

Report Esercizio 12/12/2024

Crittografia Leonardo Catalano

“Il primo esercizio era di codificare questa stringa :

QWJhIHZ6b2VidHI2bmdyIHB1ciB6ciBhciBucHBiZX Ri

Guardandola vedendola lettere e numeri ho pensato che fosse una codifica generale a base64 e usando un tool online di decode ho codificato la stringa:

Decode from Base64 format

Simply enter your data then push the decode button.

QWJhIHZ6b2VidHI2bmdyIHB1ciB6ciBhciBucHBiZX Ri

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8

Source character set.

☐

Decode each line separately (useful for when you have multiple entries).

☒ Live mode OFF

Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE >

Decodes your data into the area below.

Aba vzoebtyvngr pur zr ar nppbetb

In output è uscito però questa stringa che cmq è ancora codificata quindi probabilmente si è usata una doppia codifica :Aba vzoebtyvngr pur zr ar nppbetb.

Dopo ho usato la **codifica di Cesare** facendo varie prove spostando le lettere nell'alfabeto e arrivando **a + 13** è uscita una frase di senso compiuto

”Aba vzoebtyvngr pur zr ar nppbetb +13 (a= n, b=o ...)”

“Non Imbrogliate che me ne accorgo”

“Il 2° esercizio di oggi è installare OpenSSL e usare Python e effettuare una criptazione e una firma” .

Obiettivi dell'esercizio:

- Generare chiavi RSA.
- Estrarre la chiave pubblica da chiave privata.
- Criptare e decriptare messaggi.
- Firmare e verificare messaggi.

Strumenti utilizzati:

- OpenSSL per la generazione delle chiavi.
- Libreria cryptography in Python.

Prima di iniziare bisogna installare:

OpenSSL

```
sudo apt update
```

```
sudo apt install openssl
```

Libreria per python della crittografia

```
sudo apt install python3-pip
```

```
pip3 install cryptography
```

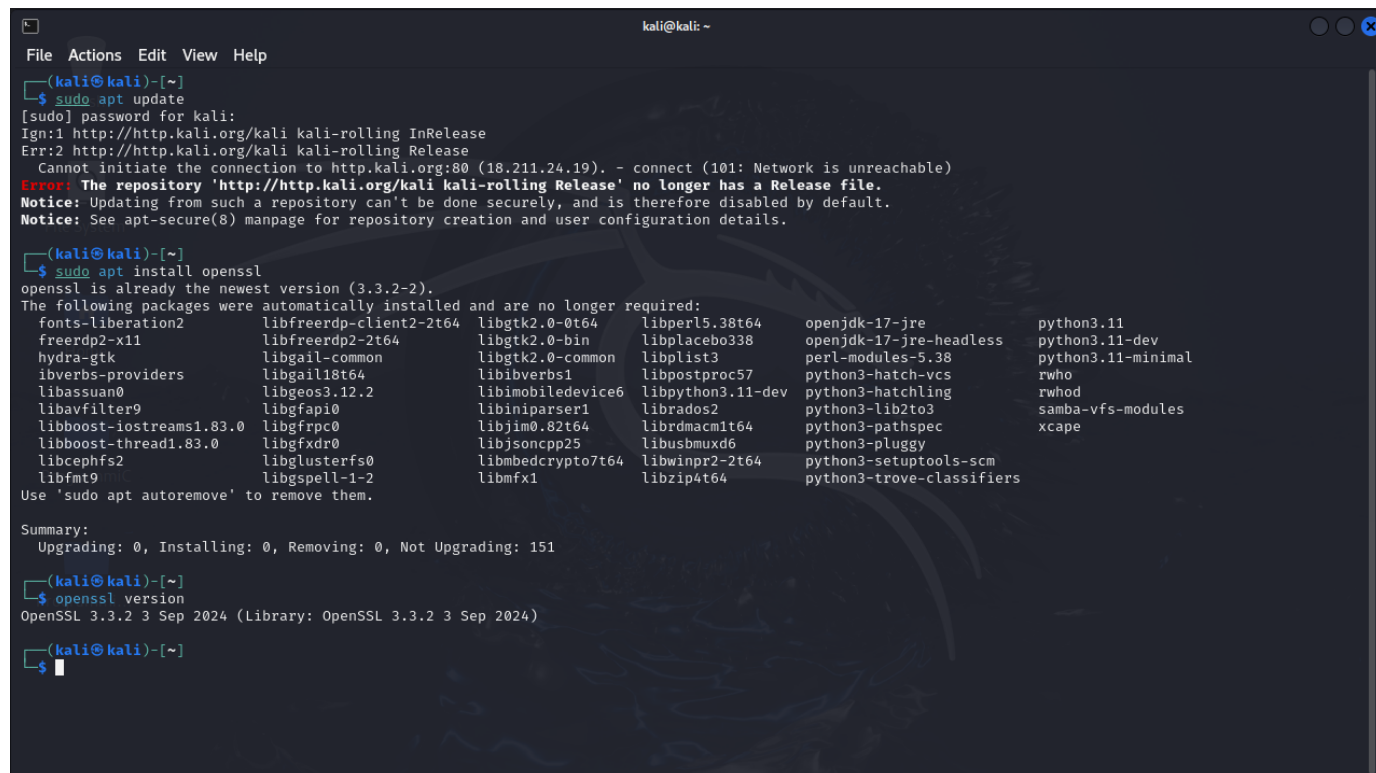
Comando per generare la chiave privata RSA

