Name: Andrea Di Dio
VUnet ID: ado380
Student Number: 2593888
E-Mail: andreadidio98@gmail.com

Question 2:

In my opinion, there is a way to circumvent the key distribution problem in symmetric-key crypto systems. One way to reach a solution is to have a "hierarchical key system". The biggest problem in symmetric key distribution is that the two hosts who are communicating have the same key, both to encrypt and decrypt (a **master key**). At the start of a connection, the two hosts agree on a key to use for the messages they have to send back and forth to one another. This means that a potential attacker could, for example, set up a side channel and work out the key, using frequency analysis or using a brute force approach. Even though this means that the attacker would need heavy computations to get access to the key, once the key is known, he could decrypt every message sent between the two hosts.

One way of circumventing this is to have many temporary keys (**session keys**), which are changed periodically between the two hosts. The amount of times we change this session key, gives us a fluctuation between performance and security. i.e., the more we change these keys (for example, in a network communication, we could change the key for every packet we need to send), the more secure our system is, however also means that we would have to do more computations which would hinder greatly the performance. On the other hand, if we change the session keys less often (e.g., use one session key for the duration of a connection-oriented service (e.g., TCP)), we would gain in performance, but it would be easier for an attacker to make more sense out of the packets being sent, as he could decrypt more packets using the same key.

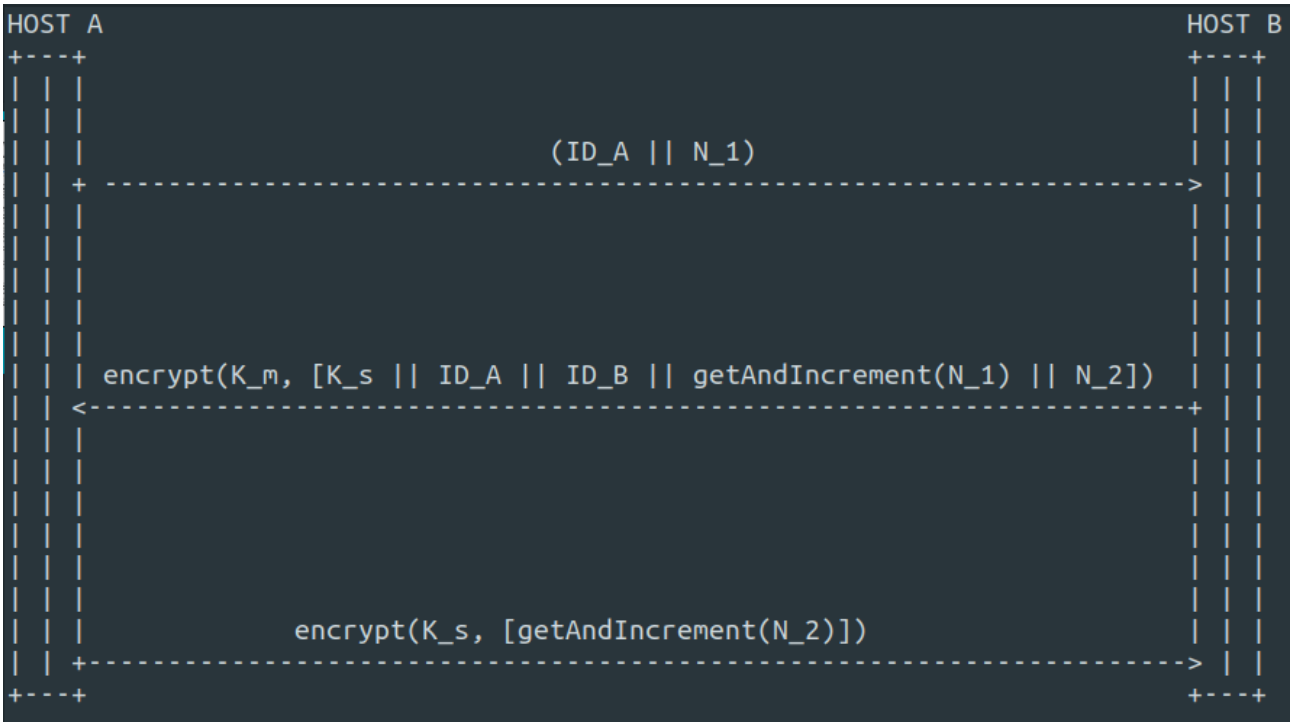
The idea is to have a single **master key** which will be used to encrypt and decrypt a **session key** when sending it to the other host, and the **session key** will be used to encrypt/decrypt the message text. We can setup the session keys by using a protocol similar to the three-way handshake in TCP: Let us have two hosts, namely **A** and **B** who want to communicate over the network:

