

II.2317 Cybersecurity (ISEP)

LAB 1

Nour El Madhoun

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 Introduction to OpenSSL

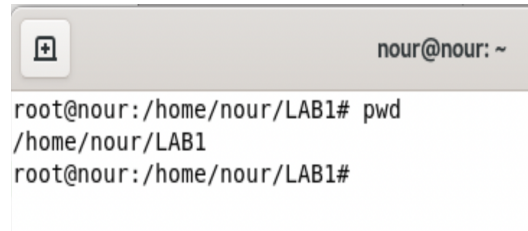
1. OpenSSL is a cryptographic toolkit implementing the SSL/TLS protocol. It offers a programming library in C to build secure client/server applications based on SSL/TLS protocol. An OpenSSL online command allows the creation of keys, the creation of a digital certificate, the calculation of a hash, the asymmetric encryption and decryption, etc. You can find all the information about OpenSSL at: <https://www.openssl.org/>
2. The general syntax of an *openssl* command is: \$ ***openssl command option***
3. The OpenSSL commands to use in this Lab are:
 - ***genrsa***: allows to generate a key pair (private key and public key)
 - ***rsautl***: allows to perform asymmetric encryption and decryption. It also allows to sign data and to verify them.
 - ***dgst***: allows to generate a hash.
 - ***rsa***: allows to extract the public key from a file that contains a key pair
4. An OpenSSL command has several options. The options of each command are very important to consult. For this reason, in order to consult the options of the command:
 - ***genrsa***, please use the command: ***openssl genrsa -help***
 - ***rsautl***, please use the command: ***openssl rsautl -help***
 - ***dgst***, please use the command: ***openssl dgst -help***
 - ***rsa***, please use the command: ***openssl rsa -help***
5. Please follow these steps to begin your practical work:
 - (a) Open a ***Terminal***.
 - (b) Create a new folder named ***LAB1*** by entering the UNIX command: ***mkdir LAB1***
 - (c) Access this folder by entering the UNIX command: ***cd LAB1***

II.2317 Cybersecurity (ISEP)

LAB 1

Nour El Madhoun

nour.el-madhoun@isep.fr

A terminal window with a title bar that says "nour@nour: ~". The terminal shows a sequence of commands and their outputs: first, "pwd" is entered, and the output is "/home/nour/LAB1"; then, another "pwd" is entered, and the output is the same: "/home/nour/LAB1".

```
nour@nour: ~  
root@nour:/home/nour/LAB1# pwd  
/home/nour/LAB1  
root@nour:/home/nour/LAB1#
```

Figure 1: Path to the folder "LAB1"

- (d) In section 2, you will need to use the path to a folder. For example, we illustrate in Figure 1 the path to our folder **LAB1** which is: `/home/nour/LAB1`. In order to retrieve the path to your folder **LAB1**, enter the UNIX command: `pwd`
 - (e) Create a new folder named **Alice** by entering the UNIX command: `mkdir Alice`
 - (f) Create a new folder named **Bob** by entering the UNIX command: `mkdir Bob`
 - (g) Access **Alice**'s folder by entering the UNIX command: `cd Alice`
 - (h) In order to retrieve the path to your folder **Alice**, enter the UNIX command: `pwd`
 - (i) Create the file **AliceDocument** by entering the UNIX command: `gedit AliceDocument`
 - (j) Write a text of your choice, for example: *Hello Bob, I'm Alice*
 - (k) Save the changes and return to the **Terminal**.
6. Make sure you are in **Alice**'s folder. We want to create a key pair for **Alice** and for this reason we will use especially the OpenSSL command `genrsa` as follows:
 - (a) Enter the OpenSSL command: `openssl genrsa -out AliceKeyPair`
 - (b) The file **AliceKeyPair** contains a pair of keys for **Alice** "a public key and a private key". Check that the file has been created correctly by entering the UNIX command: `ls`
 7. **Alice**'s public key is public and therefore it can be communicated to anyone. On the other hand, the file **AliceKeyPair** contains the private key too, so this file cannot be communicated to anyone and it must remain secret. For this reason, we will use an OpenSSL command to extract **Alice**'s public key and store it in another file.
 - (a) Enter the OpenSSL command: `openssl rsa -in AliceKeyPair -pubout -out AlicePublicKey`
 - (b) For more information about the options: `-in`, `-pubout`, `-out`, please use the command: `openssl rsa -help`
 - (c) The file **AlicePublicKey** contains the public key of Alice. Check that the file has been created correctly by entering the UNIX command: `ls`
 - (d) Now the file **AliceKeyPair** contains only the private key of Alice and then we can rename it **AlicePrivateKey** by entering the UNIX command: `mv AliceKeyPair AlicePrivateKey`
 - (e) Check that the file has been renamed correctly by entering the UNIX command: `ls`
 - (f) If you want to see the content of the files **AlicePublicKey** and **AlicePrivateKey**, please use the UNIX commands:
 - `gedit AlicePublicKey`
 - `gedit AlicePrivateKey`

