# Laborator 4

# Certificate DigitAle și semnături

**Crearea propriei autorități de certificare**

Este o operație necesară fie pentru dezvoltatori în faze de testare internă sau chiar pentru administratori atât timp cât se referă numai la un arbore de certificare intern organizației și nu este utilizat pentru a accepta conexiuni din afara acesteia. Denumirea corectă conform limbii engleze și nu numai este de autoritate de certificare totuși datorită sublimbajului ingineresc care utilizează certificate authority ce este o ambiguitate pentru că poate fi interpretat fie ca produsul autorității adică certificatul de autorizare fie ca autoritate pentru certificate - cel mai des sens asociat.

După cum am discutat la curs aceste autorități refera o entitate (companie sau organizație) care girează că identitatea electronică a unei terțe părți este cea declarată. Pentru aceasta se asociază informațiilor descriptive o pereche de chei iar rezultatul (conform lui X509 - poate îl citiți totuși) este un certificat digital.

Pentru cazuri de producție se vor folosi abordările criptografice discutate la curs. Aici este un exemplu simplu pentru a înțelege fluxul de operații necesar creării unei astfel de autorități. Vom folosii openssl pentru demonstrație. Dacă Linux este sistemul de operare de bază (mai ales cei care aveți laptop-uri configurate de la firmă este recomandat să faceți aceste teste într-o mașină virtuală pentru a nu risca să vă deranjați o posibila configurare preexistentă.

**Pasul 1 - crearea cheii private**

O metodă rapidă este prin utilizarea comenzii genrsa iar în exemplu se va utiliza des3 (triple des) atât de drag unor abordări bancare dar care nu prea oferă mare securitate în lumea reala. Lungimea cheilor va fi de 2048.

```
openssl genrsa -des3 -out CertDPrivat.key 2048
```

Se va cere o parola PEM utilizată pentru criptarea cheii private așa că nu trebuie să o pierdeți/uitați în nici un caz. Ca efect se va genera fișierul CertDPrivat.key. În nici un caz nu utilizați fără parolă! PEM vine de la "Privacy Enhanced Mail" care este o formă veche utilizată în posta electronica a lui X509.

**Pasul 2 - crearea certificatului primar (rădăcina arborelui de certificate)**

După cum am discutat la curs în caz real acesta este bine să fie furnizat de o autoritate de certificare serioasă (e.g Verisign etc) dar după cum am spus sunt situații în care acest lucru nu este necesar. Motivația sunt costurile asociate pe de o parte și pericolul propagării compromiterii în cazul în care se abuzează de un sigur certificat real pentru a genera mulți arbori interni sau externi de certificate. Pentru crearea acestui certificat primar.

```
openssl req -x509 -new -nodes -key CertDPrivat.key -sha256 -days 365 -out CertDPrivat.pem
```

```
root@test:~# openssl genrsa -des3 -out CertDPrivat.key 2048
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
root@test:~# openssl req -x509 -new -nodes -key CertDPrivat.key -sha256 -days
 365 -out CertDPrivat.key.pem
Enter pass phrase for CertDPrivat.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:Ro
State or Province Name (full name) [Some-State]:-
Locality Name (eg, city) []:Iasi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Hole S.R.L
Organizational Unit Name (eg, section) []:headquarters
Common Name (e.g. server FQDN or YOUR name) []:sefii
Email Address []:conan@holeltd.com
root@test:~#
```

Se observă din analiza comenzii că fișierul de certificare a fost creat conform standard-ului X509 și va fi valid un an (nu uitați analiza de la curs pe subiectul duratei certificatelor). Acum se poate trece la testarea acestuia adică să generăm un certificat bazat pe acest certificat primar.

**Pas 3 - generarea unei cereri de semnare a certificatului propriuzise certificate signing request (CSR)**
Ca și mai înainte avem nevoie de o cheie privată (a noii identitati validate de către acest certificat)
```
openssl genrsa -out CheiaMeaPrivata.key 2048
```
După cum se vede se va folosi RSA pentru a o genera. Acum putem trece la generarea cererii care va conduce la obținerea unui certificat pentru noi și care va utiliza aceasta cheie privată.
```
openssl req -new -key CheiaMeaPrivata.key -out CerereCertificatPersonal.csr
```
Se mai poate da o parolă de verificare (challenge) și un nume de companie dar este opțional (dacă însă există în schema organizațională trebuie totuși pus! Acum trebuie să trimitem cererea către autoritatea de certificare pentru a obține fișierul care conține certificatul.

