

Cryptography

Symmetric Encryption:

Symmetric Encryption used the same key for encryption and decryption. It useful for large data but if the key is intercepted then the security is broken.

Algorithms: AES, DES, 3DES, Blowfish

Asymmetric Encryption:

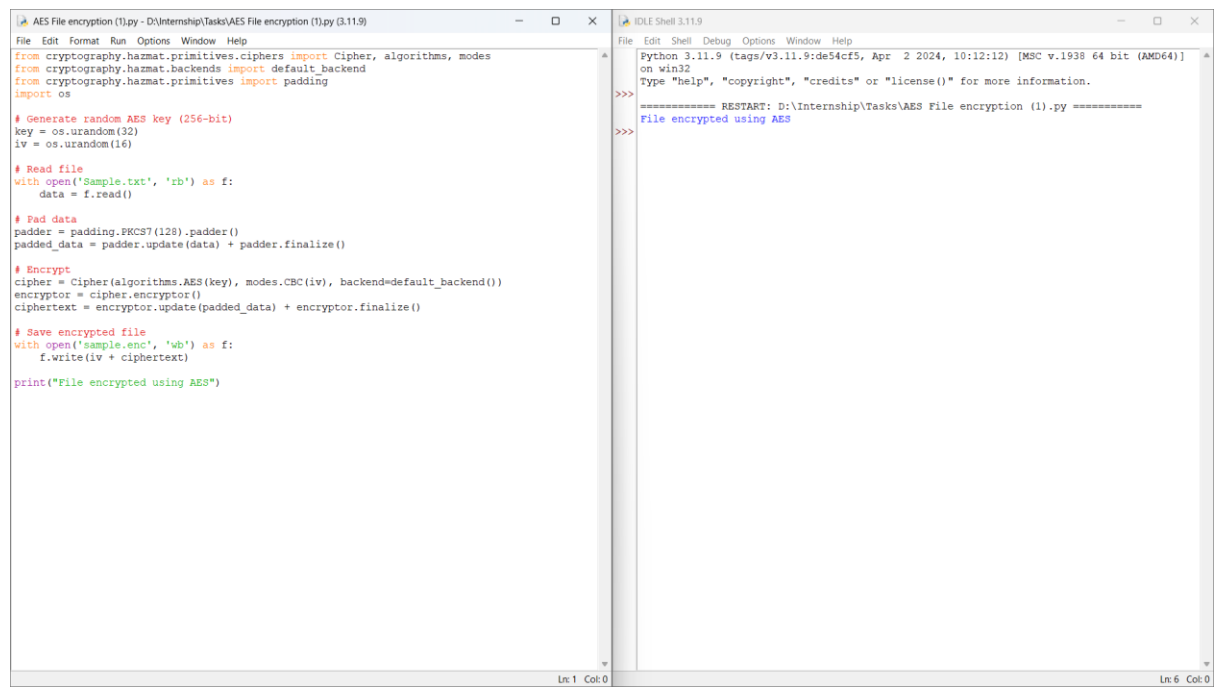
Asymmetric Encryption uses a pair of keys: one public key for encryption and one private key for decryption. Provides authentication and digital signatures. It slower than Symmetric Encryption.

Algorithms: RSA, ECC, DSA

AES Encryption:

AES stands for Advanced Encryption Standard. It is a symmetric encryption algorithm used to protect data by converts plaintext into ciphertext using secret key.

In this lab, I used python programming for AES Encryption. I encrypted a text file using python.



