RSA Public-Key Encryption and Signature Lab

Telle Miller

Liberty University – CSIS 463

9/27/2021

**Introduction**

The RSA (Rivest-Shamir-Adleman) algorithm is widely used for secure communications and is built upon number theory. The RSA algorithm can be used for encryption, decryption, digital signature generation, and digital signature verification (). The RSA algorithm performs these functions using an asymmetric cryptosystem to generate a pair of private and public keys. This key generation is performed by generating two large prime numbers. These large numbers when multiplied together give a value that is practically impossible to factor in a reasonable amount of time. This is what gives RSA it’s high level of security, albeit with a slower speed compared to some algorithms.

**Lab Setup**

The lab environment is established using an Ubuntu 20.04 virtual machine image and a virtual machine host application, such as Oracle VM VirtualBox. The image used for this lab is available on the SEED labs website. After completing the first task, I recognized that some methods would likely be repeated for the following tasks, so I spent a few minutes separating some of the reused methods into their own file. This utility file is displayed below and was used for tasks #1 through #4.

Text

Description automatically generated

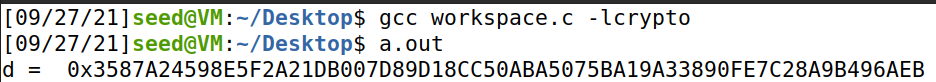
**Tasks**

Task 1: Deriving the Private Key

This task was relatively simple but laid the foundation for the rest of the assignment. I spent some time researching how RSA worked mathematically, however the lab guide was ample explanation for our purposes. The first image below shows the code used to generate the private RSA key. The second image demonstrates the command line call that compiled the code.

Text

Description automatically generated



Task 2: Encrypting a Message

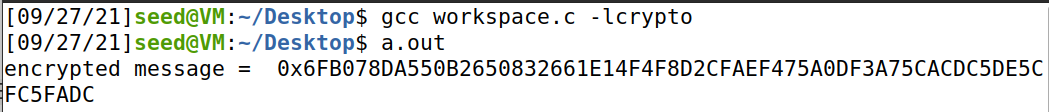
I ran into an issue in which the Python commanded referenced in the lab guide was unsupported by the virtual machine due to the transition to Python3. To circumvent this issue, I researched the method to translate ASCII text to hexadecimal in Python3. The method I found is demonstrated in the first image below. The second image shows the code used to encrypt the message. In RSA, encrypting a message is a process of mathematical operations that ultimately outputs a nearly irreversible value. The final image for this task demonstrates the output of the compiled code.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



Task 3: Decrypting a message

This task was the easiest so far. It relied heavily on the principles established in the previous two tasks, and the methods I established earlier on in the project were very useful. The first image below shows the code used to decrypt the given message. After the message was decrypted, it was in a hexadecimal format. I once again ran into an issue of unsupported functions referenced in the lab guide due to the transfer to Python3. After some research, I was able to find a method to translate the hexadecimal message into ASCII. This output is show in the second image.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

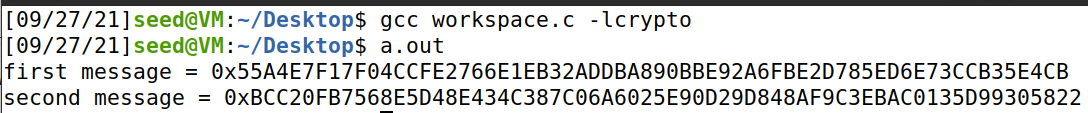
Task 4: Signing a message  
This was the most interesting task of the assignment for me. Using previously researched methods, I used Python3 to convert the two messages from ASCII to hexadecimal. This translation is shown in the first image below. In the second image, the code for signing both messages is shown. In the final image, the compiled code displays the two messages, both of which are signed. Despite only a change of one digit in the original two messages, the outputted signed message was drastically different. This is an important feature of an encryption algorithm, as subtle changes must produce noticeable results.

Text

Description automatically generated

Text

Description automatically generated



Task 5: Verifying a Signature

Once again, I used the Python3 method to translate the given message from ASCII to hexadecimal. Once I had this translated message, I attempted to verify it. After this attempt, I altered the last value of the hexadecimal message and attempted to verify this version. Both attempts are shown in the second image below. The returned value of 0 shows that the signature was valid, however the second returned value, a 1, shows that the RSA algorithm detected the change I made to the hexadecimal value. The last image for this task shows the code used to verify a given signature.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Task 6: Manually Verifying an X.509 certificate

This task was by far the most difficult for me. The previous tasks were relatively straight forward, and I was confident in my abilities to use C. This task however, required a knowledge of the command line that I was greatly lacking. I spent several hours attempting this task, following along with the lab guide and researching Ubuntu commands, but ultimately, I came up empty handed. I attached screenshots below showing the certificates I attempted to manually verify but translating the information from these certificates to a usable \*.bin file was stumping me. After an honest attempt, I decided to move on to finish the assignment. I hope to continue refining my command line skills and eventually come back to this problem to understand what I was missing.

Text

Description automatically generated

Text

Description automatically generated with medium confidence

**Discussion**

Overall, this lab was appropriately challenging and a great tool for learning the RSA encryption system. Most of my time on this lab was spent with the final task, which I ultimately failed to complete. This lab was a good refresher for my C programming skills and Linux command line abilities.

The first task was a good starter on how the RSA algorithm works. It took some research and time, but once I understood the process of going from numbers to a key, the algorithm made a lot more sense. The second task and third tasks both demonstrated the functional usage of the RSA algorithm. By both encrypting and decrypting a message, I was able to see how both sides of the system worked and what components were required to make it work. The fourth and fifth task were by far my favorite during this lab. They both demonstrated the non-repudiation aspects of this algorithm, as well as it’s sensitivity to minor changes. The idea that minor changes, even as small as a bit flip, can make large changes in the output is fascinating to me. The sixth task gave me an appreciation for the automation of cryptography we encounter every day. Networks and computers handle this hand off RSA information all the time, while manually verifying just one certificate was too much for me to handle. It gave me an improved perspective on the efficiency and reliability of computer networks.

**References**

Delfs, Hans, and Helmut Knebl. *Introduction to Cryptography: Principles and Applications.* Springer, 2007.

Rivest, R.; Shamir, A.; Adleman, L. (February 1978). *"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"* (PDF). Communications of the ACM. 21 (2): 120–126. CiteSeerX 10.1.1.607.2677. doi:10.1145/359340.359342. S2CID 2873616.

Diffie, W.; Hellman, M.E. (November 1976). *"New directions in cryptography". IEEE Transactions on Information Theory. 22 (6)*: 644–654. CiteSeerX 10.1.1.37.9720. doi:10.1109/TIT.1976.1055638. ISSN 0018-9448.

Margaret Cozzens and Steven J. Miller. "*The Mathematics of Encryption: An Elementary Introduction*". p. 180.

Surender R. Chiluka. "*Public key Cryptography*". Retrieved from <https://www.cs.uri.edu/cryptography/publickeykidkrypto.htm>.