**Pas 4 - obtinerea certificatului de la autoritatea de certificare**
Acum se va crea un certificat tip X509 pe baza cererii. Vom stabili de această dată o durată de o lună până la expirarea acestuia. În cadrul procesului vom utiliza certificatul primar și cheia lui privată unde se va cere parola pentru a o putea decripta.
```
openssl x509 -req -in CerereCertificatPersonal.csr -CA CertDPrivat.pem -CAkey
CertDPrivat.key -CAcreateserial -out CertificaPrivatPersonal.crt -days 30 -sha256
```

**Pasul 5 - utilizarea/testarea certificatului personal proaspăt generat**
Aici cele mai comune utilizări sunt importarea certificatului în lista de certificate de încredere care este menținută de către sistemul de operare sau chiar folosirea lui într-un server web. Vom prezenta un exemplu de utilizare a certificatului pentru a verifica un proces de semnare. Pentru început vom semna un text oarecare utilizând cheia privata a utilizatorului.
```
openssl dgst -sha256 -sign CheiaMeaPrivata.key -out semnatura.txt text_oarecare.txt
```
Fișierul semnătură.txt va reține semnătura asociata conținutului fișierului text_oarecare.txt.
Acum putem trece la verificarea acestei semnături utilizând certificatul utilizatorului. Pentru început vom încărca certificatul X509 în openssl și apoi vom face verificarea propriuzisă.
```
openssl x509 -in CertificaPrivatPersonal.crt
```

```
root@test:~# openssl x509 -in CertificaPrivatPersonal.crt
-----BEGIN CERTIFICATE-----
MIIDozCCAosCFARoSyUoOnNEEYeM3cTUkq4WjgT0MA0GCSqGSIb3DQEBCwUAMIGH
MQswCQYDVQQGEwJSbzEKMAgGA1UECAwBLTENMAsGA1UEBwwESWFzaTEUMBIGA1UE
CgwLSG9sZSBTLlIuTCAxFTATBgNVBAsMDGhlYWRxdWFydGVyczEOMAwGA1UEAwwF
c2VmaWkxIDAeBgkqhkiG9w0BCQEWEWNvbmFuQGhvbGVsdGQuY29tMB4XDTIxMDky
MTA5NTAxMFoXDTIxMTAyMTA5NTAxMFowgZMxCzAJBgNVBAYTAlJvMQowCAYDVQQI
DAEtMQ0wCwYDVQQHDARJYXNpMRQwEgYDVQQKDAtIb2xlIFMuUi5MLjEUMBIGA1UE
CwwLbWFpbnRlbmFuY2UxGDAWBgNVBAMMD3BhbG1hcyBwZSB0YXJsYTEjMCEGCSqG
SIb3DQEJARYUamFuaXRvcjFAaG9sZWx0ZC5jb20wggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQCpas4uOvE11MT9/M/qxvQ+yi6BVff2jNlY7tsdab1TUqnn
8FYDCMfyGjF+rDamqiCbAi0MA6zksunIxZdABA4m/POaUV/35m5eIFq6bTvbGBu6
```

Evident nu am dat tot certificatul (oare de ce?). Și acum putem realiza verificarea
```
openssl dgst -sha256 -verify <(openssl x509 -in CertificaPrivatPersonal.crt -pubkey
-noout) -signature semnatura.txt text_oarecare.txt
```
```
root@test:~# openssl dgst -sha256 -verify <(openssl x509 -in CertificaPrivatP
ersonal.crt -pubkey -noout) -signature semnatura.txt text_oarecare.txt
Verified OK
root@test:~#
```

Acum apelați un nano pentru a efectua o modificare arbitrară în fișierul text_oarecare.txt și reluați procesul.

```
root@test:~# nano text_oarecare.txt
root@test:~# openssl dgst -sha256 -verify <(openssl x509 -in CertificaPrivatP
ersonal.crt -pubkey -noout) -signature semnatura.txt text_oarecare.txt
Verification failure
```

În cazuri de producție certificatele ar fi bine să fie ținute într-o zonă sigură sau chiar într-un modul specializat extern (sau intern) HSM ( Hardware Security Module ) nu de alta dar o să vedem la exfiltrarea din cadrul fazei de exploatare ce repede poți să le găsești și să le aduci pentru analiza pe îndelete.

