Wrocław 01.10.2022

C Y B E R S E C U R I T Y

L A B 7

1. Introduction.

The application of cryptographic methods is very wide. The laboratory focuses on aspects related to digital signatures and Public Key Infrastructure (PKI). Digital signatures are commonly used daily, so in a security context, it is important to become familiar with their operation and properties. A public-key infrastructure (PKI) consists of protocols, standards, and services that establish and support the applications of such a trust system.

Digital signature is mainly used for authentication purposes. It is used to convince communicating parties with each other's identity and exchange their session keys. It is an electronic format of signature that can be used by a person to authenticate the iden tity of the message's sender or identity of the document's signer. It ensures that the original content of the message or document sent is intact. In other words, it is a cryptographic tool for signing messages, data, and verifying the authenticity of the message data. Digital signatures use a standard, accepted format, called Public Key Infrastructure (PKI), to provide the highest levels of security and universal acceptance. Moreover, digital signatures are widely used today in the business, e.g. for authorizing bank payments, for exchange of signed electronic documents, for signing transactions in the public blockchain systems, for signing digital contracts, and in many other scenarios. The digital signature provides:

- **authentication** - it allows checking if the sender created and signed the message,
- **integrity** - it allows you to check that the data has not been altered after the signature,
- **non-repudiation** - the person signing the message cannot deny his signature,

The digital signature process consists of three algorithms:

- **key generation** - this generates a set of keys, private and public, - **signing** - the algorithm signs the message with a private key, usually the message hash is encrypted.
- **verification** - the algorithm checks the authenticity of the message by verifying the signature with a public key,

Asymmetric cryptographic methods are used to sign messages/data. The digital signature process prevents several common attacks such as:

- ***key-only*** – Attacker has access to the public key
- ***known message*** – Attacker has access to valid signatures for known messages, but not those that they have chosen
- ***adaptive chosen message*** – Attacker gains access to signatures on various messages that they have chosen.

The most popular public-key cryptosystems are:

- RSA,
- DSA (Digital Signature Algorithm),
- ECDSA / EdDSA based on elliptic curves (ECC),

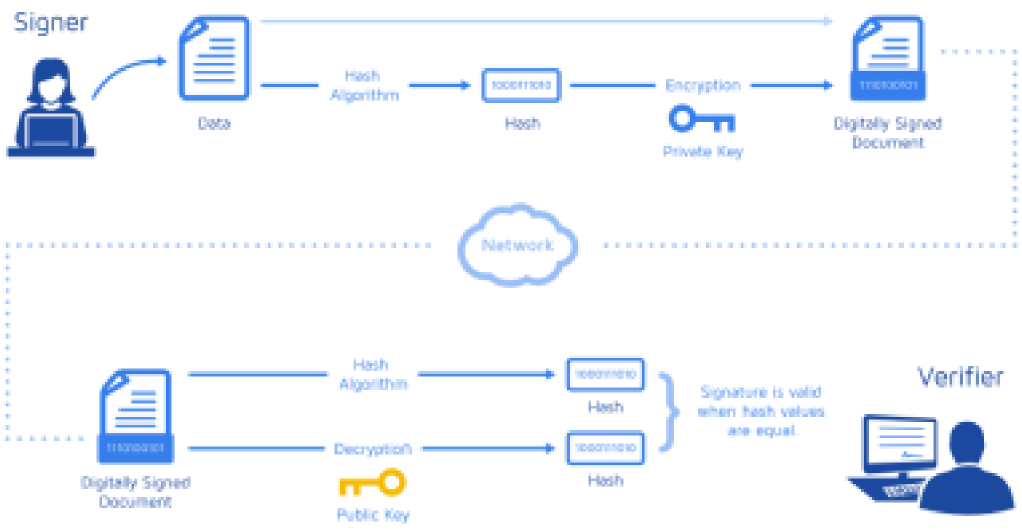


Figure 1 Flow of the digital signatures – source https://www.docusign.com/how-it works/electronic-signature/digital-signature/digital-signature-faq

Public key infrastructure (PKI) allows efficient and secure identification of public keys. It can be used within or between organizations with the help of the Internet. Different types of PKI can be deployed by varying the essential configuration details, trust rules. PKI is a set of identities, roles, policies, and actions for the creation, use, management, distribution, and revocation of public and private keys. There are several PKI standards in the industry, the most popular is the X.509 version 3. Moreover, PKI pro vides:

- authentication,
- confidentiality,
- non-repudiation,
- integrity,

PKI management model involves specifying the rules for message formats and proce dures used to communicate. The model also explains how various entities communicate with each other. The major entities of PKI management are as follows:

End Entity (EE): It can be a user or software application to which the certificate is served. It needs secure access at least to its name and private key.

Certification Authority (CA): It may be a third party or from the EE's organization that issues a certificate to the EE.

Registration Authority (RA): It is a subset of EE and is an optional component. If RA is not present, then CA performs RA's functions. RA carries out functions such as key generation, keypair management, token distribution, etc.

Certificate Revocation Lists issuer (CRL): If some certificates have to be revoked, the CRL issuer will take care of it. It is also an optional component.

Certificate Repository: Storage unit to define how to store certificates and CRLs and how it can be accessed by the EE.

The Certificate Signing Request (CSR) should be generated on the client's computer. CSR contains the public key and ID to be certified. The CSR is then sent to the RA/CA to sign the certificate. Afterward, the certificate becomes fully valuable. Generating CSR

on the customer's computer is important because only then the private key is known to the owner, and other entities do not have access to it.

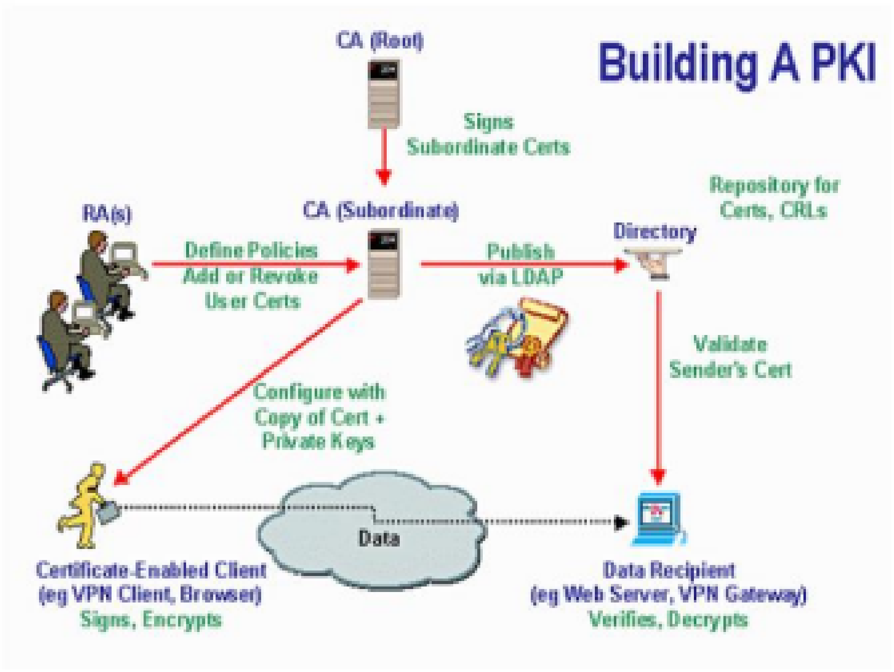


Figure 2 PKI concepts - source https://cybersecuritynews.co.uk/how-does-public key-infrastructure-pki-work

2. Environment

Virtual machines to use in this lab:

