 pm Eastern time
- Submission: Use gradescope
- Collaboration policy: refer to the last section of this document.

Overview

The learning objective of this lab is for students to get familiar with the concepts in secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, initialization vector (IV), and tools for random number generation. Moreover, students will be able to use tools and write programs to encrypt/decrypt messages. This lab covers the following topics:

- Secret-key encryption
- Encryption modes
- Programming using a crypto library
- Random number generation

These labs are modified versions of the ones on the SEED website.

Lab Tasks

Task 1 (0 pts): Encryption using Different Ciphers and Modes

In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

```
$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \
              -K 00112233445566778889aabbccddeeff \
              -iv 0102030405060708090a0b0c0d0e0f
```

Please replace the `-ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-aes-128-ctr`, `-aes-128-ofb`, etc. In this task, you should try at least 3 different ciphers. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the openssl enc command in the following:

```
-in <file>      input file
-out <file>     output file
```

```
-e              encrypt
-d              decrypt
-K/-iv          key/iv in hex is the next argument
-[pP]           print the iv/key (then exit if -P)
```

> **Cipher block chaining (CBC)**: Ehrsam, Meyer, Smith and Tuchman invented the cipher block chaining (CBC) mode of operation in 1976.[13] In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block. Wiki source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_block_chaining_(CBC)

> **Output feedback mode (OFB)**: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Output_feedback_(OFB)

Task 2 (5 pts): Encryption Modes – ECB vs. CBC

The file pic `original.bmp` can be downloaded from `src/Lab1/task-2/`, and it contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes. You shall resue the IV in task 1 and **use your own netid as the encryption key**. Since the netid may less than 16 characters (i.e. 128 bits), you need to append pound signs (#: hexadecimal value is 0x23) to the end of your netid to form a key of 128 bits. The following command shows an example for generating a key based on your netid:

```
$ echo -n "cw374###########" > key
$ xxd -p key
```

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. We can use the bless hex editor tool (already installed on our VM) to directly modify binary files. We can also use the following commands to get the header from p1.bmp, the data from p2.bmp (from offset 55 to the end of the file), and then combine the header and data together into a new file.

   ```
   $ head -c 54 p1.bmp > header
   $ tail -c +55 p2.bmp > body
   $ cat header body > new.bmp
   ```

2. Display the encrypted picture using a picture viewing program (we have installed an image viewer program called eog on our VM). Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

3. Select a picture of your choice, repeat the experiment above, and report your observations.

Task 3 (5 pts): Error Propagation – Corrupted Cipher Text

In some modes, an error in a ciphertext block results in a deciphering error only in the corresponding plaintext block while in some other modes such an error would affect two or more blocks. To understand the error propagation property of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.
4. Decrypt the corrupted ciphertext file using the correct key and IV.

How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, or OFB, respectively? Please provide a justification for each of them. (When you do this task, try to answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task.)

Task 4 (10 pts): Initialization Vector (IV)

Most of the encryption modes require an initial vector (IV) or initial random value (r). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode. The objective of this task is to understand the problems if an IV is not selected properly. Please do the following experiments:

- Task 4.1 (2 pts). A basic requirement for IV is uniqueness, which means that no IV may be reused under the same key. To understand why, please encrypt the same plaintext using (1) two different IVs, and (2) the same IV. Please describe your observation, based on which, explain why IV needs to be unique.

- Task 4.2 (2 pts). One may argue that if the plaintext does not repeat, using the same IV is safe. Let us look at the Output Feedback (OFB) mode. Assume that the attacker gets hold of a plaintext (P1) and a ciphertext (C1), can he/she decrypt other encrypted messages if the IV is always the same? You are given the following information, please try to figure out the actual content of P2 based on C2, P1, and C1.

  ```
  Plaintext  (P1): This is a known message!
  Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

  Plaintext  (P2): (unknown to you)
  Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
  ```

  The attack used in this experiment is called the known-plaintext attack, which is an attack model for cryptanalysis where the attacker has access to both the plaintext and its encrypted version (ciphertext). This is weaker than chosen-plaintext attack since the attacker does not get to choose the plaintext. If this can lead to the revealing of further secret information, the encryption scheme is not considered as secure.

