Laboratorio

24/03/2023

Crittografia simmetrica

Esercizio 1:

ex1-enc.py

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
import binascii
#import random #--> the random function is not suitable for crypto usage,
because does not generate unpredictable sequences of bytes
import os

content_file = ""
key = b"questa_La_chiave"
file_plaintext_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\file.txt"
file_encrypted_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\file.txt.enc"

#open file as text
with open(file_plaintext_path, "r", encoding="utf-8") as file_to_enc:
    print("----------------- Plaintext File (text) -----------------")
    print(file_to_enc.read())
    print("---------------------------------------------------")



#open file as byte
with open(file_plaintext_path, "rb") as file_to_enc:
    content_file = file_to_enc.read()
    print("----------------- Plaintext File (Byte) -----------------")
    print(binascii.hexlify(content_file))
    print("---------------------------------------------------")

print("----------------- Plaintext File Dimension in Byte -----------------
")
print(os.path.getsize(file_plaintext_path), "byte")
```

```python
print("---------------------------------------------------")


#padding of file with PKCS#7

p = padding.PKCS7(128).padder() #we must specify dimension of block in bit size that must be a 8-bit multiple

padded_text = p.update(content_file) + p.finalize()

print("----------------- Padded Text -----------------")
print(binascii.hexlify(padded_text))
print("---------------------------------------------------")

print("----------------- Padded Plaintext Dimension in Byte -----------------")
print(len(padded_text), "byte")
print("---------------------------------------------------")

#encrypt with AES in CBC modewith IV and KEY

iv = os.urandom(16)
print("----------------- IV -----------------")
print(binascii.hexlify(iv))
print("---------------------------------------------------")

print("----------------- IV Dimension in Byte -----------------")
print(len(iv), "byte")
print("---------------------------------------------------")

cipher = Cipher(algorithms.AES(key),modes.CBC(iv))
enc = cipher.encryptor()
ct = enc.update(padded_text)
print("----------------- Ciphertext -----------------")
print(binascii.hexlify(ct))
print("---------------------------------------------------")

print("----------------- Ciphertext Dimension in Byte -----------------")
print(len(ct), "byte")
print("---------------------------------------------------")

#save IV and ciperthext to an output file
with open(file_encrypted_path, "wb") as file_encrypted:
```

```
    content_file_enc = file_encrypted.write(iv)
    content_file_enc = file_encrypted.write(ct)

#read encrypted file
with open(file_encrypted_path, "rb") as file_encrypted:
    content_file_enc = file_encrypted.read()
    print("----------------- Encrypted File -----------------")
    print(binascii.hexlify(content_file_enc))
    print("----------------------------------------------------")

print("----------------- Plaintext File Dimension in Byte -----------------
")
print(os.path.getsize(file_encrypted_path), "byte")
print("----------------------------------------------------")
```

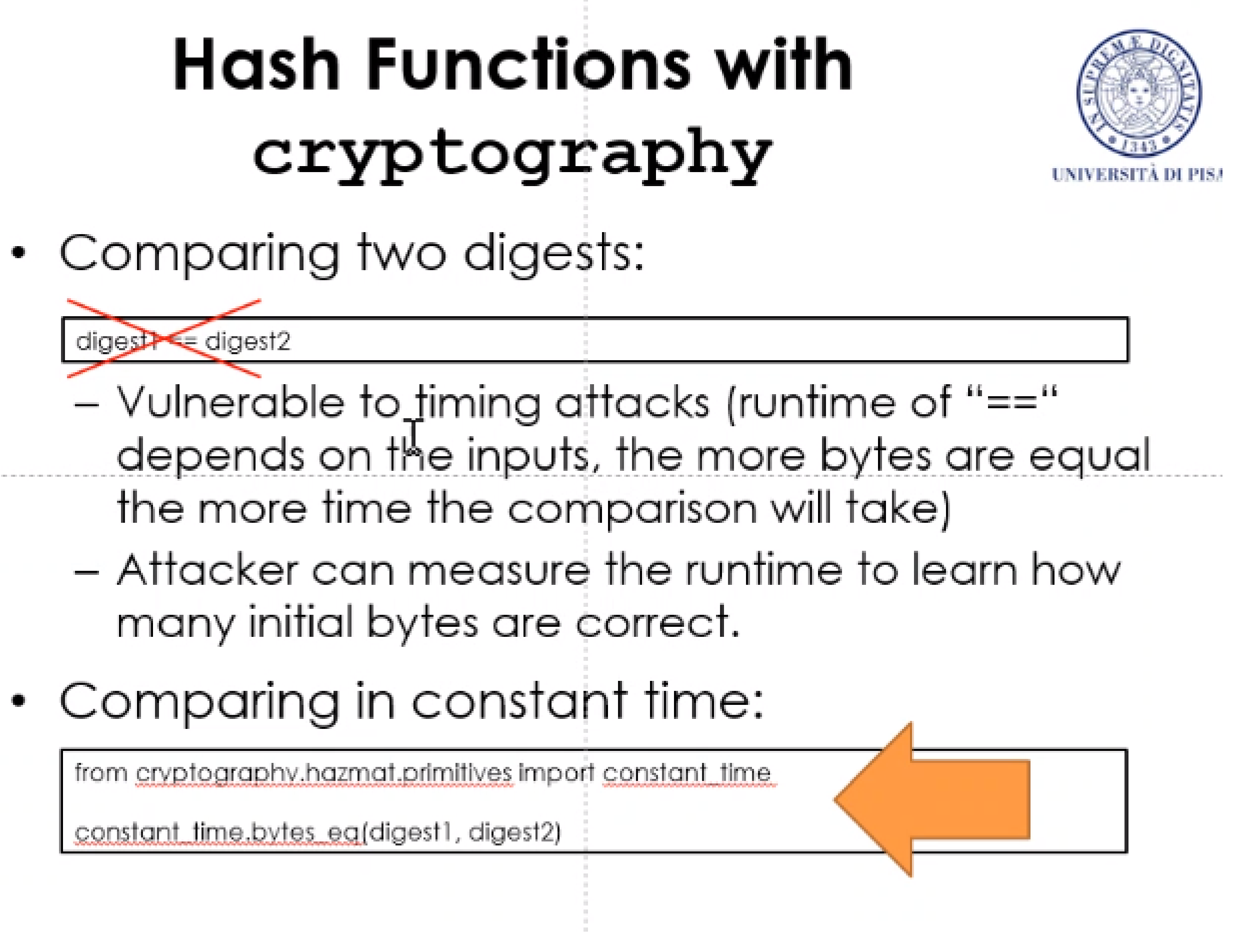
ex1-dec.py

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
import binascii
#import random #--> the random function is not suitable for crypto usage,
because does not generate unpredictable sequences of bytes
import os

content_file_enc = ""
iv = ""
ct = "" #cyphertext
key = b"questa_La_chiave"
file_decrypted_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\file.txt.enc.dec"
file_encrypted_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\file.txt.enc"

#read encrypted file
with open(file_encrypted_path, "rb") as file_encrypted:
    content_file_enc = file_encrypted.read()
    print("----------------- Encrypted File -----------------")
    print(binascii.hexlify(content_file_enc))
    print("----------------------------------------------------")

if os.path.getsize(file_encrypted_path) % 16 != 0:
    print("Encrypted file must be multiple of 16 bytes in size!")
    os.exit()
```

```python
#extract IV from encrypted file - the first 16 byte are IV
with open(file_encrypted_path, "rb") as file_encrypted:
    iv = file_encrypted.read(16)
    print("----------------- IV -----------------")
    print(binascii.hexlify(iv))
    print("---------------------------------------------------")
    ct = file_encrypted.read()
    print("----------------- Ciphertext -----------------")
    print(binascii.hexlify(ct))
    print("---------------------------------------------------")

    print("----------------- Ciphertext Dimension in Byte -----------------")
    print(len(ct), "byte")
    print("---------------------------------------------------")

#decrypt cyphertext
cipher = Cipher(algorithms.AES(key),modes.CBC(iv))
#create an encryptor
dec = cipher.decryptor()
padded_plaintext = dec.update(ct)
p = padding.PKCS7(128).unpadder() #we must specify dimension of block in bit size that must be a 8-bit multiple
unpadded_plaintext = p.update(padded_plaintext) + p.finalize()
print("----------------- Plaintext unpadded decrypted -----------------")
print(binascii.hexlify(unpadded_plaintext))
print("---------------------------------------------------")

print("----------------- Plaintext decrypted Dimension in Byte -----------------")
print(len(unpadded_plaintext), "byte")
print("---------------------------------------------------")

#save decrypted cyphertext to an output file
with open(file_decrypted_path, "wb") as file_decrypted:
    file_decrypted.write(unpadded_plaintext)

#open file as text
with open(file_decrypted_path, "r", encoding="utf-8") as file_decrypted:
    print("----------------- Decrypted File (text) -----------------")
    print(file_decrypted.read())
    print("---------------------------------------------------")
```

31/03/2023

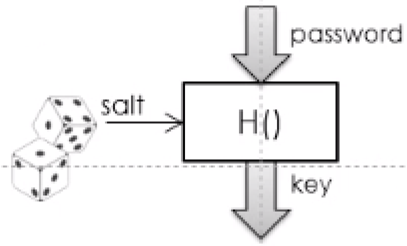
Le funzioni hash vengono usate in diversi contesti crittografici per diversi utilizzi.



Key Derivation Functions

UNIVERSITÀ DI PISA

- Logical representation:



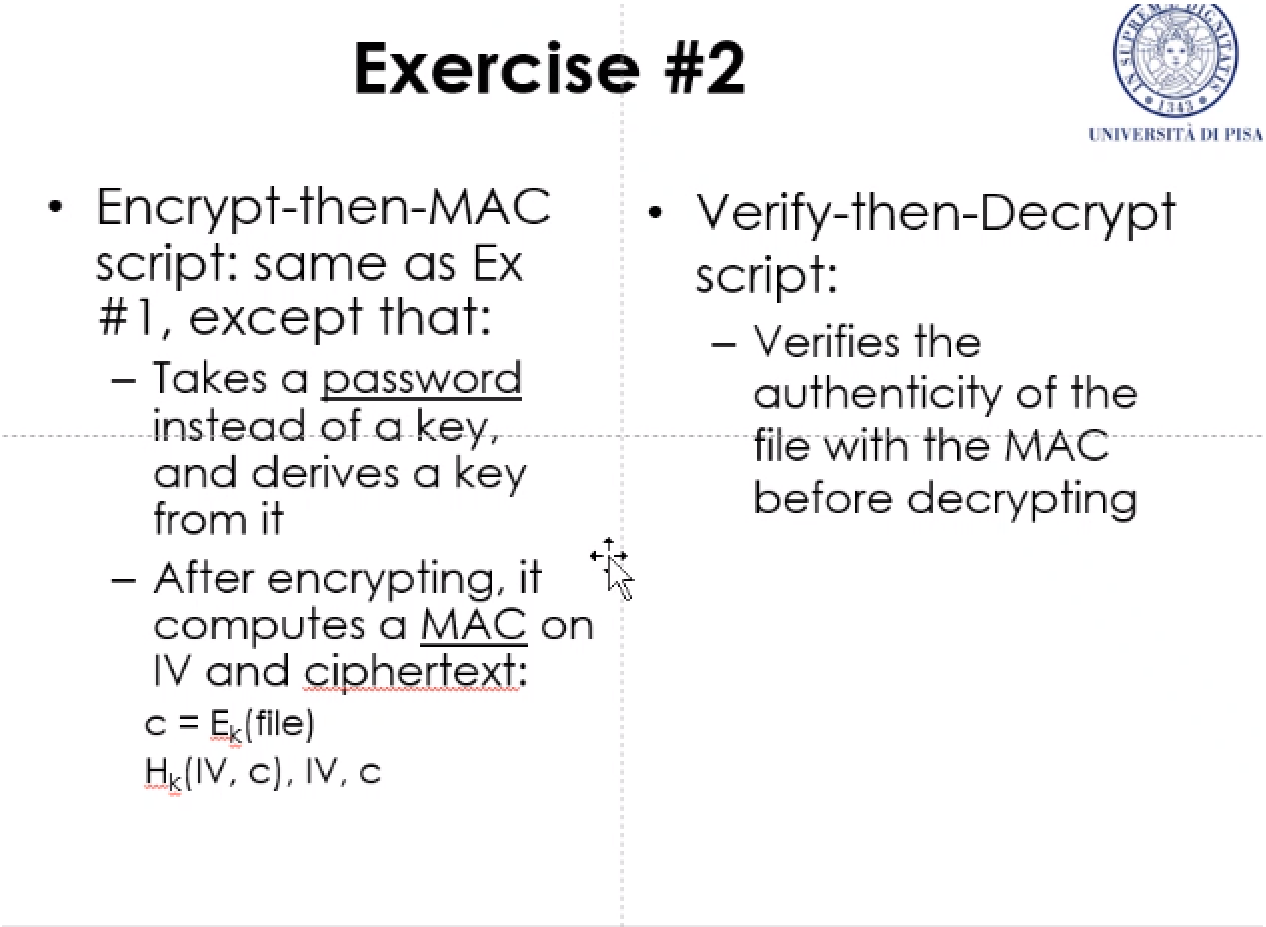
- PBKDF2 (Password-Based Key Derivation Function 2)
  - Based on a classic hash function (e.g., SHA256)
  - Repeats the classic hash function many times

Key Derivation Functions

UNIVERSITÀ DI PISA

- Two purposes:
- 1) Encrypting with a password
  - Derive an encryption key from password
- 2) Storing passwords not in-the-clear
  - Derive a randomized digest from password

Esercizio 2:



ex1-enc-hmac.py

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hmac
import binascii
#import random #--> the random function is not suitable for crypto usage, because does not generate unpredictable sequences of bytes
import os

content_file = ""
tag = ""
hash_HMAC = hashes.SHA256()
iterations_HMAC = 2**14
salt_size = 16
iv_size = 16
password = input("Type the password to encrypt:").encode()
file_plaintext_path = "C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\Lesson2\\file.txt"
file_encrypted_path = "C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\Lesson2\\file.txt.enc"
```

```python
file_encrypted_hmac_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\Lesson2\\file.txt.enc.hm
ac"

#open plaintext file as text
with open(file_plaintext_path, "r", encoding="utf-8") as file_to_enc:
    print("----------------- Plaintext File (text) -----------------")
    print(file_to_enc.read())



#open plaintext file as byte
with open(file_plaintext_path, "rb") as file_to_enc:
    content_file = file_to_enc.read()
    print("----------------- Plaintext File (Byte) -----------------")
    print(binascii.hexlify(content_file))


print("----------------- Plaintext File Dimension in Byte -----------------
")
print(os.path.getsize(file_plaintext_path), "byte")



# derive a key using PBKDF2
salt = os.urandom(salt_size)
kdf = PBKDF2HMAC(hashes.SHA256(),16,salt,iterations_HMAC)
key = kdf.derive(password)
print("----------------- Salt -----------------")
print(binascii.hexlify(salt))

print("----------------- Derived Key -----------------")
print(binascii.hexlify(key))



#padding of file with PKCS#7

p = padding.PKCS7(128).padder() #we must specify dimension of block in bit
size that must be a 8-bit multiple
padded_text = p.update(content_file) + p.finalize()
print("----------------- Padded Text -----------------")
print(binascii.hexlify(padded_text))
```

```python
print("----------------- Padded Plaintext Dimension in Byte -----------------")
print(len(padded_text), "byte")


#encrypt with AES in CBC mode with IV and KEY

iv = os.urandom(iv_size)
print("----------------- IV -----------------")
print(binascii.hexlify(iv))

print("----------------- IV Dimension in Byte -----------------")
print(len(iv), "byte")


cipher = Cipher(algorithms.AES(key),modes.CBC(iv))
enc = cipher.encryptor()
ct = enc.update(padded_text)
print("----------------- Ciphertext -----------------")
print(binascii.hexlify(ct))


print("----------------- Ciphertext Dimension in Byte -----------------")
print(len(ct), "byte")



# compute MAC on SALT+IV+CT using encrypted file of as input of HMAC

hm = hmac.HMAC(key,hash_HMAC)
hm.update(salt)
hm.update(iv)
hm.update(ct)
tag = hm.finalize()
print("----------------- HMAC Encrypted File (SALT + IV + CT) -----------------")
print(binascii.hexlify(tag))

print("----------------- HMAC Dimension in Byte -----------------")
print(len(tag), "byte")
```

```
#save SALT + HMAC + IV + CT to an output file
with open(file_encrypted_hmac_path, "wb") as file_encrypted_hmac:
    file_encrypted_hmac.write(salt)
    file_encrypted_hmac.write(tag)
    file_encrypted_hmac.write(iv)
    file_encrypted_hmac.write(ct)