- **Ubuntu Server,**

3. Digital signatures.

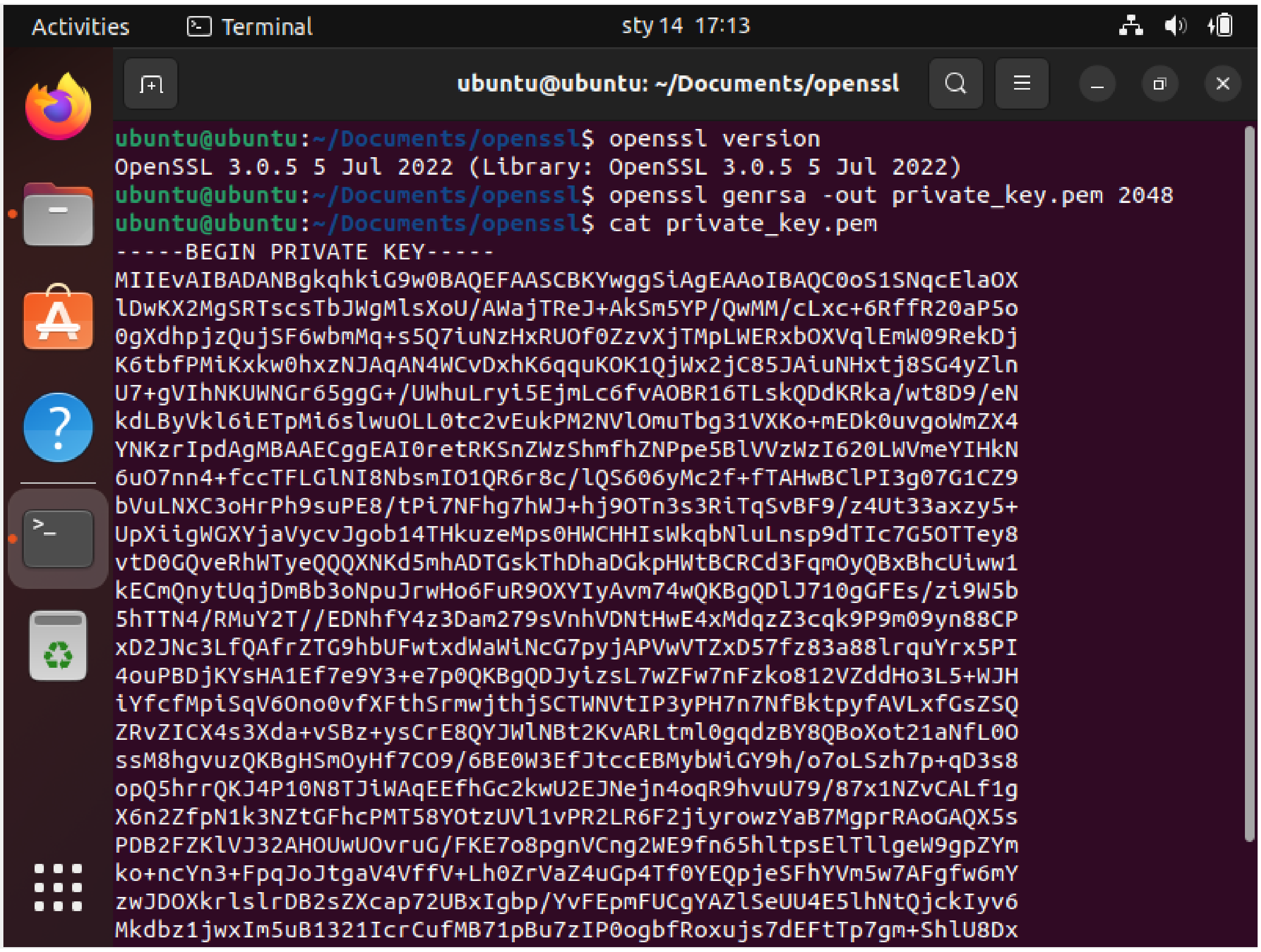
In this part of the task, a set of keys should be generated, which are necessary for further tasks. In the first part, a set of keys is to be generated using OpenSSL (RSA algorithm), in the second part, the OpenPGP tool is used.

I. **OpenSSL:**

a. Basic commands:

Use the OpenSSL tool to generate RSA private key (key length 2048bits):

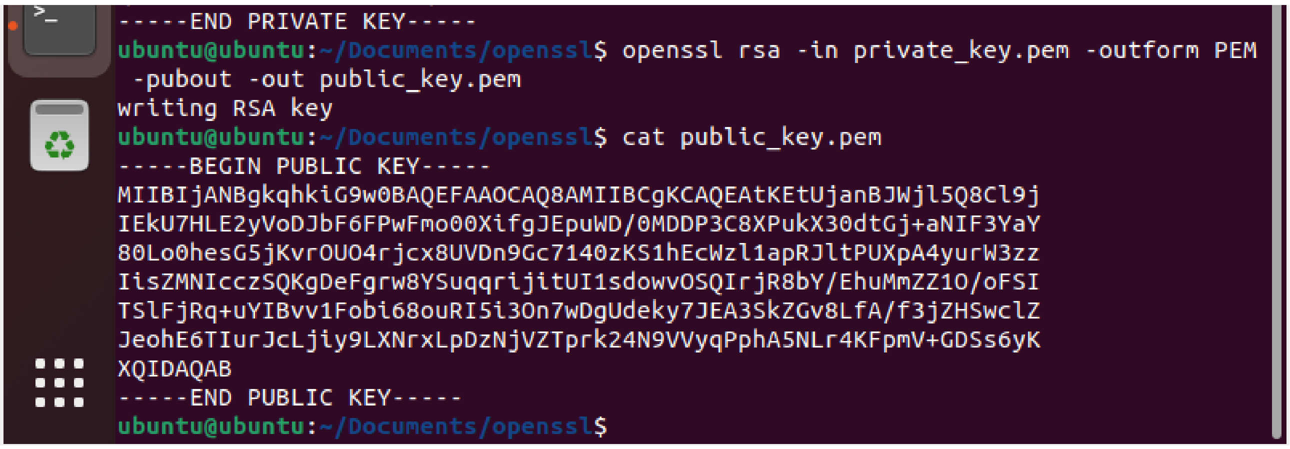
```
openssl genrsa -out private_key.pem 2048
```



```
ubuntu@ubuntu:~/Documents/openssl$ openssl version
OpenSSL 3.0.5 5 Jul 2022 (Library: OpenSSL 3.0.5 5 Jul 2022)
ubuntu@ubuntu:~/Documents/openssl$ openssl genrsa -out private_key.pem 2048
ubuntu@ubuntu:~/Documents/openssl$ cat private_key.pem
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQC0oS1SNqcElaOX
lDwKX2MgSRTscsTbJWgMlsXoU/AWajTReJ+AkSm5YP/QwMM/cLxc+6RffR20aP5o
0gXdhpjzQujSF6wbmMq+s5Q7iuNzHxRUOf0ZzvXjTMpLWERxbOXVqlEmW09RekDj
K6tbfPMiKxkw0hxzNJAqAN4WCvDxhK6qquKOK1QjWx2jC85JAiuNHxtj8SG4yZln
U7+gVIhNKUWNGr65ggG+/UWhuLryi5EjmLc6fvAOBR16TLskQDdKRka/wt8D9/eN
kdLByVkl6iETpMi6slwuOLL0tc2vEukPM2NVlOmuTbg31VXKo+mEDk0uvgoWmZX4
YNKzrIpdAgMBAAECggEAI0retRKSnZWzShmfhZNPpe5BlVVzWzI620LWVmeYIHkN
6uO7nn4+fccTFLGlNI8NbsmIO1QR6r8c/lQS606yMc2f+fTAHwBClPI3g07G1CZ9
bVuLNXC3oHrPh9suPE8/tPi7NFhg7hWJ+hj9OTn3s3RiTqSvBF9/z4Ut33axzy5+
UpXiigWGXYjaVycvJgob14THkuzeMps0HWCHHIsWkqbNluLnsp9dTIc7G5OTTey8
vtD0GQveRhWTyeQQQXNKd5mhADTGskThDhaDGkpHWtBCRCd3FqmOyQBxBhcUiww1
kECmQnytUqjDmBb3oNpuJrwHo6FuR9OXYIyAvm74wQKBgQDlJ710gGFEs/zi9W5b
5hTTN4/RMuY2T//EDNhfY4z3Dam279sVnhVDNtHwE4xMdqzZ3cqk9P9m09yn88CP
xD2JNc3LfQAfrZTG9hbUFwtxdWaWiNcG7pyjAPVwVTZxD57fz83a88lrquYrx5PI
4ouPBDjKYsHA1Ef7e9Y3+e7p0QKBgQDJyizsL7wZFw7nFzko812VZddHo3L5+WJH
iYfcfMpiSqV6Ono0vfXFthSrmwjthjSCTWNVtIP3yPH7n7NfBktpyfAVLxfGsZSQ
ZRvZICX4s3Xda+vSBz+ysCrE8QYJWlNBt2KvARLtml0gqdzBY8QBoXot21aNfL0O
ssM8hgvuzQKBgHSmOyHf7CO9/6BE0W3EfJtccEBMybWiGY9h/o7oLSzh7p+qD3s8
opQ5hrrQKJ4P10N8TJiWAqEEfhGc2kwU2EJNejn4oqR9hvuU79/87x1NZvCALf1g
X6n2ZfpN1k3NZtGFhcPMT58YOtzUVl1vPR2LR6F2jiyrowzYaB7MgprRAoGAQX5s
PDB2FZKlVJ32AHOUwUOvruG/FKE7o8pgnVCng2WE9fn65hltpsElTllgeW9gpZYm
ko+ncYn3+FpqJoJtgaV4VffV+Lh0ZrVaZ4uGp4Tf0YEQpjeSFhYVm5w7AFgfw6mY
zwJDOXkrlslrDB2sZXcap72UBxIgbp/YvFEpmFUCgYAZlSeUU4E5lhNtQjckIyv6
Mkdbz1jwxIm5uB1321IcrCufMB71pBu7zIP0ogbfRoxujs7dEFtTp7gm+ShlU8Dx
```

