

PRÁCTICA 2: Asimétrico

Seguridad de la Información

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

Se recomienda: leer todo el enunciado con calma y al detalle, incluyendo los apéndices y los tutoriales, antes de proceder a programar el protocolo.

RELACIÓN DE EJERCICIOS:

1. En este apartado se persigue un intercambio de cierta información (K1, un array de 16 bytes) entre dos entidades Alice y Bob utilizando criptografía asimétrica (RSA). Para ello, se pide realizar en los siguientes ficheros las operaciones mostradas a continuación. Dichas operaciones están reflejadas en la Figura 1.

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

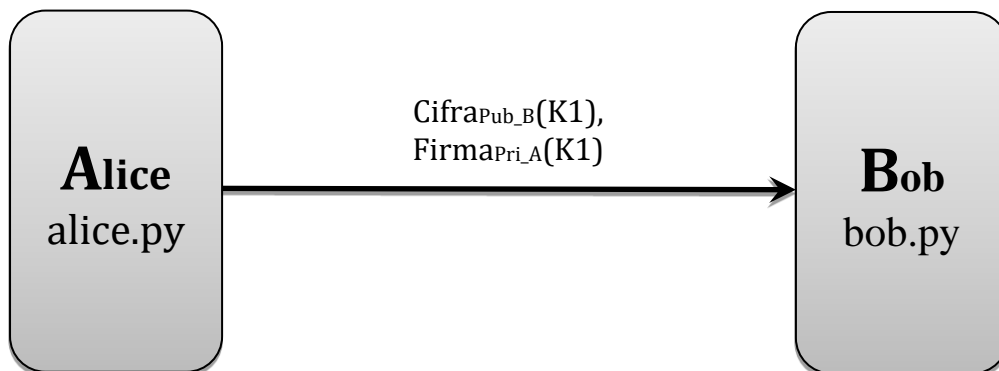
- c. Cargar la clave privada de Alice y la clave pública de Bob.
- d. Cifrar un array de 16 bytes (K1) utilizando la clave de Bob.
- e. Firmar dicho array de 16 bytes (K1) utilizando la clave de Alice.
- f. Enviar el cifrado y la firma usando un socket cliente de la clase `SOCKET_SIMPLE_TCP()`, descrita en los apéndices.

bob.py

- g. Cargar la clave privada de Bob y la clave pública de Alice.
- h. Recibir el texto cifrado y la firma digital a través del socket servidor de la clase `SOCKET_SIMPLE_TCP()`, descrita en los apéndices.
- i. Descifrar el array de 16 bytes (K1) y mostrarlo por pantalla.
- j. Comprobar la validez de la firma digital.

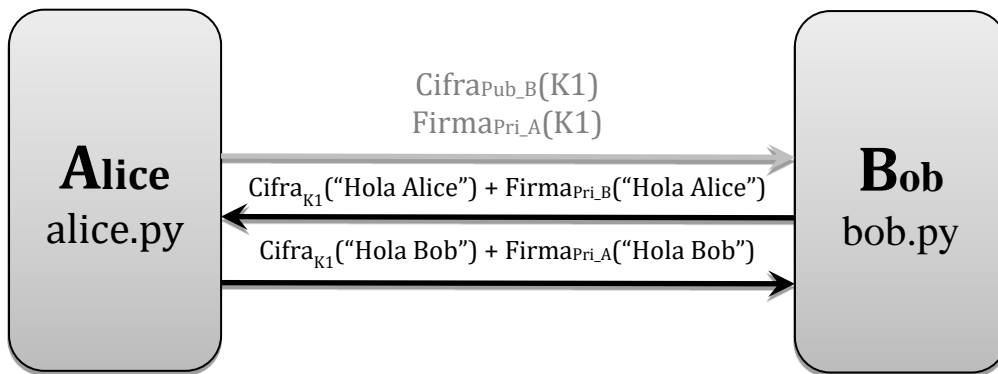
Las funciones que proporcionan el cifrado, la firma y la creación de claves RSA se encuentran en la librería **funciones_rsa**, disponible en el campus virtual y descrita en los apéndices.

NOTA: El array de 16 bytes utilizado en este apartado será utilizado como una clave simétrica AES en el siguiente apartado.



2. Completar el apartado 1 con los siguientes pasos:
- Tras el último paso del apartado 1, Bob cifrará la cadena "Hola Alice" utilizando **AES-CTR-128** con la clave K1, y firmará esa cadena con su clave privada. A continuación, enviará dicho cifrado y la firma a través del socket ya abierto en el apartado 1.
 - Alice recibirá el texto cifrado con la clave simétrica y la firma digital a través del socket ya abierto en el apartado 1. Después, descifrará la cadena de caracteres y la mostrará por pantalla tras comprobar la validez de la firma digital.
 - Finalmente, Alice repetirá los pasos a) y b) en reverso: enviando a Bob la cadena cifrada "Hola Bob" con K1 en AES-CTR-128, junto con la firma digital de dicha cadena, y Bob se encargará de descifrar y comprobar la firma digital.