#read encrypted hmac file
with open(file_encrypted_hmac_path, "rb") as file_encrypted_hmac:
    content_file_enc_hmac = file_encrypted_hmac.read()
    print("----------------- Encrypted HMAC File -----------------")
    print(binascii.hexlify(content_file_enc_hmac))


print("----------------- Encrypted HMAC File Dimension in Byte -------------
----")
print(os.path.getsize(file_encrypted_hmac_path), "byte")
```

ex1-dec-hmac.py

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hmac
import binascii
#import random #--> the random function is not suitable for crypto usage,
because does not generate unpredictable sequences of bytes
import os

content_file_enc = ""
salt = ""
tag_from_file = ""
iv = ""
ct = "" #cyphertext
hash_HMAC = hashes.SHA256()
iterations_HMAC = 2**14
tag_size = hashes.SHA256.digest_size
salt_size = 16
iv_size = 16
password = input("Type the password to decrypt:").encode()
file_decrypted_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\Lesson2\\file.txt.enc.de
c"
```

```
file_encrypted_hmac_path =
"C:\\Users\\SamuelePadula\\Desktop\\Python\\Crypto\\Lesson2\\file.txt.enc.hm
ac"

#read encrypted file
with open(file_encrypted_hmac_path, "rb") as file_encrypted:
    content_file_enc = file_encrypted.read()
    print("----------------- Encrypted HMAC File -----------------")
    print(binascii.hexlify(content_file_enc))



if os.path.getsize(file_encrypted_hmac_path) % 16 != 0:
    print("Encrypted file must be multiple of 16 bytes in size!")
    os.exit()

#read SALT + TAG + IV + CT from file
with open(file_encrypted_hmac_path, "rb") as file_encrypted_hmac:
    salt = file_encrypted_hmac.read(salt_size)
    print("----------------- Salt (from file) -----------------")
    print(binascii.hexlify(salt))

    tag_from_file = file_encrypted_hmac.read(tag_size)
    print("----------------- HMAC (from file) -----------------")
    print(binascii.hexlify(tag_from_file))

    iv = file_encrypted_hmac.read(iv_size)
    print("----------------- IV (from file) -----------------")
    print(binascii.hexlify(iv))

    ct = file_encrypted_hmac.read()
    print("----------------- Ciphertext (from file) -----------------")
    print(binascii.hexlify(ct))

    print("----------------- Ciphertext Dimension in Byte -----------------
")
    print(len(ct), "byte")



# derive a key using PBKDF2
kdf = PBKDF2HMAC(hashes.SHA256(),16,salt,iterations_HMAC)
key = kdf.derive(password)
print("----------------- Derived Key -----------------")
print(binascii.hexlify(key))
```

```python
#verify authenticity of the file
hm = hmac.HMAC(key, hash_HMAC)
hm.update(salt)
hm.update(iv)
hm.update(ct)
hm.verify(tag_from_file)
print("-----------------!! HMAC Verified !! -----------------")
print(binascii.hexlify(tag_from_file))


# If here without exception, HMAC is verified!, now decrypt

#decrypt cyphertext
cipher = Cipher(algorithms.AES(key),modes.CBC(iv))
#create an encryptor
dec = cipher.decryptor()
padded_plaintext = dec.update(ct)
p = padding.PKCS7(128).unpadder() #we must specify dimension of block in bit size that must be a 8-bit multiple
unpadded_plaintext = p.update(padded_plaintext) + p.finalize()
print("----------------- Plaintext unpadded decrypted -----------------")
print(binascii.hexlify(unpadded_plaintext))


print("----------------- Plaintext decrypted Dimension in Byte -----------------")
print(len(unpadded_plaintext), "byte")

#save decrypted cyphertext to an output file
with open(file_decrypted_path, "wb") as file_decrypted:
    file_decrypted.write(unpadded_plaintext)

#open file as text
with open(file_decrypted_path, "r", encoding="utf-8") as file_decrypted:
    print("----------------- Decrypted File (text) -----------------")
    print(file_decrypted.read())
```

01/04/2023

How to generate with openssl RSA keys

```
Generating RSA private key, 3072 bit long modulus (2 primes)
.......................................................................+
+++  <<<< generate prime number
..............................................++++ <<<< generate prime
number
e is 65537 (0x010001)
```

to make human-readble the private key:

```
openssl rsa -in rsa_privkey.pem -noout -text
RSA Private-Key: (3072 bit, 2 primes)
modulus:
    00:b6:4c:fa:09:8b:c5:ec:82:09:f2:b2:51:48:bb:
    9f:a8:42:e0:b5:22:20:7a:eb:b0:b2:f3:3b:0b:63:
    00:50:4b:19:25:55:b3:9c:55:1f:86:a0:8a:e8:0d:
    f5:38:5c:f8:8c:ba:da:59:88:12:94:ea:1a:99:f1:
    d9:b5:94:e0:3e:45:1e:04:b6:e1:12:bc:cb:7d:9e:
    ec:9b:9e:67:f1:f5:65:c5:95:1c:e6:46:64:8d:9d:
    e4:e3:4e:c8:44:56:e8:3d:e5:38:05:fc:cf:49:4d:
    e1:77:fc:c1:bc:d1:3a:16:5b:f1:3f:64:4b:76:3b:
    39:52:7a:11:72:ee:7b:b0:10:50:6b:61:16:d0:ce:
    ba:f6:ce:2d:94:44:9a:4e:db:85:ac:42:3d:4e:fc:
    07:e9:aa:37:03:76:4a:10:63:75:54:10:01:67:b0:
    a0:03:84:92:37:5c:36:9e:d4:9c:f6:af:c7:04:f3:
    af:e7:73:50:10:9d:fb:63:79:a5:71:c0:67:cc:97:
    8c:90:34:f1:12:7e:ad:6d:22:78:9f:67:f0:f6:dc:
    07:05:5b:77:a5:23:22:c7:73:45:b0:69:4e:16:f7:
    d3:1b:13:71:45:1d:96:43:08:7f:3d:e7:68:2d:61:
    22:d4:48:17:99:2a:ae:25:eb:7e:bc:0f:d1:a1:7d:
    57:b8:4f:23:49:7a:b4:92:35:ff:65:57:f9:53:b6:
    a3:27:65:05:a4:b9:44:f1:e8:b3:9f:4e:bf:cb:60:
    3b:05:d0:9b:f5:17:9d:01:eb:4a:3d:07:cc:93:15:
    65:e2:2b:f9:bd:04:66:36:9b:4c:fc:15:02:61:dc:
    97:90:6e:87:2a:b5:fd:68:53:38:67:6a:0d:b9:b8:
    20:cf:9a:59:81:62:9b:2f:ed:6d:4d:cd:96:9a:78:
    f3:82:5e:df:98:6f:e6:7c:0d:13:d6:15:81:89:75:
    90:60:42:42:78:f7:82:8d:ae:1c:61:bd:b8:49:95:
    e3:06:06:10:36:37:72:f7:71:ed
publicExponent: 65537 (0x10001)
privateExponent:
    48:30:73:53:14:66:6c:21:92:8e:e8:ce:07:5f:44:
    f9:fc:81:bf:38:a4:64:08:b1:10:2c:01:55:a0:fe:
    9e:cd:1e:48:0a:87:f5:80:3f:db:af:f7:51:ad:35:
    4d:fc:82:f0:37:8d:ff:a6:42:b5:75:7e:d3:37:52:
```

```
    5e:f5:75:57:33:47:8f:d6:5b:8a:6f:f8:a4:e6:2a:
    0b:f5:ce:73:a2:19:8b:04:61:4d:4e:d2:c2:c1:a3:
    c1:df:90:ae:7f:3b:b8:46:ec:c8:72:34:23:73:13:
    b5:d0:01:68:23:f2:3c:a8:6c:00:0e:57:53:9a:60:
    38:a8:de:00:05:30:35:a8:40:30:45:62:23:8d:b9:
    bb:c2:29:8a:6d:20:2d:da:00:35:16:85:f0:a1:1d:
    01:0a:c6:7e:38:79:5b:c4:06:d2:23:04:6b:6b:25:
    f2:3e:ad:27:fe:fc:22:29:4f:7e:e0:5c:8f:39:70:
    8e:d6:ba:fd:d9:91:92:f2:b8:f1:32:9f:3c:8a:ce:
    58:d3:7c:8b:1c:53:65:1f:5f:13:a1:5a:ad:bc:d6:
    d5:42:e4:c4:f9:1e:98:b1:59:36:a2:7b:57:3b:66:
    fb:6a:ce:2b:97:5a:f7:07:01:99:9c:f0:69:89:48:
    06:90:e3:1d:0e:8c:85:3e:4c:b7:4e:f7:bc:07:8e:
    df:4a:68:90:c0:c9:fa:f6:c3:aa:94:4d:7e:f4:20:
    05:6f:85:a9:7d:d7:01:50:8d:12:7a:7b:bc:49:63:
    39:0c:82:56:1c:f6:fb:5d:1b:01:1d:51:de:94:e7:
    4e:2f:b5:7f:ed:2c:99:ef:b1:09:69:fa:23:d0:fb:
    67:f7:c9:55:63:00:0a:56:d8:22:5a:78:93:41:9f:
    2e:18:ac:51:d4:d3:30:7b:6c:5e:75:36:46:81:48:
    1d:bb:d1:63:3a:8f:98:65:42:cc:fe:85:d0:2e:e7:
    da:41:16:cf:12:ef:ba:f1:d2:37:da:d5:e0:ff:c0:
    1b:f8:cf:3b:52:fa:6a:3a:81
prime1:
    00:e7:00:ac:f8:ee:68:a3:58:58:5c:2b:4f:f1:06:
    d8:58:68:02:87:f4:ae:08:e3:92:7b:12:aa:3b:76:
    c6:fb:ab:d3:63:3f:f1:4a:ca:03:6e:29:11:ca:1a:
    7d:ba:7d:45:2a:77:c6:74:3f:40:d6:61:01:3d:f4:
    07:b0:a0:e6:e6:f9:95:f8:c9:d3:f2:bd:31:29:25:
    e0:bc:e0:bf:06:01:bd:34:6f:12:ba:b1:93:ab:39:
    86:84:2d:a0:b4:75:ac:7c:61:90:93:e2:35:7f:c0:
    63:7e:72:91:2c:9e:2e:db:09:19:84:1d:7d:c7:94:
    35:17:17:c4:aa:d4:1c:5b:7f:0c:2c:18:92:30:db:
    e0:a4:b7:53:77:1e:7b:45:f2:c4:90:02:92:b4:83:
    c7:b2:61:1d:05:84:38:10:5f:2a:61:f2:61:81:d4:
    9d:29:a4:92:19:82:38:5a:9d:98:d9:b0:fe:22:c6:
    fd:af:02:ce:0c:8d:fd:fe:e1:b0:d9:f9:2b
prime2:
    00:ca:07:24:0d:a1:3b:6d:53:7c:82:6a:f6:ef:66:
    a3:0b:70:80:0c:c5:dc:28:1b:3a:43:61:b1:79:68:
    b9:0a:f9:50:4d:45:fd:63:b3:c8:fb:ad:f1:13:89:
    e5:a6:67:a3:6c:f6:19:c4:27:4e:cf:e8:61:2f:8d:
    47:26:36:ed:6f:8b:c4:9a:76:88:47:b6:0e:9d:d0:
    51:d9:6c:78:8d:d2:49:23:8c:cb:a2:77:0f:d3:2e:
```

```
    05:49:0a:84:d4:32:e4:65:c7:47:d9:7a:2e:a4:1a:
    15:78:a2:24:39:96:98:39:13:cc:27:52:74:ad:c4:
    00:9b:c8:9e:91:21:fa:ef:9b:1c:59:5c:15:b5:44:
    5c:9c:7a:77:03:f7:a9:69:d3:da:5c:38:da:57:ee:
    60:c8:9c:45:4b:41:b9:d9:63:40:7d:f6:71:70:55:
    8a:44:26:4c:c9:07:86:a5:0a:b1:b1:c3:89:ca:5b:
    49:f4:52:fd:f9:39:24:f5:06:50:30:85:47
exponent1:
    25:23:f5:ab:9c:61:54:89:fa:c2:ee:ef:ce:77:e4:
    46:ea:8a:25:a3:d0:6b:7b:73:6c:b8:46:88:83:03:
    61:29:72:36:4d:ec:94:b2:c0:34:71:03:fc:33:a0:
    2d:60:c0:c3:20:38:d7:2d:e8:55:cf:88:ec:96:14:
    ba:70:54:4f:a4:a7:59:35:d2:0f:00:1e:2c:58:7b:
    b6:c2:87:d4:06:69:8e:49:a1:80:44:d6:d2:3b:d0:
    85:e5:f4:25:af:99:c8:f1:c2:d6:14:13:b7:f3:8d:
    cb:a1:cd:f7:97:83:3f:12:4a:78:f4:68:e9:b2:c9:
    8a:69:f6:e3:e4:70:9e:c1:61:8a:a1:74:b7:c8:52:
    69:09:54:b1:1d:44:82:ad:92:ae:f8:ca:ef:9d:14:
    79:78:a5:ba:e2:54:45:45:97:c1:e1:bf:8d:a9:4a:
    8f:8c:77:35:04:bb:dc:cd:e3:ea:74:4b:97:f9:d8:
    85:cf:f4:a3:0e:1d:5d:62:9a:15:a1:bb
exponent2:
    1f:f9:29:57:8e:e0:dc:d8:8d:a8:06:4d:b6:6d:c3:
    f8:17:81:ec:83:93:e8:35:06:ef:8b:12:8f:68:67:
    80:b9:1c:60:5e:67:4f:d4:30:46:c4:ac:96:af:08:
    4d:61:b1:97:99:0b:52:e3:f5:b1:29:d1:d7:b8:c0:
    3d:e8:0b:83:cf:d6:f9:ab:30:be:48:ad:df:84:0c:
    b0:20:5e:a3:f3:57:e7:ec:6c:7d:f5:e1:e7:46:2d:
    47:f6:06:37:9f:26:4e:85:4f:75:b7:c4:91:ec:1e:
    e1:cc:a7:77:05:c2:69:a6:1c:75:4c:b3:72:9c:c6:
    8b:e1:20:57:4f:cd:6b:06:5d:62:37:14:a8:6f:7d:
    48:b6:89:07:73:b6:b8:2c:f3:2e:0d:41:61:11:34:
    f8:0a:e3:5c:99:b6:54:15:45:2b:aa:49:21:c2:27:
    f3:c9:2b:f5:d4:df:16:57:ae:ef:b7:46:a3:63:f7:
    3c:57:b6:22:2f:4d:0e:0a:45:be:a8:19
coefficient:
    79:f7:ca:f3:cd:b9:63:45:00:14:3b:eb:3e:26:1e:
    92:df:6b:6f:ad:0e:0c:8e:4e:90:99:c4:45:2a:bc:
    e5:95:ac:7e:91:07:da:1e:c8:65:6c:e2:78:f0:03:
    27:9f:a1:6c:93:d3:a0:7d:44:48:c5:67:d9:43:1b:
    9f:68:3f:6b:e5:3b:88:ad:d0:95:cf:4c:e9:df:55:
    f8:6f:5a:da:d4:54:5d:df:c3:d6:3a:94:e5:4f:f0:
    43:05:26:9a:62:60:64:36:e1:5a:91:00:00:e5:46:
```

```
    69:2f:ac:be:5d:62:0f:08:35:f7:34:f3:56:e8:2f:
    7b:f7:e8:5e:90:ba:38:4d:ad:e7:fb:fa:2f:9e:22:
    18:fc:b7:a5:74:8b:74:a4:78:5a:34:c5:81:af:1d:
    de:ad:71:22:ff:f1:81:30:42:b8:1d:3a:22:55:48:
    62:23:f5:92:9e:2c:65:46:62:bc:67:5c:d3:69:05:
    ee:17:a9:0b:d7:7a:3f:14:de:42:1a:61
```

To generate public key:

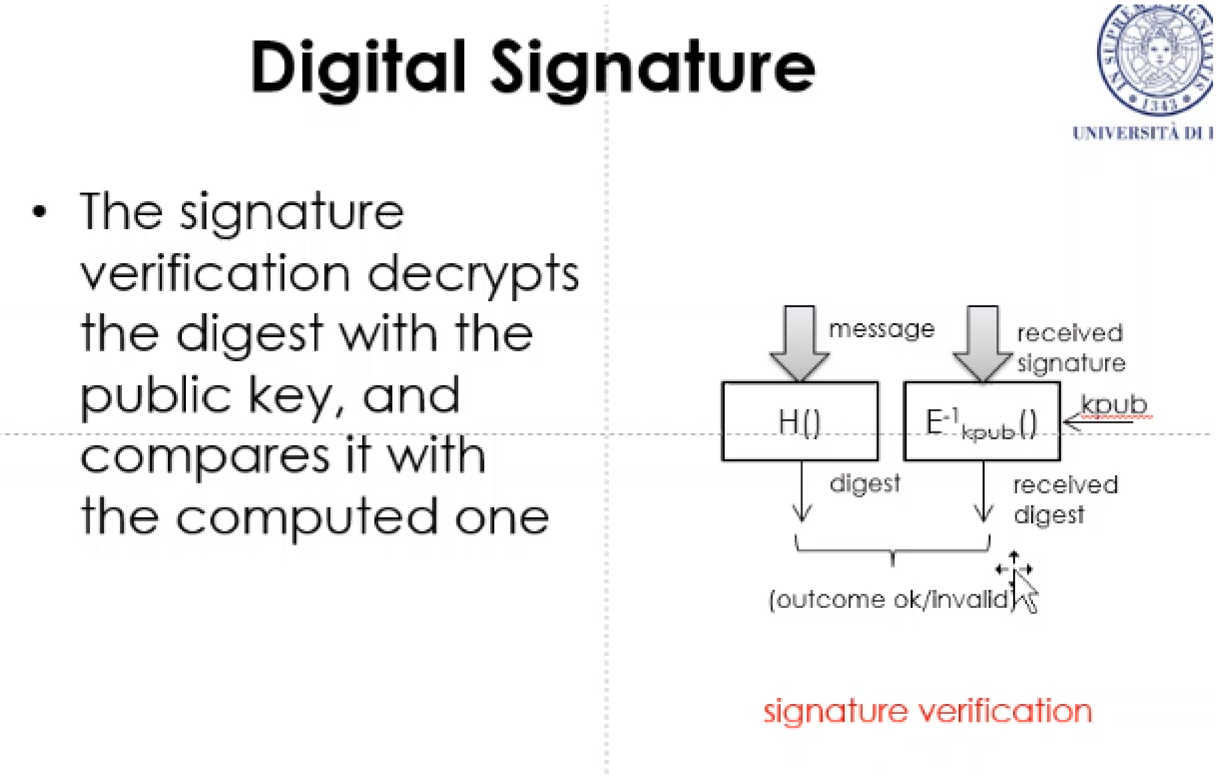
```
spadula@DESKTOP-QN8KFBP:~$ openssl rsa -in rsa_privkey.pem -pubout -out
rsa_pubkey.pem
writing RSA key
spadula@DESKTOP-QN8KFBP:~$ cat rsa_pubkey.pem
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAtkz6CYvF7IIJ8rJRSLuf
qELgtSIgeuuwsvM7C2MAUEsZJVWznFUfhqCK6A31OFz4jLraWYgSlOoamfHZtZTg
PkUeBLbhErzLfZ7sm55n8fVlxZUc5kZkjZ3k407IRFboPeU4BfzPSU3hd/zBvNE6
FlvxP2RLdjs5UnoRcu57sBBQa2EW0M669s4tlESaTtuFrEI9TvwH6ao3A3ZKEGN1
VBABZ7CgA4SSN1w2ntSc9q/HBPOv53NQEJ37Y3mlccBnzJeMkDTxEn6tbSJ4n2fw
9twHBVt3pSMix3NFsGlOFvfTGxNxRR2WQwh/PedoLWEi1EgXmSquJet+vA/RoX1X
uE8jSXq0kjX/ZVf5U7ajJ2UFpLlE8eizn06/y2A7BdCb9RedAetKPQfMkxVl4iv5
vQRmNptM/BUCYdyXkG6HKrX9aFM4Z2oNubggz5pZgWKbL+1tTc2Wmnjzgl7fmG/m
fA0T1hWBiXWQYEJCePeCja4cYb24SZXjBgYQNjdy93HtAgMBAAE=
-----END PUBLIC KEY-----
```

To print public key in text mode:

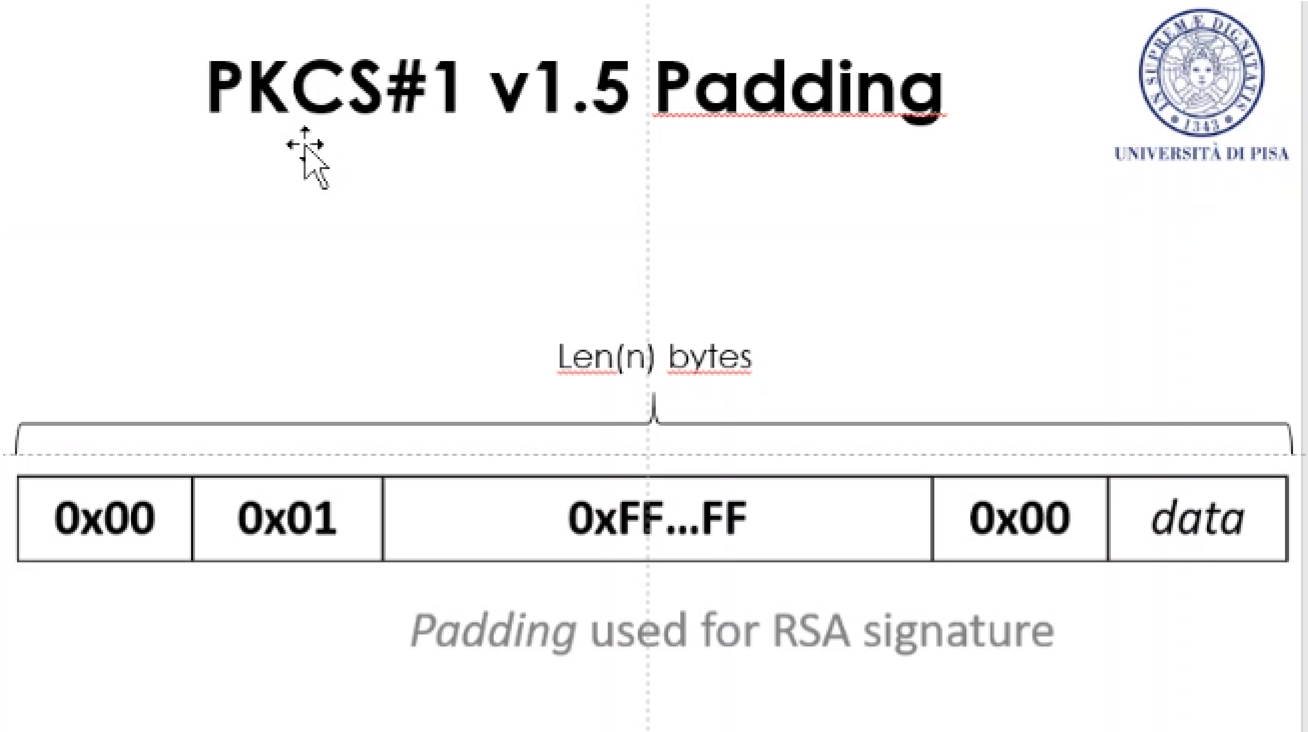
```
spadula@DESKTOP-QN8KFBP:~$ openssl rsa -pubin -in rsa_pubkey.pem -noout -
text
RSA Public-Key: (3072 bit)
Modulus:
    00:b6:4c:fa:09:8b:c5:ec:82:09:f2:b2:51:48:bb:
    9f:a8:42:e0:b5:22:20:7a:eb:b0:b2:f3:3b:0b:63:
    00:50:4b:19:25:55:b3:9c:55:1f:86:a0:8a:e8:0d:
    f5:38:5c:f8:8c:ba:da:59:88:12:94:ea:1a:99:f1:
    d9:b5:94:e0:3e:45:1e:04:b6:e1:12:bc:cb:7d:9e:
    ec:9b:9e:67:f1:f5:65:c5:95:1c:e6:46:64:8d:9d:
    e4:e3:4e:c8:44:56:e8:3d:e5:38:05:fc:cf:49:4d:
    e1:77:fc:c1:bc:d1:3a:16:5b:f1:3f:64:4b:76:3b:
    39:52:7a:11:72:ee:7b:b0:10:50:6b:61:16:d0:ce:
    ba:f6:ce:2d:94:44:9a:4e:db:85:ac:42:3d:4e:fc:
    07:e9:aa:37:03:76:4a:10:63:75:54:10:01:67:b0:
    a0:03:84:92:37:5c:36:9e:d4:9c:f6:af:c7:04:f3:
    af:e7:73:50:10:9d:fb:63:79:a5:71:c0:67:cc:97:
```

```
    8c:90:34:f1:12:7e:ad:6d:22:78:9f:67:f0:f6:dc:
    07:05:5b:77:a5:23:22:c7:73:45:b0:69:4e:16:f7:
    d3:1b:13:71:45:1d:96:43:08:7f:3d:e7:68:2d:61:
    22:d4:48:17:99:2a:ae:25:eb:7e:bc:0f:d1:a1:7d:
    57:b8:4f:23:49:7a:b4:92:35:ff:65:57:f9:53:b6:
    a3:27:65:05:a4:b9:44:f1:e8:b3:9f:4e:bf:cb:60:
    3b:05:d0:9b:f5:17:9d:01:eb:4a:3d:07:cc:93:15:
    65:e2:2b:f9:bd:04:66:36:9b:4c:fc:15:02:61:dc:
    97:90:6e:87:2a:b5:fd:68:53:38:67:6a:0d:b9:b8:
    20:cf:9a:59:81:62:9b:2f:ed:6d:4d:cd:96:9a:78:
    f3:82:5e:df:98:6f:e6:7c:0d:13:d6:15:81:89:75:
    90:60:42:42:78:f7:82:8d:ae:1c:61:bd:b8:49:95:
    e3:06:06:10:36:37:72:f7:71:ed
Exponent: 65537 (0x10001)
```

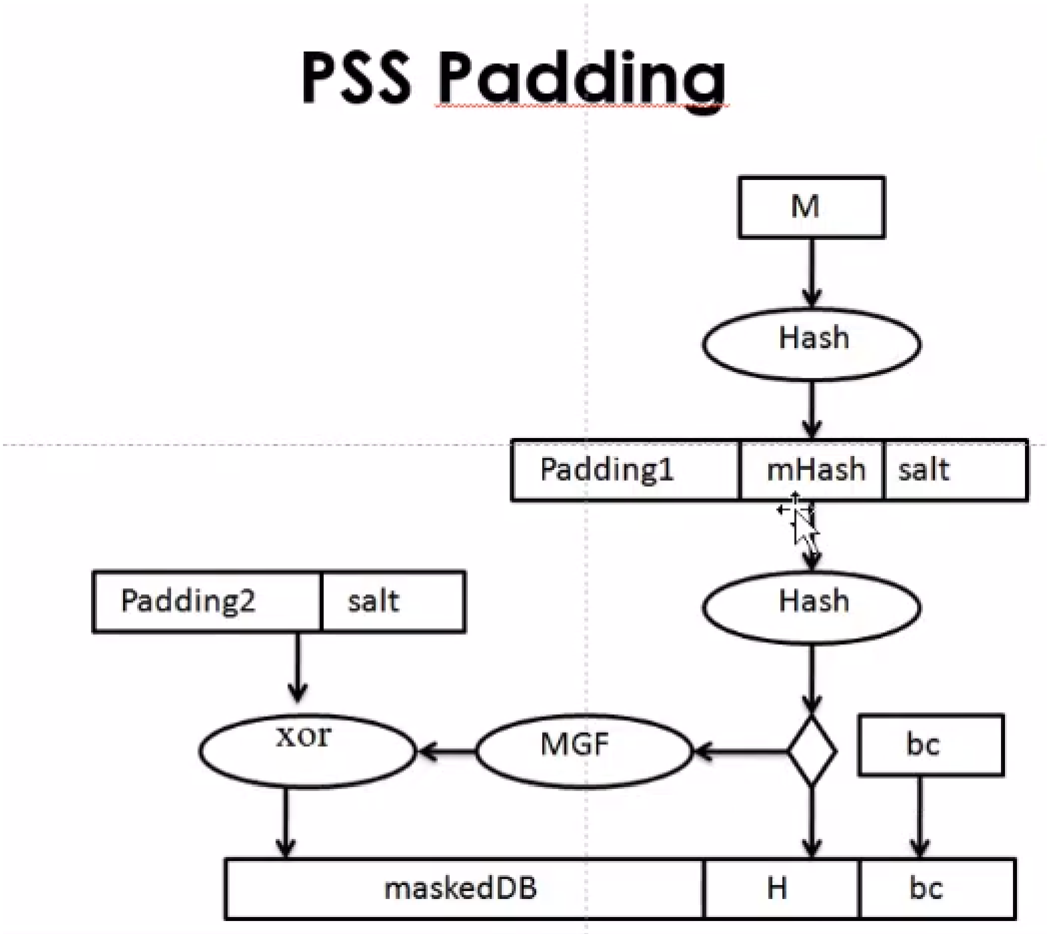
Digital Signature



RSA così come si studia in ambito didattico non è sicuro ne in cifratura ne in firma digitale. Esistono diversi tipi di attacchi come ad esempio se viene cifrato un plaintext di tutti 0 byte il cyphertext sarebbe tutti 0. RSA da sola è solo una trapdoor function quindi bisogna calcolare il padding non solo per fargli raggiungere la lunghezza necessaria ma anche per difenderlo da semplici attacchi. Il padding più semplice è quello presentato di seguito PKCS#1 usato per la firma digitale:



vengono aggiunti dei dati prima.
Questo padding risulta sicuro ma esistono degli attacchi abbastanza difficili da realizzare in pratica che potrebbero in qualche modo diminuirne la sicurezza. Questo padding è deterministico ovvero firmando due messaggi con la stessa chiave otteniamo sempre lo stesso messaggio.
Per ottenere una maggiore sicurezza dimostrabile bisogna usare un altro padding di tipo probabilistico, ottenendo due firme diverse anche se usato sullo stesso messaggio, il **PSS**:



in pratica si tratta non di un padding ma di una trasformazione probabilistica del messaggio originale.

RSA ha però un grande problema di performance, ovvero mandare le firme digitale su un socket costa molto in termini computazionali e tutte le operazioni con la chiave privata sono molto lente (a differenza di quelle fatte con la chiave pubblica che sono molto più veloci).

Elliptic Curve

Elliptic-Curve Cryptography

UNIVERSITÀ DI PI

RSA key length	Equivalent EC key length*	NIST** curve name	OpenSSL*** curve name	Effective strength	Recommen-dation
1536 bits	192 bits	P-192	prime192v1	96 bits	Low security
2048 bits	224 bits	P-224	secp224r1	112 bits	Medium security
3072 bits	256 bits	P-256	prime256v1	128 bits	Good security
7680 bits	384 bits	P-384	secp384r1	192 bits	TOP SECRET security****

* The RSA/EC security equivalence is taken from SEGC official documents.
** National Institute of Standards and Technology.
*** Supported by OpenSSL 1.0.1k.
**** Elliptic curve P-384 is recommended by NSA for TOP SECRET documents.

Il livello di sicurezza effetivo è la metà del numero di bit della curva ellittica usata.
Con molti meno bit (256) si ottiene lo stesso livello di sicurezza di 3072 bits con RSA con livelli di prestazioni maggiori.

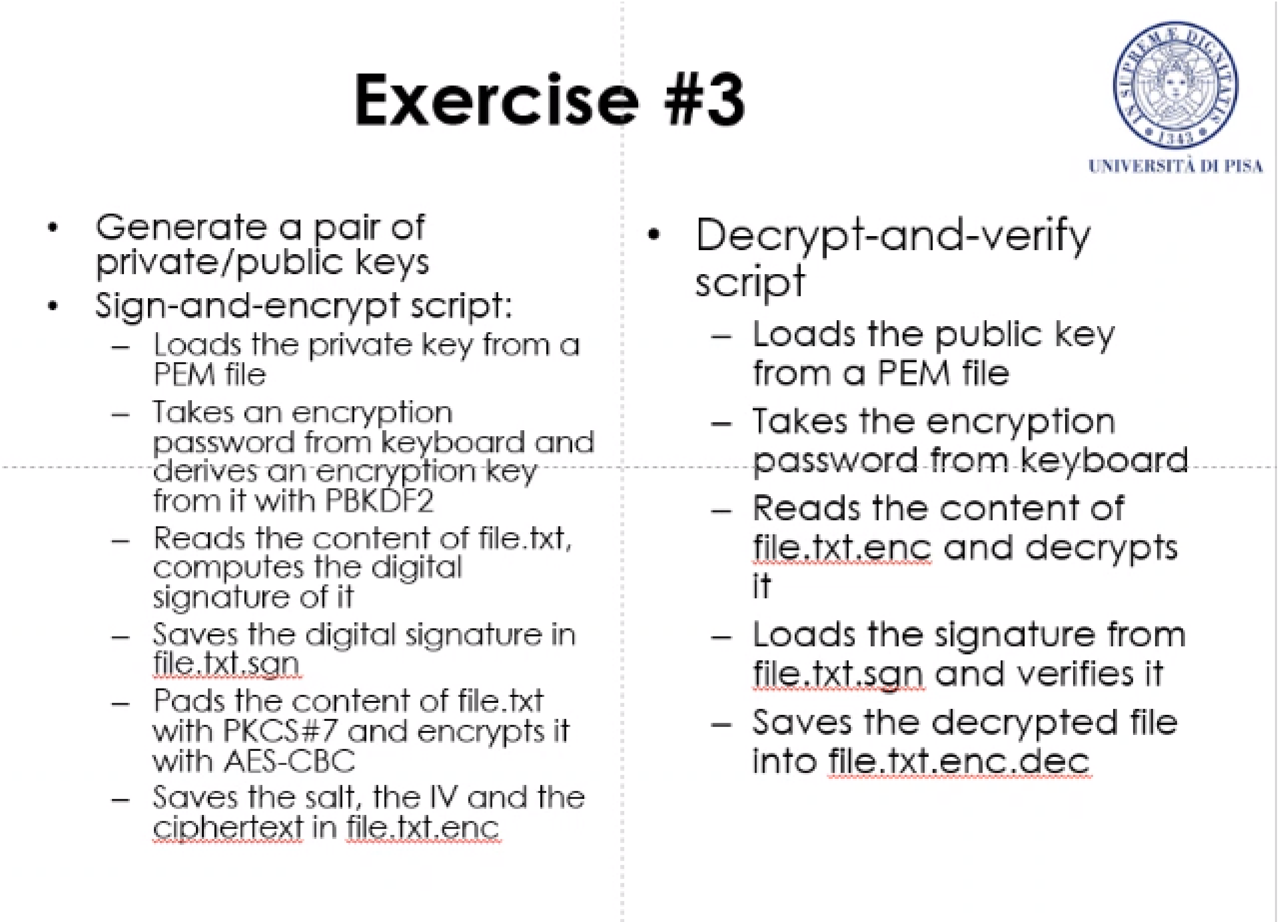
```
spadula@DESKTOP-QN8KFBP:~$ openssl ecparam -genkey -name prime256v1 -out ec_privkey.pem
spadula@DESKTOP-QN8KFBP:~$ cat ec_privkey.pem
-----BEGIN EC PARAMETERS-----
BggqhkjOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEINtptx2JaBln7FN9eg2u9d8AH3Qj8Yy6MGRUhkwmXQNpoAoGCCqGSM49
AwEHoUQDQgAEYqUJ+uoRtsWKXhzohRLGH5U1z30NRNOuioJNpVdv0vXYuRCXUht4
olFcopkCykXXDvi91tnQU187WgHbexfSPg==
-----END EC PRIVATE KEY-----
```

```
spadula@DESKTOP-QN8KFBP:~$ openssl ec -in ec_privkey.pem -noout -text
read EC key
Private-Key: (256 bit)
priv:
```

```
    db:69:b7:1d:89:68:19:67:ec:53:7d:7a:0d:ae:f5:
    df:00:1f:74:23:f1:8c:ba:30:64:54:86:4c:26:5d:
    03:69
pub:
    04:62:a5:09:fa:ea:11:b6:c5:8a:5e:1c:e8:85:12:
    c6:1f:95:35:cf:7d:0d:44:d3:ae:8a:82:4d:a5:57:
    6f:d2:f5:d8:b9:10:97:52:1b:78:a2:51:5c:a2:99:
    02:ca:45:d7:0e:f8:bd:d6:d9:d0:53:5f:3b:5a:01:
    db:7b:17:d2:3e
ASN1 OID: prime256v1
NIST CURVE: P-256
```

