II.2317 Cybersecurity (ISEP)

Nour El Madhoun **LAB 3** nour.el-madhoun@isep.fr

- **Important remarks:**
  - Firstly, we will give you the exact syntax for the commands to give you a general idea of how OpenSSL commands work. Afterwards, you have to do some research on the internet to find the right syntax and answer the questions.
  - The **three LABs** must be worked on by a team of **two students**.
  - Do not copy the commands as they are written in the pdfs because this can generate errors. It will be better if you write them.
  - You have to provide a **single report** for your work for the **three LABs**.
  - The report must contain **screenshots of all the parts with** *.
  - Do not forget to indicate **your names** in the report.
  - The report must be submitted on moodle before the **deadline that will be given during the session** by your supervisor.
  - If you have any questions, please contact your supervisor (and copy the e-mail to Nour El Madhoun).

1 Exercise "Preparation of cryptographic security elements for each actor" → (2.5 points)

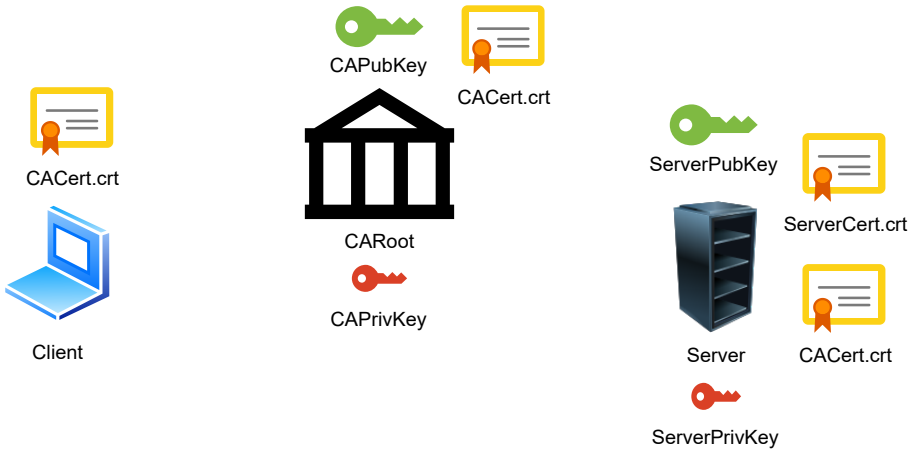

Figure 1: Cryptographic security elements for each actor

1. Please follow these steps to begin your practical work:
   (a) Open a ***Terminal***.
   (b) Create a new folder named ***LAB3*** and access this folder.
   (c) Create a new folder named ***CARoot***.
   (d) Create a new folder named ***Server***.
   (e) Create a new folder named ***Client***.
2. The scenario for this exercise is as follows (see Figure 1):

(a) We have a root certification authority named ***CARoot***.

(b) We have a ***Server*** and a ***Client***.

(c) The ***CARoot*** has a key pair ***CAPubKey*** and ***CAPrivKey*** and a self-signed certificate ***CACert.crt***. * We ask you to generate for ***CARoot*** these cryptographic security elements → **(0.5 pt)**.

(d) The ***Server*** has also a key pair ***ServerPubKey*** and ***ServerPrivKey***. * We ask you to generate for the ***Server*** these cryptographic security elements → **(0.5 pt)**.

(e) The ***Client*** does not have any key.

(f) The ***Server*** needs to create a request for a certificate ***ServerRequest.csr***. Afterwards, it will send this request to ***CARoot***. * We ask you to create for the ***Server ServerRequest.csr*** and send it to ***CARoot*** (by using the copy command as we have seen in the previous LAB) → **(0.5 pt)**.

(g) ***CARoot*** will generate ***ServerCert.crt*** and send it to the ***Server***. * We ask you to generate ***ServerCert.crt*** and send it to the ***Server*** → **(0.5 pt)**.

(h) ***CARoot*** will also send ***CACert.crt*** to the ***Server*** and the ***Client***. Consequently, The ***Server*** and ***Client*** store ***CACert.crt*** as a Trusted Third Party. * We ask you to send ***CACert.crt*** to the ***Server*** and the ***Client*** → **(0.5 pt)**.