Based on private key, generate public.key:

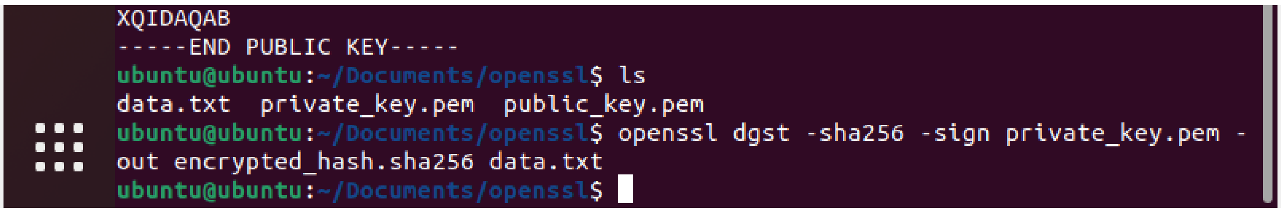
```
openssl rsa -in private_key.pem -outform PEM -pubout -out
public_key.pem
```



```
-----END PRIVATE KEY-----
ubuntu@ubuntu:~/Documents/openssl$ openssl rsa -in private_key.pem -outform PEM
 -pubout -out public_key.pem
writing RSA key
ubuntu@ubuntu:~/Documents/openssl$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtKEtUjanBJWjl5Q8Cl9j
IEkU7HLE2yVoDJbF6FPwFmo00XifgJEpuWD/0MDDP3C8XPukX30dtGj+aNIF3YaY
80Lo0hesG5jKvrOUO4rjcx8UVDn9Gc7140zKS1hEcWzl1apRJltPUXpA4yurW3zz
IisZMNIcczSQKgDeFgrw8YSuqqrijitUI1sdowvOSQIrjR8bY/EhuMmZZ1O/oFSI
TSlFjRq+uYIBvv1Fobi68ouRI5i3On7wDgUdeky7JEA3SkZGv8LfA/f3jZHSwclZ
JeohE6TIurJcLjiy9LXNrxLpDzNjVZTprk24N9VVyqPphA5NLr4KFpmV+GDSs6yK
XQIDAQAB
-----END PUBLIC KEY-----
ubuntu@ubuntu:~/Documents/openssl$
```

Sign the file with the following commands:

```
openssl dgst -sha256 -sign private_key.pem -out en
crypted_hash.sha256 data.txt
```

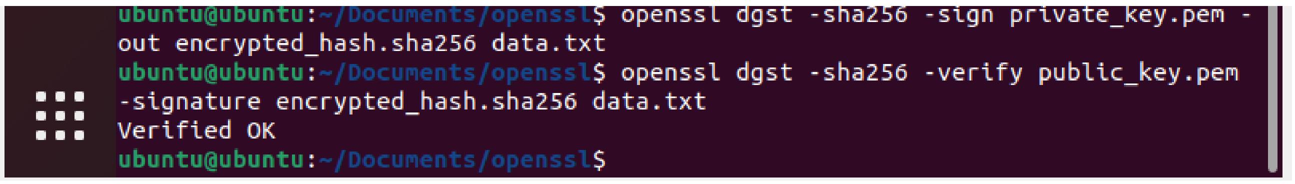


```
XQIDAQAB
-----END PUBLIC KEY-----
ubuntu@ubuntu:~/Documents/openssl$ ls
data.txt  private_key.pem  public_key.pem
ubuntu@ubuntu:~/Documents/openssl$ openssl dgst -sha256 -sign private_key.pem -
out encrypted_hash.sha256 data.txt
ubuntu@ubuntu:~/Documents/openssl$ 
```

To verify the signature, run the following command:

```
openssl dgst -sha256 -verify public_key.pem -signature en
```

```
crypted_hash data.txt
```



```
ubuntu@ubuntu:~/Documents/openssl$ openssl dgst -sha256 -sign private_key.pem -
out encrypted_hash.sha256 data.txt
ubuntu@ubuntu:~/Documents/openssl$ openssl dgst -sha256 -verify public_key.pem
-signature encrypted_hash.sha256 data.txt
Verified OK
ubuntu@ubuntu:~/Documents/openssl$
```

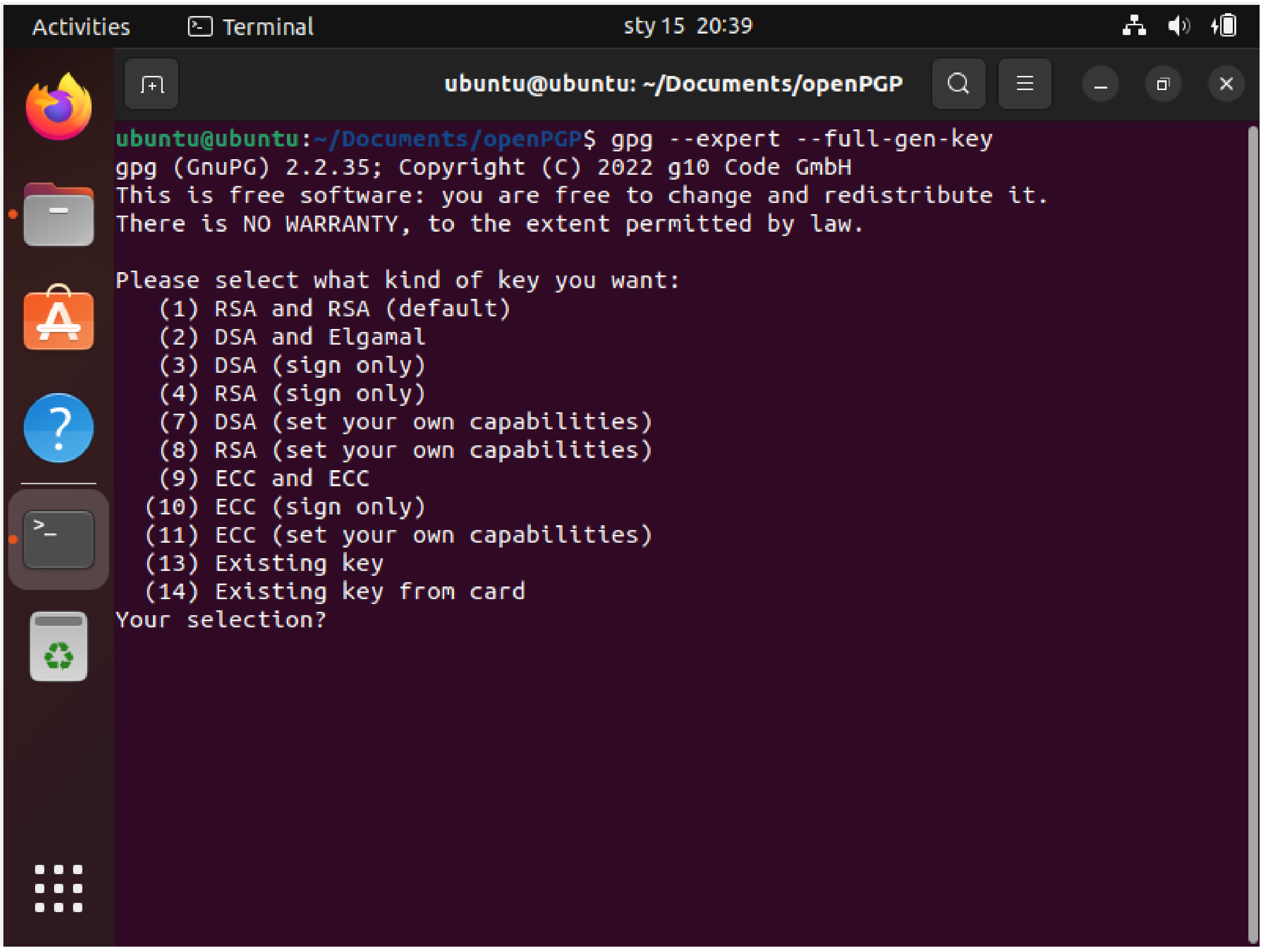
OpenSSL documentation is available here: https://www.openssl.org/docs/

