PRÁCTICA 2: Criptografía (Parte 3)

Seguridad de la Información
Curso 2018-2019

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

RELACIÓN DE EJERCICIOS:

1. (**5 puntos**) El código Python descrito en el apéndice B (y en las transparencias relacionadas con esta práctica) muestra el funcionamiento del algoritmo RSA para el cifrado/descifrado y para la firma/comprobación de firmas. Utilizando como base ese código, crear una clase llamada RSA_OBJECT que tenga los métodos indicados en el apéndice A, 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()
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")
```

2. Realizar los siguientes apartados:

   a. (**2.5 puntos**) Implementar el siguiente protocolo haciendo uso de la clase RSA_OBJECT(), siendo el *mensaje* "Hola Amigos de la Seguridad":

      i. A: c = $\text{Cifrado}_{\text{Publica\_B}}$(*mensaje*)
      ii. A: s = $\text{Firma}_{\text{Privada\_A}}$(c)
      iii. A → B: c , s
      iv. Si B: $\text{Comprobar}_{\text{Publica\_A}}$(c, s)
      v. B: mensaje = $\text{Descifrado}_{\text{Privada\_B}}$(c)

      NOTA: En el paso iii, tanto c como s deben ser guardados en un fichero, y leídos posteriormente para realizar los pasos iv. y v. Para ello, hay que tener en cuenta que el tamaño por defecto de una firma s es de 256 bytes.

   b. (**2.5 puntos**) Haciendo uso del protocolo del apartado anterior, hacer que A y B compartan una clave de sesión de AES de tamaño 128 bits. Posteriormente, hacer uso de esa clave de sesión para enviar un mensaje cifrado (“Hola amigos de la seguridad”) con cualquier modo de operación de AES. Comparar el tamaño del mensaje cifrado de este apartado con el apartado anterior, y explicar sus diferencias.

APÉNDICE A: 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."""
```

APÉNDICE B: 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, RSAKey, 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, RSAKey):
    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):
    h = SHA256.new(texto.encode("utf-8")) # Ya veremos los hash la semana que viene
    print(h.hexdigest())
    signature = pss.new(key_private).sign(h)

    return signature

def comprobarRSA_PSS(texto, firma, key_public):
    h = SHA256.new(texto.encode("utf-8")) # Ya veremos los hash la semana que viene
    print(h.hexdigest())
    verifier = pss.new(key_public)
    try:
        verifier.verify(h, firma)
        return True
    except (ValueError, TypeError):
        return False

# Main
# Crear clave RSA
# y guardar en ficheros la clave privada (protegida) y publica
password = "password"
fichero_privado = "rsa_key.pem"
fichero_publico = "rsa_key.pub"
```

```
key = crear_RSAKey()
guardar_RSAKey_Privada(fichero_privado, key, password)
guardar_RSAKey_Publica(fichero_publico, key)

# Cargar la clave RSA privada del fichero, y muestra ambas en pantalla
# (La estructura de la clave privada tambien guarda la clave publica)
key = cargar_RSAKey_Privada(fichero_privado, password)
print(key.publickey().export_key())
print(key.export_key())

# Cifrar y Descifrar con PKCS1 OAEP
cadena = "Lo desconocido es lo contrario de lo conocido. Pasalo."
cifrado = cifrarRSA_OAEP(cadena, key)
print(cifrado)
descifrado = descifrarRSA_OAEP(cifrado, key)
print(descifrado)

# Firmar y comprobar con PKCS PSS
firma = firmarRSA_PSS(cadena, key)

keypub = cargar_RSAKey_Publica(fichero_publico)
if comprobarRSA_PSS(cadena, firma, keypub):
    print("La firma es valida")
else:
    print("La firma es invalida")
```