**Laboratorio N°03**

Alumno: Chañi Laura, Jose Carlos

**Laboratorio N°3 - Ing. Oscar Ramirez**

**ACTIVIDADES**

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**

* 1. Crear una clave pública y una clave privada RSA de 2048 bits para Alice. Guardar cada clave en un fichero.
  2. Crear una clave pública y una clave privada RSA de 2048 bits para Bob. Guardar cada clave en un fichero.

key\_alicia= crear\_RSAKey()

guardar\_RSAKey\_Publica("alicia\_keyPubli", key\_alicia)

guardar\_RSAKey\_Privada("alicia\_keyPriv", key\_alicia, "aliciaPrivada")

key\_bob =crear\_RSAKey()

guardar\_RSAKey\_Publica("bob\_keyPubli", key\_bob)

guardar\_RSAKey\_Privada("bob\_keyPriv", key\_bob, "bobPrivada")

**alice.py**

* 1. Cargar la clave privada de Alice y la clave pública de Bob.
  2. Cifrar el texto “Hola amigos de la seguridad” utilizando la clave de Bob.
  3. Firmar el texto “Hola amigos de la seguridad” utilizando la clave de Alice.
  4. Guardar en unos ficheros, el texto cifrado y la firma digital.

def saveData(fichero, data):

file\_out = open(fichero, "wb")

file\_out.write(data)

file\_out.close()

def loadData(fichero):

data = open(fichero, "rb").read()

return data

alice\_keyPriv = cargar\_RSAKey\_Privada("alicia\_keyPriv","aliciaPrivada")

bob\_keyPubli = cargar\_RSAKey\_Publica("bob\_keyPubli")

texto = "Hola amigos de la seguridad"

textoCifrado= cifrarRSA\_OAEP(texto,bob\_keyPubli)

saveData("cifraBobPub", textoCifrado)

textoFirmado = firmarRSA\_PSS(texto,alice\_keyPriv)

saveData("firmaAlicePriv", textoFirmado)

**bob.py**

* 1. Cargar la clave privada de Bob y la clave pública de Alice.
  2. Cargar el texto cifrado y la firma digital.
  3. Descifrar el texto cifrado y mostrarlo por pantalla.
  4. Comprobar la validez de la firma digital.

bob\_keyPriv = cargar\_RSAKey\_Privada("bob\_keyPriv", "bobPrivada")

alice\_keyPubli = cargar\_RSAKey\_Publica("alicia\_keyPubli")

textCipher = loadData("cifraBobPub")

firmDigital = loadData("firmaAlicePriv")

textDecipher=descifrarRSA\_OAEP(textCipher, bob\_keyPriv)

print("Texto Descifrado con la clave privada de Bob: ",textDecipher)

prueba = comprobarRSA\_PSS(textDecipher, firmDigital,alice\_keyPubli)

print("Resultado prueba de texto: ",prueba)

1. 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 cifrado1 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>

from Crypto.PublicKey import ECC

from Crypto.Hash import SHA256

from Crypto.Signature import DSS

def crear\_ECCKey():

key = ECC.generate(curve='P-256')

return key

def guardar\_ECCKey\_Privada(fichero, key, password):

f = open(fichero,'wt')

f.write(key.export\_key(format='PEM',passphrase=password,protection="scryptAndAES128-CBC"))

f.close()

def cargar\_ECCKey\_Privada(fichero, password):

keyCifra = open(fichero,'rt').read()

key = ECC.import\_key(keyCifra,passphrase=password)

return key

def guardar\_ECCKey\_Publica(fichero, key):

key\_pub = key.public\_key().export\_key(format='PEM')

file\_out = open(fichero, "wt")

file\_out.write(key\_pub)

file\_out.close()

def cargar\_ECCKey\_Publica(fichero):

keyFile = open(fichero, "rt").read()

key\_pub = ECC.import\_key(keyFile)

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 3.9.0

# 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 3.9.0

# return cadena

def firmarECC\_PSS(texto, key\_private):

h = SHA256.new(texto.encode("utf-8"))

print(h.hexdigest())

signature = DSS.new(key\_private, 'fips-186-3').sign(h)

return signature

# ...

# return 0

def comprobarECC\_PSS(texto, firma, key\_public):

h= SHA256.new(texto.encode("utf-8"))

verifier = DSS.new(key\_public,'fips-186-3')

try:

verifier.verify(h, firma)

print("Correcto")

return True

except (ValueError, TypeError):

print("Incorrecto")

return False

def saveData(fichero, data):

file\_out = open(fichero, "wb")

file\_out.write(data)

file\_out.close()

def loadData(fichero):

data = open(fichero, "rb").read()

return data

#key = crear\_ECCKey()

#guardar\_ECCKey\_Publica("ECCPublic.pem", key)

