Jillian Fliedner

IA5010

January 20, 2018

Lab 2 Report - Cryptography

1. Bash script crypto.sh is included in the zip!
2. Also included in the zip! The receiver’s public and private keys are test\_pub.key and test\_priv.key. The sender’s public and private keys are test\_pub\_s.key and test\_priv\_s.key.
3. For my scheme, I took some inspiration from the slides! For this assignment, it was discouraged to use RSA to encrypt the data, because this becomes an issue as the data gets bigger. This lead me to decide to use AES-128 for encrypting the data, but this creates an issue of how you share the symmetric key and IV with the receiver. The solution I used with this was to use Pretty Good Privacy (PGP). When given the data along with the receiver’s public key, you generate a random AES key and IV and you use this public RSA key to encrypt the key and IV in two separate files for the receiver to decrypt with their private key. The receiver can then use this to decrypt the data that was encrypted with AES with a key of size 128. I chose this key size, because it is a good balance between security and efficiency. For this, we do not need an unnecessarily large key size, and AES 128 is still secure. Along with encrypting the actual data, I also hashed (using SHA256) the plaintext and decrypted plaintext for guaranteeing integrity. Through hashing, the text is made into a unique hash that will make it obvious if the data has been modified because the two hashes will not be the same; this is how I guarantee integrity. If the hashes do not match up, I make the user aware there is an integrity issue. Of course, the data’s confidentiality is guaranteed through the encryption by utilizing both asymmetric and symmetric encryption by using PGP. Only the person with the corresponding RSA private key can retrieve the AES key and IV to decrypt the data. However, there is still an issue of authentication, which I solve using signing. Before I encrypt the plaintext, I sign it using the sender’s private RSA key, creating the /tmp/sign file. During the decryption, after I decrypt I use the plaintext to verify the signature using the sender’s public RSA key. I check to make sure the output of the command for verifying the signing outputs “Verified OK”. If it doesn’t, I make the user aware that there’s an authenticity issue. Utilizing RSA encryption, AES 128 encryption, hashing, and signing I maintain confidentiality, integrity, and authenticity of the data.
4. The lab script begins with checking to make sure the number of arguments doesn’t equal zero. If it does, it issues an error message and exits the script. If the arguments do not equal 0 (can assume the lab id is passed as an argument), the script tries sshing into ia5010\_master using $1, which is the first argument containing the lab id. If this is a success, the script outputs to the terminal that it’s downloading the files. It copies the files from the passed lab id’s directory in ia5010\_master onto ~/Desktop/Labs. After this, it checks if a setup.sh script exists in ~/Desktop/Labs/$1/. If there is, it outputs to the terminal that it is executing the setup. It changes the permissions of setup.sh to be executable by the user. After this, it uses sudo to gain the privileges of a root user and it executes setup.sh.