- Task 4.3 (6 pts). From the previous tasks, we now know that IVs cannot repeat. Another important requirement on IV is that IVs need to be unpredictable for many schemes, i.e., IVs need to be randomly generated. In this task, we will see what is going to happen if IVs are predictable.

Assume that Bob just sent out an encrypted message, and Eve knows that its content is either Yes or No; Eve can see the ciphertext and the IV used to encrypt the message, but since the encryption algorithm AES is quite strong, Eve has no idea what the actual content is. However, since Bob uses predictable IVs, Eve knows exactly what IV Bob is going to use next. The following summarizes what Bob and Eve know:

```
Encryption method: 128-bit AES with CBC mode.
Key (in hex):00112233445566778899aabbccddeeff (known only to Bob)
Ciphertext (C1): bef65565572ccee2a9f9553154ed9498 (known to both)
IV used on P1 (known to both)
    (in ascii): 1234567890123456
    (in hex) : 31323334353637383930313233343536
Next IV (known to both)
    (in ascii): 1234567890123457
    (in hex)  : 31323334353637383930313233343537

128-bit plaintext corresponding to Yes (in hex):
5965730d0d0d0d0d0d0d0d0d0d0d0d0d
128-bit plaintext corresponding to No (in hex):
4e6f0e0e0e0e0e0e0e0e0e0e0e0e0e0e
```

A good cipher should not only tolerate the known-plaintext attack described previously, it should also tolerate the chosen-plaintext attack, which is an attack model for cryptanalysis where the attacker can obtain the ciphertext for an arbitrary plaintext. Since AES in CBC mode is a strong cipher that can tolerate the chosen-plaintext attack, Bob does not mind encrypting any plaintext given by Eve; he does use a different IV for each plaintext, but unfortunately, the IVs he generates are not random, and they can always be predictable.

Your job is to construct a message P2 and ask Bob to encrypt it and give you the ciphertext. Your objective is to use this opportunity to figure out whether the actual content of P1 is Yes or No.

Task 5 (10 pts): Programming using the Crypto Library

In this task, you are given a plaintext and a ciphertext, and your job is to find the key that is used for encryption. You do know the following facts:

- The `aes-128-cbc` cipher is used for the encryption.
- The key used to encrypt this plaintext is an English word shorter than 16 characters; the word can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), pound signs (#: hexadecimal value is 0x23) are appended to the end of the word to form a key of 128 bits. We have provided you a word list at `src/Lab1/task-5/words.txt`.

Your goal is to write a program to find out the encryption key. The plaintext, ciphertext, and IV are listed in the following:

```
Plaintext (total 21 characters):    you captured the flag

Ciphertext (in hex format): 46beb3b832973495f79b860884245e43
                            1d73c2d3f7e3a7632dce894ed14ff62b
```

```
IV (in hex format):         aabbccddeeff00998877665544332211
```

You need to pay attention to the following issues:

- If you choose to store the plaintext message in a file, and feed the file to your program, you need to check whether the file length is 21. If you type the message in a text editor, you need to be aware that some editors may add a special character to the end of the file. The easiest way to store the message in a file is to use the following command (the `-n` flag tells `echo` not to add a trailing newline):

  ```
  $ echo -n "you captured the flag" > file
  ```

- You can write the program in Java or Python. If using Python, you can use libraries pyCrypto and PyCryptodome. Sample code can be found from the following URL: https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html If you are using Java, you can refer to the documentation in `javax.crypto` package.

