```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import serialization, hashes
import os
import textwrap
```

Symmetric encryption (AES-CTR mode)

The code below shows the working of symmetric encryption using the **AES-256** algorithm in **CTR (Counter)** mode.

In Symmetric encryption, we use the **same key** is used for both encryption and decryption.

Here, we are using a 256-bit key and a 128-bit IV (nonce) that are randomly generated. We have a plaintext that is encrypted into ciphertext and then decrypted back to the original message.

We are using the CTR mode because it is a stream cipher mode and can handle data of any length without the need for any padding. This makes it efficient for encrypting large amounts of data or text very quickly.

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

def generate_key_iv():
    # Generate a random key and initialization vector (IV) or nounce (for CTR mode)
    key = os.urandom(32)  # 256-bit key for AES-256
    iv = os.urandom(16)   # 128-bit IV (nounce in CTR mode)
    return key, iv

def encrypt(key, iv, plaintext):
    # We are using AES-256 in CBC mode for encryption
    cipher = Cipher(algorithms.AES(key), modes.CTR(iv))
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()
    return ciphertext

def decrypt(key, iv, ciphertext):
    # Decrypting the ciphertext using AES-256 in CBC mode
    cipher = Cipher(algorithms.AES(key), modes.CTR(iv))
    decryptor = cipher.decryptor()
    decrypted_message = decryptor.update(ciphertext) + decryptor.finalize()
    return decrypted_message

def main():
    # Generate key and IV
    key, iv = generate_key_iv()

    # Lets define the following message for encryption
    plaintext = b"Hello, this is Blockchain homework 1!"

    # Encrypt the message
    ciphertext = encrypt(key, iv, plaintext)

    # Decrypt the message
    decrypted_text = decrypt(key, iv, ciphertext)

    # Print results
    print(f"Key: {key.hex()}\n")
    print(f"IV (nounce): {iv.hex()}\n")
    print(f"Original Text: {plaintext.decode()}\n")
    print(f"Encrypted Text: {ciphertext.hex()}\n")
    print(f"Decrypted Text: {decrypted_text.decode()}\n")

if __name__ == "__main__":
    main()
```

```
Key: 76e05afb595c64719db6abb73f129a2db46797bf47fe8acbe50faca8cab88e5c

IV (nounce): bf1f977d93a37a8513cbf956214c5732

Original Text: Hello, this is Blockchain homework 1!

Encrypted Text: 00e669defcacc49b4ad6ccbbb19ce545cdb109fd53a4de78825d42f4e6699690a3296f5be9

Decrypted Text: Hello, this is Blockchain homework 1!
```

Asymmetric encryption (RSA)

This code shows the working of asymmetric encryption using the **RSA** algorithm.

Unlike Symmetric encryption, Asymmetric encryption uses **key pairs**: a **public key** for encryption and a **private key** for decryption.

Here we are generating the RSA key pair with a 2048-bit key size, and this encryption uses **OAEP padding** with **SHA-256** for security.

The plaintext is encrypted with the public key and decrypted with the private key.

The RSA algorithm is computationally heavy, therefore it is mainly used for encrypting small amounts of data, such as symmetric keys, or digital signatures.

```
def generate_key_pair():
    # Generate a private and public key pair using rsa algorithm
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    public_key = private_key.public_key()
    return public_key, private_key

def serialize_public_key(public_key):
    # Serialize the public key to PEM format
    pem_public = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    return pem_public

def encrypt_message(public_key, plaintext):
    # We are encrypting the plaintext using the public key we generated
    ciphertext = public_key.encrypt(
        plaintext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return ciphertext

def decrypt_message(private_key, ciphertext):
    # Decrypt the ciphertext using the private key
    decrypted_text = private_key.decrypt(
        ciphertext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_text

def main():
    # Generate public & private key pair
    public_key, private_key = generate_key_pair()

    # Serialize the public key to view in PEM format
    pem_public = serialize_public_key(public_key)

    # Define the plaintext message
    plaintext = b"Hello, this is Blockchain homework 1!"

    # Encrypt the message
    ciphertext = encrypt_message(public_key, plaintext)

    # Decrypt the message
    decrypted_text = decrypt_message(private_key, ciphertext)

    # Print results
    print(f"Public Key (PEM):\n{pem_public.decode()}\n")
    print(f"Original Text: {plaintext.decode()}\n")

    # Print encrypted text
    encrypted_hex = ciphertext.hex()
    wrapped_text = textwrap.fill(encrypted_hex, width=64)  # 64 characters per line
```

```
    print("Encrypted Text:")
    print(wrapped_text + "\n")

    print(f"Decrypted Text: {decrypted_text.decode()}")

if __name__ == "__main__":
    main()
```

```
Public Key (PEM):
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAx++YH1anStJs9In9BSW4
oeTIeDKVhV1aClo3CZlHLFkCxWJG0dkUy0JPr+kL9k/y7Fdi6+zGbjnp8Hm10yI3
06PGuiNEzdJOIC64asDnlr4lYqYK3kxjsm2Wo28PdYAKEEFS1IOoM4nEYH9lb2tK
/PugPT7Edrw6LEALFVosQQxSbUKIUSufXyKKHD5BA9hQ8dSP2+dHdhk9gkezH4tW
CTxafz+t0uDTCVibRmlcayj/o6eVMc9+b/9zZ0shU9YDrW4X1jN0FCuWMDEU+E2c
8ufE1L8NIK1s3EOzqLahqhi3bG7C1+4FyaPtFPdDcZ0/+OJ4Xw3BeLhb8sY0lTkU
8wIDAQAB
-----END PUBLIC KEY-----


Original Text: Hello, this is Blockchain homework 1!

Encrypted Text:
6b58deb9463443ceecde242ad89d302a379b618087c2fbb100f26bf12517c46d
eb870c494128e58afeb6af7cd06e567329340f8e3f2bd66c018404ca23c0cd98
268c18347451fe852c0d3eba367f13bbd7882d7b85b0e5a23e6bbe5e3b7255e2
743ee30fa2e5488b0179c9f58bbf35db54502ed2e6ae367065f72c4947076bc2
4436a1640a480a1c4677fac32237a54c7a55e2b9d8ac8fc6a753e6a827e42b88
d838e33e522ebde9f3b14b2b8055a1ea37abf86cf085ebf6c182472090edbbfa
1fcd5d4f965e64274f8879e67e6d811435b5e9b561a4aec3064c8e2d89be8122
8103b7cc2a9d326bc949c0386230a796992d8a5ff168a186d226195979655840

Decrypted Text: Hello, this is Blockchain homework 1!
```

```
Start coding or generate with AI.
```