Georgia Gwinnett College
School of Science and Technology
ITEC 4320: Internet Security
OpenSSL Lab 1: Private-Key Encryption with AES

Introduction

OpenSSL is an open-source software library for security applications. The library includes an implementation of all widely used cryptographical tools for confidentiality and integrity. In this lab, you will use OpenSSL commands on the Ubuntu operating system to perform data encryption and decryption with 256-bit AES. You will complete the lab on a virtual machine in the ITEC Lab, and therefore need a VPN connection.

Instructions

1. Open the Cisco AnyConnect Secure Mobile Client on your computer and connect to vpn.ggc.edu. Then choose the ITECLAB group, enter your GGC username and password, and press OK to complete the VPN connection.

2. From the application PuTTY or the command prompt, login to 172.20.1.106 by SSH, using *only* your GGC username as **both** the username and password.

3. Copy the file **Lab1.zip** from the directory **/home/yding** to your home directory using the command **cp /home/yding/Lab1.zip .**

4. Unzip **Lab1.zip** using the command **unzip Lab1.zip**, then change directory to **Lab1** using the command **cd Lab1**

Exercise 1 **[10 Points]**

Change directory to **Ex1**. The file **cipher.bin** contains a ciphertext that was created using **256-bit AES** in the **CBC** mode, with the password "welcome" and the option **-pbkdf2** to derive an initialization vector (IV) and a one-time key from the password. Using a proper OpenSSL command, decrypt **cipher.bin** to a file named **plain.txt**. View the plaintext using the command **more plain.txt**. Take a screenshot of your command and result. Name the image file **Lab1Ex1**.

Exercise 2 **[30 Points]**

1. Change directory to **Ex2**.

2. Generate a random 256-bit key and write it to a file named **aesKey.bin**. Take a screenshot of your command and name the image **Lab1Ex2p2**.

3. Using **256-bit AES** with the **CBC** mode, encrypt the file **msg.txt** with the key **aesKey.bin** from Step 2, using the option **-pbkdf2** to derive a one-time key and IV from **aesKey.bin**, and write the ciphertext to a file named **cipher.bin**. Take a screenshot of your command and result. Name the image **Lab1Ex2p3**.

4. Using a proper command, decrypt **cipher.bin** from the Step 3 and write the plaintext to a file named **plain.txt**. If your command is correct, **plain.txt** should be identical to **msg.txt**. Take a screenshot of your command and result. Name the image **Lab1Ex2p4**.

Exercise 3 [30 Points]

In this exercise, we take a closer look at **AES** in the **CBC** mode.

1. Copy **aesKey.bin** from **Ex2** to **Ex3**. Change directory to **Ex3**.

2. Use the command from Step 3 of Exercise 2 to encrypt the file **msg.txt** and write the ciphertext to a file named **cipher1.bin**, but this time add the option **-p** to display the *random* 64-bit *salt*, as well as the 256-bit *one-time* key and the 128-bit IV derived from **aesKey.bin** and the salt, using **pbkdf2**.

3. Repeat the previous step, this time writing the ciphertext to a file named **cipher2.bin**. Take a screenshot of your commands and results from the previous step and this step. Name the image **Lab1Ex3p2-3**.

   You should notice that the salt value is different in the two runs. Consequently, the IV and one-time key, which are derived from the *master* key **aesKey.bin** and the salt, are also different in the two runs.

4. Use the command **hexdump -vC** to display **cipher1.bin** and **cipher2.bin** in the hexadecimal representation. Take a screenshot of your commands and results. Name the image **Lab1Ex3p4**.

   Notice that the salt values for **cipher1.bin** and **cipher2.bin** generated in Step 3 appear in the second half of the first block of the files. The actual ciphertext starts in the second block. By comparing the hex values, you can see that because the salt values (consequently the one-time key and IV) are *different*, the two ciphertexts are *different*, even though they encrypt the same plaintext.

   When a ciphertext is decrypted with the same *master* key **aesKey.bin**, **AES** would read the salt from the file, derive the same one-time key and IV from **aesKey.bin** and the salt, and use the one-time key and IV to decrypt the ciphertext.