The screenshot displays a Python IDE with two windows. The left window, titled 'AES File encryption (1).py - D:\Internship\Tasks\AES File encryption (1).py (3.11.9)', contains the following Python code:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import padding
import os

# Generate random AES key (256-bit)
key = os.urandom(32)
iv = os.urandom(16)

# Read file
with open('Sample.txt', 'rb') as f:
    data = f.read()

# Pad data
padder = padding.PKCS7(128).padder()
padded_data = padder.update(data) + padder.finalize()

# Encrypt
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
encryptor = cipher.encryptor()
ciphertext = encryptor.update(padded_data) + encryptor.finalize()

# Save encrypted file
with open('sample.enc', 'wb') as f:
    f.write(iv + ciphertext)

print("File encrypted using AES")
```

The right window, titled 'IDLE Shell 3.11.9', shows the execution output:

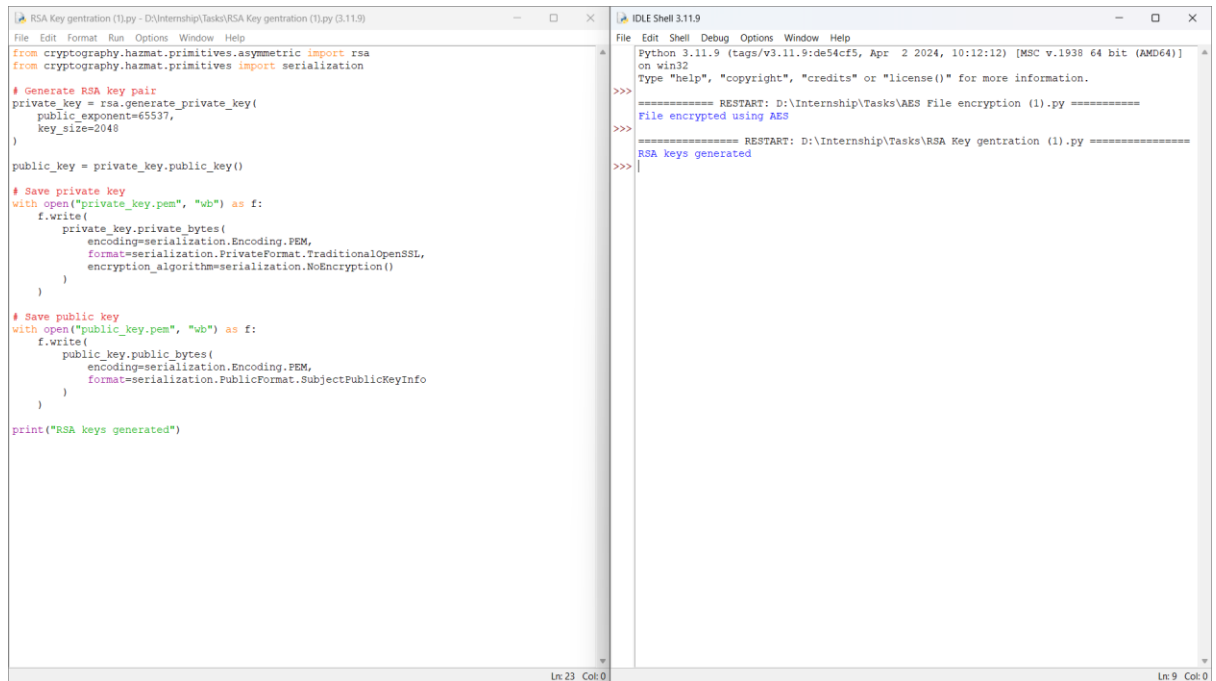
```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Internship\Tasks\AES File encryption (1).py =====
>>> File encrypted using AES
```

The status bar at the bottom indicates 'Ln: 1 Col: 0' for the editor and 'Ln: 6 Col: 0' for the shell.

RSA Key Generation:

It is an asymmetric encryption algorithm. It uses two keys for encryption and decryption.

In this lab also I used python for generate RSA key. This program generates public key and private key and stored those keys in files.



The screenshot shows a Python script for RSA key generation and its execution in IDLE Shell 3.11.9. The script generates a private key and a public key, saving them to 'private_key.pem' and 'public_key.pem' respectively. The output shows the successful generation of RSA keys.

