**Georgia Gwinnett College**

**School of Science and Technology**

**ITEC 3300: Information Security**

**OpenSSL Lab 2: Public-Key Encryption with RSA**

**Introduction**

In this lab, you will use OpenSSL commands on the Ubuntu operating system to perform encryption and decryption with the RSA public-key encryption scheme, as well as hybrid encryption and decryption involving both RSA and AES.

**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 **Lab2.zip** from the directory **/home/yding** to your home directory using the command **cp /home/yding/Lab2.zip .**
4. Unzip **Lab2.zip** using the command **unzip Lab2.zip**, then change directory to **Lab2** using the command **cd Lab2**

**Exercise 1: RSA Encryption & Decryption [50 Points]**

1. Change directory to **Ex1**.
2. ![A screenshot of a cell phone

   Description automatically generated]()Generate a pair of RSA keys, using **3072 bits** as the key length, and write the keys to a file named **rsaKey.bin**. Take a screenshot of your command and include your image here.
3. ![A close up of a logo

   Description automatically generated]()View the pairs of RSA keys in **rsaKey.bin** as plaintext and redirect the result to a file named **rsaKey.txt**. View **rsaKey.txt** using the command **more rsaKey.txt**. Take a screen shot of the two commands as well as the beginning portion of **rsaKey.txt** that includes the public key. (You may need to enlarge the application window of PuTTY or command prompt to include all the above in one image.) Include your image here.
4. ![A picture containing table

   Description automatically generated]()Extract the public RSA key from **rsaKey.bin** and write the public key to a file named **pubKey.bin**. View **pubKey.bin** as plaintext. The extracted public key should be identical to the one in **rsaKey.bin.** Take a screen shot of the two commands as well as the public key. (You may need to enlarge the application window of PuTTY or command prompt to include all the above in one image.) Include your image here.
5. Encrypt the file **msg.txt** and write the ciphertext to a file named **cipher1.bin**. (Which key should you use for encryption?) **the public key**
6. ![A picture containing drawing

   Description automatically generated]()Repeat Step 5 but this time write the ciphertext to a file named **cipher2.bin**. Use the command **diff cipher1.bin cipher2.bin** to check if the two ciphertexts are different. The result of the command should show that they are indeed *different*, even though they encrypt the same message. This indicates that by default the RSA cryptosystem you are using is *randomized*. Take a screenshot of your commands and results from Step 5 and Step 6. Include your image here.
7. **![A screenshot of a cell phone

   Description automatically generated]()**Decrypt **cipher1.bin** and **cipher2.bin**, writing the resulting plaintexts to files named **plain1.txt** and **plain2.txt** respectively. (Which key should you use for decryption?) View the files **plain1.txt** and **plain2.txt**. 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** Take a screenshot of your commands and results. Include your image here.

**Exercise 2: Hybrid Encryption & Decryption [50 Points]**

1. Change directory to **Ex2**. In the directory, the file **rsaKey.bin** contains a pair of RSA keys that has been generated, and the file **pubKey.bin** contains the public key that has been extracted from **rsaKey.bin**.
2. ![A screenshot of a cell phone

   Description automatically generated]()Attempt to encrypt the file **msg.txt** using **pubKey.bin**. The command will fail and receive an error message “data too large for the key size”. This is because RSA is a block encryption scheme. The length of the message encrypted cannot exceed the key length, however the size of **msg.txt** is much larger than the key length (3072 bits). Take a screenshot of your command and the error message. Include the image here.

As described in class, because public-key encryption is slow, a common approach for encryption is *hybrid encryption*, that is, use a public-key cryptosystem (like RSA) to exchange a secret key for a private-key encryption scheme (like AES), then encrypt the data using private-key encryption.

1. Generate a random 256-bit key for **AES** and write it to a file named **aesKey.bin**.
2. Using **256-bit AES** with the **CBC** mode, encrypt the file **msg.txt** with the key **aesKey.bin**, 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 commands from Step 3 and Step 4. Include the image here.  **[see number five below!]**
3. ![A screenshot of a cell phone

   Description automatically generated]()Encrypt the key **aesKey.bin** using the RSA public key **pubKey.bin** and write the ciphertext to a file named **cipherSK.bin**. Take a screenshot of your command and include the image here.

In practice, a sender would send to a receiver **cipher.bin** which is the actual ciphertext of data, and **cipherSK.bin** which is the ciphertext of the key **aesKey.bin** that was used to encrypt the data.

Now suppose that the receiver has received **cipher.bin** and **ciphersSK.bin**.

1. ![A screenshot of a cell phone

   Description automatically generated]()Decrypt **cipherSK.bin** using **rsaKey.bin** and write the plaintext to a file named **sk.bin**. Use the **diff** command to verify that **sk.bin** and **aesKey.bin** are identical. Take a screenshot of your commands and results. Include your image here.
2. ![A screenshot of a cell phone

   Description automatically generated]()Decrypt **cipher.bin** using **sk.bin** from Step 6 and write the plaintext to a file named **plain.txt**. If your commands are correct, **plain.txt** should be identical to **msg.txt**. Take a screenshot of your commands and results. Include your image here.

**Exercise 3: Deterministic Public-Key Encryption is Insecure [50 Points]**

Recall that an encryption scheme is *deterministic* if under the same encryption key, the same plaintext would always have the same ciphertext. We already mentioned several times that deterministic encryption is *insecure* because it reveals whether two ciphertexts encrypt the same plaintext. In this exercise, you will see that the situation is even worse when deterministic public-key encryption is used, and in the scenario given below, you can easily decrypt a ciphertext *without the private decryption key*.

For public-key encryption in OpenSSL, the option **-raw** specifies that no randomness is used for encryption. In other words, the encryption is deterministic. This is the equivalent of the **-nosalt** option for private-key encryption. **In practice the option** **-raw should never be used**. It should be used only for the purpose of testing.

Change directory to Ex3. The file **pubKey.bin** contains a public RSA Key that was used with the option **-raw** to create six ciphertexts **cipher1.bin**, …, **cipher6.bin** which are also in the directory. Yet the private decryption key is *not* available. You know in addition that each of the six ciphertexts is the encryption of one of the three known files **red.txt**, **green.txt** and **blue.txt** which are in the directory as well.

You task for this exercise is to decrypt each of the six ciphertext *without the private decrypt key*. That is, using only the information given above as well as proper OpenSSL and Linux commands, decide, for each of the six ciphertexts, which one of the three files it encrypts. Take a screenshot of all your commands and results. Include your images and answer here.

**[see below]**

**As, seen below, the ciphertexts in order from 1 to 6 are:** ***Green****,* ***Blue****,* ***Red****,* ***Green****,* ***Red****,* ***Blue***

![A screenshot of text

Description automatically generated]()