

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:

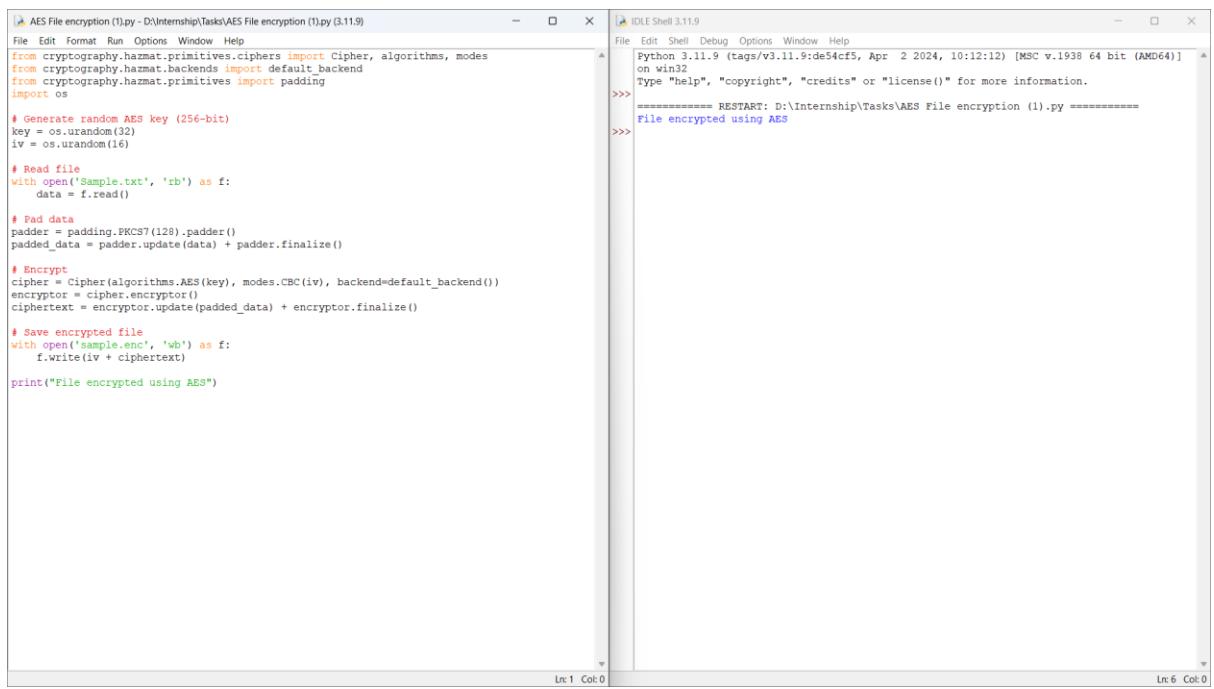
Asymmetric Encryption uses a pair of keys: one public key for encryption and one private key for decryption. Provides authentication and digital signatures. It slower than Symmetric Encryption.

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 shows a Python development environment with two windows. On the left is a code editor window titled "AES File encryption (1).py - D:\Internship\Tasks\AES File encryption (1).py (3.11.9)". The code implements AES encryption using the PyCryptodome library. It generates a random 256-bit key and IV, reads a file named "Sample.txt", pads the data, encrypts it using CBC mode, and writes the result to "sample.enc". The right window is the "IDLE Shell 3.11.9" window, which shows the command-line interface where the script was run. The output includes the Python version, build date, and a message indicating the file was encrypted using AES.

```
# AES File encryption (1).py - D:\Internship\Tasks\AES File encryption (1).py (3.11.9)
File Edit Format Run Options Window Help
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)

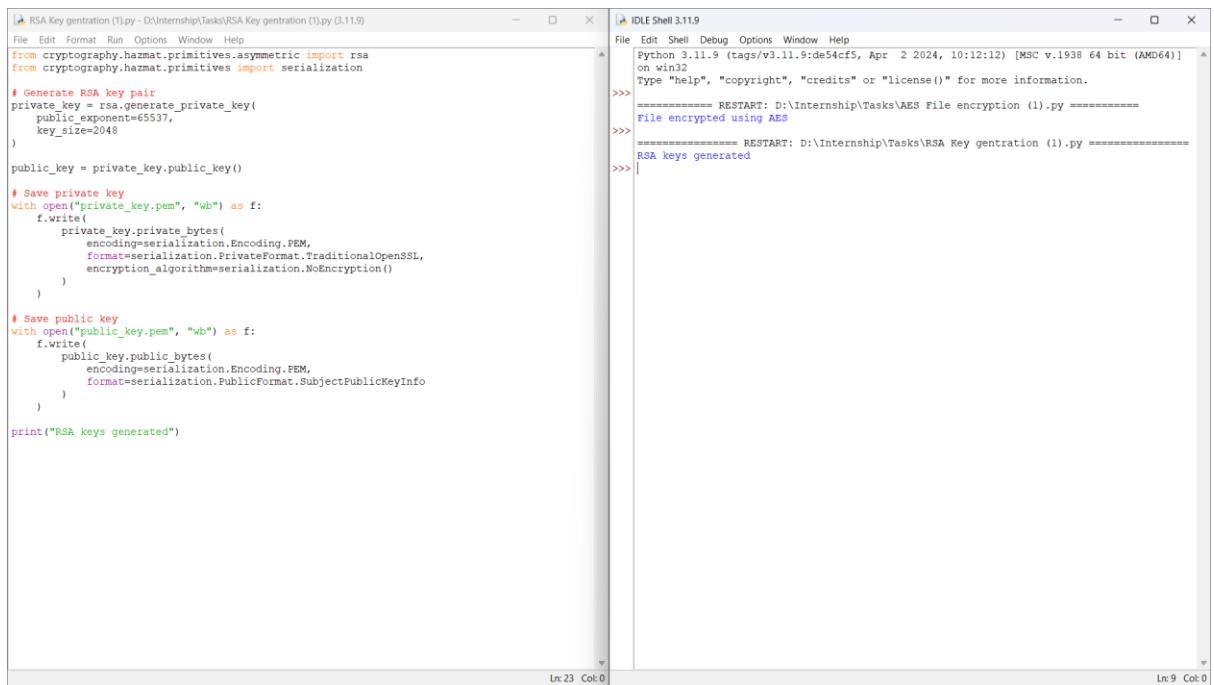
print("File encrypted using AES")
```

```
IDLE Shell 3.11.9
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
>>>
```