#guardar\_ECCKey\_Privada("ECCPrivada.pem",key,"password")

clavPriv = cargar\_ECCKey\_Privada("ECCPrivada.pem","password")

clavPubli = cargar\_ECCKey\_Publica("ECCPublic.pem")

firma = firmarECC\_PSS("Hola Amigos", clavPriv)

comprobarECC\_PSS("Hola Amigos",firma,clavPubli)

1. 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.Hash import SHA256

from Crypto.Signature import pss

class RSA\_OBJECT:

def \_\_init\_\_(self):

self.key = None

self.private\_key=None

self.public\_key = None

def create\_KeyPair(self):

self.key = RSA.generate(2048)

self.private\_key = self.key

self.public\_key = self.key.publickey()

def save\_PrivateKey(self, file, password):

key\_cifrada = self.private\_key.export\_key(passphrase=password, pkcs=8, protection="scryptAndAES128-CBC")

file\_out = open(file, "wb")

file\_out.write(key\_cifrada)

file\_out.close()

def load\_PrivateKey(self, file, password):

key\_cifrada = open(file, "rb").read()

self.private\_key = RSA.import\_key(key\_cifrada, passphrase=password)

def save\_PublicKey(self, file):

key\_cifrada = self.public\_key.export\_key()

file\_out = open(file, "wb")

file\_out.write(key\_cifrada)

file\_out.close()

def load\_PublicKey(self, file):

keyFile = open(file, "rb").read()

self.public\_key = RSA.import\_key(keyFile)

def cifrar(self, datos):

#data = datos.encode("utf-8")

engineRSACipher = PKCS1\_OAEP.new(self.public\_key)

cifrado = engineRSACipher.encrypt(datos)

return cifrado

#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):

engineRSACipher = PKCS1\_OAEP.new(self.private\_key)

datos = engineRSACipher.decrypt(cifrado)

#datos = datos.decode('utf-8')

return datos

# 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):

h= SHA256.new(datos)

print(h.hexdigest())

signature = pss.new(self.private\_key).sign(h)

return signature

# 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):

h = SHA256.new(text)

print(h.hexdigest())

verificador = pss.new(self.public\_key)

try:

verificador.verify(h,signature)

return True

except(ValueError, TypeError):

return False

# 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.

# 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"))

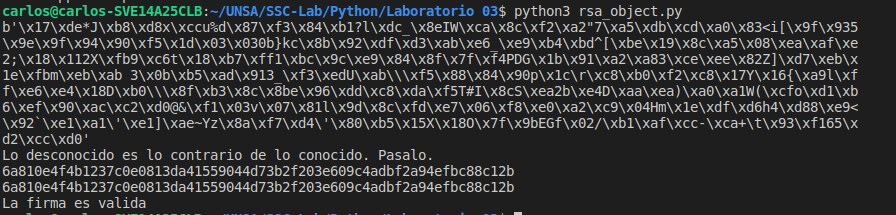
if RSA\_public.comprobar(cadena.encode("utf-8"), firma):

print("La firma es valida")

else:

print("La firma es invalida")

Ejecución:



**ANEXOS**

**ca.py**

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

key\_alicia= crear\_RSAKey()

guardar\_RSAKey\_Publica("alicia\_keyPubli", key\_alicia)

guardar\_RSAKey\_Privada("alicia\_keyPriv", key\_alicia, "aliciaPrivada")

key\_bob =crear\_RSAKey()

guardar\_RSAKey\_Publica("bob\_keyPubli", key\_bob)

guardar\_RSAKey\_Privada("bob\_keyPriv", key\_bob, "bobPrivada")

**alice.py**

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

def saveData(fichero, data):

file\_out = open(fichero, "wb")

file\_out.write(data)

file\_out.close()

def loadData(fichero):

data = open(fichero, "rb").read()

return data

alice\_keyPriv = cargar\_RSAKey\_Privada("alicia\_keyPriv","aliciaPrivada")

bob\_keyPubli = cargar\_RSAKey\_Publica("bob\_keyPubli")

texto = "Hola amigos de la seguridad"

textoCifrado= cifrarRSA\_OAEP(texto,bob\_keyPubli)

saveData("cifraBobPub", textoCifrado)

textoFirmado = firmarRSA\_PSS(texto,alice\_keyPriv)

saveData("firmaAlicePriv", textoFirmado)

**bob.py**

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

def saveData(fichero, data):

file\_out = open(fichero, "wb")

file\_out.write(data)

file\_out.close()

def loadData(fichero):

data = open(fichero, "rb").read()

return data

bob\_keyPriv = cargar\_RSAKey\_Privada("bob\_keyPriv", "bobPrivada")

alice\_keyPubli = cargar\_RSAKey\_Publica("alicia\_keyPubli")

textCipher = loadData("cifraBobPub")

firmDigital = loadData("firmaAlicePriv")

textDecipher=descifrarRSA\_OAEP(textCipher, bob\_keyPriv)

print("Texto Descifrado con la clave privada de Bob: ",textDecipher)

prueba = comprobarRSA\_PSS(textDecipher, firmDigital,alice\_keyPubli)

print("Resultado prueba de texto: ",prueba)