Estriamo la chiave pubblica come fatto prima con RSA:

```
spadula@DESKTOP-QN8KFBP:~$ openssl ec -in ec_privkey.pem -pubout -out
ec_pubkey.pem
read EC key
writing EC key
spadula@DESKTOP-QN8KFBP:~$ cat ec_pubkey.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEYqUJ+uoRtsWKXhzohRLGH5U1z30N
RNOuioJNpVdv0vXYuRCXUht4olFcopkCykXXDvi91tnQU187WgHbexfSPg==
-----END PUBLIC KEY-----
```



Esercizio 3

ex-enc.py

```
from cryptography.hazmat.primitives import serialization #allow to serialize
and deserialize keys from PEM format
from cryptography.hazmat.primitives.asymmetric import padding as
asym_padding
from cryptography.hazmat.primitives import hashes
import binascii
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
import os
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

content_file = ""
key = ""
salt = ""
iv = ""
hash_HMAC = hashes.SHA256()
iterations_HMAC = 2**14
salt_size = 16
iv_size = 16
enc_password = input("Type the password to encrypt:").encode()
file_plaintext_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt"
file_encrypted_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt.enc"
file_privkey_ec_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\ec_privkey.pem"
file_pubkey_ec_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\ec_pubkey.pem"
file_digital_signature_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt.sgn"

with open(file_privkey_ec_path,"rb") as f:
    content = f.read()
    print("[+] EC Private Key serialized")
    prvkey = serialization.load_pem_private_key(content, None)


# derive a key using PBKDF2
```

```python
salt = os.urandom(salt_size)
kdf = PBKDF2HMAC(hashes.SHA256(),16,salt,iterations_HMAC)
key = kdf.derive(enc_password)

print("[+] Salt generated:")
print(binascii.hexlify(salt))

print("[+] Key derived:")
print(binascii.hexlify(key))



#open plaintext file as text
with open(file_plaintext_path, "r", encoding="utf-8") as file_to_enc:
    print("[+] Plaintext file:")
    print(file_to_enc.read())




#open plaintext file as byte
with open(file_plaintext_path, "rb") as file_to_enc:
    content_file = file_to_enc.read()
    print("[+] Plaintext file (byte):")
    print(binascii.hexlify(content_file))

digital_signature = prvkey.sign(content_file, ec.ECDSA(hashes.SHA256()))
print("[+] Digital signature Elliptic Curve (ECDSA)")
print(binascii.hexlify(digital_signature))

#save digital signature to an output file
with open(file_digital_signature_path, "wb") as digital_signature_file:
    digital_signature_file.write(digital_signature)

#padding of file with PKCS#7

#open plaintext file as byte
with open(file_plaintext_path, "rb") as file_to_enc:
    content_file = file_to_enc.read()

p = padding.PKCS7(128).padder() #we must specify dimension of block in bit size that must be a 8-bit multiple

padded_text = p.update(content_file) + p.finalize()
```

```python
print("[+] Padded text (",len(padded_text), "byte", ")")
print(binascii.hexlify(padded_text))

#encrypt with AES in CBC modewith IV and KEY

iv = os.urandom(16)
print("[+] IV (",len(iv), "byte", ")")
print(binascii.hexlify(iv))


cipher = Cipher(algorithms.AES(key),modes.CBC(iv))
enc = cipher.encryptor()
ct = enc.update(padded_text)
print("[+] Ciphertext (",len(ct), "byte", ")")
print(binascii.hexlify(ct))

#save SALT + IV + ciperthext to an output file
with open(file_encrypted_path, "wb") as file_encrypted:
    file_encrypted.write(salt)
    file_encrypted.write(iv)
    file_encrypted.write(ct)

#read encrypted file
with open(file_encrypted_path, "rb") as file_encrypted:
    content_file_enc = file_encrypted.read()
    print("[+] Encrypted file (",len(content_file_enc), "byte", ")")
    print(binascii.hexlify(content_file_enc))
```

ex-dec.py

```python
from cryptography.hazmat.primitives import serialization #allow to serialize and deserialize keys from PEM format
from cryptography.hazmat.primitives.asymmetric import padding as asym_padding
from cryptography.hazmat.primitives import hashes
import binascii
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
import os
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

content_file = ""
signature = ""
```

```python
key = ""
salt = ""
iv = ""
hash_HMAC = hashes.SHA256()
iterations_HMAC = 2**14
salt_size = 16
iv_size = 16
dec_password = input("Type the password to decrypt:").encode()
file_plaintext_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt"
file_encrypted_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt.enc"
file_decrypted_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt.enc.dec"
file_privkey_ec_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\ec_privkey.pem"
file_pubkey_ec_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\ec_pubkey.pem"
file_digital_signature_path =
r"C:\Users\SamuelePadula\Desktop\Python\Crypto\Lesson3\file.txt.sgn"

with open(file_pubkey_ec_path,"rb") as f:
    content = f.read()
    print("[+] EC Public Key serialized")
    pubkey = serialization.load_pem_public_key(content, None)

#read SALT + IV + CT from file
with open(file_encrypted_path, "rb") as file_encrypted:
    salt = file_encrypted.read(salt_size)
    print("[+] Salt from file (",len(salt), "byte", ")")
    print(binascii.hexlify(salt))

    iv = file_encrypted.read(iv_size)
    print("[+] IV from file (",len(iv), "byte", ")")
    print(binascii.hexlify(iv))

    ct = file_encrypted.read()
    print("[+] Ciphertext from file (",len(ct), "byte", ")")
    print(binascii.hexlify(ct))


# derive a key using PBKDF2
kdf = PBKDF2HMAC(hashes.SHA256(),16,salt,iterations_HMAC)
```

```
key = kdf.derive(dec_password)
print("[+] Key derived:")
print(binascii.hexlify(key))


#decrypt cyphertext
cipher = Cipher(algorithms.AES(key),modes.CBC(iv))

#create an decryptor
dec = cipher.decryptor()
padded_plaintext = dec.update(ct)
p = padding.PKCS7(128).unpadder() #we must specify dimension of block in bit size that must be a 8-bit multiple
unpadded_plaintext = p.update(padded_plaintext) + p.finalize()
print("[+] Unpadded plaintext from file (",len(unpadded_plaintext), "byte", ")")
print(binascii.hexlify(unpadded_plaintext))

with open(file_pubkey_ec_path,"rb") as file_publickey:
    content = file_publickey.read()
    print("[+] EC Public Key serialized")
    pubkey = serialization.load_pem_public_key(content, None)

with open(file_digital_signature_path,"rb") as signature_file:
    signature = signature_file.read()
    print("[+] EC Signature serialized")

try:
    pubkey.verify(signature,unpadded_plaintext, ec.ECDSA(hashes.SHA256()))
except:
    print("[!!!] Invalid signature")
    exit()

print("[+] Valid signature")

#save decrypted cyphertext to an output file
with open(file_decrypted_path, "wb") as file_decrypted:
    file_decrypted.write(unpadded_plaintext)

#open file as text
with open(file_decrypted_path, "r", encoding="utf-8") as file_decrypted:
    print("[+] Decrypted File (text)")
    print(file_decrypted.read())
```

Certification Authority

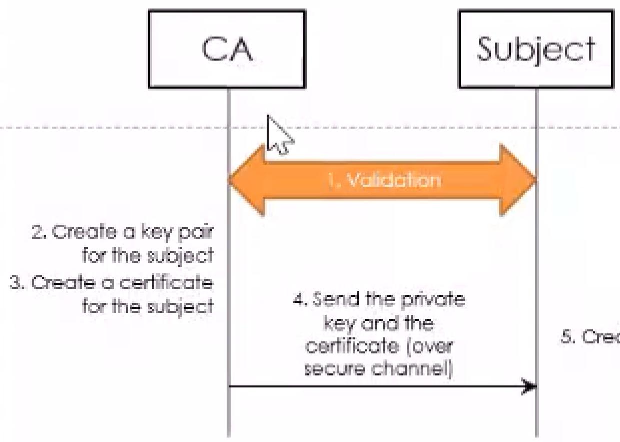
UNIVERSITÀ DI PISA

- A main problem in asymmetric cryptography is to be sure that a certain subject uses a certain public quantity
- For example, be sure that a server reachable at a certain domain "www.server.com" uses a certain RSA public key
- Simply sending the RSA public key over Internet is not safe, because a man in the middle could change it
- We need a trusted third entity called *certification authority* (CA)
  - Everyone trusts the CA
  - Everyone knows the CA's public key
  - The CA releases signed certificates, which bind a given subject to a given public quantity
- The most common type of certificates are *public key certificates*, which bind a given subject (usually an Internet domain or a company) to a given public key

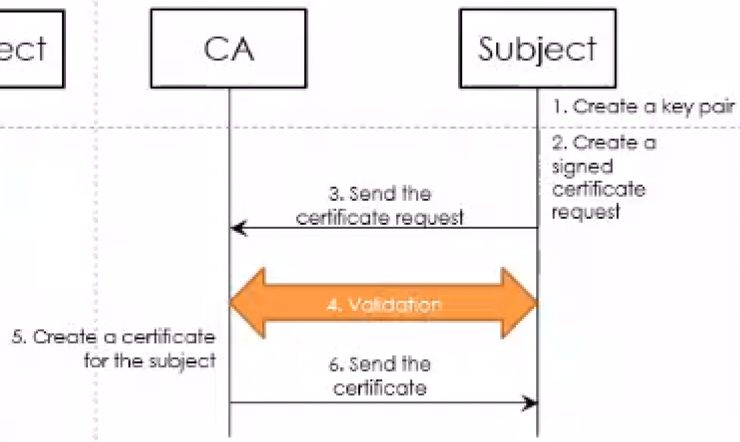
Creating Certificates

UNIVERSITÀ DI PISA

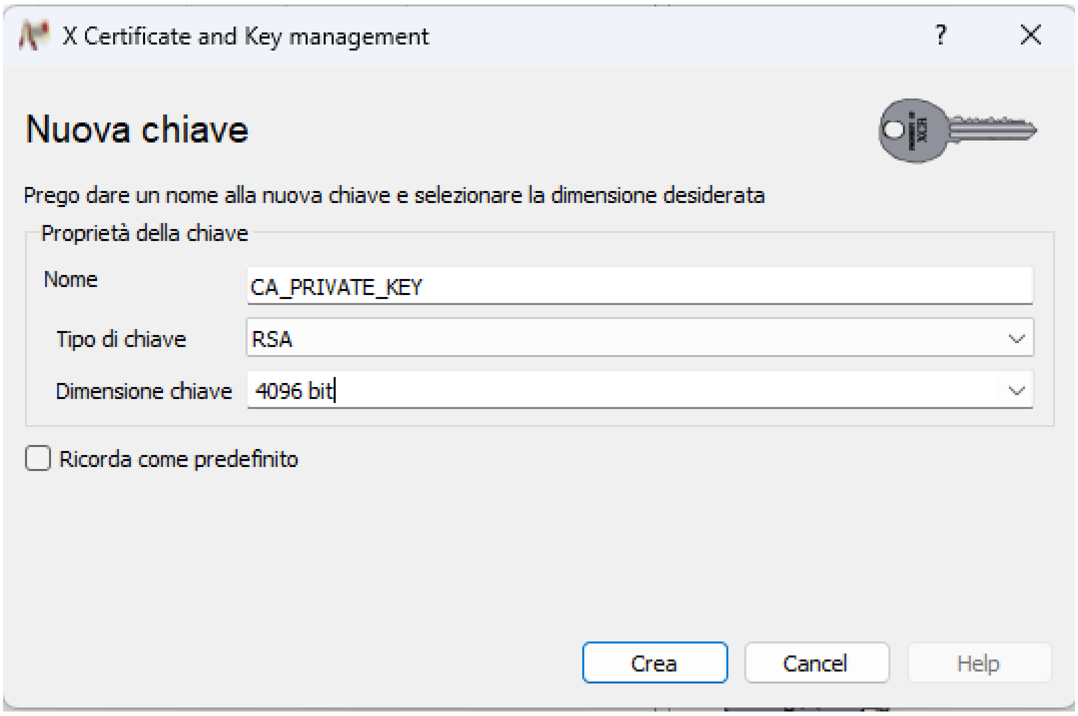
- Simple way:



- Secure way:



Bisogna creare una chiave privata per la CA:

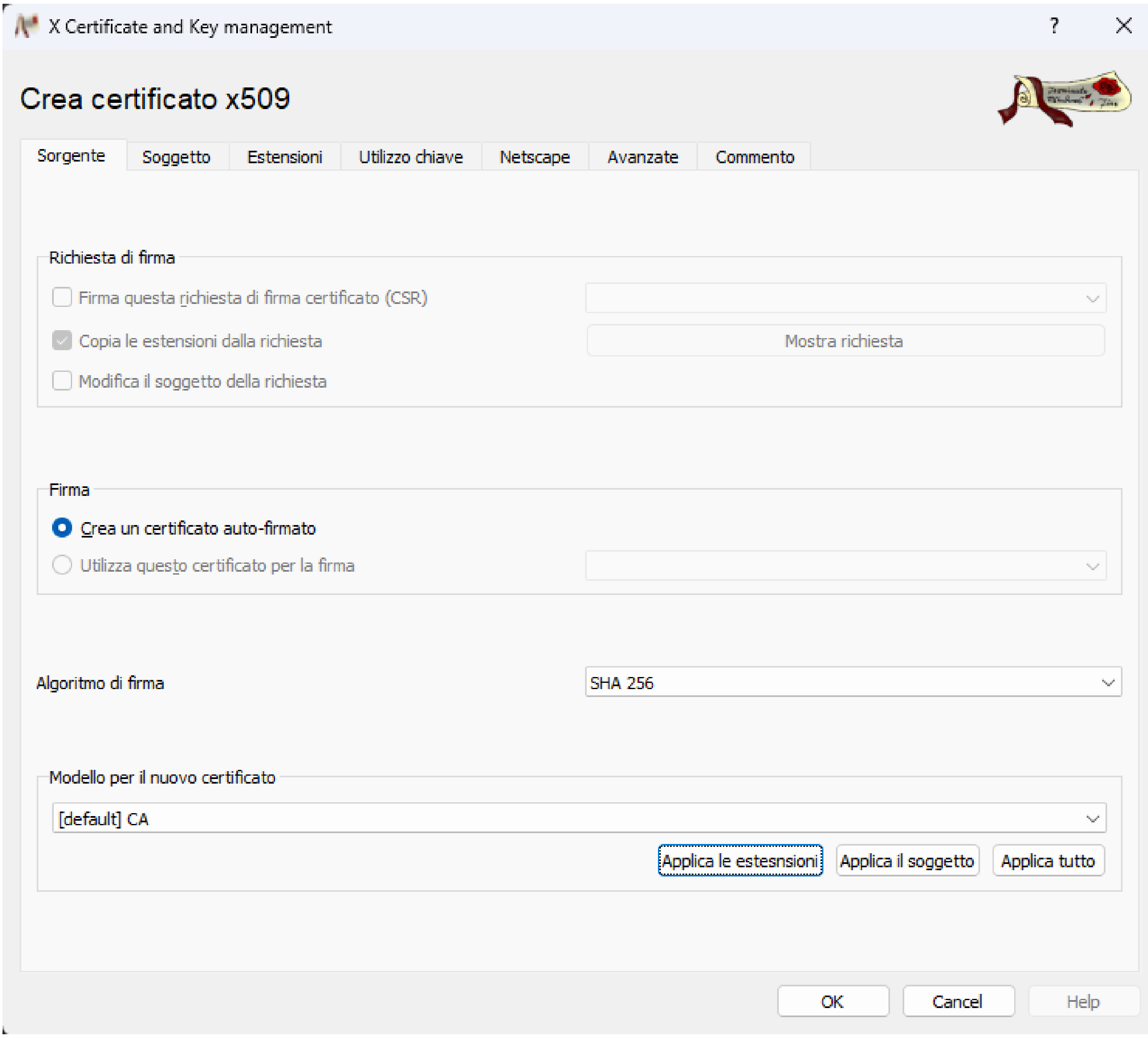


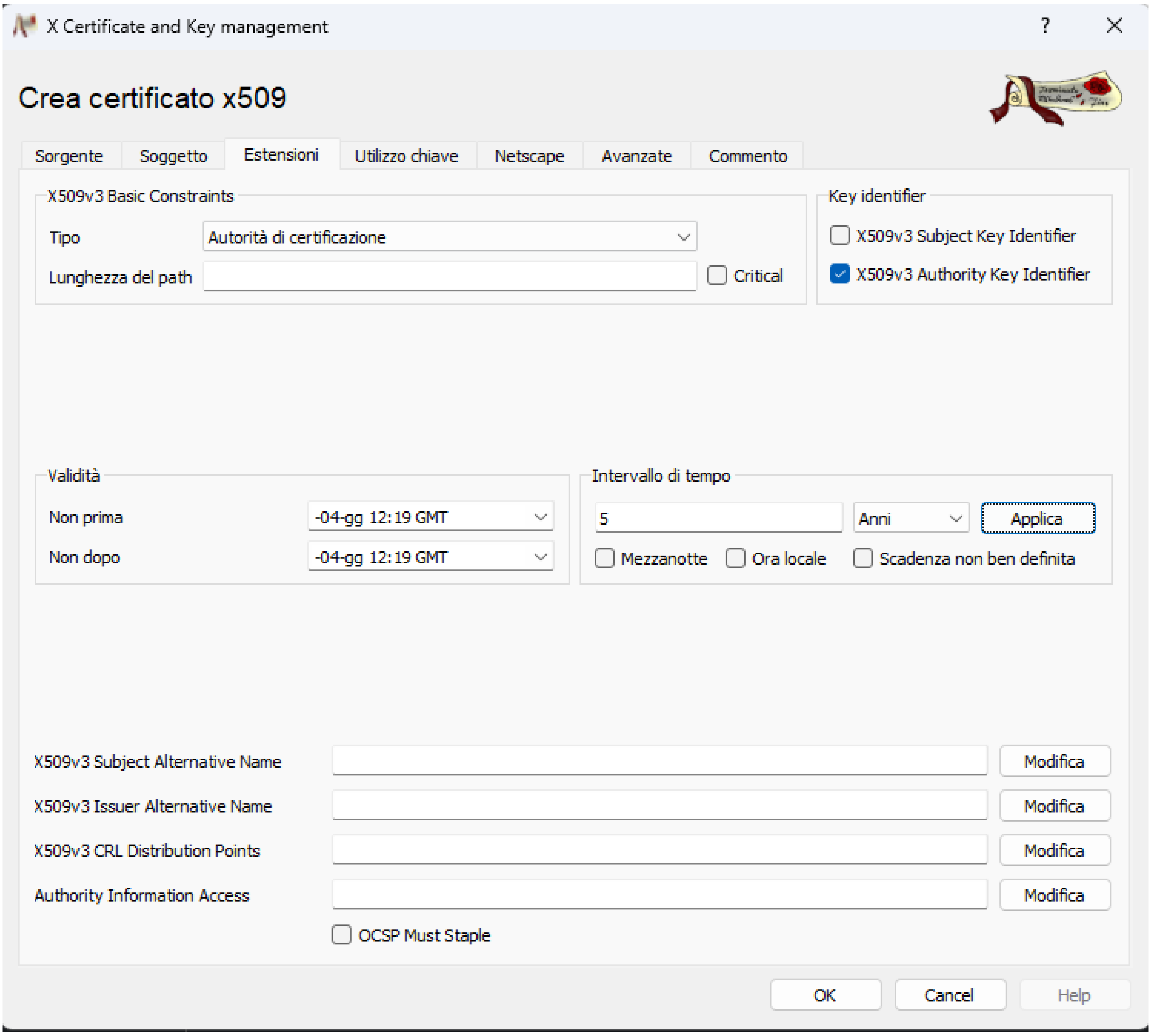
E' possibile esportare la chiave privata (per convenzione si usa esportare la chiave usando l'estensione .key):

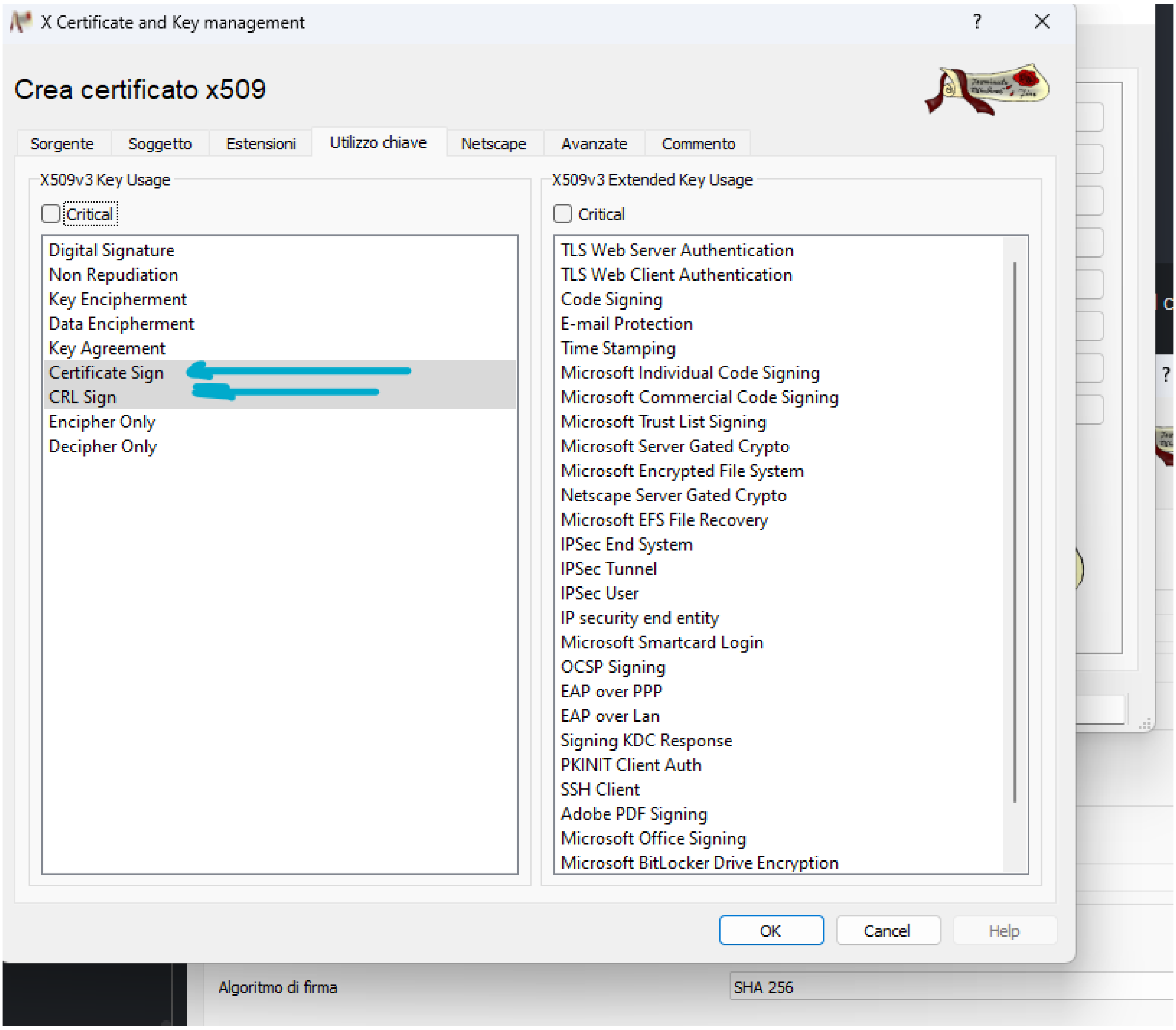

```
-----BEGIN RSA PRIVATE KEY-----
MIIJJwIBAAKCAgEAuoZFBKtP88iTZv6HaHdK1V8EZbHGpCWZuTz1w4QVF1U2Vuy8
mFH9bfIzSJMdy+5rjr7/2/k852x61bA+clYEwBLa3VuiaATX8pyP5OGdzJ+7eC1I
JBFq46hS4eZ6ptlHFSkkcrCxT2Qq/qm3WYhmPNKNTWa9ZLMVygemGujuf586XsMX
t+p4XjgQTrasPMIEkChtOiLuAHSqRXCR/TKHDc4sXqHE+k6sJYsTbBaTxOCXHpzP
0W9D0sSzB4oGd+8RKmhLYdbRSE6RWCWd/pQ3BxLgB8MS/dlJYsH1E271rrV9EMYq
Z99AECQYIWtM6JhkmuJ36ardJEgyXPcNfohLOZgW+AzcjaN5BMhkuv+24kJHq4hw
eLgE/sVM4BiHDoWuvqyt846eSpvoborizRp1XHWTn5ue/H3w/U8YygmyZ1IyZQz2
WVu6fjZCMs3fcnGIOU9f+nK1J8O90MSVlvLwBUxFc6ZwoJ33yPA8OW5/hdI1RyQ4
kfklGfsz0ImusHoeR5i3UKmwFMhvo6WTE/aUFBRchUaa0bSoFKUBQtOSnXzq3hTK
buTk6uBiBsjCijxePmbSAv+6yNyexA1yqoBKaHewnD/mhbxsGgFMbspa2fb4ft+/
NTYj67ArD48dc6F6sfR/arHVnuIiUB3hss6uJ5HAs9UFx/cEF0N0E4N6rr0CAwEA
AQKCAgAOrWQe8vAOqdjXgO3czY5I7folmnCcGA1Zoy9fnDQgqubkvio0/jaoASgB
7PmqQ2+ZQvRVNe6R81pTGFRBoP0stA8e3ggQkVkA0UsNYqeNI4CX+1Ay7l/v3B3x
grZiDLToOF2S3M6HBcXQVkCEPhR1csgFsDD1DJaJWEK82VlzF297Z3SlV0DzgQvf
dcZl3YqIYXX+3iLGTGfdoPcL6n7wGioppCcg3zHynlf/6GLmWhnqIcLEozG+ExZX
YIgTxe8e3CgaYdIEE5KiUgt1reoKl3Sjh7kl2oRIziRfzBpvDf7yY/WQSOP8oSp4
7r2Vi/BWs71l2LkgEsJLyHD6ZPUuR3faFyDYL+VlB3WCZ+Qt5OvSq8gaFZREJaE0
m6I95A+WQD+kge52Ls2KcK+xudrbijHWG7c5UyRrJ/IvIfgbTkyQmyA9FrKrwr0C
IYZiDxYXQqp9gpoMDzPXeOTJ6PFriTIRmrVYrecAXxZmTZUc7uxAOmsiQ8UJ/YLZ
cnBFkbnevYGx22dOL2OQ+LzBCeh+Dyf46OJK2eaN2p5bFTIQEqQJKH3CiBCqerDO
SDNY8wahUAJwciYQD42tghy8RXaZTvG0yTBzusoLx6ExMi2LQ2TUdnfojnG+Rav4
YJD2ep4YAi1ODKf+xzVDKOmkblChbGBPweCRmPmWH79J6VkywQKCAQEA8SOGN02r
sBtnifTXIOdWP7IxnAMGS0MlE/VF6PfjzTJ317bjf1PItfNS5X1/jfOPHRNvGr7E
```

0xxN8Se2TtBeeV8GkYCkdqXrJ35OLdkqUxHPwSH+gZMgjSjK7sDAfkLpLbWHJV58
Zhh0DzzFEksK3vYzQar72z6wmZLGGZ5SEvwbTGy2zmXjGOuRF+Om+WbnAJ6NbpIS
l8Z/s2W5h5vEzkQtGrBnMbGrqTQKf9PtmmpXV2YhUb9NeTy7z57b7WrCHGo8X2Ps
Br/P1WM5V5gyJcWIPPEbUqiSezpuIzFVFyJi/xc8NCzYBk5T2JXBM7mUm2XUhVMO
yfGhyWiqGhJe0QKCAQEAxgUW177evKHTNhITKJkmlodl/MVRlAGNvyyzT+7Mr2Cc
a+3w20S+TG5FLz4qRrtqgeIEcbgBikdkJHCzaxj5SgOgBipKd5Vrn0hK9Xs1IfrJ
0nYrWEKCqJzukRC2/xv8EfkS8AWDavgWxfRqL0JN+TG3eCdkCJa/1EI6YH2UQ4TW
M4xoQpXRckVbj4kdIgiXkzghzEJX8t1TKRiY9LzAuGAchf6tNnEfYLcYlIElOhMP
fHIp1C5N4t/rXFg7FGTl+D1nITsFkXIuY3YABBLBL9w/S8P5G5x5J7ktx8qb9H/K
s2sMr+Te7LQ/FELK79bArnzR/4LlLk3TLtFMfSjELQKCAQBWcrzgQLknVnvFCoDB
bA4QocqFOtRb0QOO5ySCA5qoaspqDEf80sWm7UevvFEpS8Ln0prHRNL9OC19IhaK
pMrpyjZpnWvYmVz3eKGAcFVrGHyZqZ07SMqnsJMoCvQ3j7dWyrhbnkcMtwGMoOWp
zDtmeW8gwLKwBAZ92A+rCYY1BiqnOGZFEmPbAECxBs1Kpih0oWLk2/tMbD5Fy1c7
FY31wJ1G7yzftOlsrJqC/zA0ZqFPVO7nBqU4rJxML5B6ygYy96cTL5hjRwq0XnEl
RQvdXLad2nZIKOTyxpzLgxkVRR+mgeb0cYs0n4oRoIZ2C7cKCvSoo4EuxrCQqzQo
SqOxAoIBAB3iuL7Y9L7dYYYgljmjW5qOVssecKB4147JzUo8DTJOz2zOnXJKXrok
TlbB+BlywbFWjjsnfTwEaE3DoKCCRWVxRJ6JlXGU4IhKnd2MuckmE32rDgGlEBko
jizgq+22qIWB5OTKwDnNtYosyDXXuPLqGPmOYF+XeN4tHKhha5YBH17qSvX5rIGl
jBsOo5H2YAH9D8THIoTp+FoUd0lAj4mEH+ntNPEpg4XSPGh1UUwBgm3SwRNf5atf
BiOKwWfjjn23rq/qO33nEK88KOZ0eimiAP+LVTZgmDxxi0JDBuSQwOPxPRny3d41
meuR5RTrgWsUNZFtjO8/GadQOU4OcAUCggEAS8lKByd7QZuiSil4qChWWEzhG49A
7rRDxV+uUQfX5L8e0k9GYeQohleqQcnm/So4Q403V09qZUTKv8itji/TUags3P+3
2uKN5kEWoK5HFvrqOes4LiVVPJN6XSQw824dt6FFgJj5DoVGTwTBVkgbDzkJIrZY
gKsqR94RWAchns80Oha535qgir7ic8BBZ0VNThF6ji24u330W2u5LxO9P5ZVky8L
VWkh5uo25Zi7fADL4PjDxpQdEJVnQ8EpbmRW7EHBg9G+oWFcbGIX8Eq08Mt1+2f8
sPODbYxs5Dc7mPz4V0gZvxf+x6fupsAk24qBXGz7WrDlu84sVP4WyLDgTA==
-----END RSA PRIVATE KEY-----

All'interno del certificato invece sarà contenuta la chiave pubblica e tutte le informazioni sull'issuer del certificato e sul soggetto al quale il certificato è stato rilasciato:


X Certificate and Key management
Crea certificato x509
Sorgente
Soggetto
Estensioni
Utilizzo chiave
Netscape
Avanzate
Commento
Richiesta di firma
Firma questa richiesta di firma certificato (CSR)
Copia le estensioni dalla richiesta
Mostra richiesta
Modifica il soggetto della richiesta
Firma
Crea un certificato auto-firmato
Utilizza questo certificato per la firma
Algoritmo di firma
SHA 256
Modello per il nuovo certificato
[default] CA
Applica le estesnsioni
Applica il soggetto
Applica tutto
OK
Cancel
Help


