Nouvelles Technologies et Société
Introduction to Cybersecurity
Nour El Madhoun **LAB 3** nour.el-madhoun@epita.fr

- **Important remarks:**
  - In this lab, you will be autonomous to answer all questions. Therefore, you need to go back to **LAB1**, **LAB2** and the **video provided** and do some research on the internet to find the correct syntax and answer the questions.
  - The report must contain **screenshots of all the parts with** *.
  - Do not forget to indicate **your name** and **group number** in the report.
  - The report must be sent before the **end of your third session** to the **email address provided by your supervisor**.

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

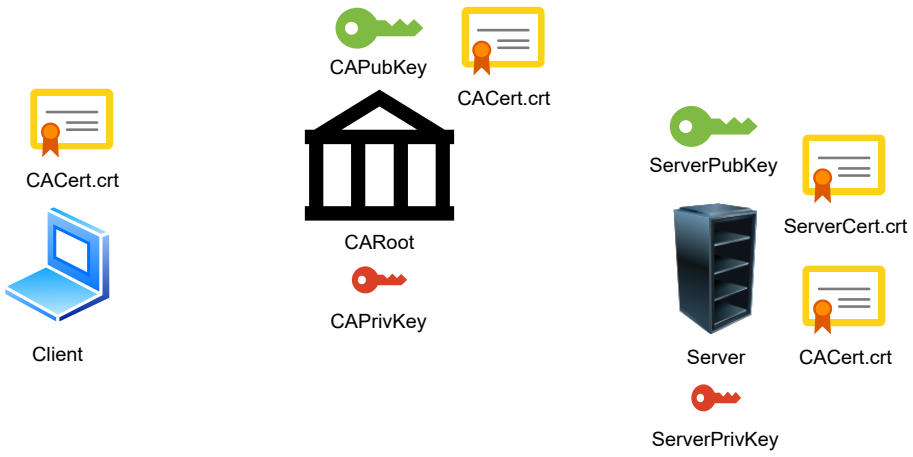

Figure 1: Cryptographic security elements for each actor

1. Please follow these steps to begin your practical work:
   - Launch the Debian operating system or the Debian virtual machine.
   - Open a ***Terminal***.
   - Enter the UNIX command ***sudo su*** (default password : root) or ask your Lab supervisor to give you the right password.
   - Create a new folder named ***LAB3*** and access this folder.
   - Create a new folder named ***CARoot***.
   - Create a new folder named ***Server***.
   - Create a new folder named ***Client***.
2. The scenario for this exercise is as follows (see Figure 1):
   - We have a root certificate authority named ***CARoot***.
   - We have a ***Server*** and a ***Client***.

- 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)**.
- 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)**.
- The ***Client*** does not have any key.
- 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 last session) → **(0.5 pt)**.
- ***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)**.
- ***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)**.

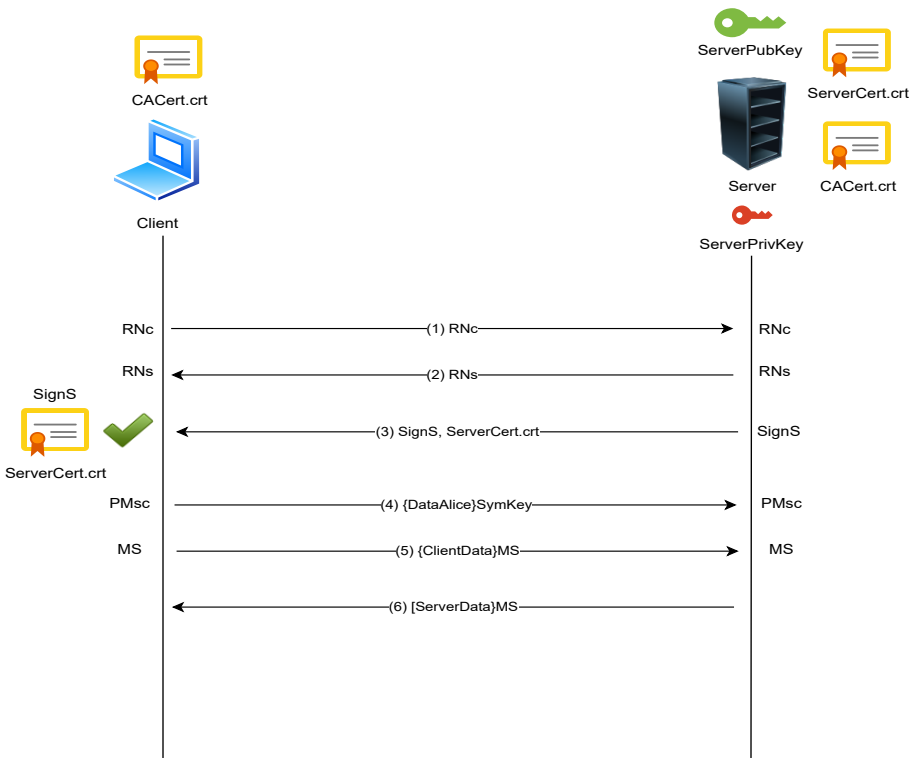

Figure 2: TLS Protocol (Messages exchanged)

2 Exercise "TLS Protocol" → (7.5 points)

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

Nouvelles Technologies et Société
Introduction to Cybersecurity
Nour El Madhoun **LAB 3** nour.el-madhoun@epita.fr

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 last session) → **(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***:
   - Make a concatenation of ***RNc*** and ***RNs*** by entering the UNIX command: ***cat RNc RNs > RNcRNs***
   - * Apply the hash function ***SHA256*** on ***RNcRNs*** to find its hash ***HashRNcRNs*** → **(0.5 pt)**.
   - * We ask you to proceed to generate ***SignS*** thanks to ***ServerPrivKey*** → **(0.5 pt)**.
   - * You can send now ***SignS*** and ***ServerCert.crt*** to the ***Client*** → **(0.25 pt)**.
   - The ***Client*** needs to verify ***ServerCert.crt***. * We ask you to verify it as you did in the last session → **(0.5 pt)**.
   - 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 for verifying a signature that 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***:
   - * We ask you to generate ***PMsc*** → **(0.5 pt)**.
   - * Encrypt ***PMsc*** thanks to ***ServerPubKey*** by naming the encrypted symmetric key ***PMscEncrypted*** → **(0.25 pt)**.
   - * Send to the ***Server PMscEncrypted*** → **(0.25 pt)**.
   - * Decrypt ***PMscEncrypted*** thanks to ***ServerPrivateKey*** by naming the decrypted key ***PMsc*** → **(0.5 pt)**.
5. Both the ***Client*** and ***server*** will generate the symmetric Master key ***MS*** which is calculated from the hash of ***PMsc***, ***RNc*** and ***RNs***:
   - * We ask you to calculate ***MS*** → **(0.75 pt)**.
   - * We ask you to create the file ***ClientData*** and encrypt it thanks to ***MS*** → **(0.5 pt)**.
   - * Send the encrypted result to the ***Server*** and decrypt it → **(0.25 pt)**.
6. The ***Server*** will answer to the ***Client***:
   - * Create the file ***ServerData*** and encrypt it thanks to ***MS*** → **(0.5 pt)**.
   - * Send the encrypted result to the ***Client*** and decrypt it → **(0.25 pt)**.