```python
# (OPCIONAL) Usando como base el código del apartado 1 (RSA), crear un fichero rsa_object.py
# que contenga una clase llamada RSA_OBJECT, la cual tenga los métodos indicados en el
# apéndice C, y que ejecute correctamente el código de prueba mostrado a continuación:

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Signature import pss
from Crypto.Hash import SHA256

class RSA_OBJECT:
    def __init__(self):
        #Inicializa un objeto RSA, sin ninguna clave
        # Nota: Para comprobar si un objeto (no) ha sido inicializado, hay
        # que hacer "if self.public_key is None:"
        if self.load_PublicKey is None:
            self.public_key = None
            self.private_key = None
        else:
            print("El objeto ya fue inicializado")

    def create_KeyPair(self):
        self.private_key=RSA.generate(2048)
        self.public_key = self.private_key.public_key()
    # Crea un par de claves publico/privada, y las almacena dentro de la instancia

    def save_PrivateKey(self, file, password):
        private_key_c = self.private_key.export_key(passphrase = password,pkcs=8, protection="scryptAndAES128-CBC")
        file_out = open(file, "wb")
        file_out.write(private_key_c)
        file_out.close()

    # Guarda la clave privada self.private_key en un fichero file, usando una contraseña
    # password

    def load_PrivateKey(self, file, password):
    # Carga la clave privada self.private_key de un fichero file, usando una contraseña
    # password
        private_key_c = open(file, "rb").read()
        self.private_key = RSA.import_key(private_key_c, passphrase=password)


    def save_PublicKey(self, file):
    # Guarda la clave publica self.public_key en un fichero file
        public_key = self.public_key.export_key()
        file_out = open(file, "wb")
        file_out.write(public_key)
        file_out.close()



    def load_PublicKey(self, file):
    # Carga la clave publica self.public_key de un fichero file
        public_key = open(file, 'rb').read()
        self.public_key = RSA.import_key(public_key)



    def cifrar(self, datos):
    # Cifra el parámetro datos (de tipo binario) con la clave self.public_key, y devuelve
```

```
    # el resultado. En caso de error, se devuelve None
        engineRSACifrado = PKCS1_OAEP.new(self.public_key)
        cifrado = engineRSACifrado.encrypt(datos)
        return cifrado


    def descifrar(self, cifrado):
    # Descrifra el parámetro cifrado (de tipo binario) con la clave self.private_key, y
    # Devuelve el resultado (de tipo binario). En caso de error, se devuelve None
        engineRSADescifrado = PKCS1_OAEP.new(self.private_key)
        datos = engineRSADescifrado.decrypt(cifrado)
        return datos


    def firmar(self, datos):
    # Firma el parámetro datos (de tipo binario) con la clave self.private_key, y devuelve
    # el resultado. En caso de error, se devuelve None.
    # La firma se realiza sobre el hash del texto (h)
        h = SHA256.new(datos)
        print(h.hexdigest())
        signature = pss.new(self.private_key).sign(h)
        return signature


    def comprobar(self, text, signature):
    # Comprueba el parámetro text (de tipo binario) con respecto a una firma signature
    # (de tipo binario), usando para ello la clave self.public_key.
    # Devuelve True si la comprobacion es correcta, o False en caso contrario o
    # en caso de error.
    # Comprobamos que la firma coincide con el hash (h)
        h = SHA256.new(text)
        print(h.hexdigest())
        verifier = pss.new(self.public_key)
        try:
            verifier.verify(h, signature)
            return True
        except (ValueError, TypeError):
            return False


# Crear clave RSA
# y guardar en ficheros la clave privada (protegida) y publica
password = "password"
private_file = "rsa_key.pem"
public_file = "rsa_key.pub"

RSA_key_creator = RSA_OBJECT()
RSA_key_creator.create_KeyPair()
RSA_key_creator.save_PrivateKey(private_file, password)
RSA_key_creator.save_PublicKey(public_file)

# Crea dos clases, una con la clave privada y otra con la clave publica
RSA_private = RSA_OBJECT()
RSA_public = RSA_OBJECT()
RSA_private.load_PrivateKey(private_file, password)
RSA_public.load_PublicKey(public_file)

# Cifrar y Descifrar con PKCS1 OAEP
cadena = "Lo desconocido es lo contrario de lo conocido. Pasalo."
cifrado = RSA_public.cifrar(cadena.encode("utf-8"))
print(cifrado)
```

```
descifrado = RSA_private.descifrar(cifrado).decode("utf-8")
print(descifrado)

# Firmar y comprobar con PKCS PSS
firma = RSA_private.firmar(cadena.encode("utf-8"))
if RSA_public.comprobar(cadena.encode("utf-8"), firma):
 print("La firma es valida")
else:
 print("La firma es invalida")

```