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
```

Comando per estrarre la chiave pubblica:

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

OpenSSL:



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)~  
$ sudo apt update  
[sudo] password for kali:  
Ign:1 http://http.kali.org/kali kali-rolling InRelease  
Err:2 http://http.kali.org/kali kali-rolling Release  
       Cannot initiate the connection to http.kali.org:80 (18.211.24.19). - connect (101: Network is unreachable)  
Error: The repository 'http://http.kali.org/kali kali-rolling Release' no longer has a Release file.  
Notice: Updating from such a repository can't be done securely, and is therefore disabled by default.  
Notice: See apt-get(8) manpage for repository creation and user configuration details.  
(kali@kali)~  
$ sudo apt install openssl  
openssl is already the newest version (3.3.2-2).  
The following packages were automatically installed and are no longer required:  
  fonts-liberation2  libfreerdp-client2-2t64  libgtk2.0-0t64  libperl5.38t64  openjdk-17-jre  python3.11  
  freerdp2-x11       libfreerdp2-2t64       libgtk2.0-bin  libplacebo338   openjdk-17-jre-headless  python3.11-dev  
  hydra-gtk          libgail-common         libgtk2.0-common  libplist3       perl-modules-5.38      python3.11-minimal  
  ibverbs-providers libgail18t64          libibverbs1      libpostproc57   python3-hatch-vcs      rwho  
  libassuan0         libgeos3.12.2         libimobiledevice6  libpython3.11-dev  python3-hatchling      rwhod  
  libavfilter9       libgfapi0             libiniparser1     librados2        python3-lib2to3         samba-vfs-modules  
  libboost-iostreams1.83.0  libgfrc0             libjim0.82t64     librdmacm1t64   python3-paths-spec     xcace  
  libboost-thread1.83.0    libgfxdr0            libjsoncpp25      librdmucmd6     python3-pluggy         python3-setuptools-scm  
  libcephfs2           libglusterfs0         libmbedcrypto7t64  libwinpr2-2t64  python3-trove-classifiers  
  libfmt9              libgspell-1-2         libmfx1           libzip4t64  
Use 'sudo apt autoremove' to remove them.  
  
Summary:  
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 151  
(kali@kali)~  
$ openssl version  
OpenSSL 3.3.2 3 Sep 2024 (Library: OpenSSL 3.3.2 3 Sep 2024)  
(kali@kali)~  
$
```

