***Phase 2-1, Summary and Explanation***

**How do we secure the traffic?**

What was discovered in Wireshark (a network packet sniffing tool) was how susceptible a message is to being intercepted by a foreign attacker or ‘man in the middle’ attack. This was a clear indication of how this data transferal process is insecure and that a solution will be put in place to secure and uphold integrity for the data and data transferal process.

In order to secure the data so that it cannot be as susceptible to a man in the middle attack is to implement encryption. The process of encrypting can vary depending on the technique chosen, however, to maintain the highest security in the data transferal process this involves the use of a public and private key.

In addition to implementing encryption, another piece of information will be added to the data in order to protect the data’s data integrity as well as its authenticity. To complete this a message authentication code or ‘MAC’ will be used. A ‘MAC’ is a code or tag that is attached to the data. A ‘MAC’ itself can be implemented in three different ways, either being built into the cipher (MAC-then-encrypt or MtE) with it first being appended to the cleartext and then encrypted. Another manner of implementation can be by appending the ‘MAC’ to the cipher text, with the ‘MAC’ produced based on the encrypted cipher text (encrypt-then-MAC or EtM). A similar process for the ‘MAC’ to be based on the plaintext however the plaintext is encrypted without the ‘MAC’, only for the ‘MAC’ to be appended to the cipher text (Encrypt-and-MAC or E&M). An alternative method to the ‘MAC’ based authentication is the process of hashing the data followed by encrypting the hashed data (hash-then-Encrypt).

A final solution that will secure and uphold the integrity of the data and data transferal process will involve implementing both an encryption method as well as a message authentication code.

Figure 1: Different Techniques

Diagram

Description automatically generated

**Technique Summary and Justification:**

The approach that will be used in order to secure and uphold the integrity of the data and data transferal process will be to firstly encrypt the plaintext followed by then creating a message authentication code ‘MAC’ from the ciphertext which that ‘MAC’ will then be appended to the ciphertext. This is the ‘Encrypt-then-MAC’ (EtM) technique.

The main reason for choosing the ‘EtM’ approach is because this technique is the most secure and yields the best data integrity out of the four techniques. This technique offers better security for the encryption and decryption process, this is largely because of how the ‘MAC’ is generated. With its generation technique preventing invalid ciphertexts from being fed through the process. Which in turn protects the data and receiver of the data from ciphertext attacks. It also avoids any problems regarding confidentiality on the plaintext as the ‘MAC’ is generated on the encrypted cipher and has no direct relation to the plaintext itself, regardless of the quality of the plaintext. This is great when it comes to security and data integrity, which the other techniques available cannot boast. As the other techniques base their message authentication code directly off of the plain text hence making the level of protection lower.

The encrypt-then-Mac technique also fits very well with the asymmetric encryption process. As only small amounts of data need to be sent in the data transferal process the time to complete the actions will not be as greatly affected as if you were to send over large amounts of data. Which would lend more to the symmetric encryption process. A great benefit for the use of asymmetric encryption is that it is far more secure than other encryption techniques. For example, the recommended key size in asymmetric encryption is 2048 bits or larger, whereas in the symmetric encryption most keys are typically 128 or 256 bits depending on the use case. As asymmetric encryption incorporates a dual key process where one of the keys is strictly known to only one of the parties in the data transferal process it is far more secure than just having a single public key which can consequently compromise the data if the key was retrieved.

Figure 2: Asymmetric Encryption, with ‘Encrypt-then-MAC’

Diagram

Description automatically generated

Figure 3: Asymmetric visual example

Diagram

Description automatically generated

***Note: The encrypted message’s character contents is a placeholder and is not mathematically derived.***

**How this technique is applied into the system:**

To implement the encryption-then-MAC technique the encryption process must first be completed. This will be completed by incorporating the ‘cryptography’ library in python. As asymmetric encryption is being used for the encryption process a private key will be first generated using and ‘RSA’ algorithm.

Since the private key has just been created a public key is now needed. To generate a public key, you must generate another RSA key based on the original private key that was first generated.

Since both keys have been generated, they must be stored. To do this the private key must first be serialised. Once serialised the key can then be written to a PEM file (priv\_key.pem). The public key then follows a similar process whereby it is first serialised then the public key is written to a different PEM file (pub\_key.pem).

To encrypt the message, the private key is first used in combination with the cryptography library’s existing functions to encrypt the message.

Since encryption-then-MAC is being used, the ‘MAC’ now needs to be generated from the ciphertext. This process will be implemented by incorporating a hashing function that uses SHA256 (Secure Hash Algorithm 256 bit). As the ‘MAC’ has now been generated, it can be appended to the original ciphertext. Allowing for the ciphertext with the appended mac to now be sent securely across the network.

On the other end of the data transferal process, i.e., the retrieval of the message involves the public key and the decryption process. To complete this, again the cryptography library will be used. Specifically, it contains a decryption function which allows for the public key to be used in order to perform the decryption process. Once the decryption process is completed the message is now readable.

***References:***

Figure 1: Stevensen, A. (2021). Technique overview Image. Retrieved 29 October 2021, made using Lucid Chart

Figure 2: Stevensen, A. (2021). Asymmetric Encryption with Encryption-then-MAC Image. Retrieved 28 October 2021, made using Lucid Chart

Figure 2: Stevensen, A. (2021). Asymmetric Encryption Image. Retrieved 28 October 2021, made using Lucid Chart

When to Use Symmetric Encryption vs. Asymmetric Encryption. (2021). Retrieved 1 November 2021, from <https://www.keyfactor.com/blog/symmetric-vs-asymmetric-encryption/>

An example of asymmetric encryption in python using a public/private keypair - utilizes RSA from PyCrypto library. (2021). Retrieved 29 October 2021, from <https://gist.github.com/syedrakib/241b68f5aeaefd7ef8e2>

Vollebregt, B. (2021). Asymmetric Encryption and Decryption in Python. Retrieved 29 October 2021, from <https://nitratine.net/blog/post/asymmetric-encryption-and-decryption-in-python/>

(2021). Retrieved 29 October 2021, from <https://coderzcolumn.com/tutorials/python/hmac-hash-based-message-authentication-code-using-pythons>