**Automatizarea semnării unui certificat utilizând o autoritate de certificare existentă cu ajutorul Python**

Pentru cei care sunt deja obișnuiți cu lucrul cu certificatele în Linux există niște biblioteci wrapper în Python care permit automatizarea oricărui proces de acest tip.
În exemplul de mai jos am utilizat modulul PyOpenSSL. Ca și în exemplul anterior este recomandat a nu se utiliza în producție ceea ce va fi prezentat (dar sunt un punct de plecare). Dacă ați înțeles documentația de X509 veți vedea că o implementare completă conform standardului ar fi condus la ceva fără eficiență academică.
Pentru a vedea certificatele generate (Pycharm-ul vi le pune automat la dispoziție în zona fișierelor proiect) dar se poate utiliza și la nivel de consolă.

```
from OpenSSL import crypto, SSL
import time
def gen_cert():
    #cream perechea de chei
    k = crypto.PKey()
    k.generate_key(crypto.TYPE_RSA, 4096)
    # cream certificatul autosemnat
    cert = crypto.X509()
    cert.get_subject().C = numeTara
    cert.get_subject().ST = statUSsauProvincie
    cert.get_subject().L = numeLocalitate
    cert.get_subject().O = numeOrganizatie
    cert.get_subject().OU = numeDepartament
    cert.get_subject().CN = numePrenume
    cert.get_subject().emailAddress = adresaEmail
    cert.set_serial_number(numarInregistrare)
    cert.gmtime_adj_notBefore(0)
    unixTime=int(time.time())
    dataExpirarii=unixTime+secundeTimpExpirare
    cert.gmtime_adj_notAfter(dataExpirarii)
    cert.set_issuer(cert.get_subject())
    cert.set_pubkey(k)
    cert.sign(k, 'sha512')
    with open(CERT_FILE, "wt") as f:
        f.write(crypto.dump_certificate(crypto.FILETYPE_PEM, cert).decode("utf-8"))
    with open(KEY_FILE, "wt") as f:
        f.write(crypto.dump_privatekey(crypto.FILETYPE_PEM, k).decode("utf-8"))

if __name__=="__main__":
    adresaEmail = input('Adresa de e-mail:')
    numePrenume = input('Numele:')
    numeTara = input('Numele Tarii: ')
    numeLocalitate= input('Numele Localitatii: ')
    statUSsauProvincie= input('NUme Stat (pt US) sau provincie: ')
    numeOrganizatie = input ('Numele organizatiei: ')
    numeDepartament = input('Numele Departamentului: ')
    numarInregistrare = int(input('Numarul de serie: '))
    zileNumarExpirare = int(input('Cate zile este valid certificatul? '))
    secundeDeCandDevineValid=0
    secundeTimpExpirare=zileNumarExpirare*864000
```

```
    KEY_FILE = "cheiePrivata.key"
    CERT_FILE="certificatAutosemnat.crt"
    gen_cert()
```

Acum că ați înțeles primul exemplu se poate trece la următorul un pic mai complicat.

```
from cryptography import x509
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.x509.oid import NameOID
import datetime
import uuid
if __name__=="__main__":
    numePrenume = input('Numele si Prenumele detinatorului:')
    numeEmitent = input('Numele si Prenumele emitentului:')
    anulExpirarii = int(input("Anul Expirarii: "))
    lunaEXpirarii = int(input("Luna Expirarii: "))
    ziuaExpirarii = int(input("Ziua Expirarii: "))
    numeOrganizatie = input('Numele organizatiei: ')
    numeDepartament = input('Numele Departamentului: ')
    one_day = datetime.timedelta(1, 0, 0)
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    builder = x509.CertificateBuilder()
    builder = builder.subject_name(x509.Name([
        x509.NameAttribute(NameOID.COMMON_NAME, numePrenume),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, numeOrganizatie),
        x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, numeDepartament),
    ]))
    builder = builder.issuer_name(x509.Name([
        x509.NameAttribute(NameOID.COMMON_NAME, numeEmitent),
    ]))
    builder = builder.not_valid_before(datetime.datetime.today() - one_day)
    builder = builder.not_valid_after(datetime.datetime(anulExpirarii, lunaEXpirarii, ziuaExpirarii))
    builder = builder.serial_number(int(uuid.uuid4()))#parca suna mai bine sa-l cerem de la sistem nu?
    builder = builder.public_key(public_key)
    builder = builder.add_extension(
        x509.BasicConstraints(ca=True, path_length=None), critical=True,
    )
    certificate = builder.sign(
        private_key=private_key, algorithm=hashes.SHA256(),
        backend=default_backend()
    )
    print(isinstance(certificate, x509.Certificate))

    with open("CheiaCertificatului.key", "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.BestAvailableEncryption(b"dupa ureche ca de obicei")
        ))
```

```
    with open("CertificatulPropriuzis.crt", "wb") as f:
        f.write(certificate.public_bytes(
            encoding=serialization.Encoding.PEM,
        ))
```