2 Exercise "Asymmetric cryptography" → (10 points)

In this exercise, the interest is to encrypt a document with a public key and decrypt it thanks to the private key.

1. We want to do for **Bob** the same steps we did for **Alice**:
 - (a) You have to go back to the parent folder **LAB1** by entering the UNIX command: `cd ..`
 - (b) You need to access **Bob**'s folder by entering the UNIX command: `cd Bob`
 - (c) * Now, we ask you to repeat the same steps for **Bob** as we did for **Alice**. At the end, you must have two files for **Bob**: **BobPublicKey** and **BobPrivateKey** → (1.5 pt).
2. **Alice** wants to encrypt **AliceDocument** thanks to **Bob**'s public key, for this reason you need to proceed as follows:
 - (a) Make sure you are in **Bob**'s folder. In order to retrieve the path to your folder **Bob**, enter the UNIX command (see section 1): `pwd`
 - (b) **Alice** does not have **Bob**'s public key, and then you must copy **Bob**'s public key to **Alice**'s folder by entering the command:
`cp /home/UserName/LAB1/Bob/BobPublicKey /home/UserName/LAB1/Alice/`
 - (c) * Go back to the parent folder **LAB1** and access **Alice**'s folder → (0.25 pt).
 - (d) * Check that **BobPublicKey** has been copied correctly to **Alice**'s folder → (0.25 pt).
 - (e) You will now proceed to encrypt **AliceDocument** thanks to **BobPublicKey** by naming the encrypted document **AliceDocumentEncrypted**. To do this, please enter the OpenSSL command:
`openssl rsautl -encrypt -in AliceDocument -pubin -inkey BobPublicKey -out AliceDocumentEncrypted`
 - (f) For more information about the options: `-encrypt`, `-in`, `-pubin`, `-inkey`, `-out`, please use the command: `openssl rsautl -help`
 - (g) * Check that **AliceDocumentEncrypted** has been created correctly → (0.25 pt).
 - (h) * Check the content of the file **AliceDocumentEncrypted**. What do you notice? → (0.25 pt).
3. The objective in this step is to decrypt **AliceDocumentEncrypted** thanks to **BobPrivateKey**:
 - (a) * Make sure you are in **Alice**'s folder. We ask you to copy **AliceDocumentEncrypted** to **Bob**'s folder → (0.5 pt).
 - (b) * Go back to the parent folder **LAB1** and access **Bob**'s folder → (0.25 pt).
 - (c) * Check that **AliceDocumentEncrypted** has been copied correctly to **Bob**'s folder → (0.25 pt).
 - (d) * You will now proceed to decrypt **AliceDocumentEncrypted** thanks to **BobPrivateKey** by naming the decrypted document **AliceDocumentDecrypted**. To do this, please use the OpenSSL command `openssl rsautl` and the options: `-decrypt`, `-in`, `-inkey`, `-out` → (2 pt).
 - (e) * Check that **AliceDocumentDecrypted** has been created correctly → (0.25 pt).
 - (f) * Check the content of the file **AliceDocumentDecrypted**. What do you notice? → (0.25 pt).
4. The objective of this step is to show you that asymmetric encryption cannot be applied to large files:
 - (a) * Go back to the parent folder **LAB1** and access **Alice**'s folder → (0.25 pt).

