**Georgia Gwinnett College**

**School of Science and Technology**

**ITEC 3300: Information Security**

**OpenSSL Lab 3: Digital Signature with RSA**

**Introduction**

In this lab, you will use OpenSSL commands on the Ubuntu operating system to perform digital signature and verification using RSA.

**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 **Lab3.zip** from the directory **/home/yding** to your home directory using the command **cp /home/yding/Lab3.zip .**
4. Unzip **Lab3.zip** using the command **unzip Lab3.zip**, then change directory to **Lab3** using the command **cd Lab3**

**Exercise 1: [100 Points]**

1. Generate a pair of RSA keys, using **3072 bits** as the key length, and write the keys to a file named **sigKey.bin**. Note that **sigKey.bin** contains both the private signing key and the public verification key.
2. Extract the public verification key from **sigKey.bin** and write it to a file named **verKey.bin**. Take a screen shot of the two commands from Steps 1 and 2. Include your image here.
3. Sign the file **msg.txt** without the hash function option and write the signature to a file named **sig.bin**. (Which key should you use for signing?)
4. Repeat Step 4 but this time explicitly using **sha256** as the hash function and write the signature to a file named **sig256.bin**. Use the **diff** command to check whether **sig.bin** and **sig256.bin** are different. You should see that they are the same. This is because OpenSSL does **hash-then-sign** and by default **sha256** is the hash function. Take a screen shot of the three commands from Steps 3 and 4. Include your image here.

Note that while encryption must be randomized to be secure, digital signature can be deterministic, that is, it is fine that signing the same message twice yields the same signature.

1. Compute the hash value of the file **msg.txt** using the hash function **sha512** and output the result to the console.
2. Sign the file **msg.txt** using the hash function **sha512** and write the signature to a file named **sig512.bin**.
3. Verify whether **sig512.bin** is a valid signature of **msg.txt**. Your command should output **Verified OK** because **sig512.bin** is indeed the signature of **msg.txt**, and neither the file nor the signature has been modified. Take a screen shot of the three commands and results from Steps 5 – 7. Include your image here.
4. Copy **msg.txt** to a different file named **msg2.txt**.
5. Compute the hash value of **msg2.txt** using **sha512** and output the result to the console. Then verify whether **sig512.bin** is a valid signature of **msg2.txt**. You should see that at this point the hash value of **msg2.txt** is the same as that of **msg.txt** from Step 5, and your command for signature verification still outputs **Verified OK**. Do you see why? Take a screen shot of the three commands and results from Steps 8 and 9. Include your image here.
6. Open **msg2.txt** using your favorite editor and **make a small change** to the file. **Repeat Step 9**. Now you should notice that the hash value of **msg2.txt** is different from that of **msg.txt** from Step 5, and your command for signature verification outputs **Verification Failure**, because the file **msg2.txt** was modified. Take a screen shot of the two commands and results. Include your image here.