**Tema pornind de la exemplele prezentate scrieți un program Python care să permită criptarea și semnarea unui fișier utilizând certificate.**

**PGP**

Deoarece Pretty Good Privacy (PGP) este o alternativă decentă la abordările bazate pe RSA va trebui studiat și acesta. În caz că nu îl folosiți sau lucrați în mașină virtuală atunci trebuie început prin a instala suportul necesar la nivelul sistemului de operare. Cred că deja v-ați obișnuit ca înainte de fiecare instalare să dați un upgrade al sistemului de operare deci

```
apt update
apt upgrade
```

Apoi trecem la instalarea GNU Privacy Guard GnuPG care este o implementare completă și gratuită a standardului OpenPGP definit în RFC4880 (referit de multe ori ca PGP). Despre PGP original ați studiat deja la alte materii deci nu mai insist.

```
apt install gnupg
```

Ca și în celelalte cazuri se începe prin a crea o pereche de chei

```
gpg --gen-key
```

Nu uitați - să nu pierdeți parola. Dacă o uitați sau cheia secretă a fost compromisă este bine să creăm și o cheie pentru revocare

```
gpg --output ~/revocation.crt --gen-revoke your_email@address.com
```

```
Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 1
Enter an optional description; end it with an empty line:
> i was dumb
>
Reason for revocation: Key has been compromised
i was dumb
Is this okay? (y/N) y
ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

Trebuie apoi restricționat accesul la respectivul certificat

```
chmod 600 ~/revocation.crt
```

Dacă am nevoie să import cheile publice de la alții (necesare pentru a decripta sau verifica nonrepudierea unui mesaj/fișier primit de la aceștia atunci

```
gpg --import name_of_pub_key_file
```

Dacă nu mi s-a furnizat cheia publică dar omul le are păstrate pe un server atunci le pot găsi direct prin căutarea în respectivul server

```
gpg --keyserver nume_server_key_pgp  --search-keys parametri_cautare
```

**Verificarea și semnarea cheilor**

După cum am văzut cheile mele publice le pot distribui public sau măcar să furnizez o manieră de acces la acestea astfel încât ceilalți să mă poată contacta într-o manieră cât de cât sigură. Totuși este important să existe o manieră de verificare a identității celui care își asuma o cheie. Cea mai rapidă metodă de verificare a

cheilor între părți peste un canal nesigur este să le comparam semnăturile asociate (amprentele). Acestea se obțin imediat cu

```
gpg --fingerprint your_email@address.com
```

```
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2023-09-22
pub   rsa3072 2021-09-22 [SC] [expires: 2023-09-22]
      9027 C3B2 8809 25BD AC54  679A CDEC EED1 5765 C20D
uid           [ultimate]
sub   rsa3072 2021-09-22 [E] [expires: 2023-09-22]
```

Acum să trecem la semnarea cheii

```
gpg --sign-key email@address.com
```

Trebuie să permiteți persoanei a cărei cheie o semnați să beneficieze de această relație de încredere prin trimiterea înapoi a cheii semnate

```
gpg --output ~/signed.key --export --armor email@example.com
```

Se va cere parola din nou. Dacă doresc să pun la dispoziția publicului larg cheia publică sistemul pgp poate face și acest lucru și anume

```
gpg --output ~/mygpg.key --armor --export your_email@address.com
```

Iar apoi fișierul generat poate fi pus unde doriți în zona publică. Criptarea și decriptarea mesajelor cu gpg sunt relativ simple de exemplu pentru criptare

```
gpg --encrypt --sign --armor -r person@email.com name_of_file
```

Acesta comandă va cripta mesajul utilizând cheia publică a destinatarului, o semnează cu cheia voastră privată pentru a garanta nonrepudierea și generează un fișier text cu extensia .asc. Parametrul -r este necesar în caz că doriți să puteți citi mesajul criptat. Pentru decriptarea unui mesaj procesul este cam la fel

```
gpg file_name.asc
```

Dacă nu este un fișier ci numai un mesaj text acesta poate fi direct copiat în linia de comandă după gpg apoi la Ctrl+D gpg va considera EOF și va trece la decriptare

**Crearea unui server ssh**