II.2317 Cybersecurity (ISEP)

LAB 1

Nour El Madhoun

nour.el-madhoun@isep.fr

- (b) Create a file with large size named **LargeFile** by entering the OpenSSL command:
`openssl rand -out LargeFile -base64 $((230 * 3/4))`**
 - (c) * Try now to encrypt **LargeFile** by using **BobPublicKey** and by naming the encrypted file **LargeFileEncrypted** → (0.5 pt).
 - (d) You will notice that asymmetric encryption is not possible on large files. This is because asymmetric cryptography is very slow and requires a lot of resources. Asymmetric encryption is a very expensive encryption in computing power and is therefore made to encrypt small amounts of data. This is why it is recommended to use symmetric encryption for large files. For more information on the comparison between the two cryptographies, please visit: [Link 1](#), [Link 2](#)
5. The objective of this step is to show you how **Alice** can generate an electronic signature in order to: authenticate herself to **Bob**, ensure the non-repudiation for herself and guarantee the integrity of the signed data.
- (a) * Make sure you are in **Alice's** folder. Create a file named **AuthData** and write a text of your choice → (0.25 pt).
 - (b) * Check that **AuthData** has been created correctly → (0.25 pt).
 - (c) * Copy **AlicePublicKey** to **Bob's** folder → (0.25 pt).
 - (d) You will now proceed to apply the hash function **SHA256** on the document **AuthData** to find its hash **HashAuthData**. To this this, please enter the OpenSSL command: **`openssl dgst -sha256 -out HashAuthData AuthData`**
 - (e) * Check that **HashAuthData** has been created correctly → (0.25 pt).
 - (f) * Check the content of **HashAuthData** → (0.25 pt).
 - (g) * You will now proceed to sign **HashAuthData** thanks to **AlicePrivateKey** by naming the signature **AliceSignature**. To do this, please use the OpenSSL command **`openssl rsautl`** and the options: **`-sign, -in, -inkey, -out`** → (2 pt)
 - (h) For more information about the option: **`-sign`**, please use the command: **`openssl rsautl -help`**
 - (i) Check that **AliceSignature** has been created correctly.
 - (j) Copy **AliceSignature** and **AuthData** to **Bob's** folder.
 - (k) Go back to the parent folder **LAB1** and access **Bob's** folder.
 - (l) Check that **AliceSignature** and **AuthData** have been copied correctly to **Bob's** folder.
 - (m) You will now proceed to verify **AliceSignature** thanks to **AlicePublicKey**. To do this, you need to:
 - Retrieve **HashAuthData** by entering the OpenSSL command: **`openssl rsautl -verify -in AliceSignature -pubin -inkey AlicePublicKey -out HashAuthData`**
 - Check that **HashAuthData** has been retrieved correctly.
 - Calculate a new hash **HashBob** on **AuthData** thanks to the hash function **SHA256**, by entering the OpenSSL command: **`openssl dgst -sha256 -out HashBob AuthData`**
 - Check that **HashBob** has been created correctly.
 - Compare **HashAuthData** with **HashBob** by entering the UNIX command: **`diff HashBob HashAuthData`**
 - (n) You will notice that the **diff** command gives you no return (null) and this is because the two hashes are equal. Therefore, **Bob** will confirm the authenticity of **Alice**, non-repudiation for **Alice** and the integrity of **AuthData**.
 - (o) You can try changing some characters in **HashBob** or **HashAuthData** and re-run the **diff** command to see the difference.