# TP Class Assignment - 11/Feb/2019

Each group must answer the questions of the following exercises in the Github area of their group until the end of 19/Fev/2018. For each day of delay, 0.15 points will be deducted from the grade of this assignment.

Note that the virtual machine provided for this course can be used to solve these exercises.

## Exercises

### 1. Random / pseudorandom numbers

#### Experience 1.1

Run the following command, which generates 1024 pseudorandom bytes: `openssl rand -base64 1024`

#### Question P1.1

Test the following commands, which will get 1024 pseudorandom bytes of the system and present them in base64:

- `head -c 32 /dev/random | openssl enc -base64`
- `head -c 64 /dev/random | openssl enc -base64`
- `head -c 1024 /dev/random | openssl enc -base64`
- `head -c 1024 /dev/urandom | openssl enc -base64`

What conclusions can you draw? What are the basis for these conclusions?

#### Question P1.2

The haveged - http://www.issihosts.com/haveged/index.html - is an entropy daemon adapted from the HAVEGE algorithm (*HArdware Volatile Entropy Gathering and Expansion*) - http://www.irisa.fr/caps/projects/hipsor/ -.

Install the package haveged on the virtual machine with the following command: `sudo apt-get install haveged`.

Re-test the following commands, which will get 1024 pseudorandom bytes from the system and display them in base64:

- `head -c 1024 /dev/random | openssl enc -base64`
- `head -c 1024 /dev/urandom | openssl enc -base64`

What conclusions can you draw? What are the basis for these conclusions?

#### Experience 1.2

The example using *java.security.SecureRandom* seen in class can be found in the class directory (Aula2/PseudoAleatorio) in the *RandomBytes.java* file.

Review, compile, and run this example.

Note: If you have problems with the javac or javac version run the following commands:

- `sudo update-alternatives --config java`
- `sudo update-alternatives --config javac`

and choose the 9/Oracle version.

### Question P1.3

In the class directory (Aula2/PseudoAleatorio) you will find the *generateSecret-app.py* file based on the eVotUM.Cripto module (https://gitlab.com/eVotUM/Cripto-py) already installed on the virtual machine in /home/user/API/Crypto-py/eVotUM/Crypto.

1. Analyze and execute this random secret generator program and show the reason why the output only contains letters and digits (not containing punctuation characters or others).
2. What would you have to do to not limit the output to letters and digits?

## 2. Secret Sharing/Splitting

### Experience 2.1

The example using the *genSharedSecret.php* seen in class, is in the class directory (Aula2/SecretSharing), in the file with the same name.

Review and run this example. Check what happens if you try to rebuild the secret (*reconstroiSecret.php*) with more or less than the expected components.

### Experience 2.2

The example using the *shares.pl* seen in class, is in the class directory (Aula2/SecretSharing), in the file with the same name.

Review and run this example. Check what happens if you try to rebuild the secret (*reconstruct.pl*) with more or less than the expected components.

### Question P2.1

In the class directory (Aula2/PseudoAleatorio) you will find the *createSharedSecret-app.py*, *recoverSecretFromComponents-app.py* and *recoverSecretFromAllComponents-app.py* files based on the eVotUM.Cripto module (https://gitlab.com/eVotUM/Cripto-py) already installed on the virtual machine in /home/user/API/Crypto-py/eVotUM/Crypto.

A. Analyze and execute these programs, showing what you had to do to divide the secret "Now we have an extremely confidential secret" in 8 parts, with a quorum of 5 to rebuild the secret.

Note that the use of this program is
`python createSharedSecret-app.py number_of_shares quorum uid private-key.pem` where:

- number_of_shares - parts in which you want to split the secret
- quorum - number of parts needed to rebuild the secret
- uid - identifier of the secret (so to ensure that when you rebuild the secret, you are providing the parts of the same

secret)
- private-key.pem - private key, since each part of the secret is returned in a base 64 JWT signed object

B. Describe also the difference between *recoverSecretFromComponents-app.py* and *recoverSecretFromAllComponents-app.py*, and in which situations you may need to use *recoverSecretFromAllComponents-app.py* instead of *recoverSecretFromComponents-app.py*.

