Davide Stefani

Consenga

EPICODE SPACCA!

DATA: 28/08/2024

Esercizio: Criptazione e Firmatura con OpenSSL e Python

Oggi impareremo a fare delle operazioni magiche con i computer chiamate criptazione e firmatura. Queste tecniche ci aiutano a tenere i messaggi segreti e a confermare che provengono da chi dice di averli inviati. Non preoccuparti, faremo tutto passo per passo in modo semplice!

Obiettivi dell'esercizio:

1. **Generare chiavi RSA: Crea una coppia di chiavi, una per nascondere (cripto) e l'altra per rivelare (decripto) messaggi.**
2. **Estrarre la chiave pubblica dalla chiave privata: Crea una chiave che puoi condividere con gli altri.**
3. **Criptare e decriptare messaggi: Nascondi un messaggio (cripto) e poi riportalo alla luce (decripto).**
4. **Firmare e verificare messaggi: Metti una firma su un messaggio per dire "questo l'ho scritto io", e controlla che la firma sia valida.**

Strumenti utilizzati:

- **OpenSSL per generare le chiavi.**
- **Libreria cryptography in Python per lavorare con i messaggi.**

Passo 1: Generare le Chiavi RSA con OpenSSL

Cos'è una Chiave RSA?

Immagina di avere una scatola con un lucchetto (la chiave pubblica) e una chiave segreta (la chiave privata). Puoi dare la scatola a chiunque, ma solo tu, con la tua chiave segreta, puoi aprirla.

Come generare le Chiavi?

1. **Apri il Terminale: Questo è il luogo dove puoi dare istruzioni al tuo computer. Se usi Windows, cerca "Prompt dei comandi" o "PowerShell". Su Mac o Linux, cerca**

"Terminale".

2. **Genera la chiave privata: Copia e incolla questo comando nel terminale e premi "Invio":**
3. **bash**
4. **Copia codice**
5. **openssl genpkey -algorithm RSA -out chiave_privata.pem -pkeyopt rsa_keygen_bits:2048**
6. **Questo comando chiede al computer di creare una chiave privata. Verrà salvata in un file chiamato chiave_privata.pem.**
7. **Estrai la chiave pubblica: Ora, copia e incolla questo comando per estrarre la chiave pubblica dalla chiave privata:**

```
openssl rsa -pubout -in chiave_privata.pem -out chiave_pubblica.pem
```

1. **Questo crea un file chiamato chiave_pubblica.pem che contiene la tua chiave pubblica.**

Passo 2: Criptare e Decriptare Messaggi con Python

Preparazione

Assicurati di avere installato Python e la libreria cryptography. Se non l'hai ancora fatto, apri il terminale e digita:

```
pip install cryptography
```

Come Criptare un Messaggio

Crea un file Python (ad esempio cripta_messaggio.py) e incolla il seguente codice:

```python
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend

# Leggiamo la chiave pubblica
with open("chiave_pubblica.pem", "rb") as key_file:
    public_key = serialization.load_pem_public_key(key_file.read(), backend=default_

# Il messaggio che vogliamo criptare
messaggio = b"Questo è un messaggio segreto!"

# Criptiamo il messaggio
messaggio_criptato = public_key.encrypt(
    messaggio,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

print("Messaggio criptato:", messaggio_criptato)
```

Questo codice legge la chiave pubblica e la usa per criptare un messaggio segreto.

Come Decriptare un Messaggio

Crea un altro file Python (ad esempio decripta_messaggio.py) e incolla il seguente codice:

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.backends import default_backend

# Leggiamo la chiave privata
with open("chiave_privata.pem", "rb") as key_file:
    private_key = serialization.load_pem_private_key(key_file.read(), password=None, b

# Decriptiamo il messaggio
messaggio_decriptato = private_key.decrypt(
    messaggio_criptato,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

print("Messaggio decriptato:", messaggio_decriptato.decode())
```

Questo codice legge la chiave privata e decripta il messaggio, riportandolo alla sua forma originale.

Passo 3: Firmare e Verificare Messaggi con Python

Come Firmare un Messaggio

Aggiungi questo codice nel tuo file Python per firmare un messaggio:

```python
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

# Firmiamo il messaggio
firma = private_key.sign(
    messaggio,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)

print("Firma digitale:", firma)
```

Questo codice usa la tua chiave privata per firmare il messaggio.

Come Verificare la Firma

Verifica la firma aggiungendo questo codice:

```python
try:
    public_key.verify(
        firma,
        messaggio,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("Firma verificata: Il messaggio è autentico!")
except:
    print("Firma non valida: Il messaggio è stato modificato!")
```

Questo codice controlla se la firma è valida, usando la chiave pubblica.