X Certificate and Key management
Crea certificato x509
Sorgente
Soggetto
Estensioni
Utilizzo chiave
Netscape
Avanzate
Commento
X509v3 Basic Constraints
Tipo
Autorità di certificazione
Lunghezza del path
Critical
Key identifier
X509v3 Subject Key Identifier
X509v3 Authority Key Identifier
Validità
Non prima
-04-gg 12:19 GMT
Non dopo
-04-gg 12:19 GMT
Intervallo di tempo
5
Anni
Applica
Mezzanotte
Ora locale
Scadenza non ben definita
X509v3 Subject Alternative Name
Modifica
X509v3 Issuer Alternative Name
Modifica
X509v3 CRL Distribution Points
Modifica
Authority Information Access
Modifica
OCSP Must Staple
OK
Cancel
Help

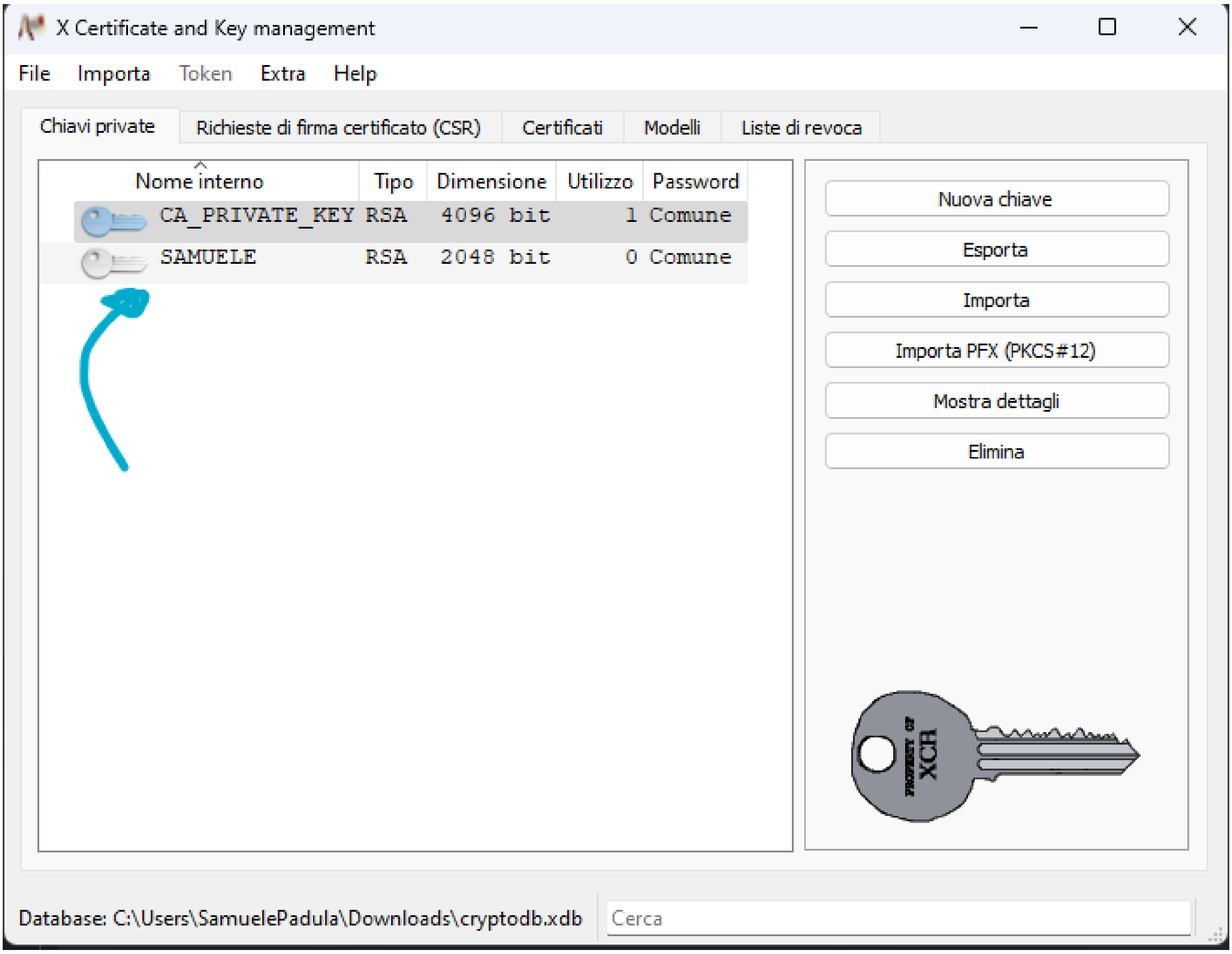



```
root@root:$openssl x509 -in CA.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4627467661766718247 (0x403811522d367327)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IT, ST = PI, L = Pisa, O = CA, CN = CA, emailAddress =
ca@ca.com
        Validity
            Not Before: Apr 14 12:19:00 2023 GMT
            Not After : Apr 14 12:19:00 2033 GMT
        Subject: C = IT, ST = PI, L = Pisa, O = CA, CN = CA, emailAddress =
ca@ca.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
```

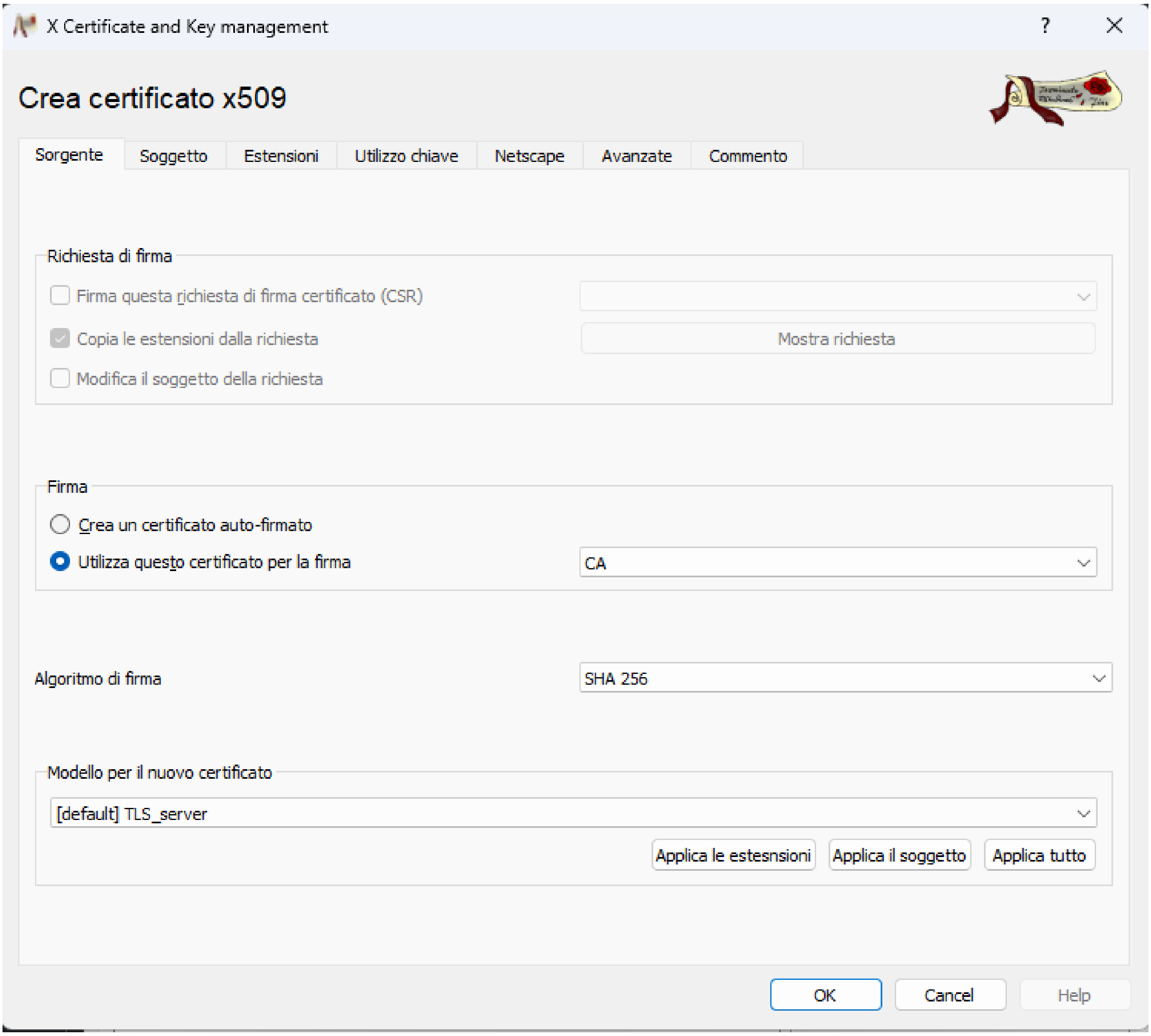
```
                    00:ba:86:45:04:ab:4f:f3:c8:93:66:fe:87:68:77:
                    4a:d5:5f:04:65:b1:c6:a4:25:99:b9:3c:f5:c3:84:
                    15:17:55:36:56:ec:bc:98:51:fd:6d:f2:33:48:93:
                    1d:cb:ee:6b:8e:be:ff:db:f9:3c:e7:6c:7a:d5:b0:
                    3e:72:56:04:c0:12:da:dd:5b:a2:68:04:d7:f2:9c:
                    8f:e4:e1:9d:cc:9f:bb:78:2d:48:24:11:6a:e3:a8:
                    52:e1:e6:7a:a6:d9:47:15:29:24:72:b0:b1:4f:64:
                    2a:fe:a9:b7:59:88:66:3c:d2:8d:4d:66:bd:64:b3:
                    15:ca:07:a6:1a:e8:ee:7f:9f:3a:5e:c3:17:b7:ea:
                    78:5e:38:10:4e:b6:ac:3c:c2:04:90:28:6d:3a:22:
                    ee:00:74:aa:45:70:91:fd:32:87:0d:ce:2c:5e:a1:
                    c4:fa:4e:ac:25:8b:13:6c:16:93:c4:e0:97:1e:9c:
                    cf:d1:6f:43:d2:c4:b3:07:8a:06:77:ef:11:2a:68:
                    4b:61:d6:d1:48:4e:91:58:25:9d:fe:94:37:07:12:
                    e0:07:c3:12:fd:d9:49:62:c1:f5:13:6e:f5:ae:b5:
                    7d:10:c6:2a:67:df:40:10:24:18:21:6b:4c:e8:98:
                    64:9a:e2:77:e9:aa:dd:24:48:32:5c:f7:0d:7e:88:
                    4b:39:98:16:f8:0c:dc:8d:a3:79:04:c8:64:ba:ff:
                    b6:e2:42:47:ab:88:70:78:b8:04:fe:c5:4c:e0:18:
                    87:0e:85:ae:be:ac:ad:f3:8e:9e:4a:9b:e8:6e:8a:
                    e2:cd:1a:75:5c:75:93:9f:9b:9e:fc:7d:f0:fd:4f:
                    18:ca:09:b2:67:52:32:65:0c:f6:59:5b:ba:7e:36:
                    42:32:cd:df:72:71:88:39:4f:5f:fa:72:b5:27:c3:
                    bd:d0:c4:95:96:f2:f0:05:4c:45:73:a6:70:a0:9d:
                    f7:c8:f0:3c:39:6e:7f:85:d2:35:47:24:38:91:f9:
                    25:19:fb:33:d0:89:ae:b0:7a:1e:47:98:b7:50:a9:
                    b0:14:c8:6f:a3:a5:93:13:f6:94:14:14:5c:85:46:
                    9a:d1:b4:a8:14:a5:01:42:d3:92:9d:7c:ea:de:14:
                    ca:6e:e4:e4:ea:e0:62:06:c8:c2:8a:3c:5e:3e:66:
                    d2:02:ff:ba:c8:dc:9e:c4:0d:72:aa:80:4a:68:77:
                    b0:9c:3f:e6:85:bc:6c:1a:01:4c:6e:ca:5a:d9:f6:
                    f8:7e:df:bf:35:36:23:eb:b0:2b:0f:8f:1d:73:a1:
                    7a:b1:f4:7f:6a:b1:d5:9e:e2:22:50:1d:e1:b2:ce:
                    ae:27:91:c0:b3:d5:05:c7:f7:04:17:43:74:13:83:
                    7a:ae:bd
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                27:F5:9C:3C:28:C9:A0:22:B9:21:D2:FD:8E:96:D0:BF:F7:BD:B2:E6
            X509v3 Key Usage:
                Certificate Sign, CRL Sign
```

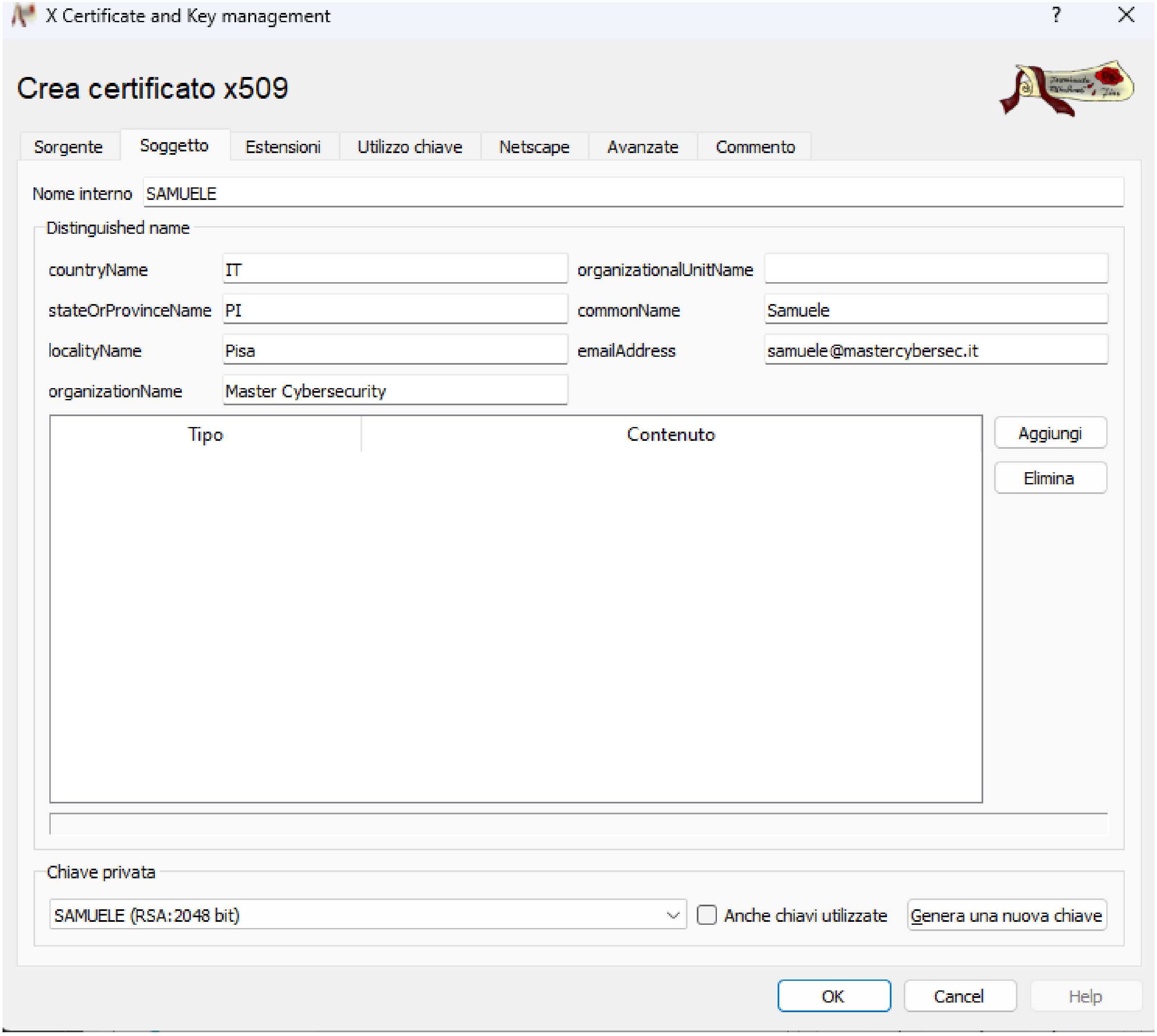
```
            Netscape Cert Type:
                SSL CA, S/MIME CA, Object Signing CA
            Netscape Comment:
                xca certificate
    Signature Algorithm: sha256WithRSAEncryption
         9a:58:01:8e:53:e4:f3:53:28:fb:2a:7c:72:2d:65:19:58:cb:
         9f:7d:2e:64:54:a3:c8:b2:e3:62:56:0e:9b:a6:b8:a8:c7:49:
         0d:e1:6c:4b:d4:b4:13:5d:27:39:c4:1c:28:78:35:c7:d4:36:
         dd:d4:72:36:d9:2f:f6:80:15:e6:cd:91:c3:d2:e7:b2:fc:c8:
         55:ad:63:2e:15:98:66:f7:1c:cd:0a:96:02:f4:13:54:ae:f7:
         0e:a0:d7:68:13:fe:a0:18:42:92:0e:e2:ed:7b:be:cf:9a:54:
         81:4c:53:9e:0c:27:e7:87:e2:65:3a:30:8d:30:3e:9b:2d:7b:
         72:85:3d:b1:8e:14:e9:af:eb:fc:45:fd:3b:0b:9d:89:65:25:
         c2:fc:b5:c5:13:fb:7e:51:e1:15:e1:57:88:a5:40:e8:09:82:
         47:92:55:ef:f0:58:16:44:22:a5:af:0a:9a:86:3d:4d:3e:ac:
         d7:a4:32:7d:15:db:41:32:0e:57:a1:06:b9:c0:89:6b:f1:72:
         65:93:5c:07:57:0f:06:a0:7b:ff:6c:71:f8:c7:ec:48:fe:d3:
         60:d7:e6:f9:28:02:d4:22:81:13:78:e3:ae:8a:d6:b8:68:e8:
         06:63:ee:d7:4b:f3:13:b3:b8:74:24:82:23:fd:89:79:15:29:
         d0:0a:2c:05:5a:19:ea:96:68:a3:59:22:a2:52:05:e7:e4:e6:
         e6:6e:9b:10:94:da:46:17:e2:8b:d3:10:82:0e:40:01:87:a4:
         f5:bc:de:ae:c7:23:49:f4:21:c8:7f:b6:76:50:9a:f2:75:d0:
         94:37:02:db:37:03:da:29:91:d6:97:ad:0a:4a:6b:32:1a:0c:
         d3:3f:0c:5e:e2:1e:17:9f:6a:b5:dc:67:f7:56:43:16:6e:8c:
         9f:19:51:14:f5:2e:58:17:2b:64:71:2e:f7:0c:f7:41:5d:c3:
         34:24:01:35:3f:22:dc:a2:4a:b9:40:e3:30:6a:74:29:00:2f:
         f3:32:9b:bf:c6:78:e5:22:b9:25:8e:b8:fc:6c:5e:06:06:4a:
         7f:77:19:56:28:70:23:d2:01:c6:f7:ad:b9:2e:76:4a:74:0c:
         9c:b9:30:b2:65:8f:57:93:94:b7:b0:96:18:df:5b:f3:9e:3d:
         8c:93:15:f8:1b:12:40:98:04:a4:c2:09:13:86:4d:7f:17:df:
         85:0f:c2:8e:f8:4f:28:88:42:79:39:1f:1c:f4:ea:d1:52:9b:
         a2:49:01:af:bf:1d:67:5a:91:93:6d:00:96:78:43:de:12:d2:
         f4:f0:11:f2:64:3b:d0:a6:36:c1:70:a3:fe:90:d7:6d:e0:c4:
         d5:f9:b1:72:6e:7c:cf:c3
```