Note: Remember that the generation of the key pair can be done with the command `openssl genrsa -aes128 -out mykey.pem 1024`. The corresponding certificate can be generated with the command `openssl req -key mykey.pem -new -x509 -days 365 -out mykey.crt`

## 3. Authenticated Encryption

Scenario:

> Your company wants to place a service on the market with the following characteristics:
>
> - encryption (with symmetric cipher) of secrets;
> - decryption of the previously encrypted secret;
> - the customer can decipher the secret(s) he encrypted during the time he pays the service;
> - in order the customer knows what he encrypted, he can label the secret.

> For this, your company has acquired specific cipher / decipher hardware, where the cipher key is automatically changed every day, being identified by "year.month.day". This hardware also performs HMAC_SHA256 and has an API with the following functions:
>
> - cipher (plaintext_secret), and returns cyphertext_secret
> - decipher (cyphertext_secret, cipher_key), and returns plaintext_secret or error
> - hmac (k, str), and returns HMAC_SHA256 of the str with the secret key k

### Question P3.1

Based on the scenario identified, how would you suggest to cipher and decipher the secret(s) in order to ensure confidentiality, integrity and authenticity of the secret and its label using *Authenticated Encryption*? Include in your answer the algorithm, which using the API, you would ask the development area of your company to implement in order to encrypt and decrypt the secret.

## 4. Algorithms and Key sizes

The website https://webgate.ec.europa.eu/tl-browser/ provides the list of Entities with qualified trust services, according to Regulation EU 910/2014 (eIDAS) - we will talk about this Regulation in another class.

Among these services is the digital qualified certificate issuing service, called "QCert for ESig".

### Question P4.1

Each group listed below should identify the algorithms and key sizes used in the certificates of the Certification

Authorities (CA) that issue qualified digital certificates, and verify if they are the most appropriate (and if not, propose the most appropriate ones):

- Group 1 - Austria, for 2 CAs that issue "QCert for ESig" certificates;
- Group 2 - Croácia, for 2 CAs that issue "QCert for ESig" certificates;
- Group 3 - France, for the CAs "Ministère de la Justice", "Imprimerie Nationale";
- Group 4 - Hungary, for 2 CAs that issue "QCert for ESig" certificates;
- Group 5 - Italy, for the CAs "Banca d'Italia", "Ministero della Difesa";
- Group 6 - Lituania, for 2 CAs that issue "QCert for ESig" certificates;
- Group 7 - Netherland, for the CAs "Ministerie van Defensie", "QuoVadis Trustlink B.V.";
- Group 8 - Portugal, for the CAs "Cartão de Cidadão", "ECCE";
- Group 9 - Eslovénia, for the CAs "Ministry of Defence of Slovenia", "Republika Slovenija";
- Group 10 - Belgium, for the CAs "Zetes S.A./N.V.", "Portima s.c.r.l. c.v.b.a.";
- Group 11 - Germany, for the CAs "D-Trust GmbH", "Deutscher Sparkassen Verlag GmbH";
- Group 12 - Norway, for the CAs "Nordea Bank Norge ASA", "Statoil ASA";
- Group 13 - Romania, for the CAs "CENTRUL DE CALCUL SA", "AlfaTrust Certification S.A.";
- Group 14 - Spain, for the CAs "Dirección General de la Policía", "Fábrica Nacional de Moneda y Timbre";
- Group 15 - Bulgary, for the CAs "Information Services Plc.", "BORICA AD";
- Group 16 - Chec Republic, or 2 CAs that issue "QCert for ESig" certificates.

Note 1: For Certification Authorities that already have multiple CA certificates, consider only the last certificate issued.

Note 2: To obtain the size of the keys and algorithms used, you should:

1. choose the CA certificate,
2. select *Base 64-encoded*,
3. copy the contents of the *Base 64-encoded* and write to file (e.g. cert.crt),
4. Insert ----- BEGIN CERTIFICATE ----- at the beginning of the file,
5. Insert ----- END CERTIFICATE ----- at the end of the file,
6. Run the following command `openssl x509 -in cert.crt -text -noout` (replace cert.crt with the name you gave the file in step 3.)