In questo caso ce l'avevo già e ho controllato la versione con il comando

Python:

```

kali@kali:~$ sudo apt install python3-pip
python3-pip is already the newest version (24.3.1+dfsg-1).
python3-pip set to manually installed.
The following packages were automatically installed and are no longer required:
 fonts-liberation2 libfreerdp-client-2t64 libgtk2.0-0t64 libperl5.38t64 openjdk-17-jre python3.11
 freerdp2-x11 libfreerdp2-2t64 libgtk2.0-bin libplacebo338 openjdk-17-jre-headless python3.11-dev
 hydra-gtk libgail-common libgtk2.0-common libplist3 perl-modules-5.38 python3.11-minimal
 ibverbs-providers libgail18t64 libibverbs1 libpostproc57 python3-hatch-vcs rwho
 libassuan0 libgeo3.12.2 libimobiledevice6 libpython3.11-dev python3-hatchling python3-lib2to3 rwhoed
 libavfilter9 libgfpapi0 libiniiparser1 librados2 python3-lib2to3 samba-vfs-modules
 libboost-iostreams1.83.0 libgfrpc0 libjbig0.82t64 librdmacm1t64 python3-paths-spec xcapi
 libboost-thread1.83.0 libgfdx0 libbsoncpp25 libusbmuxd6 python3-pluggy
 libcephfs2 libicusterfs0 libmbdedcrypto7t64 libwinpr-2t64 python3-setup-tools-scm
 libfmt9 libgspell1-2 libmbfx1 libzip4t64 python3-trove-classifiers

Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 151

```

In questo caso ce l'avevo già aggiornato anche con la libreria 3-pip

Chiave privata RSA:

[illegible]

Estrazione chiave pubblica:

[illegible]

```
(kali㉿kali)-[~/Desktop/ProgrammiPython]
$ openssl rsa -pubout -in private_key.pem -out public_key.pem
writing RSA key
Home
(kali㉿kali)-[~/Desktop/ProgrammiPython]
$
```

Questi 2 comandi ci creano 2 file.py dove all'interno c'è in uno la chiave privata e nel secondo la chiave pubblica.

Successivamente si crea un file `encdec.py` e al suo interno importiamo:

Padding che è un processo che aggiunge dati extra a un messaggio per far sì che abbia una lunghezza adeguata per essere crittografato.

Serialization per leggere le chiavi private e pubbliche.

base64 per visualizzare il criptato in base64.

con open leggiamo le chiavi e le trasferiamo nelle variabili:

private_key public_key

```
GNU nano 8.2 encdec.py
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization

import base64

#Carica la chiave privata
#dal file private_key.pem

with open('private_key.pem', 'rb') as key_file:
    private_key = serialization.load_pem_private_key(
        key_file.read(),
        password=None)

#Carica la chiave pubblica
#dal file public_key.pem

with open('public_key.pem', 'rb') as key_file:
    public_key = serialization.load_pem_public_key(key_file.read())

message = "Ciao, Epicode spacca!"

# Criptazione con la chiave pubblica
encrypted = public_key.encrypt(message.encode(), padding.PKCS1v15())

# Decriptazione con la chiave privata
decrypted = private_key.decrypt(encrypted, padding.PKCS1v15())

print ("Messaggio originale:", message)

print ("Messaggio criptato:", base64.b64encode(encrypted).decode("utf-8"))

print ("Messaggio decriptato:", decrypted.decode("utf-8"))
```

Per verificare che il file e le chiavi siano state create si usa il comando ls sulla directory

```
(kali@kali)-[~/Desktop/ProgrammiPython]
$ touch encdec.py

(kali@kali)-[~/Desktop/ProgrammiPython]
$ nano encdec.py

(kali@kali)-[~/Desktop/ProgrammiPython]
$ ls
analisi_parole.py  encdec.py  media_mobile.py  programma_data_ora.py  Programma_Geometria.py  programma_socket.py.save
band_musicale.py  esercizio1.py  pari_dispari.py  programmaesempio1.py  programmaprovaifstrano.py  public_key.pem
dizionario.py     liste.py    private_key.pem  programmaesempio.py  programma_socket.py      somma3numeri.py
```