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.



```
RSA Key generation (1).py - D:\Internship\Tasks\RSA Key generation (1).py (3.11.9)
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()
        )
    )

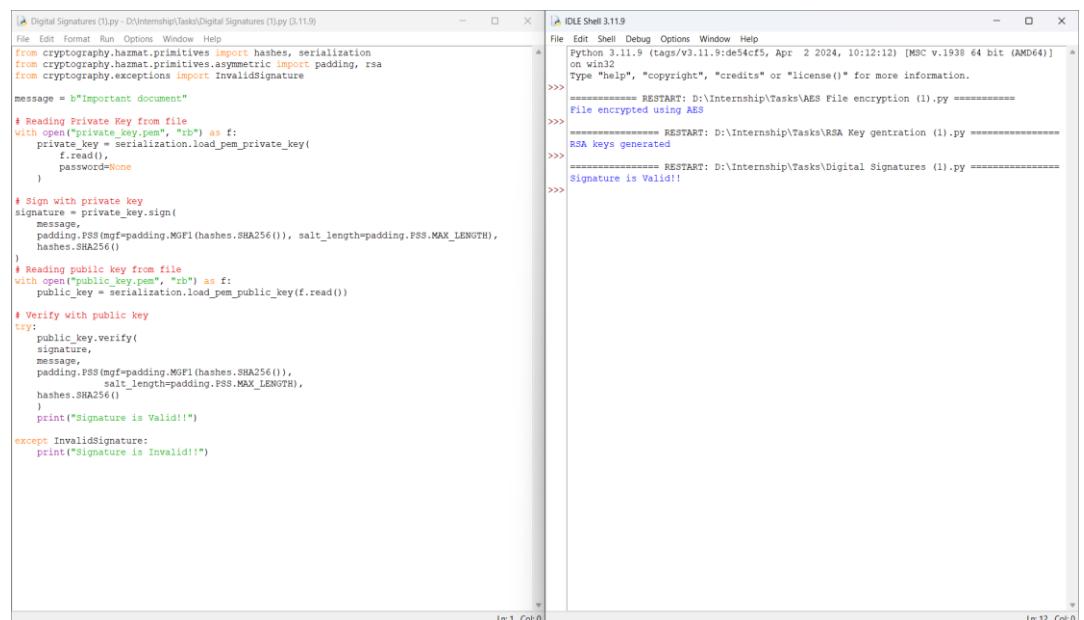
# Save public key
with open("public_key.pem", "wb") as f:
    f.write(
        public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
    )

print("RSA keys generated")
```

```
IDLE Shell 3.11.9
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.



```
Digital Signatures (1).py - D:\Internship\Tasks\Digital Signatures (1).py (3.11.9)
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!!")
```

```
IDLE Shell 3.11.9
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 screenshot shows a dual-pane interface. The left pane is a code editor titled "Hash (1).py - D:\Internship\Tasks\Hash (1).py (3.11.9)" containing Python code to generate a SHA256 hash of a file named "sample.txt". The right pane is a terminal window titled "IDLE Shell 3.11.9" showing the execution of three different scripts: "AES File encryption (1).py", "RSA Key generation (1).py", and "Digital Signatures (1).py". Each script is run with a restart command, and the terminal displays the generated file hash or key.

```
Hash (1).py - D:\Internship\Tasks\Hash (1).py (3.11.9)
File Edit Format Run Options Window Help
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

IDLE Shell 3.11.9
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!?
>>> ===== RESTART: D:\Internship\Tasks\Hash (1).py =====
b33aa4e9ca45c0749765501c593afed3ba3210c5f3fe9ae81bc3495d3183bace
>>>
```