- Recall that AES works with blocks of 128 bits (16 bytes). However, when the input string is not a multiple of 16 bytes, we need to convert it to 16 bytes by padding it. We will use PKCS5/PKCS7 padding for this purpose. With this padding, if the string is k bytes short of being a multiple of 16 for k > 0, then we pad the string with k bytes each with a value of k (represented in hex). If the string has length which is multiple of 16, then another 16 bytes of padding is added. An example of this was used in task 4, where `5965730d0d0d0d0d0d0d0d0d0d0d0d0d` corresponds to the string 'Yes'. The first three bytes correspond to the three characters and subsequent bytes contain 0x0d (13 in hex). You will need to pad the plaintext string using appropriate *existing* libraries when performing this task (PyCryptodome in Python or specifying appropriate padding in Java).

Task 6 (7 pts). Pseudo Random Number Generation

Task 6.1 (2 pts). Measure the Entropy of Kernel

In the virtual world, it is difficult to create randomness, i.e., software alone is hard to create random numbers. Most systems resort to the physical world to gain the randomness. Linux gains the randomness from the following physical resources:

```
void add_keyboard_randomness(unsigned char scancode);
void add_mouse_randomness(__u32 mouse_data);
void add_interrupt_randomness(int irq);
void add_blkdev_randomness(int major);
```

The first two are quite straightforward to understand: the first one uses the timing between key presses; the second one uses mouse movement and interrupt timing; the third one gathers random numbers using the interrupt timing. Of course, not all interrupts are good sources of randomness. For example, the timer interrupt is not a good choice, because it is predictable. However, disk interrupts are a better measure. The last one measures the finishing time of block device requests.

The randomness is measured using entropy, which is different from the meaning of entropy in the information theory. Here, it simply means how many bits of random numbers the system currently has. You can find out how much entropy the kernel has at the current moment using the following command.

```
$ cat /proc/sys/kernel/random/entropy_avail
```

Let us monitor the change of the entropy by running the above command via watch, which executes a program periodically, showing the output in fullscreen. The following command runs the cat program every 0.1 second.

```
$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

Please run the above command. While it is running, move your mouse, click your mouse, type something, read a large file, visit a website. What activities increases the entropy significantly. Please describe your observation in your report.

Task 6.2 (2 pts). Get Pseudorandom Numbers from /dev/random

Linux stores the random data collected from the physical resources into a random pool, and then uses two devices to turn the randomness into pseudo andom numbers. These two devices are /dev/random and /dev/urandom. They have different behaviors. The /dev/random device is a blocking device. Namely, every time a random number is given out by this device, the entropy of the randomness pool will be decreased. When the entropy reaches zero, /dev/random will block, until it gains enough randomness.

Let us design an experiment to observe the behavior of the /dev/random device. We will use the cat command to keep reading pseudo random numbers from /dev/random. We pipe the output to hexdump for nice printing.

```
$ cat /dev/random | hexdump
```

Please run the above command and at the same time use the watch command to monitor the entropy. What happens if you do not move your mouse or type anything. Then, randomly move your mouse and see whether you can observe any difference. Please describe and explain your observations.

Task 6.3 (3 pts): Get Random Numbers from /dev/urandom

Linux provides another way to access the random pool via the /dev/urandom device, except that this device will not block. Both /dev/random and /dev/urandom use the random data from the pool to generate pseudo random numbers. When the entropy is not sufficient, /dev/random will pause, while /dev/urandom will keep generating new numbers. Think of the data in the pool as the "seed", and as we know, we can use a seed to generate as many pseudorandom numbers as we want.

Let us see the behavior of /dev/urandom. We again use cat to get pseudo random numbers from this device. Please run the following command, and the describe whether moving the mouse has any effect on the outcome.

```
$ cat /dev/urandom | hexdump
```

Let us measure the quality of the random number. We can use a tool called ent, which has already been installed in our VM. According to its manual, "ent applies various tests to sequences of bytes stored in files and reports the results of those tests. The program is useful for evaluating pseudo-random number generators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a file is of interest". Let us first generate 1 MB of pseudo random number from /dev/urandom and save them in a file. Then we run ent on the file. Please describe your outcome, and analyze whether the quality of the random numbers is good or not.