1. Hosts A and B agree upon a function to modify the crypto nonce (e.g., a getAndIncrement()) and agree upon a shared master key, $\mathbf{K_m}$.
2. Host A initiates the connection with host B and sends a request for a session key by also appending a **crypto nonce,** $\mathbf{N_1}$ (which is used in a similar way to the sequence number/ACK mechanism of TCP).
3. Host B sends a packet back to A containing the session key, $\mathbf{K_s}$, the evaluation of the calculation of the "nonce-function" (e.g., $\mathbf{N_1 + 1}$) and its crypto nonce, $\mathbf{N_2}$. This packet is encrypted using the master key.
4. Host A acknowledges the receival of the session key by sending a packet back to host B containing the evaluation of the calculation of nonce-function (e.g., $\mathbf{N_2 + 1}$). This value is encrypted using the newly established session key.

The above sequence of events also depends however on the type of connection. When using a connectionless-transport protocol like UDP, in the packets, we should also include a Host ID (just like port numbers in UDP).

Sequence Diagram:

To visualise these steps, assume the “nonce-function” is a getAndIncrement, which increments by one the nonce, and assume we have a function encrypt() which takes as arguments a key and a message, and encrypts the message using the key, consider the following sequence diagram:



Note that in the above diagram, theoretically, if using a connection-oriented service such as TCP, we could omit the “IDs”.

Performance:

As previously mentioned, the performance of this system, greatly depends on how often we decide to make a new session key to use for encryption/decryption.
The upper limit on the amount of master keys that have to be held by n hosts is modeled by the function $\frac{(n*(n-1))}{2}$. This is reached if every host communicates with every other host.
At any instant in time, there are the same amount of session keys as there are master keys. However, if we want to calculate how many session keys we have created over time, we would need to make each host hold a counter, $\mathbf{C_n}$ to hold the amount of session keys used by a host n. Then to calculate the amount of session keys in total we could use a formula such as: $\frac{\sum_{n=1}^{n} C_n}{2}$.

Example:
Consider a system with 5 hosts, 1, 2, 3, 4, 5 where the hosts communicate with each other as follows:
1 → 2 ($C_1 = 3$, $C_2 = 3$)
3 → 4 ($C_3 = 2$, $C_4 = 2$)
3 → 5 ($C_3 = 4$, $C_5 = 4$)

With C_n being the amount of session keys used by a particular host over the lifetime of a communication.
We get: $(C_1 + C_2 + C_3 + C_4 + C_5) / 2$
$= (3 + 3 + 6 + 2 + 4) / 2 = 9$ session keys for the entire system.

Therefore, for such a system the most important measure on which to agree when exchanging messages among hosts, is the lifetime of a session key, and depending on the application we are using such a system for, we can choose to either have a more secure system, which due to the many computations and session key setups would add overhead to the performance, or we could choose to have a less secure system which would perform better.

Sources Used:

- Stallings, W. (2014). *Cryptography and Network Security: Principles and Practice*(7th ed.). Boston: Pearson.
- Introduction to cyber security: Stay safe online. (n.d.). Retrieved from https://www.open.edu/openlearn/ocw/mod/oucontent/view.php?id=48322§ion=1.3