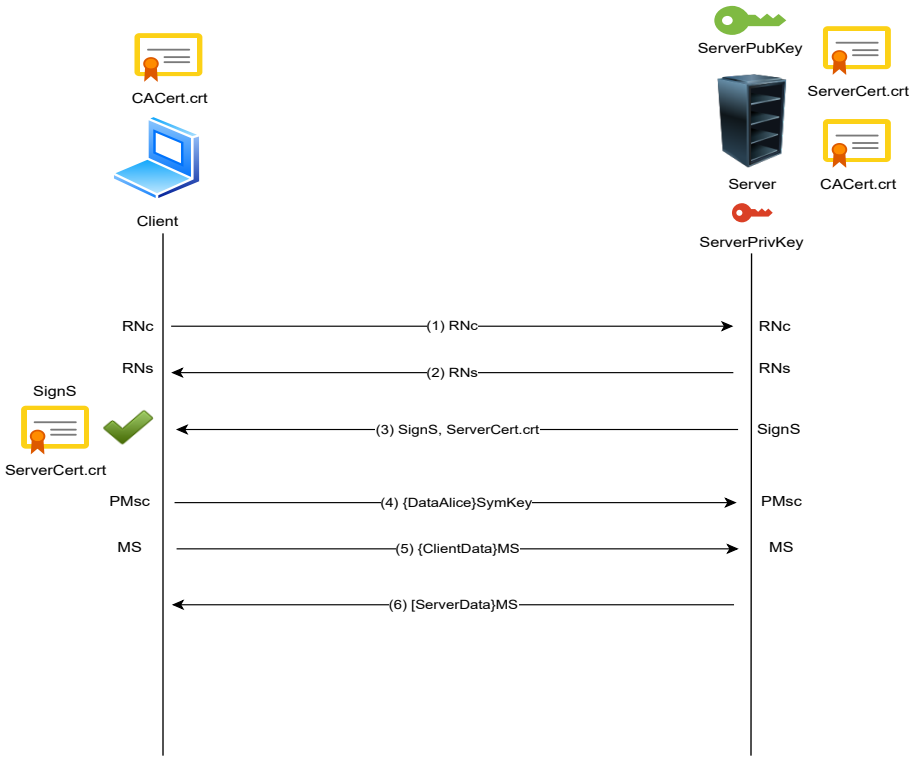

Figure 2: TLS Protocol (Messages exchanged)

2 Exercise "TLS Protocol" → (7.5 points)

The scenario for this exercise is as follows (see Figure 2):

1. The ***Client*** generates a file ***RNc***. * We ask you to create ***RNc*** and write a random number of your choice. Afterwards, you need to send it to the ***Server*** (by using the copy command as we have seen in the previous LAB) → **(0.25 pt)**.
2. The ***Server*** generates a file ***RNs***. * We ask you to create ***RNs*** and write a random number of your choice. Afterwards, you need to send it to the ***Client*** → **(0.25 pt)**.
3. The ***Server*** will generate ***SignS*** on the hash of ***RNc*** and ***RNs*** thanks to its private key ***ServerPrivKey***:
   (a) Make a concatenation of ***RNc*** and ***RNs*** by entering the UNIX command: ***cat RNc RNs > RNcRNs***
   (b) * Apply the hash function ***SHA256*** on ***RNcRNs*** to find its hash ***HashRNcRNs*** → **(0.5 pt)**.
   (c) * We ask you to proceed to generate ***SignS*** thanks to ***ServerPrivKey*** → **(0.5 pt)**.
   (d) * You can send now ***SignS*** and ***ServerCert.crt*** to the ***Client*** → **(0.25 pt)**.
   (e) The ***Client*** needs to verify ***ServerCert.crt***. * We ask you to verify it as you did in the previous LAB → **(0.5 pt)**.
   (f) The ***Client*** needs to verify ***SignS***. We ask you to verify it as follows :
   - * Extract ***ServerPubKey*** from ***ServerCert.crt*** → **(0.5 pt)**
   - * Repeat the same steps to verify a signature as you did in **part 5 of Exercise 2 in LAB1** → **(1 pt)**
4. The ***Client*** will generate a pre-master symmetric key ***PMsc*** and sends it encrypted to the ***Server***:
   (a) * We ask you to generate ***PMsc*** → **(0.5 pt)**.
   (b) * Encrypt ***PMsc*** thanks to ***ServerPubKey*** by naming the encrypted symmetric key ***PMscEncrypted*** → **(0.25 pt)**.
   (c) * Send ***PMscEncrypted*** to the ***Server*** → **(0.25 pt)**.
   (d) * Decrypt ***PMscEncrypted*** thanks to ***ServerPrivateKey*** by naming the decrypted key ***PMsc*** → **(0.5 pt)**.
5. Both the ***Client*** and ***server*** will generate the Master Symmetric key ***MS*** which is calculated from the hash of ***PMsc***, ***RNc*** and ***RNs***:
   (a) * We ask you to calculate ***MS*** → **(0.75 pt)**.
   (b) * We ask you to create the file ***ClientData*** and encrypt it thanks to ***MS*** → **(0.5 pt)**.
   (c) * Send the encrypted result to the ***Server*** and decrypt it → **(0.25 pt)**.
6. The ***Server*** will answer to the ***Client***:
   (a) * Create the file ***ServerData*** and encrypt it thanks to ***MS*** → **(0.5 pt)**.
   (b) * Send the encrypted result to the ***Client*** and decrypt it → **(0.25 pt)**.