PRÁCTICA 2: Criptografía (Parte 2)

Seguridad en la Información

Lenguajes y Ciencias de la Computación.
E.T.S.I. Informática, Universidad de Málaga

PRÁCTICA 2: Criptografía (Parte 2)

Seguridad en la Información

RELACIÓN DE EJERCICIOS:

1. El código Python descrito en el apéndice A (y en las transparencias relacionadas con esta práctica) muestra el funcionamiento del algoritmo RSA, junto con el funcionamiento de las funciones Hash.

   Utilizando dicho código como base, se pide realizar los siguientes ficheros con las siguientes operaciones:

   `ca.py`

   a. Crear una clave pública y una clave privada RSA de 2048 bits para Alice. Guardar cada clave en un fichero.
   b. Crear una clave pública y una clave privada RSA de 2048 bits para Bob. Guardar cada clave en un fichero.

   `alice.py`

   c. Cargar la clave privada de Alice y la clave pública de Bob.
   d. Cifrar el texto "Hola amigos de la seguridad" utilizando la clave de Bob.
   e. Firmar el texto "Hola amigos de la seguridad" utilizando la clave de Alice.
   f. Guardar en unos ficheros, el texto cifrado y la firma digital[1].

   `bob.py`

   g. Cargar la clave privada de Bob y la clave pública de Alice.
   h. Cargar el texto cifrado y la firma digital.
   i. Descifrar el texto cifrado y mostrarlo por pantalla.
   j. Comprobar la validez de la firma digital.

2. La criptografía de curvas elípticas (o ECC) es una variante de la criptografía asimétrica basada en las matemáticas de las curvas elípticas. Al igual que RSA, esta clase de criptografía permite tanto realizar operaciones de cifrado[2] como de firma.

   Se pide implementar en el fichero `ecc.py` las funciones indicadas en el apéndice B utilizando criptografía de curvas elípticas. Para ello, se deberá consultar la documentación de la librería pycryptodome:

   https://pycryptodome.readthedocs.io/en/latest/src/public_key/ecc.html
   https://pycryptodome.readthedocs.io/en/latest/src/signature/dsa.html

3. (**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:

```
# 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()
```

[1] Ver https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files junto al código fuente del apéndice A para ver como guardar arrays de bytes en ficheros.
[2] Nótese que la operación de cifrado con ECC **aún no está implementada** en pycryptodome.

```
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")
```

APÉNDICE A: Código de ejemplo del uso de RSA, y del (des)cifrado y firma/comprobación

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

def crear_RSAKey():
    key = RSA.generate(2048)

    return key

def guardar_RSAKey_Privada(fichero, key, password):
    key_cifrada = key.export_key(passphrase=password, pkcs=8, protection="scryptAndAES128-CBC")
    file_out = open(fichero, "wb")
    file_out.write(key_cifrada)
    file_out.close()

def cargar_RSAKey_Privada(fichero, password):
    key_cifrada = open(fichero, "rb").read()
    key = RSA.import_key(key_cifrada, passphrase=password)

    return key

def guardar_RSAKey_Publica(fichero, key):
    key_pub = key.publickey().export_key()
    file_out = open(fichero, "wb")
    file_out.write(key_pub)
    file_out.close()

def cargar_RSAKey_Publica(fichero):
    keyFile = open(fichero, "rb").read()
    key_pub = RSA.import_key(keyFile)

    return key_pub

def cifrarRSA_OAEP(cadena, key):
    datos = cadena.encode("utf-8")
    engineRSACifrado = PKCS1_OAEP.new(key)
    cifrado = engineRSACifrado.encrypt(datos)

    return cifrado

def descifrarRSA_OAEP(cifrado, key):
    engineRSADescifrado = PKCS1_OAEP.new(key)
    datos = engineRSADescifrado.decrypt(cifrado)
    cadena = datos.decode("utf-8")

    return cadena

def firmarRSA_PSS(texto, key_private):
    # La firma se realiza sobre el hash del texto (h)
    h = SHA256.new(texto.encode("utf-8"))
    print(h.hexdigest())
    signature = pss.new(key_private).sign(h)

    return signature

def comprobarRSA_PSS(texto, firma, key_public):
    # Comprobamos que la firma coincide con el hash (h)
    h = SHA256.new(texto.encode("utf-8"))
    print(h.hexdigest())
    verifier = pss.new(key_public)
    try:
        verifier.verify(h, firma)
        return True
    except (ValueError, TypeError):
        return False
```

APÉNDICE B: Funciones a implementar de ECC

```
from Crypto.PublicKey import ECC
from Crypto.Hash import SHA256
from Crypto.Signature import DSS

# Ver https://pycryptodome.readthedocs.io/en/latest/src/public_key/ecc.html
# Ver https://pycryptodome.readthedocs.io/en/latest/src/signature/dsa.html

def crear_ECCKey():
    # Use 'NIST P-256'
    return key

def guardar_ECCKey_Privada(fichero, key, password):
    # ...

def cargar_ECCKey_Privada(fichero, password):
    # ...
    return key

def guardar_ECCKey_Publica(fichero, key):
    # ...

def cargar_ECCKey_Publica(fichero):
    # ...
    return key_pub

# def cifrarECC_OAEP(cadena, key):
    # El cifrado con ECC (ECIES) aun no está implementado
    # Por lo tanto, no se puede implementar este método aun en la versión 3.9.7
    return cifrado

# def descifrarECC_OAEP(cifrado, key):
    # El cifrado con ECC (ECIES) aun no está implementado
    # Por lo tanto, no se puede implementar este método aun en la versión 3.9.7
    return cadena

def firmarECC_PSS(texto, key_private):
    # ...
    return signature

def comprobarECC_PSS(texto, firma, key_public):
    # ...
    try:
        # ...
        return True
    except (ValueError, TypeError):
        return False
```

APÉNDICE C: Definición de la clase RSA_OBJECT()

```
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:"

    def create_KeyPair(self):
        """Crea un par de claves publico/privada, y las almacena dentro de la instancia"""

    def save_PrivateKey(self, file, password):
        """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"""

    def save_PublicKey(self, file):
        """Guarda la clave publica self.public_key en un fichero file"""

    def load_PublicKey(self, file):
        """Carga la clave publica self.public_key de un fichero file"""

    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"""

    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"""

    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."""

    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."""
```