```
apt install openssh-server
```

Activarea serverului

```
systemctl enable ssh
```

Dezactivarea serverului - eu nu recomand acces extern decât dacă este musai

```
systemctl disable ssh
```

Pornirea serviciului ssh

```
systemctl start ssh
```

Introducerea regulilor asociate în firewall

```
ufw allow ssh
ufw enable
ufw status
```

Sau o puteți face din gufw (pentru fiii windows-ului). În caz ca nu i-ați atribuit o cheie întâi o generăm

ssh-keygen

Apoi o instalează local sau într-un server la distanță (dăm pe a doua)

```
ssh-copy-id user@remote-server-ip-name
```

Acum pot utiliza

```
ssh user@remote-server-ip-name
```

Pentru serverul local se pot genera, cheile cu

```
ssh-keygen -t rsa
```

Aceasta va crea doua fișiere ~/.ssh/id_rsa : cheia privată și ~/.ssh/id_rsa.pub : cheia publică. Evident este bine ca parola să respecte toate regulile de creare despre care am mai discutat la curs și să nu o pierdeți. Dacă se dorește schimbarea parolei

```
ssh-keygen -p
```

Sau

```
cd ~/.ssh/
ssh-keygen -f id_rsa -p
```

Dacă nu am ce face și îmi trebuie server ssh atunci este necesar să dezactivăm autentificarea bazată pe parolă

```
nano /etc/ssh/sshd_config
```

Și modific câmpul

```
PasswordAuthentication no
```

Verificați că următoarele câmpuri au valorile care trebuie

```
PubkeyAuthentication yes
ChallengeResponseAuthentication no
```

Apoi salvați fișierul de configurare și

```
systemctl reload sshd
```

Pentru a accesa un astfel de server trebuie să îi știți adresa publică și parola. Dacă ați instalat o cheie SSH pentru autentificare mai este nevoie și de cheia privată pentru contul de root.

**Utilizarea GPG din Python**

Pentru aceste exemple vom utiliza modului python-gnupg și fs deci importați-le în proiectul PyCharm.

```
import os
import fs
from fs import open_fs
import gnupg

if __name__ == '__main__':
    gpg = gnupg.GPG(gnupghome="/home/nume_cont_root_sau_cu_drepturi/.gnupg")
    home_fs = open_fs(".")

    if os.path.exists("semnaturi/"):
        print("Exista directorul cu semnaturi")
    else:
        home_fs.makedir(u"semnaturi")
        print("Am creat directorul pentru semnaturi")

    files_dir = []

    files = [f for f in os.listdir(".") if os.path.isfile(f)]
    for f in files:
        files_dir.append(f)

    for x in files_dir:
        with open(x, "rb") as f:
            stream = gpg.sign_file(f,passphrase="parola mea",detach = True,
output=files_dir[files_dir.index(x)]+".sig")
            os.rename(files_dir[files_dir.index(x)]+".sig", "semnaturi/"+files_dir[files_dir.index(x)]+".sig")
            print(x+" ", stream.status)
```

Programul de mai sus ne ajută să creăm semnături separate pentru fișiere de test și o va face asupra fiecărui fișier găsit în directorul în care este executat. Este recomandabil să se lucreze cu un utilizator cu drepturi de root (vezi calea referită).

**Pentru criptarea fișierelor**

Se merge pe aceeași abordare simplă adică tot ce se află într-un director va fi criptat și salvat într-un director numit criptate. Cheia publică utilizată este cea asociată adresei de mail dată în program.

```
import os
import fs
from fs import open_fs
import gnupg

if __name__ == '__main__':
    gpg = gnupg.GPG(gnupghome="/home/user_cu_drepturi/.gnupg")
```

```
    home_fs = open_fs(".")

    if os.path.exists("criptate/"):
        print("Directorul cu fisere criptate exista")
    else:
        home_fs.makedir(u"criptate")
        print("Am creat directorul pentru fisierele criptate")

    files_dir = []

    files = [f for f in os.listdir(".") if os.path.isfile(f)]
    for f in files:
        files_dir.append(f)

    for x in files_dir:
        with open(x, "rb") as f:
            status = gpg.encrypt_file(f,recipients=["user_cu_drepturi@acasa.org"],output=
files_dir[files_dir.index(x)]+".gpg")
            print("ok: ", status.ok)
            print("status: ", status.status)
            print("stderr: ", status.stderr)
            os.rename(files_dir[files_dir.index(x)] + ".gpg", "encrypted/" +files_dir[files_dir.index(x)] +
".gpg")
```

Dacă există mai multe chei păstrate în directorul .gnupg și se dorește utilizarea unei anume chei atunci trebuie să modificați tabloul recipients fie prin adăugare fie prin înlocuirea referirii curente.

**Decriptarea fișierelor**