II. OpenPGP:

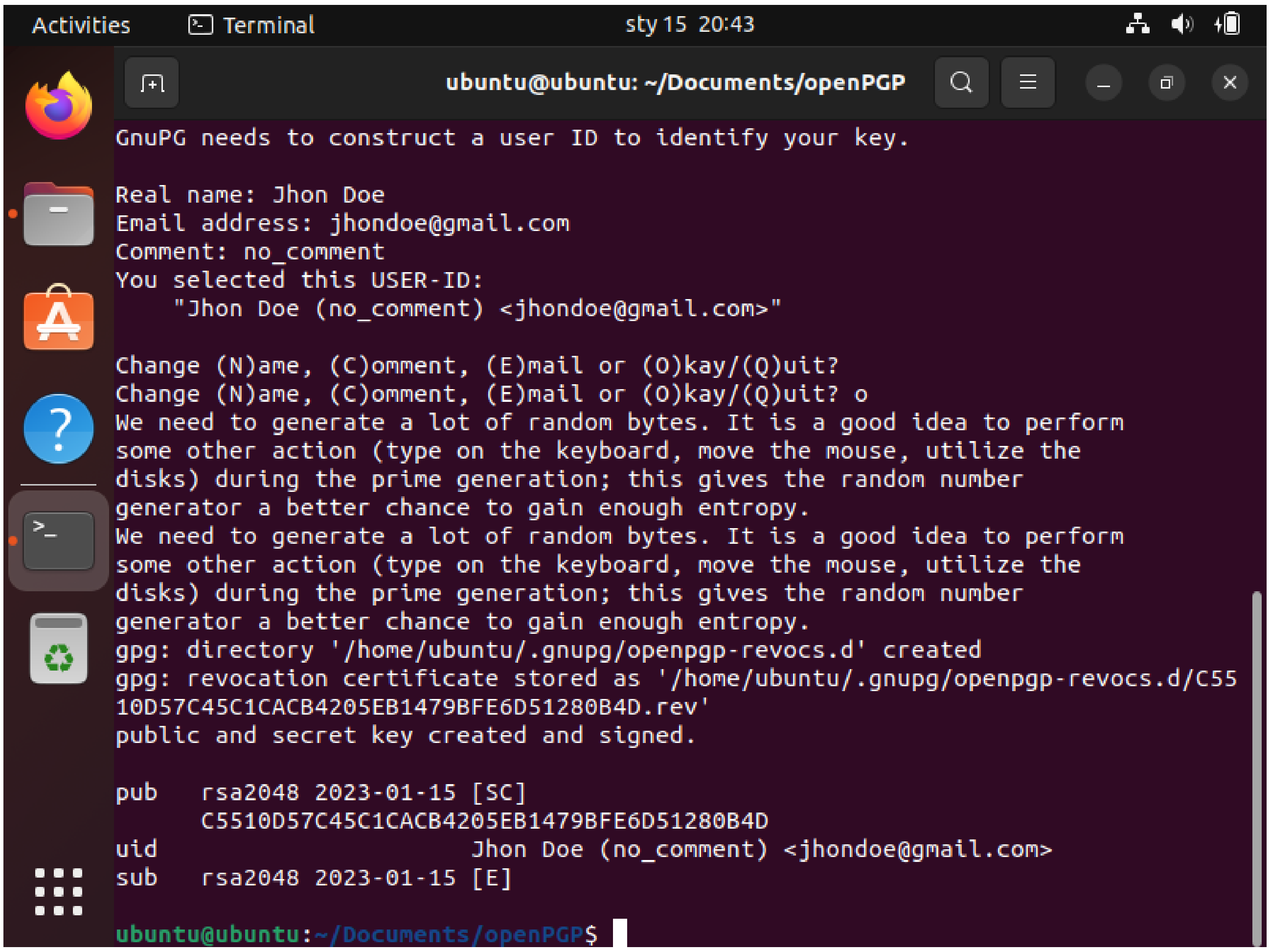
a. Basic commands:

Generate key pair using GPG tool:

```
gpg --expert --full-gen-key
```



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --expert --full-gen-key
gpg (GnuPG) 2.2.35; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (7) DSA (set your own capabilities)
   (8) RSA (set your own capabilities)
   (9) ECC and ECC
  (10) ECC (sign only)
  (11) ECC (set your own capabilities)
  (13) Existing key
  (14) Existing key from card
Your selection?
```



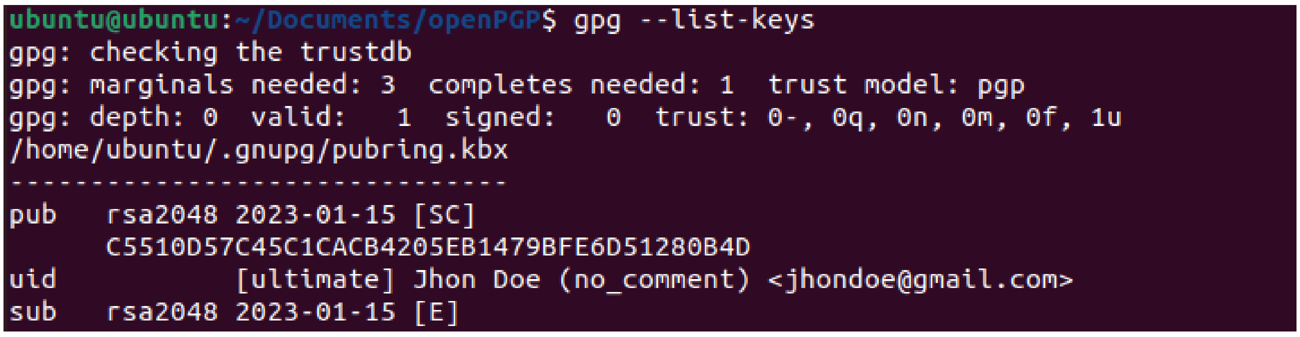
```
GnuPG needs to construct a user ID to identify your key.

Real name: Jhon Doe
Email address: jhondoe@gmail.com
Comment: no_comment
You selected this USER-ID:
    "Jhon Doe (no_comment) <jhondoe@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: directory '/home/ubuntu/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/ubuntu/.gnupg/openpgp-revocs.d/C55
10D57C45C1CACB4205EB1479BFE6D51280B4D.rev'
public and secret key created and signed.

pub   rsa2048 2023-01-15 [SC]
      C5510D57C45C1CACB4205EB1479BFE6D51280B4D
uid                      Jhon Doe (no_comment) <jhondoe@gmail.com>
sub   rsa2048 2023-01-15 [E]

ubuntu@ubuntu:~/Documents/openPGP$ 
```

Before generating the keys, you are obligated to complete the following fields: appropriate algorithm, properties of an algorithm (e.g. key length, elliptic curve, etc.), expiry date, name, e-mail address, and passphrase.

To list available keys use:

- public:

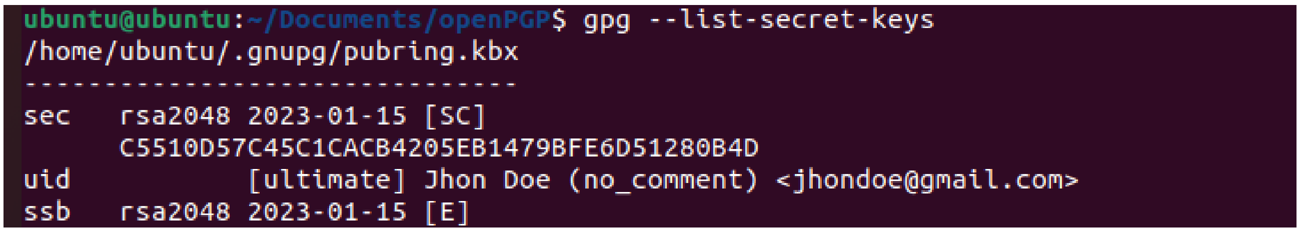
```
gpg --list-keys,
```



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --list-keys
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
/home/ubuntu/.gnupg/pubring.kbx
------------------------------
pub   rsa2048 2023-01-15 [SC]
      C5510D57C45C1CACB4205EB1479BFE6D51280B4D
uid           [ultimate] Jhon Doe (no_comment) <jhondoe@gmail.com>
sub   rsa2048 2023-01-15 [E]
```

- private:

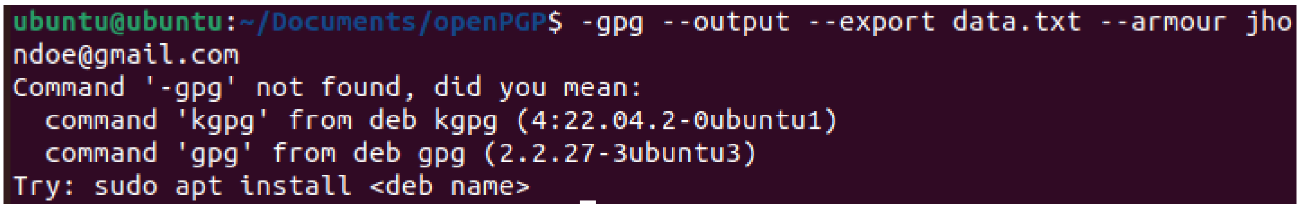
```
gpg --list-secret-keys,
```



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --list-secret-keys
/home/ubuntu/.gnupg/pubring.kbx
------------------------------
sec   rsa2048 2023-01-15 [SC]
      C5510D57C45C1CACB4205EB1479BFE6D51280B4D
uid           [ultimate] Jhon Doe (no_comment) <jhondoe@gmail.com>
ssb   rsa2048 2023-01-15 [E]
```

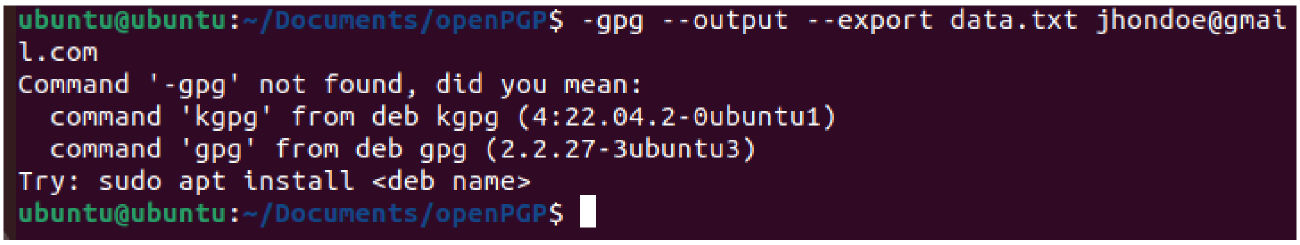
Export a public key:

- gpg --output --export *<file name>* --armour *<email_address>* - text form - ASCII,



```
ubuntu@ubuntu:~/Documents/openPGP$ -gpg --output --export data.txt --armour jho
ndoe@gmail.com
Command '-gpg' not found, did you mean:
  command 'kgpg' from deb kgpg (4:22.04.2-0ubuntu1)
  command 'gpg' from deb gpg (2.2.27-3ubuntu3)
Try: sudo apt install <deb name>
```

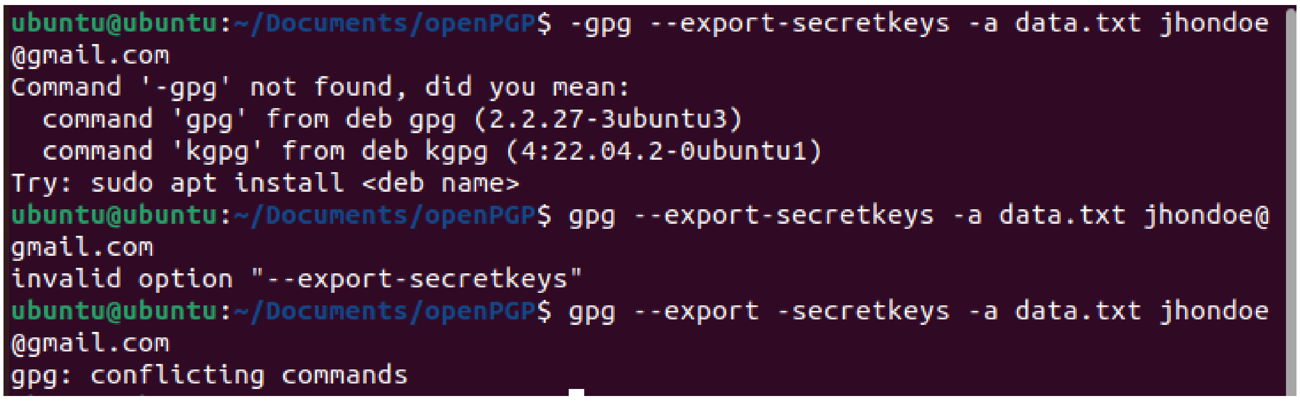
- gpg --export --output *<file name>* *<email_address>* - binary form,



```
ubuntu@ubuntu:~/Documents/openPGP$ -gpg --output --export data.txt jhondoe@gmai
l.com
Command '-gpg' not found, did you mean:
  command 'kgpg' from deb kgpg (4:22.04.2-0ubuntu1)
  command 'gpg' from deb gpg (2.2.27-3ubuntu3)
Try: sudo apt install <deb name>
ubuntu@ubuntu:~/Documents/openPGP$
```

Export a private key:

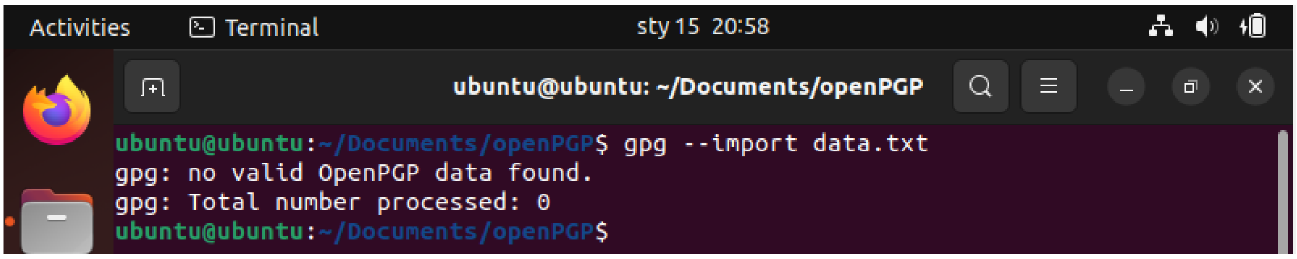
- gpg --export-secret-keys -a *<file name>* *<email address>*,



```
ubuntu@ubuntu:~/Documents/openPGP$ -gpg --export-secretkeys -a data.txt jhondoe
@gmail.com
Command '-gpg' not found, did you mean:
  command 'gpg' from deb gpg (2.2.27-3ubuntu3)
  command 'kgpg' from deb kgpg (4:22.04.2-0ubuntu1)
Try: sudo apt install <deb name>
ubuntu@ubuntu:~/Documents/openPGP$ gpg --export-secretkeys -a data.txt jhondoe@
gmail.com
invalid option "--export-secretkeys"
ubuntu@ubuntu:~/Documents/openPGP$ gpg --export -secretkeys -a data.txt jhondoe
@gmail.com
gpg: conflicting commands
```

Importuj klucz:

gpg --import *<file name>*

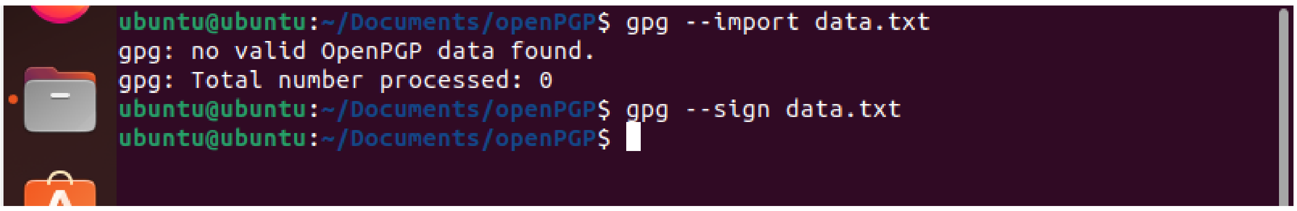


```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --import data.txt
gpg: no valid OpenPGP data found.
gpg: Total number processed: 0
ubuntu@ubuntu:~/Documents/openPGP$
```

Sign a file:

- gpg --sign *<file name>*,

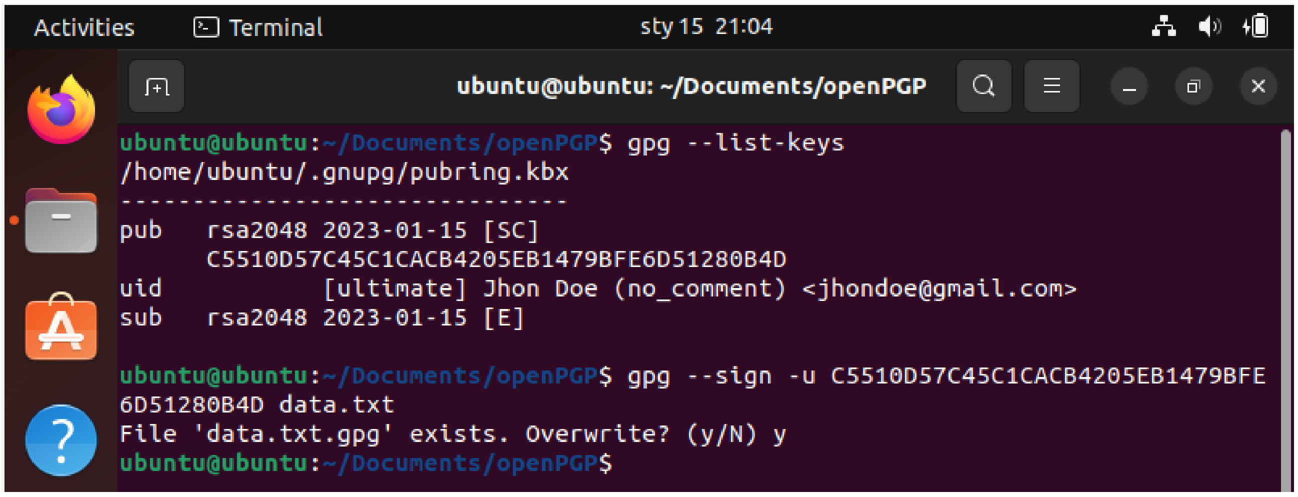
Passphrase - `passphrase`



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --import data.txt
gpg: no valid OpenPGP data found.
gpg: Total number processed: 0
ubuntu@ubuntu:~/Documents/openPGP$ gpg --sign data.txt
ubuntu@ubuntu:~/Documents/openPGP$
```