Poi si fa partire il programma con il comando python encdec.py :

```
(kali@kali)-[~/Desktop/ProgrammiPython]
$ python encdec.py
Messaggio originale: Ciao, Epicode spacca!
Messaggio criptato: IIBzfAaNffRkUo2UciTGPua17ee1ag4kkINTZsRvpC7u8F9dbmtLhA2oqL+DPQbM1wc9j5UH2wxdt05mgmonYa0FmJ2arFVWjHTsNMErb8COCQMX+vMLMm7XEF8lNGFc/foVs9V
Tj6Kc36xCm6rIEaAaxpA6NIwLFMmombfzgLCTCMmrsz817uImP7uTE5fEpLb9CQ5B0sisR1DYmhJ8CRKzoRp78e4Pj0LkWAuV2t4SFpYNYL0+V4QTTDM8AUKQ2+B+LUIURO/vog2ax+aqM7GImLckA0oG
1eXhnhXRp00KRzTXixmPMnrXze40IfP/YX6WjXewPKXXqiSF6eWA=
Messaggio decriptato: Ciao, Epicode spacca!
```

Successivamente si va a creare il file della firma digitale (firma.py)

Il file è identico al precedente ma con delle modifiche :

hashes che serve a creare l'hash del messaggio.

con open leggiamo le chiavi e le trasferiamo nelle variabili: private_key public_key

```
(kali@kali)-[~/Desktop/ProgrammiPython]
$ touch firma.py

(kali@kali)-[~/Desktop/ProgrammiPython]
$ nano firma.py
```

```
GNU nano 8.2 firma.py *
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization

import base64

#Carica la chiave privata
$ cat file private_key.pem

with open('private_key.pem', 'rb') as key_file:
    private_key = serialization.load_pem_private_key (
        key_file.read(),
        password=None)

#Carica la chiave pubblica
$ cat file public_key.pem

with open('public_key.pem', 'rb') as key_file:
    public_key = serialization.load_pem_public_key(key_file.read())

message = "Ciao, Epicode spacca!"

#Firma con la chiave privata

signed = private_key.sign(message.encode(), padding.PKCS1v15(), hashes.SHA256())

# Verifica della firma con la chiave pubblica

try:
    encrypted_b64 = base64.b64encode(signed).decode("utf-8")
    public_key.verify(signed, message.encode(), padding.PKCS1v15(), hashes.SHA256())

    print("Base64 della firma:", encrypted_b64)

    print("Messaggio originale da confrontare:", message)

    print("La firma è valida.")
except Exception as e:
    print ("La firma non è valida.", str(e))
```

Poi si fa partire il programma con il comando python firma.py

```
(kali@kali)-[~/Desktop/ProgrammiPython]
$ python firma.py
Base64 della firma: N6ph03tXujHua+8yLsjEiWYF1QhJ2t8BZ58GZ22k5bkQUBUg07aa9TyfK3xySfci5fbuV0YaPXwnNR6MACotmf39Uv5z7+oQlaifpVmcz00i8j244I7+3tH0NJZswuOXuo/knvs
cRMN08nw061XcNiPm/9enTXKI fHZFVGfsxb3EjqH+8oMu02wwtGU46fmODTfTvokIFoTYTUUngh3DneSjFhe60Jnc/jwq0S04uIhQnQx5T5TdKJFO/5/mSgMUR0Agz7fjLu/yXBsGp4KP6bx2rWUoFiqRf5n
x5Kana4t+yHQM9hyF++rGgbeAjiWvflcTvHt8Cazi5zMxxhprQ=
Messaggio originale da confrontare: Ciao, Epicode spacca!
La firma è valida.
```

Se il programma è privo di errori riuscirà ad eseguire tutto il programma e non avremo in output errori.