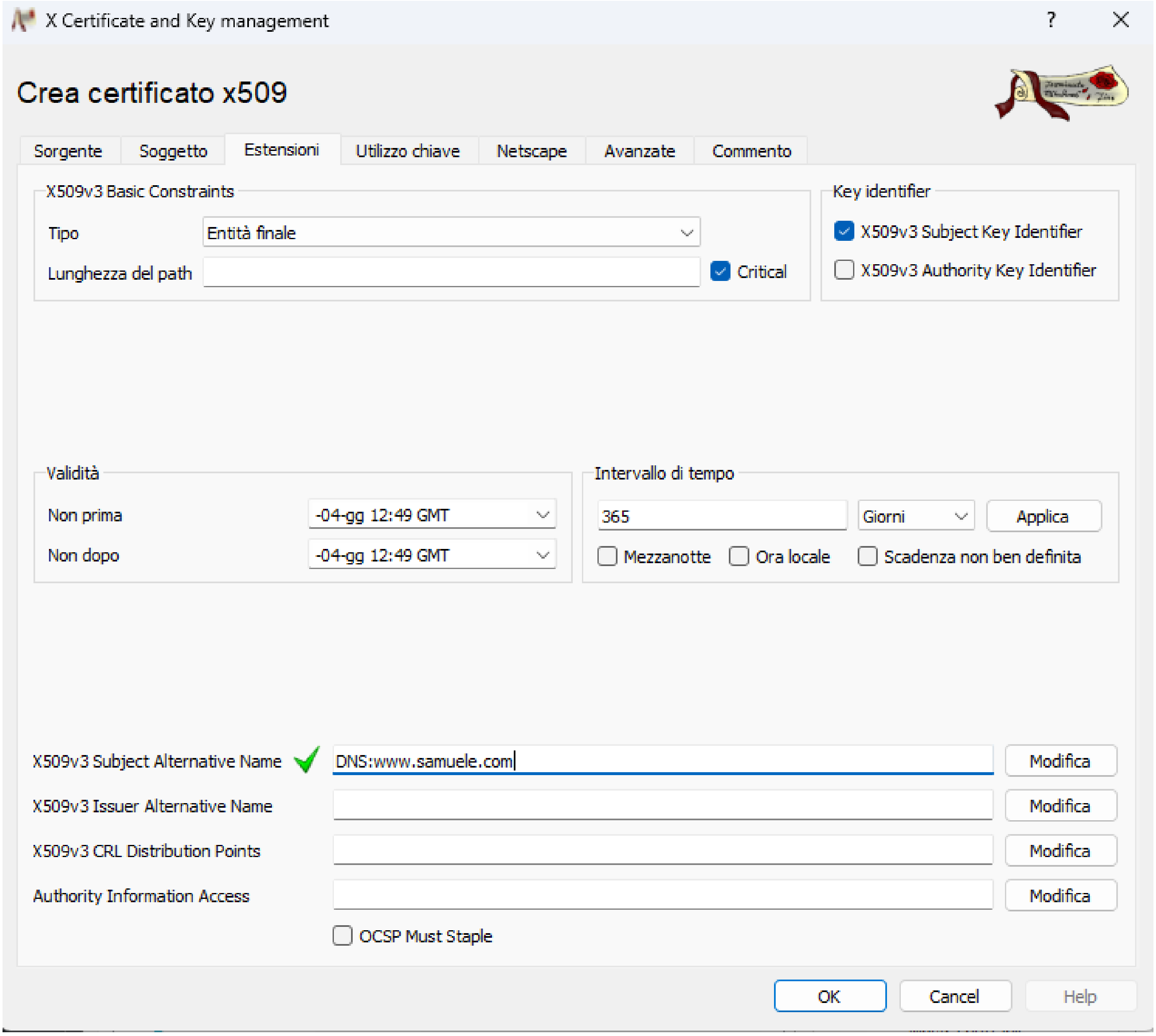
Generiamo una chiave privata protetta da password per il cliente ipotetico che ci ha richiesto un nuovo certificato:

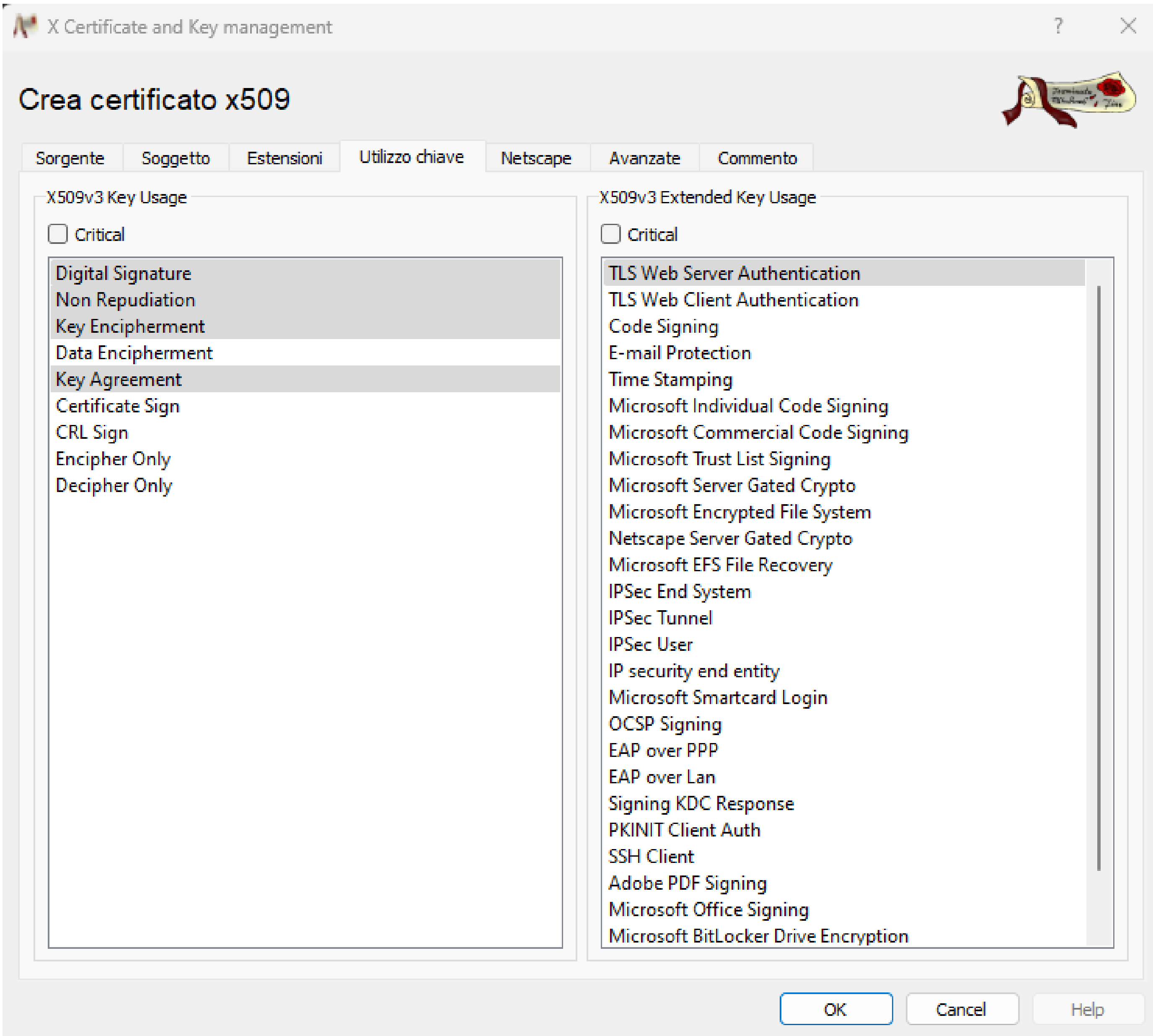

X Certificate and Key management
File
Importa
Token
Extra
Help
Chiavi private
Richieste di firma certificato (CSR)
Certificati
Modelli
Liste di revoca
Nome interno
Tipo
Dimensione
Utilizzo
Password
CA_PRIVATE_KEY RSA 4096 bit 1 Comune
SAMUELE RSA 2048 bit 0 Comune
Nuova chiave
Esporta
Importa
Importa PFX (PKCS#12)
Mostra dettagli
Elimina
XCA
Database: C:\Users\SamuelePadula\Downloads\cryptodb.xdb
Cerca

Per generare il certificato per il cliente:




X Certificate and Key management
Crea certificato x509
Sorgente
Soggetto
Estensioni
Utilizzo chiave
Netscape
Avanzate
Commento
Nome interno
SAMUELE
Distinguished name
countryName
IT
stateOrProvinceName
PI
localityName
Pisa
organizationName
Master Cybersecurity
organizationalUnitName
commonName
Samuele
emailAddress
samuele@mastercybersec.it
Tipo
Contenuto
Aggiungi
Elimina
Chiave privata
SAMUELE (RSA:2048 bit)
Anche chiavi utilizzate
Genera una nuova chiave
OK
Cancel
Help


X Certificate and Key management
Crea certificato x509
Sorgente
Soggetto
Estensioni
Utilizzo chiave
Netscape
Avanzate
Commento
X509v3 Basic Constraints
Tipo
Entità finale
Lunghezza del path
Critical
Key identifier
X509v3 Subject Key Identifier
X509v3 Authority Key Identifier
Validità
Non prima
-04-gg 12:49 GMT
Non dopo
-04-gg 12:49 GMT
Intervallo di tempo
365
Giorni
Applica
Mezzanotte
Ora locale
Scadenza non ben definita
X509v3 Subject Alternative Name
DNS:www.samuele.com
Modifica
X509v3 Issuer Alternative Name
Modifica
X509v3 CRL Distribution Points
Modifica
Authority Information Access
Modifica
OCSP Must Staple
OK
Cancel
Help



Proviamo ora invece a seguire il metodo di generazione della CSR tramite openssl:

```
spadula@DESKTOP-QN8KFBP:/mnt/c/Users/SamuelePadula/Downloads$ openssl req -
config openssl.cnf -new -newkey rsa:2048 -keyout SAMUELE2.key -out
SAMUELE2.csr
Generating a RSA private key
.............................................................................
.............................................................................
.+++++
............................................................................+++++
writing new private key to 'SAMUELE2.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:IT
State or Province Name (full name) [Some-State]:PI
Locality Name (eg, city) []:Pisa
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Master Cybersec
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:SAMUELE
Email Address []:samuele@mastercybersecurity.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:
```

La challenge password è una password che andrà in chiaro nella richiesta di certificazione e che la CA una volta acquisita può usare in un secondo momento per ri-autenticarmi, per comunicazioni future tra SAMUELE e la CA.

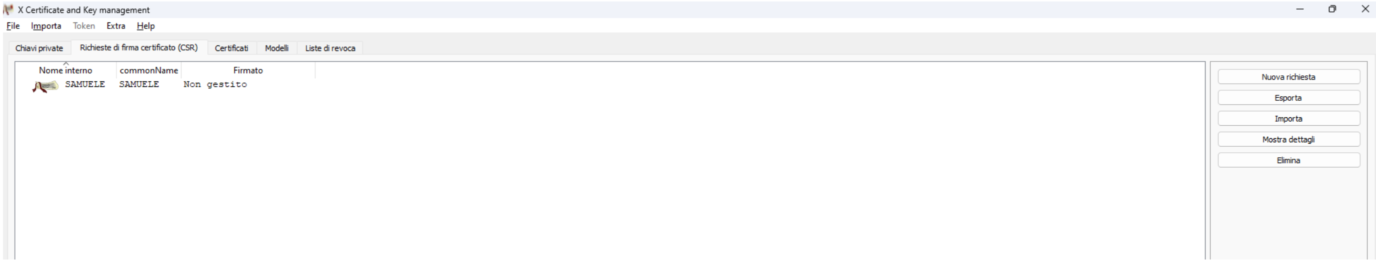
```
spadula@DESKTOP-QN8KFBP:/mnt/c/Users/SamuelePadula/Downloads$ openssl req -
config openssl.cnf  -in SAMUELE2.csr
```


```
-----BEGIN CERTIFICATE REQUEST-----
MIIC5DCCAcwCAQAwgYUxCzAJBgNVBAYTAklUMQswCQYDVQQIDAJQSTENMAsGA1UE
BwwEUGlzYTEYMBYGA1UECgwPTWFzdGVyIEN5YmVyc2VjMRAwDgYDVQQDDAdTQU1V
RUxFMS4wLAYJKoZIhvcNAQkBFh9zYW11ZWxlQG1hc3RlcmN5YmVyc2VjdXJpdHku
Y29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2HbfCcJAlo8t3FyJ
b2QPrbiHVXR/rYH5VdHOFZojuq2FYjbXaNg4rv7+OR6K9zb1CIPnGxP9Xrq+Jj9x
fLBM6GS2A7JcSVhjFNRbXG+CcoeF2oD3RkUkbX1sgO6HyLsI6jBCXHzBoNcAVIqE
f1j26C6vJ22WEycn+CkGVNsoaSr0sbPRjkJ7RYhbA5m22RqhWaihBJetY7YAVc7G
0JmobbIKLf5wanhI+WAxLlQCcxG72F2kfxmjj/wnpx/tSwr4haPlAoQ3ZUF9OvqV
xFOcRGwGcq9OHKhlucMmBWhdWcHRG+Rt+Gapjv25UAnxNAB0nJ7QBh9ZrZv5BdLC
jctbnQIDAQABoBkwFwYJKoZIhvcNAQkHMQoMCHBhc3N3b3JkMA0GCSqGSIb3DQEB
CwUAA4IBAQCIjeZ3kPqXjsQj+KHHvOq3HUo50UjUwRiyhVExjeRPhs3LZBO7dxKu
Xcw7qaVKzcEIHWl48VlBG7gWCdpModbstEfTV97c8kLwcrPOxbiedLzZJqY06JJv
/5PBvysbQNHOiofy1Gu9s0ryjXlw1CnQK1iDgZcINcXe8CMala+Hx4k+BgYzOT49
k/b1ajidmtqPshyRA2DlEN56GKr9RmSleTkhRCEvy+QzTlAjPp/iJD+Gc9IMt8hv
ucqvfZ8C1VdGeUp0n5Cal1QYHUsSQDcFZpVWTVIjFLwHXkmY1P0yVkOmWGnNqvJP
tOj7NGB7xCWjIxrXPiaOy5Ci7p5sZx0Z
-----END CERTIFICATE REQUEST-----
```