```
File Edit Format Run Options Window Help
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# Generate RSA key pair
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)

public_key = private_key.public_key()

# Save private key
with open("private_key.pem", "wb") as f:
    f.write(
        private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        )
    )

# Save public key
with open("public_key.pem", "wb") as f:
    f.write(
        public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
    )

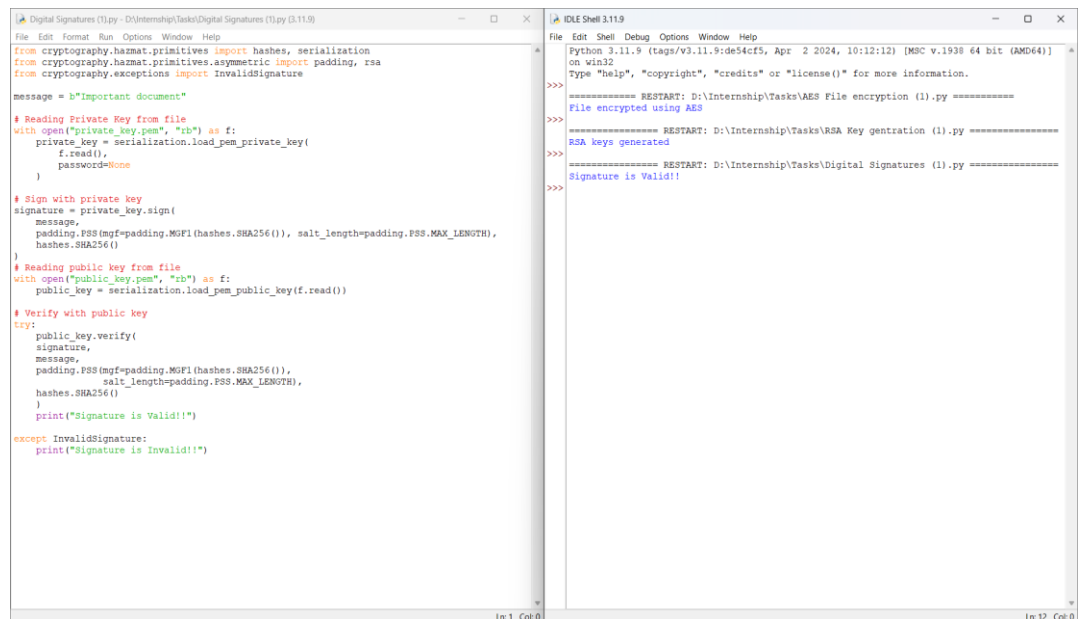
print("RSA keys generated")

Ln: 23 Col: 0
```

```
File Edit Shell Debug Options Window Help
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Internship\Tasks\AES File encryption (1).py =====
File encrypted using AES
>>>
===== RESTART: D:\Internship\Tasks\RSA Key generation (1).py =====
RSA keys generated
>>>
```

Digital Signature Verification:

The RSA keys are the signatures of the file. When the signature is mismatched, those files can't decrypt. I used python for this lab also.



The screenshot shows a Python script for digital signature verification and its execution in IDLE Shell 3.11.9. The script reads a message, signs it with a private key, and then verifies the signature using the public key. The output shows that the signature is valid.

```
File Edit Format Run Options Window Help
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import padding, rsa
from cryptography.exceptions import InvalidSignature

message = b"Important document"

# Reading Private Key from file
with open("private_key.pem", "rb") as f:
    private_key = serialization.load_pem_private_key(
        f.read(),
        password=None
    )

# Sign with private key
signature = private_key.sign(
    message,
    padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt_length=padding.PSS.MAX_LENGTH),
    hashes.SHA256()
)

# Reading public key from file
with open("public_key.pem", "rb") as f:
    public_key = serialization.load_pem_public_key(f.read())

# Verify with public key
try:
    public_key.verify(
        signature,
        message,
        padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt_length=padding.PSS.MAX_LENGTH),
        hashes.SHA256()
    )
    print("Signature is Valid!!")
except InvalidSignature:
    print("Signature is Invalid!!")