Sign a file with a selected key:

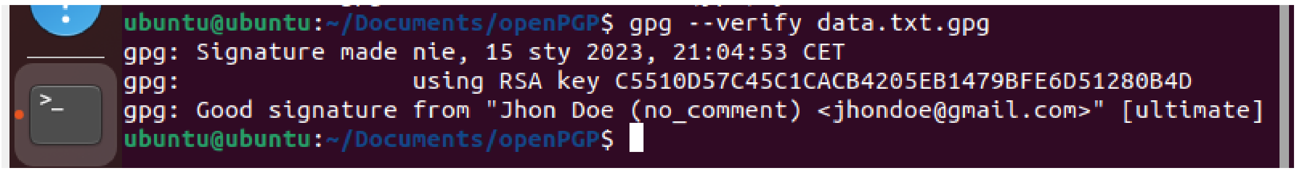
- gpg --sign -u *<key_id> <file name>*,



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --list-keys
/home/ubuntu/.gnupg/pubring.kbx
-------------------------------
pub   rsa2048 2023-01-15 [SC]
      C5510D57C45C1CACB4205EB1479BFE6D51280B4D
uid           [ultimate] Jhon Doe (no_comment) <jhondoe@gmail.com>
sub   rsa2048 2023-01-15 [E]

ubuntu@ubuntu:~/Documents/openPGP$ gpg --sign -u C5510D57C45C1CACB4205EB1479BFE
6D51280B4D data.txt
File 'data.txt.gpg' exists. Overwrite? (y/N) y
ubuntu@ubuntu:~/Documents/openPGP$
```

Verify a signature:

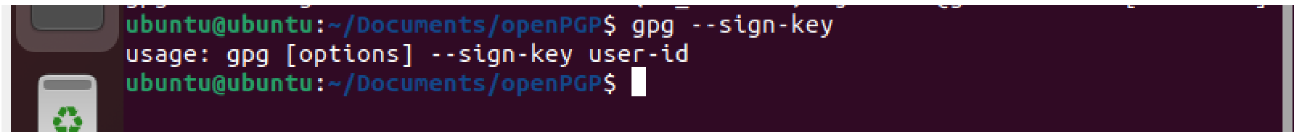
- gpg --verify *<file name>*.gpg



```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --verify data.txt.gpg
gpg: Signature made nie, 15 sty 2023, 21:04:53 CET
gpg:                using RSA key C5510D57C45C1CACB4205EB1479BFE6D51280B4D
gpg: Good signature from "Jhon Doe (no_comment) <jhondoe@gmail.com>" [ultimate]
ubuntu@ubuntu:~/Documents/openPGP$
```

Sign a imported key:

- gpg --sign-key

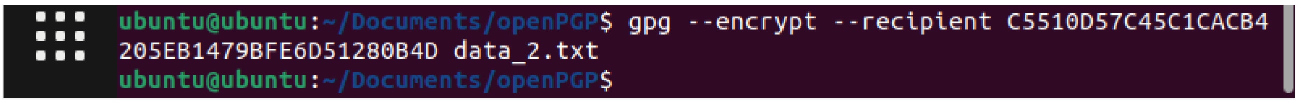