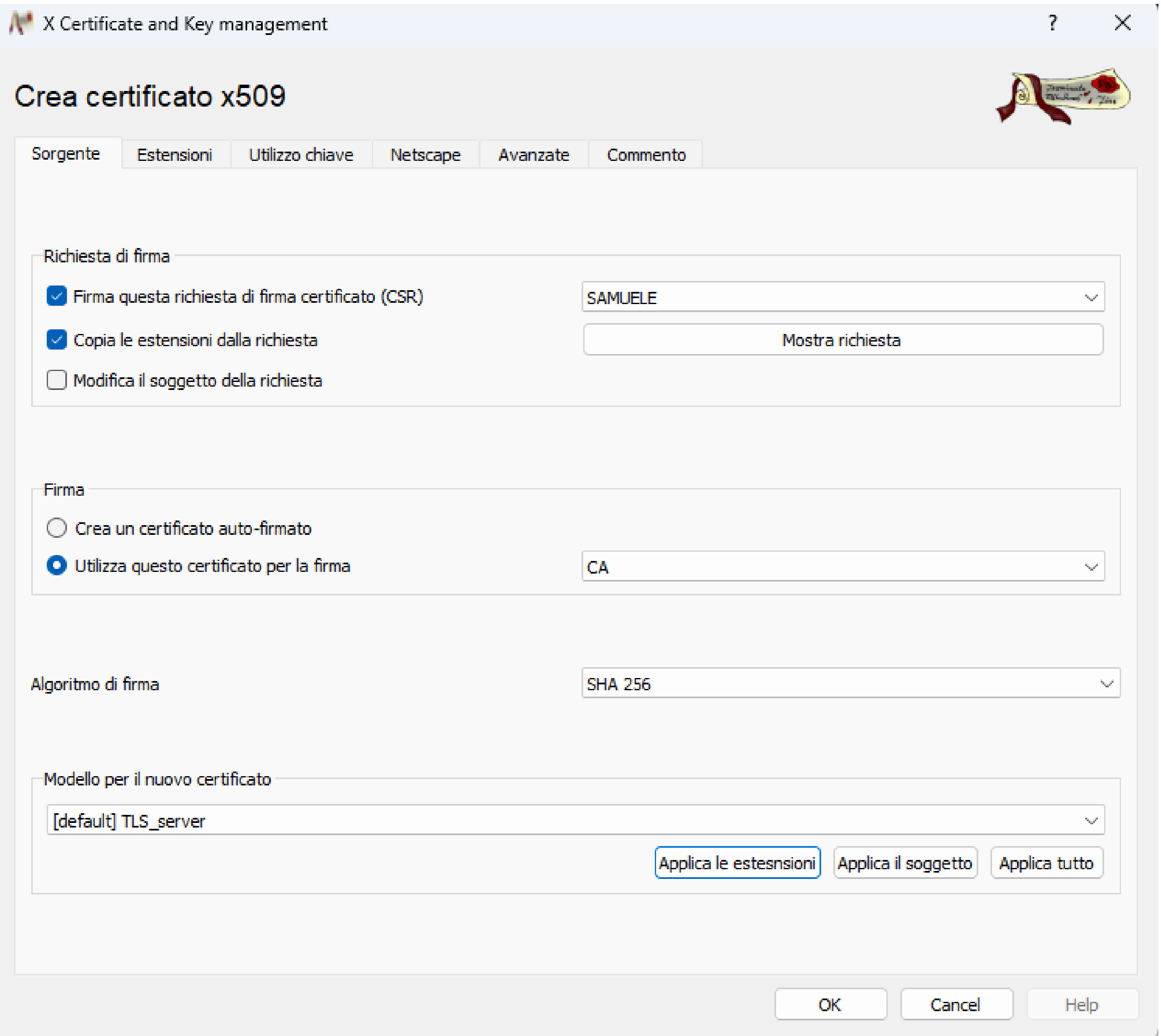
```
spadula@DESKTOP-QN8KFBP:/mnt/c/Users/SamuelePadula/Downloads$ openssl req -
config openssl.cnf  -in SAMUELE2.csr -noout -text
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = IT, ST = PI, L = Pisa, O = Master Cybersec, CN =
SAMUELE, emailAddress = samuele@mastercybersecurity.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d8:76:df:09:c2:40:96:8f:2d:dc:5c:89:6f:64:
                    0f:ad:b8:87:55:74:7f:ad:81:f9:55:d1:ce:15:9a:
                    23:ba:ad:85:62:36:d7:68:d8:38:ae:fe:fe:39:1e:
                    8a:f7:36:f5:08:83:e7:1b:13:fd:5e:ba:be:26:3f:
                    71:7c:b0:4c:e8:64:b6:03:b2:5c:49:58:63:14:d4:
                    5b:5c:6f:82:72:87:85:da:80:f7:46:45:24:6d:7d:
                    6c:80:ee:87:c8:bb:08:ea:30:42:5c:7c:c1:a0:d7:
                    00:54:8a:84:7f:58:f6:e8:2e:af:27:6d:96:13:27:
                    27:f8:29:06:54:db:28:69:2a:f4:b1:b3:d1:8e:42:
                    7b:45:88:5b:03:99:b6:d9:1a:a1:59:a8:a1:04:97:
                    ad:63:b6:00:55:ce:c6:d0:99:a8:6d:b2:0a:2d:fe:
                    70:6a:78:48:f9:60:31:2e:54:02:73:11:bb:d8:5d:
                    a4:7f:19:a3:8f:fc:27:a7:1f:ed:4b:0a:f8:85:a3:
                    e5:02:84:37:65:41:7d:3a:fa:95:c4:53:9c:44:6c:
                    06:72:af:4e:1c:a8:65:b9:c3:26:05:68:5d:59:c1:
                    d1:1b:e4:6d:f8:66:a9:8e:fd:b9:50:09:f1:34:00:
                    74:9c:9e:d0:06:1f:59:ad:9b:f9:05:d2:c2:8d:cb:
                    5b:9d
                Exponent: 65537 (0x10001)
        Attributes:
            challengePassword        :password
    Signature Algorithm: sha256WithRSAEncryption
         88:8d:e6:77:90:fa:97:8e:c4:23:f8:a1:c7:bc:ea:b7:1d:4a:
         39:d1:48:d4:c1:18:b2:85:51:31:8d:e4:4f:86:cd:cb:64:13:
         bb:77:12:ae:5d:cc:3b:a9:a5:4a:cd:c1:08:1d:69:78:f1:59:
         41:1b:b8:16:09:da:4c:a1:d6:ec:b4:47:d3:57:de:dc:f2:42:
         f0:72:b3:ce:c5:b8:9e:74:bc:d9:26:a6:34:e8:92:6f:ff:93:
         c1:bf:2b:1b:40:d1:ce:8a:87:f2:d4:6b:bd:b3:4a:f2:8d:79:
         70:d4:29:d0:2b:58:83:81:97:08:35:c5:de:f0:23:1a:95:af:
         87:c7:89:3e:06:06:33:39:3e:3d:93:f6:f5:6a:38:9d:9a:da:
         8f:b2:1c:91:03:60:e5:10:de:7a:18:aa:fd:46:64:a5:79:39:
         21:44:21:2f:cb:e4:33:4e:50:23:3e:9f:e2:24:3f:86:73:d2:
```

```
0c:b7:c8:6f:b9:ca:af:7d:9f:02:d5:57:46:79:4a:74:9f:90:
9a:97:54:18:1d:4b:12:40:37:05:66:95:56:4d:52:23:14:bc:
07:5e:49:98:d4:fd:32:56:43:a6:58:69:cd:aa:f2:4f:b4:e8:
fb:34:60:7b:c4:25:a3:23:1a:d7:3e:26:8e:cb:90:a2:ee:9e:
6c:67:1d:19
```

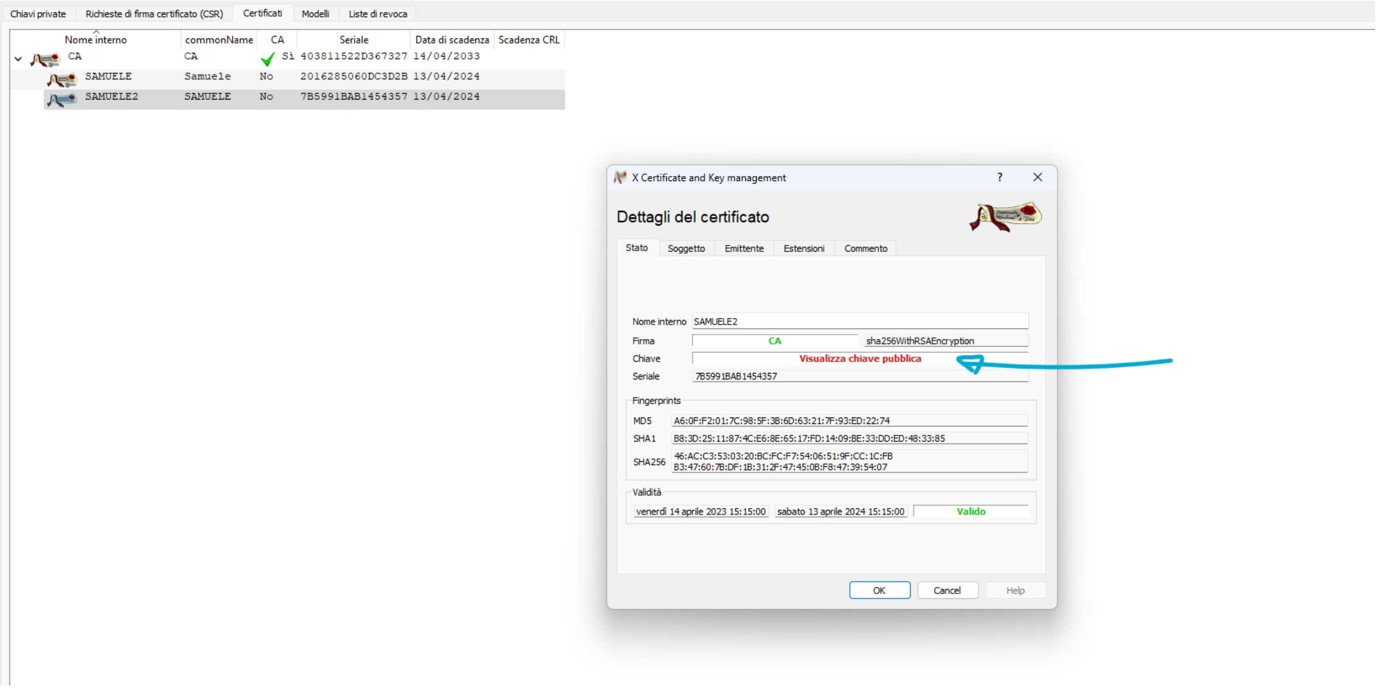
Importiamo la CSR:



Creiamo il certificato a partire dalla CSR firmandolo con la chiave privata della CA:



Del certificato appena creato non è disponibile la chiave privata:



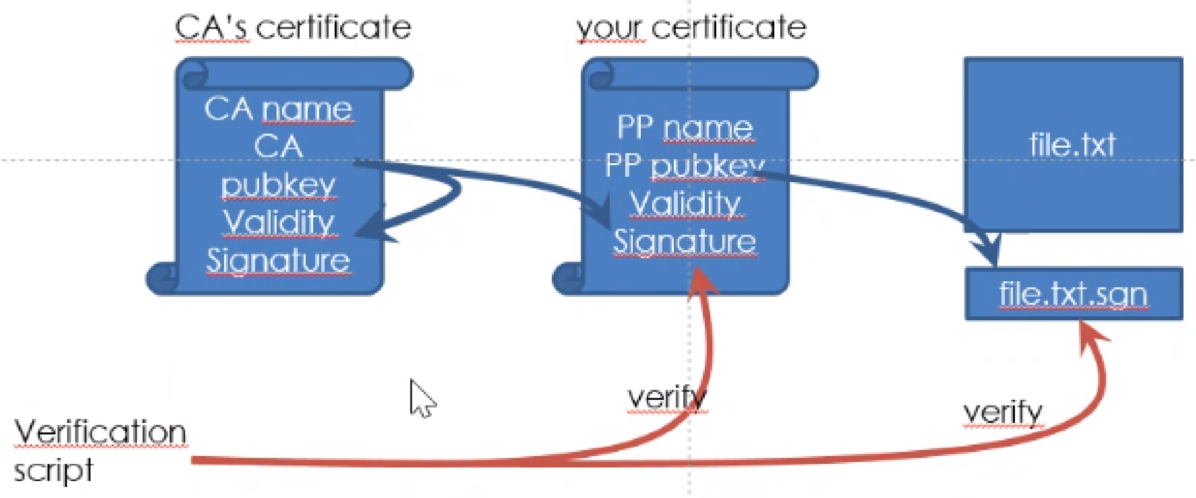
in quanto quella è stata generata dal cliente che ha richiesto il certificato tramite CSR.

Esercizio 4

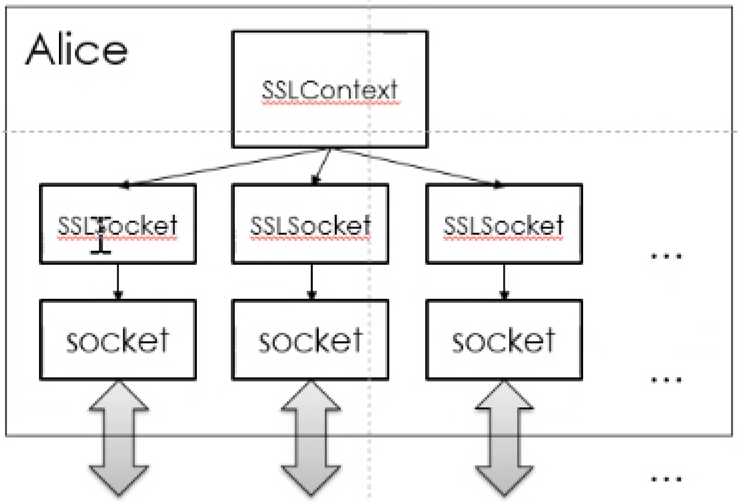
Final Exercise

UNIVERSITÀ DI PISA

- Set up a certification authority with XCA:
  - Create a private key and a certificate for yourself
  - Save the CA certificate, your certificate, and your private key in PEM-format files
- Signature script:
  - Sign a file with your private key
  - Save the signature in another file
- Verification script:
  - Load the CA certificate and your certificate
  - Verify the time validity of the CA certificate, and the signature and the time validity of your certificate
  - Check that the issuer of your certificate is actually the CA by comparing common names
  - Verify that your certificate is really yours by checking the common name
  - Verify the signature on the file



Transport Layer Security with ssl

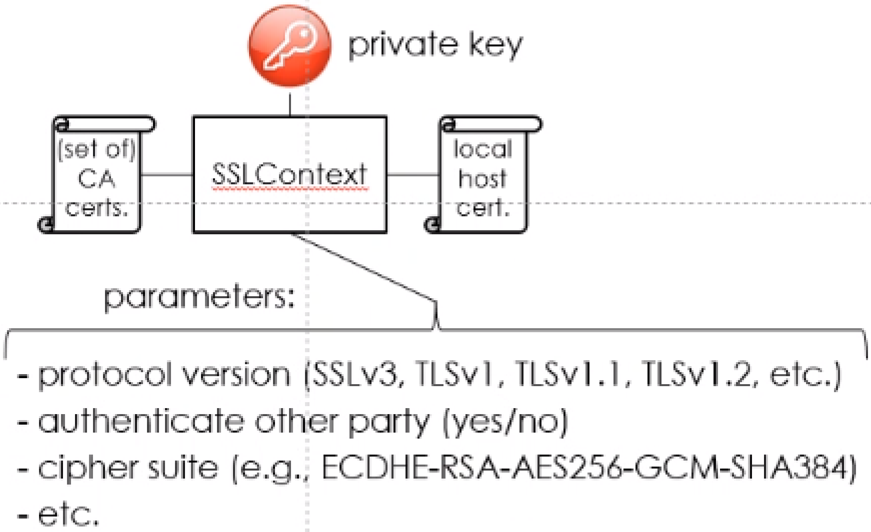


Su un SSL socket si potranno fare le medesime operazioni di un socket classico, la sendall_ssl() che viene fatta ad esempio, prima cifra i dati, ne calcola un tag e poi chiama una seconda sendall classica che invia i dati cifrati sul socket. Sul socket classico non lavoreremo più, perchè mascherato dal socket SSL.

Con SSL context viene creata una configurazione di base dal quale poi ricavare più socket sicuri, questo per evitare di dover configurare ogni volta le CA trusted oppure le chiavi private da usare nel socket ad esempio.

La configurazione minimale:

Transport Layer Security with `ssl`



Esercizio 5

Final Exercise #5

- Server-side script
  - A server which receives some data from a TLS connection and sends back 'OK…' + the received data
  - With XCA, create private key + certificate for the server
- Client-side script
  - A client that connects to the server and sends 'CIAO'
- First version: only server authenticates
- Second version: client and server mutually authenticates
  - With XCA, create private key + certificate for the client