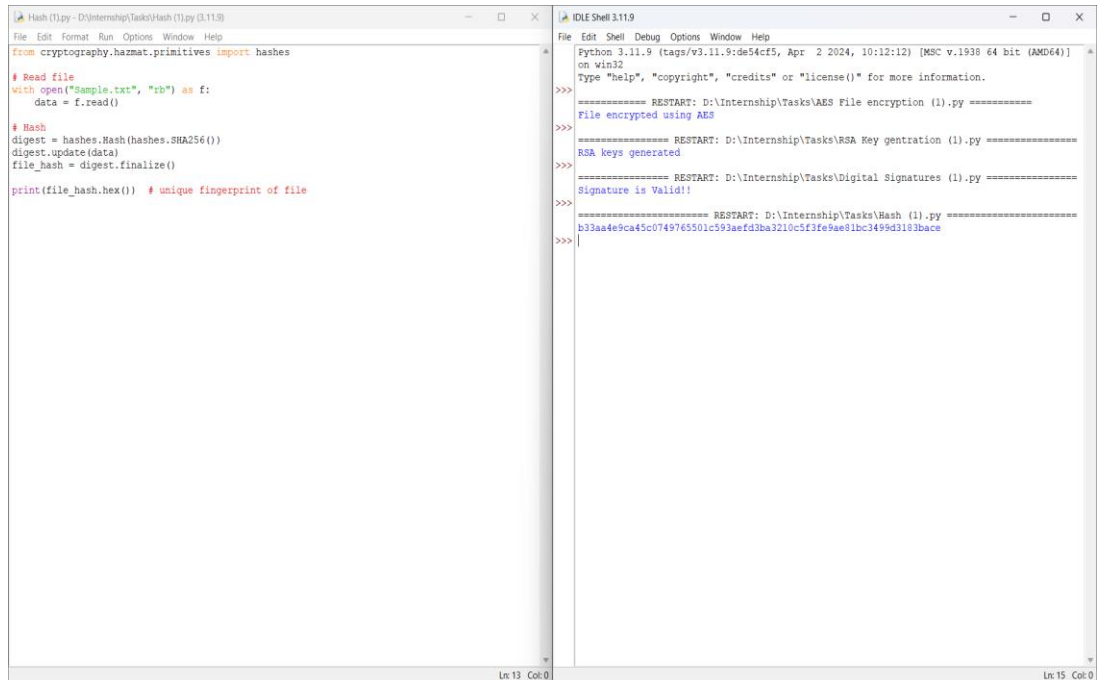
Ln: 1 Col: 0
```

```
File Edit Shell Debug Options Window Help
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Internship\Tasks\AES File encryption (1).py =====
File encrypted using AES
>>>
===== RESTART: D:\Internship\Tasks\RSA Key generation (1).py =====
RSA keys generated
>>>
===== RESTART: D:\Internship\Tasks\Digital Signatures (1).py =====
Signature is Valid!!
>>>
```

Hash Generation:

Hash is the fingerprint of the file. We need to generate hash for the file first when the file was created. The receiver also needs to generate hash for the same file before he decrypts it. If it changed then the file must intercept by someone.

Like before labs in this lab also I used python for generate hash.



The image shows two side-by-side windows from a Python IDE. The left window, titled 'Hash (1).py', contains the following Python code:

```
from cryptography.hazmat.primitives import hashes

# Read file
with open("Sample.txt", "rb") as f:
    data = f.read()

# Hash
digest = hashes.Hash(hashes.SHA256())
digest.update(data)
file_hash = digest.finalize()

print(file_hash.hex()) # unique fingerprint of file
```

The right window, titled 'IDLE Shell 3.11.9', shows the output of the script. It includes a help message, followed by several restart prompts and the final output of the hash generation script:

```
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Internship\Tasks\AES File encryption (1).py =====
File encrypted using AES
>>>
===== RESTART: D:\Internship\Tasks\RSA Key generation (1).py =====
RSA keys generated
>>>
===== RESTART: D:\Internship\Tasks\Digital Signatures (1).py =====
Signature is Valid!!
>>>
===== RESTART: D:\Internship\Tasks\Hash (1).py =====
b33aa4e9ca45c0749765501c593aefd3ba3210c5f3fe9ae81bc3499d3183bace
>>>
```

The status bars at the bottom of the windows indicate 'Ln 13 Col 0' for the left window and 'Ln 15 Col 0' for the right window.