5. Use the command in Step 4 of Exercise 2 to decrypt **cipher1.bin** and **cipher2.bin**, writing the resulting plaintexts to files named **plain1.txt** and **plain2.txt** respectively, also adding the option **-p** to display the salt, one-time key and IV. Take a screenshot of your commands and results. Name the image **Lab1Ex3p5**.

Notice that for each of **cipher1.bin** and **cipher2.bin**, the salt, one-time key and IV match exactly those from encryption. Also notice that even though **cipher1.bin** and **cipher2.bin** are different, they decrypt to the same plaintext – **plain1.txt** and **plain2.txt** are the same and are identical to **msg.txt**.

<u>Exercise 4</u>: Deterministic Encryption is Insecure [20 Points]

1. Copy **aesKey.bin** from **Ex3** to **Ex4**. Change directory to **Ex4**.

2. Encrypt **msg.txt** as in Steps 2 and 3 of Exercise 3, however this time add the **-nosalt** option. Take a screenshot of your commands and results. Name the image **Lab1Ex4p2**.

   The option **-nosalt** specifies that no random salt will be used for key derivation. When the master key **aesKey.bin** is fixed, **pbkdf2** would always derive the same one-time key and IV. Consequently, the encryption is deterministic, namely, the same plaintext would always have the same ciphertext. This is insecure because it reveals whether two ciphertexts encrypt the same plaintext. In a context, such patterns may allow an attacker to break the encryption. Therefore, **in practice the option -nosalt should never be used**. It should be used only for the purpose of testing.

3. Use the command **hexdump -vC** to display **cipher1.bin** and **cipher2.bin**. You should notice that the two files are identical. Also use the command **diff** to check whether **cipher1.bin** and **cipher2.bin** are different. The command should return no output, meaning that the files are identical. Take a screenshot of your commands and results. Name the image **Lab1Ex4p3**.

<u>Exercise 5</u>: The ECB Mode is Insecure [30 Points]

The file **msg.txt** for this exercise contains the text "UsernamePasswordUsernamePassword". It consists of two *identical* 128-bit blocks, each block being "UsernamePassword". Using the command **hexdump -vC msg.txt**, you can see the hexadecimal representations of the two blocks in the first two lines of the output. In this exercise, you will use 256-bit AES in the **ECB** mode to encrypt the file, and observe that the ciphertext also contains two *identical* blocks.

1. Copy **aesKey.bin** from **Ex4** to **Ex5**, and change directory to **Ex5**.

2. Encrypt **msg.txt** as in Steps 2 of Exercise 3, however using **aes-256-ecb** *instead of* **aes-256-cbc**. Take a screenshot of your commands and results.

3. Use the command **hexdump -vC cipher1.bin** to display **cipher1.bin**. Take a screenshot of your commands and results from the previous step and this step. Name the image **Lab1Ex5p2-3**.

   The ciphertexts of the two blocks in **msg.txt** appear in the second and third blocks of **cipher1.bin**. Notice that the two ciphertext blocks are also *identical*.

4. Repeat Steps 2 and 3, this time writing the ciphertext to a file named **cipher2.bin**. Take a screenshot of your commands and results. Name the image **Lab1Ex5p4**.

   Because of the random salt, **cipher2.bin** uses a different salt value from **cipher1.bin**. Consequently, the two ciphertexts are different. However, what does not change is that the two ciphertext blocks in **cipher2.bin** are *still identical*, even though they are different from the blocks in **cipher1.bin**. This exercise shows that the **ECB mode is insecure** because the ciphertext reveals whether the plaintext has repeated blocks. In a context, such patterns may allow an attacker to break the encryption.

5. Repeat Steps 2 and 3, but this time using **aes-256-cbc** instead. Write the ciphertext to a file named **cipher3.bin**. Take a screenshot of your commands and results. Name the image **Lab1Ex5p5**.

   Notice that even though **msg.txt** has two identical blocks, the corresponding ciphertext blocks in **cipher3.bin** (appearing in the second and third blocks) are *different*.

After finishing all the exercises, **upload all the screenshots (the image files) to D2L**.