```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --sign-key
usage: gpg [options] --sign-key user-id
ubuntu@ubuntu:~/Documents/openPGP$
```

Encrypt a file with gpg:

- gpg --encrypt --recipient <email address> --output <output file> <input file>

"This command did not work, so i took help from internet"



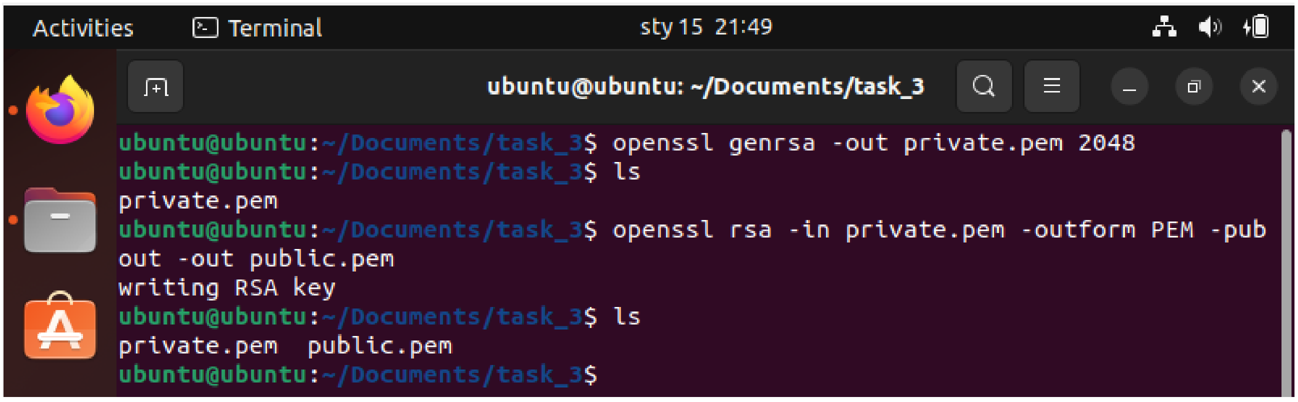
```
ubuntu@ubuntu:~/Documents/openPGP$ gpg --encrypt --recipient C5510D57C45C1CACB4
205EB1479BFE6D51280B4D data_2.txt
ubuntu@ubuntu:~/Documents/openPGP$
```

Other useful operations of gpg:

- gpg --fingerprint
- gpg --edit-key

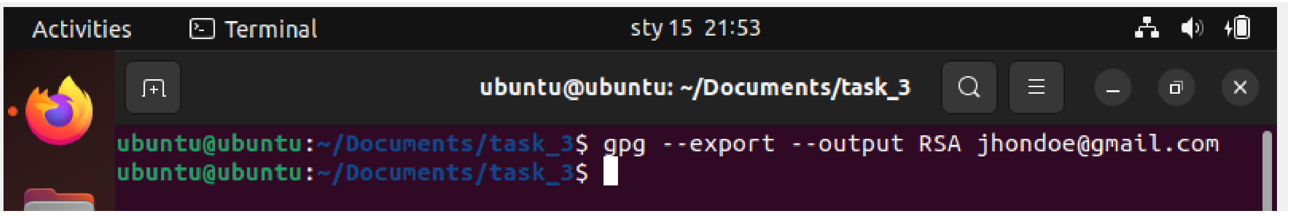
III. Tasks:

1. Generate a set of keys using OpenSSL (RSA) and OpenPGP (ECC and RSA).



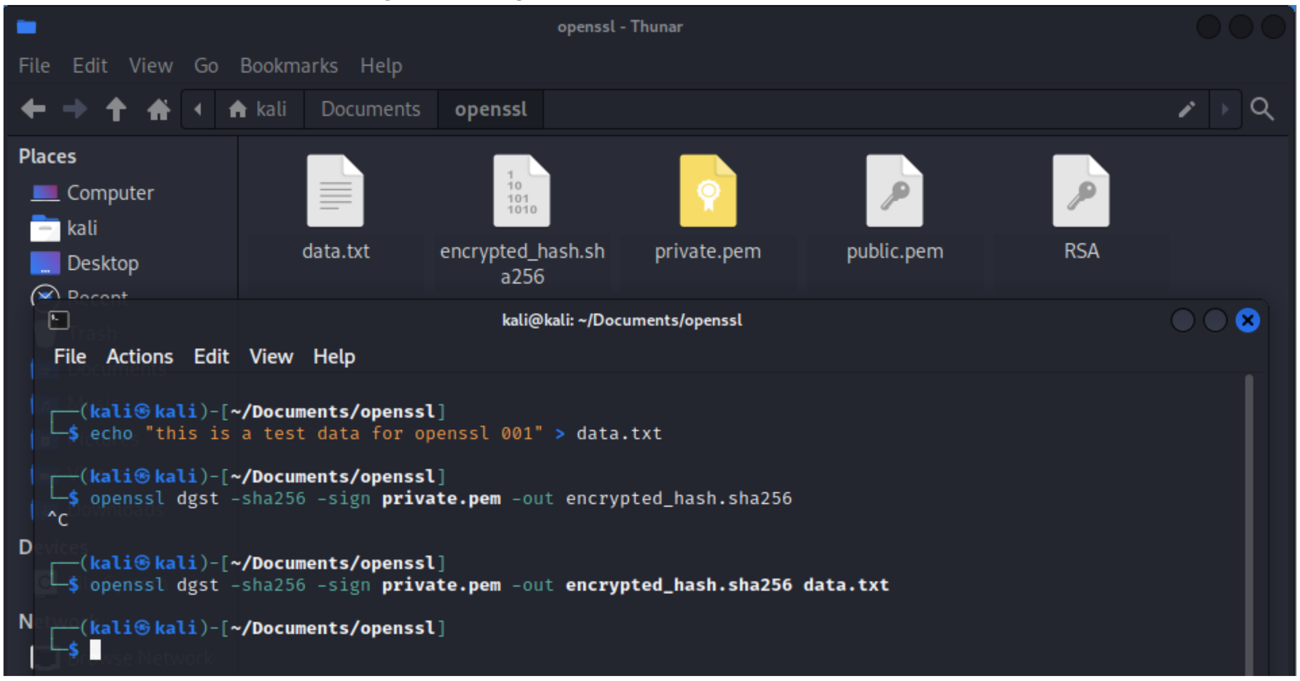
```
ubuntu@ubuntu:~/Documents/task_3$ openssl genrsa -out private.pem 2048
ubuntu@ubuntu:~/Documents/task_3$ ls
private.pem
ubuntu@ubuntu:~/Documents/task_3$ openssl rsa -in private.pem -outform PEM -pub
out -out public.pem
writing RSA key
ubuntu@ubuntu:~/Documents/task_3$ ls
private.pem  public.pem
ubuntu@ubuntu:~/Documents/task_3$
```

2. Export a public key from GPG (ACSII format).



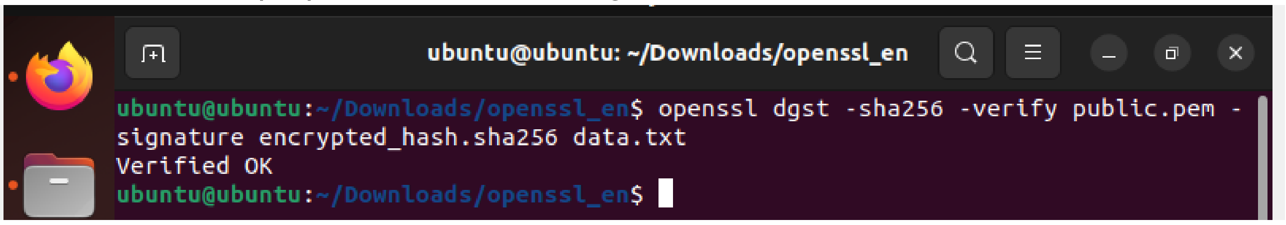
```
ubuntu@ubuntu:~/Documents/task_3$ gpg --export --output RSA jhondoe@gmail.com
ubuntu@ubuntu:~/Documents/task_3$
```

3. Transfer your public key (OpenSSL and OpenPGP) to another machine/user account.
> Ubuntu to Kali
4. For OpenPGP, a public key should be imported and signed with a local private key. Verify a fingerprint of a certificate on both machines/ user accounts.
5. Create a text file and sign it using both methods.



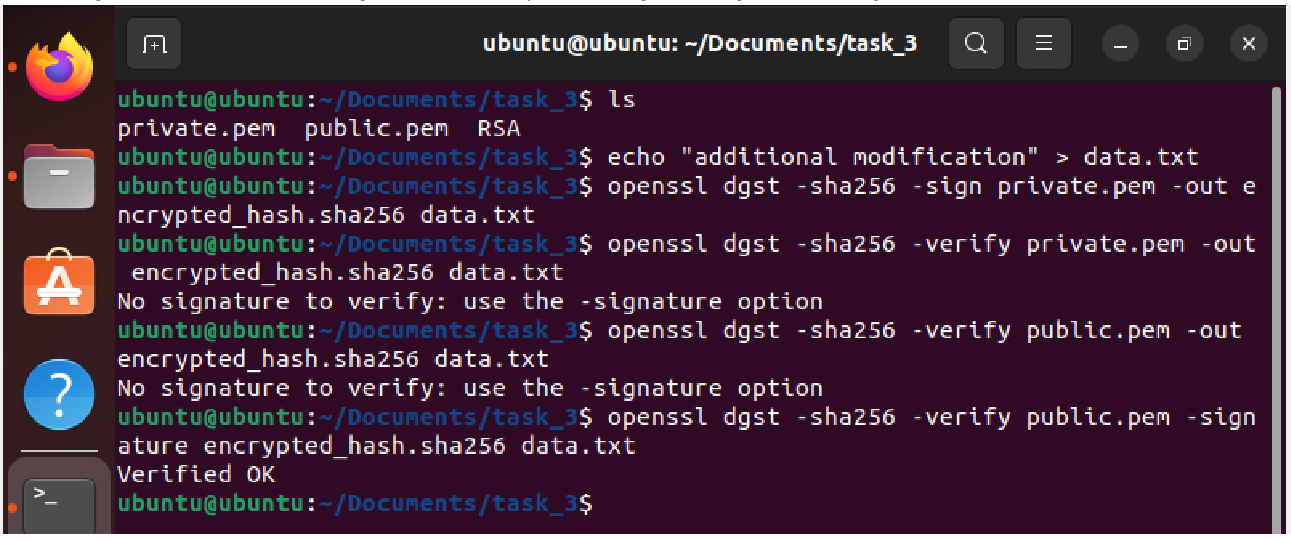
```
┌──(kali㉿kali)-[~/Documents/openssl]
└─$ echo "this is a test data for openssl 001" > data.txt

┌──(kali㉿kali)-[~/Documents/openssl]
└─$ openssl dgst -sha256 -sign private.pem -out encrypted_hash.sha256
^C

┌──(kali㉿kali)-[~/Documents/openssl]
└─$ openssl dgst -sha256 -sign private.pem -out encrypted_hash.sha256 data.txt

┌──(kali㉿kali)-[~/Documents/openssl]
└─$
```

6. Transfer your file and signatures to another machine/user account and verify the signatures.
> I have another laptop with ubuntu running, i tested on that



```
ubuntu@ubuntu:~/Downloads/openssl_en$ openssl dgst -sha256 -verify public.pem -
signature encrypted_hash.sha256 data.txt
Verified OK
ubuntu@ubuntu:~/Downloads/openssl_en$
```

7. After the signature has been successfully verified, modify the content of the original document, sign and verify the digital signature again.