```
$   head -c 1M /dev/urandom > output.bin
$   ent output.bin
```

Theoretically speaking, the /dev/random device is more secure, but in practice, there is not much difference, because the "seed" used by /dev/urandom is random and non-predictable (/dev/urandom does re-seed whenever new random data become available). A big problem of the blocking behavior of /dev/random is that blocking can lead to denial of service attacks. Therefore, it is recommended that we use /dev/urandom to get random numbers. To do that in our program, we just need to read directly from this device file. The following code snippet shows how.

```
#define LEN 16  // 128 bits
unsigned char *key = (unsigned char *) malloc(sizeof(unsigned char)*LEN);
FILE* random = fopen("/dev/urandom", "r");
fread(key, sizeof(unsigned char)*LEN, 1, random); fclose(random);
```

Please modify the above code snippet to generate a 256-bit encryption key. Please compile and run your code; print out the numbers and include a screenshot in the report.

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide some necessary files generated from your Lab. A detailed description is provided as follows.

- Task 2
  - Submit the encrypted picture with .bmp suffix for task 2.1, please rename your encrypted picture as `t2_enc_YOURNETID.bmp`
  - Submit Your picture you selected by your own and the corresponding encrypted picture for task 2.3, please rename your picture as `t2_pic_YOURTNETID.bmp`, and the encrypted pic as `t2_enc_pic_YOURNETID.bmp`
  - Your Lab report should answer question 2.2.
- Task 3
  - Submit the text file you created, please rename it as `t3_text_YOURNETID.txt`

    - Submit the encrypted file you created based on the generated text, please rename it as `t3_enc_text_YOURNETID`.
    - Submit your key and IV in a file named `t3_key_iv.txt`
    - Your Lab report should answer questions asked in task 3
- Task 4
    - You need to answer each question in task 4.1, 4.2, and 4.3, and include them in your Lab report.
- Task 5
    - Submit your code for cracking the key, please name your code as `t5_crack_YOURNETID.py` (It can be a java file, if you're coding in Java, please change the file extension accordingly.).
    - Please submit a readme describe how to compile and run your code, we encourage you to submit a Makefile to compile your code.
    - In your Lab report, you need to briefly introduce your code, and report the key you recovered.
- Task 6
    - Submit your code for task 6.3, please name your code as `t6_rd_YOURNETID.c`.
    - In your Lab report, you need to answers all questions in task 6 and provide your observation required by each task.

List of required files to be included in your submission:

1. Your lab report
2. `t2_enc_YOURNETID.bmp`
3. `t2_pic_YOURTNETID.bmp`
4. `t2_end_pic_YOURTNETID.bmp`
5. `t3_text_YOURNETID.txt`
6. `t3_enc_text_YOURNETID`
7. `t3_key_iv.txt`
8. `t5_crack_YOURNETID.py` (It can be a java file, if you're coding in Java, please change the file extension accordingly.)
9. `Readme` doc that explains how to compile and run `t5_crack_YOURNETID.py`
10. `t6_rd_YOURNETID.c`

Collaboration Policy

We have provided you with sufficient information to perform the labs. There may be some material online that may be useful: you are not prohibited from looking at these materials. Though when you hand in the lab, we expect that all the work/code is your own. In particular, you should be able to explain any part of your solution in detail, and why you chose to do it that way and not some other way. If you referred to resources online/included (fragments of) source code from elsewhere, you must add a comment indicating it.

Academic dishonesty. You can discuss the problems and the high-level approach with other students, but only after you have tried to solve it yourself. If you do discuss, you should acknowledge the discussion in your lab report. You cannot share solutions/code with other students. Do not obtain solutions from online sources.