```
import os
import fs
from fs import open_fs
import gnupg
if __name__ == '__main__':
    gpg = gnupg.GPG(gnupghome="/home/looser_cu_drepturi/.gnupg")
    home_fs = open_fs(".")

    files_dir = []
    files_dir_clean = []

    if os.path.exists("decriptate/"):
        print("Exista directorul pentru fisierele decriptate")
    else:
        home_fs.makedir(u"decriptate/")
        print("Am creat directorul pentru fisierele decriptate")

    files = [f for f in os.listdir(".") if os.path.isfile(f)]
    for f in files:
        files_dir.append(f)

    for x in files_dir:
        length = len(x)
        endLoc = length - 4
        clean_file = x[0:endLoc]
        files_dir_clean.append(clean_file)
```

```
    for x in files_dir:
        with open(x, "rb") as f:
            status = gpg.decrypt_file(f, passphrase="parola
mea",output=files_dir_clean[files_dir.index(x)])
            print("ok: ", status.ok)
            print("status: ", status.status)
            print("stderr: ", status.stderr)
            os.rename(files_dir_clean[files_dir.index(x)], "decriptate/" +
files_dir_clean[files_dir.index(x)])
```

Se observă că se caută în directorul curent și adaugă tot ce găsește în tabloul files_dir. Apoi deoarece fișierele criptate au extensia .gpg aceasta este eliminată pentru fiecare fișier iar rezultatul este depus în file_dir_cleaned.

**Verificarea semnăturilor separate**

```
import os
import fs
from fs import open_fs
import gnupg
if __name__ == '__main__':
    gpg = gnupg.GPG(gnupghome="/home/utilizatir_cu_drepturi/.gnupg")
    home_fs = open_fs(".")

    files_dir = []

    files = [f for f in os.listdir(".") if os.path.isfile(f)]
    for f in files:
        files_dir.append(f)

    for i in files_dir:
        with open("../../semnaturi/" + i + ".sig", "rb") as f:
            verify = gpg.verify_file(f, i)
            print(i + " ", verify.status)
```

Este trivial nu am ce explica.

**Verificarea fișierelor**

```
import os
import sys
import fs
from fs import open_fs
import gnupg

if __name__ == '__main__':
    gpg = gnupg.GPG(gnupghome="/home/nume_cont_looser_cu_drepturi/.gnupg")
    home_fs = open_fs(".")

    script_to_run = str(sys.argv[1])

    with open("../../semnaturi/" + script_to_run + ".sig", "rb") as f:
        verify = gpg.verify_file(f, script_to_run)
        print(script_to_run + " ", verify.status)
        if verify.status == "semnatura valida":
            print("fisier ok lansez procesarea")
            exec(open(script_to_run).read())
        else:
```

```
            print("semnatura invalida sau lipseste")
            print("opresc procesarea")
```
La fel de simplu ca și în cazul anterior

**Testare**

Le fac executabile (scripturile Pyton)

```
chmod +x *.py
```

Acum doresc sa le myt in directorul cu acces general /usr/local/bin/

Dacă vreau să fiu pinguin adevărat adaug la începutul programelor

```
#!usr/bin/env python3
```

Această comandă va ajuta sistemul de operare să identifice programul care trebuie să le execute și apoi le scot extensia.py (nu mai este nevoie). În sfârșit le mut în respectivul director (/usr/local/bin/) și atunci pot fi executate de oriunde.

**Temă laborator - testați programele exemplu și creați o aplicație care monitorizează un director semnează fișierele de acolo și dacă un fișier a fost modificat detectează acest lucru și îl resemnează.**

**Tema acasă**

**De acum aveți un punct de plecare și suficiente cunoștințe astfel încât să schimbați RSA cu eliptice, să utilizați DH cu eliptice, să schimbați versiunea de pgp utilizată, sa modificați conform standardelor lungimile cheilor și ce vă mai vine prin cap pentru a creste mult securitatea respectivelor metode. Python este aproape pseudocod așa că de acum puteți rescrie la nivel de c/cpp/golang/rust/ruby/java/.net orice s-a prezentat.**