```
ubuntu@ubuntu:~/Documents/task_3$ ls
private.pem  public.pem  RSA
ubuntu@ubuntu:~/Documents/task_3$ echo "additional modification" > data.txt
ubuntu@ubuntu:~/Documents/task_3$ openssl dgst -sha256 -sign private.pem -out e
ncrypted_hash.sha256 data.txt
ubuntu@ubuntu:~/Documents/task_3$ openssl dgst -sha256 -verify private.pem -out
 encrypted_hash.sha256 data.txt
No signature to verify: use the -signature option
ubuntu@ubuntu:~/Documents/task_3$ openssl dgst -sha256 -verify public.pem -out
encrypted_hash.sha256 data.txt
No signature to verify: use the -signature option
ubuntu@ubuntu:~/Documents/task_3$ openssl dgst -sha256 -verify public.pem -sign
ature encrypted_hash.sha256 data.txt
Verified OK
ubuntu@ubuntu:~/Documents/task_3$
```

8. Only for OpenPGP. Encrypt the file using a public key and then decrypt the encrypted file. Afterward, change one character in the encrypted file and try to decrypt it again.

IV. **Questions:**

1. Is it possible to generate only one asymmetric key (e.g. key private)? Would that make sense?
> It is possible to generate only one asymmetric key, such as a private key, but it would not make sense as asymmetric encryption relies on a pair of keys, a public and a private one, where the public key is used for encrypting the data and the private key is used for decrypting it. Without the corresponding public key, the private key would be useless

2. What form does the GPG key have? How does a PEM key differ from an Open PGP key?
> GNU Privacy Guard keys come in the form of a public key and a private key. A PEM key is a format used for storing private keys and public keys in an ASCII format, while OpenPGP is a standard for encrypting and signing messages. PEM keys can be used with OpenPGP, but OpenPGP keys are not limited to the PEM format.

3. Is the exchange of private keys justified?
> The exchange of private keys is not generally justified, as private keys should be kept secret and should not be shared with anyone.

4. What is the fingerprint?
>A fingerprint is a short sequence of characters that serves as a unique identifier for a GPG key. It can be used to verify the authenticity of a key, by comparing the fingerprint with the one provided by the key's owner.

5. Describe what has changed and why in task 7?
> Nothing seemed to change.

6. Did you succeed in decrypting the ciphertext after the character replacement in task 8?
> No, after changing the encrypted file decrypting it was not possible.

7. What is the difference in the signature (OpenPGP) for ECC and RSA algorithms?
> The difference in the signature (OpenPGP) for ECC and RSA algorithms is that ECC is based on the mathematics of elliptic curves and is considered to provide the same level of security as RSA with much shorter key lengths, while RSA is based on the mathematical properties of large prime numbers.

8. Can a message be signed with a public key? If so, what could the consequences be?
> A message cannot be signed with a public key, as signing a message requires access to the private key. If a message were signed with a public key, it would be possible for anyone to forge signatures, as the public key is freely available.

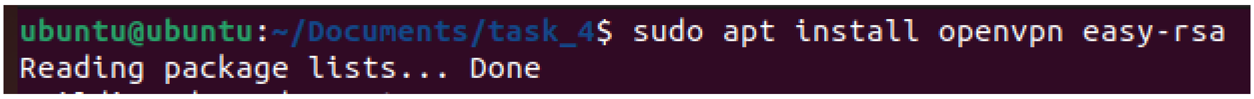
4. PKI - key generation in X.509 standard.

OpenVPN utilizes certificates to encrypt traffic between the server and clients. The simple certificate authority (CA) is built to issue trusted certificates. The Easy-RSA is a tool that is used to build and manage a PKI CA. Easy-RSA is based on the OpenSSL library.

I. Build CA.

The EasyRSA tool is located in the home directory (/home/server). The tool is used to build and manage OpenVPN certificates:

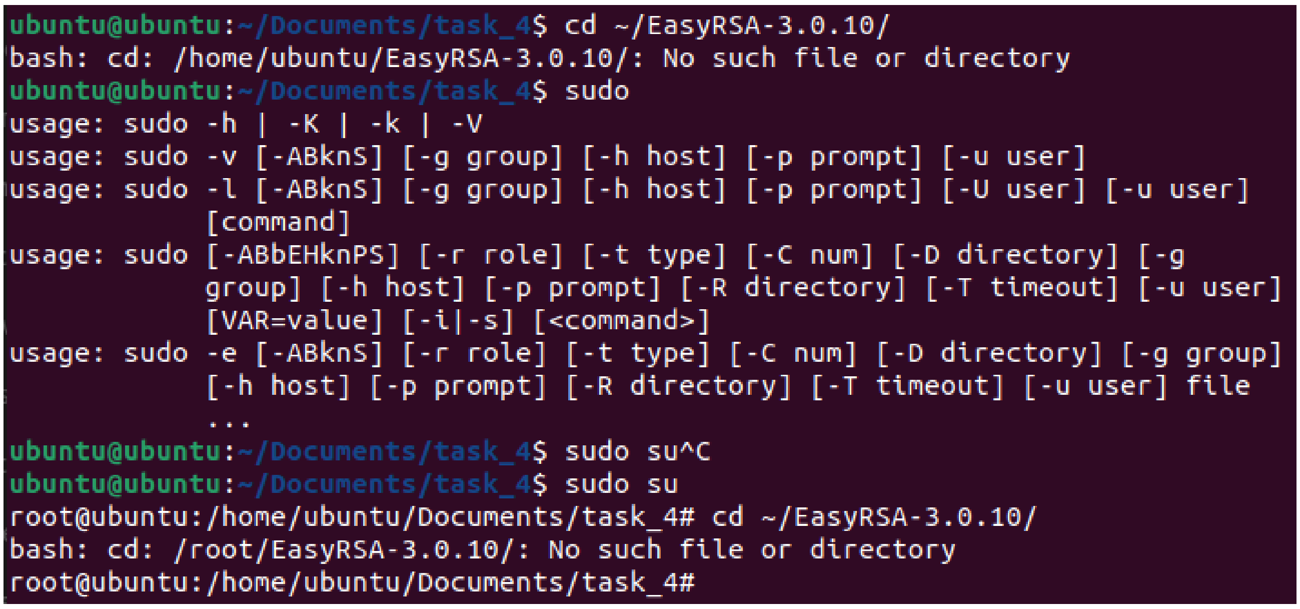
Generation the sets of certificates:



```
ubuntu@ubuntu:~/Documents/task_4$ sudo apt install openvpn easy-rsa
Reading package lists... Done
```

Navigate to the EasyRSA directory:

```
cd ~/EasyRSA-3.0.10/
```



```
ubuntu@ubuntu:~/Documents/task_4$ cd ~/EasyRSA-3.0.10/
bash: cd: /home/ubuntu/EasyRSA-3.0.10/: No such file or directory
ubuntu@ubuntu:~/Documents/task_4$ sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-ABknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-ABknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
            [command]
usage: sudo [-ABbEHknPS] [-r role] [-t type] [-C num] [-D directory] [-g
            group] [-h host] [-p prompt] [-R directory] [-T timeout] [-u user]
            [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-ABknS] [-r role] [-t type] [-C num] [-D directory] [-g group]
            [-h host] [-p prompt] [-R directory] [-T timeout] [-u user] file
            ...
ubuntu@ubuntu:~/Documents/task_4$ sudo su^C
ubuntu@ubuntu:~/Documents/task_4$ sudo su
root@ubuntu:/home/ubuntu/Documents/task_4# cd ~/EasyRSA-3.0.10/
bash: cd: /root/EasyRSA-3.0.10/: No such file or directory
root@ubuntu:/home/ubuntu/Documents/task_4#
```

Copy the vars.example file:

```
cp vars.example vars
```

Edit the vars file:

```
nano vars
```

Remove # to uncomment lines and adjust values as follows:

```
#set_var EASYRSA_REQ_COUNTRY "PL"
#set_var EASYRSA_REQ_PROVINCE "Wroclaw"
```

```
#set_var EASYRSA_REQ_CITY "Wroclaw"
#set_var EASYRSA_REQ_ORG "PWR"
#set_var EASYRSA_REQ_EMAIL "<student e-mail>"
#set_var EASYRSA_REQ_OU "PWR"
```

Run this script with the `init-pki` option to initiate the public key infrastructure on the CA server:

```
./easyrsa init-pki
```

To build CA, run the following command - it generates ca.key (private.key) and ca.crt (public certificate). It is very important to keep the key in a safe place because the security of the VPN network depends on it.

```
./easyrsa build-ca nopass
```

Note that the common name should be correctly set according to its intended use. To revoke a client certificate, you need to revoke the client on CA:

```
./easyrsa revoke <client’s common name>
```

Then generate a CRL file and transfer it onto the server:

```
./easyrsa gen-crl
```

II. **Generate the server and client certificates.**

Run the command to generate a server private key (server.key) and a certificate signing request - CSR (server.req):

```
./easyrsa gen-req server nopass
```

Sign a server request using CA, it generates server.crt:

```
./easyrsa sign-req server server
```

Run the command to generate a client private key (client1.key) and a certificate signing request (client1.req):

```
./easyrsa gen-req client1 nopass
```

Sign a client signing request using CA, it generates client1.crt:

```
./easyrsa sign-req client client1
```

Generate Diffie-Hellman key (dh.pem) which is used to exchange keys: `./easyrsa gen-dh`

Generate an HMAC signature (ta.key) to strengthen the server’s TLS integrity verifica tion capabilities:

```
openvpn --genkey --secret ta.key
```

The server files:

- server.key – server private key,
- server.crt – server certificate,
- ta.key HMAC signature,
- dh.pem - Diffie-Hellman key,
- ca.crt - CA certificate,

The client files:

- client.key – server private key,
- client.crt – server certificate,
- ta.key - HMAC signature,
- ca.crt - CA certificate,

Check the properties of one of the certificates and answer the following questions:

1. What is the signature algorithm used?

> from the context i could say SHA256

2. What is the public key algorithm used?

> (RSA)

3. What version of the X.509 standard has been used to generate a certificate?

> did not fnd!

4. What else can be read from the server/client certificate?

> key, certificate signature and serial can be found (google)