Las funciones que proporcionan las operaciones de AES necesarias para la realización de este ejercicio se encuentran en la librería **funciones_aes**, disponible en el campus virtual y descrita en los apéndices.



Librería “SOCKET_SIMPLE_TCP”

La librería “SOCKET_SIMPLE_TCP” es una librería simple¹ creada por los profesores de la asignatura que permiten enviar y recibir mensajes a través de sockets TCP, cuyo código fuente se incluye en este enunciado. Estos mensajes se envían y reciben como array de bytes.

Un servidor se crea e intercambia mensajes de la siguiente forma:

```
socketserver = SOCKET_SIMPLE_TCP('127.0.0.1', 5551)
socketserver.escuchar()
array_bytes = socketserver.recibir()
socketserver.enviar(array_bytes)
socketserver.cerrar()
```

Mientras que un cliente se conecta e intercambia mensajes de la siguiente forma:

```
socketclient = SOCKET_SIMPLE_TCP('127.0.0.1', 5551)
socketclient.conectar()
socketclient.enviar(array_bytes)
array_bytes = socketclient.recibir()
socketclient.cerrar()
```

Librería “funciones_aes” y “funciones_rsa”

Estas librerías creadas por los profesores de la asignatura permiten realizar varias operaciones criptográficas simétricas y asimétricas (p. ej., cifrar y descifrar con AES en los modos de operación GCM y CTR, cifrar y firmar usando los modos de operación de RSA).

Para utilizar una librería hay que importarla (`import funciones_aes`), y para utilizar cada una de sus funciones hay que colocar el nombre de la librería antes de cada método (p. ej., `aes_engine = funciones_aes.iniciarAES_CTR_cifrado(k1)`).

Respecto al uso de la librería, la entidad que envía los datos (**emisor**) debe inicializar el objeto `aes` con la función `iniciarAES_GCM_cifrado()`, mientras que la entidad que recibe los datos (**receptor**) debe iniciar el objeto `aes` con la función `iniciarAES_GCM_descifrado()`. El **emisor únicamente** puede cifrar, y el **receptor únicamente** puede descifrar.

Si una entidad funciona como **emisor y receptor**, entonces debe tener *un objeto aes para el cifrado, y un objeto aes para el descifrado*.

¹ No utilizable en entornos de producción

APENDICE: Código fuente funciones RSA

```
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(datos, key):
    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)

    return datos

def firmarRSA_PSS(datos, key_private):
    # La firma se realiza sobre el hash de los datos (h)
    h = SHA256.new(datos) # Le hacemos el hash a los datos
    print(h.hexdigest()) # Imprimimos por pantalla el hash en un formato visible por personas
    signature = pss.new(key_private).sign(h)

    return signature

def comprobarRSA_PSS(datos, firma, key_public):
    # Comprobamos que la firma coincide con el hash (h)
    h = SHA256.new(datos) # Le hacemos el hash a los datos
    print(h.hexdigest()) # Imprimimos por pantalla el hash en un formato visible por personas
    verifier = pss.new(key_public)
    try:
        verifier.verify(h, firma)
        return True
    except (ValueError, TypeError):
        return False
```

APENDICE: Código fuente clase SOCKET_SIMPLE_TCP()

```

"""
Clase de ejemplo para el envio y la recepcion de mensajes en un canal TCP
No utilizar en un entorno de produccion
"""

import socket
import struct
import sys

class SOCKET_SIMPLE_TCP:

    def __init__(self, host, puerto):
        """Inicializa un objeto socket TCP, proporcionando un host y a un puerto"""
        self.host = host
        self.puerto = puerto
        self.server = None

    def conectar(self):
        """Convierte el objeto socket en un cliente, y se conecta a un servidor"""
        self.socket = socket.create_connection((self.host, self.puerto))

    def escuchar(self):
        """Convierte el objeto socket en un servidor, y recibe la peticion de un cliente"""
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_address = (self.host, self.puerto)
        self.server.bind(server_address)
        self.server.listen(1)
        self.socket, dir_cliente = self.server.accept()
        return dir_cliente

    def __recvall(self, count):
        """PRIVADO: Recibe "count" bytes del buffer de entrada"""
        buffer = b''
        while count:
            # Puede que no reciba "count", asi que tengo que leer hasta recibirlo todo
            newbuf = self.socket.recv(count)
            if not newbuf: return None
            buffer += newbuf
            count -= len(newbuf)
        return buffer

    def enviar(self, datos):
        """Envia un array de bytes "datos" del origen al destino."""
        longitud = len(datos)
        self.socket.sendall(struct.pack('!I', longitud)) # unsigned int en formato de red(!)
        self.socket.sendall(datos)

    def recibir(self):
        """Recibe un array de bytes "datos" del destino al destino."""
        lenbuf = self.__recvall(4)
        longitud, = struct.unpack('!I', lenbuf)
        return self.__recvall(longitud)

    def cerrar(self):
        """Cierra la conexcion"""
        if self.socket != None:
            self.socket.close()
        if self.server != None:
            self.server.close()

"""
Prepara socket en puerto 3333: s = SOCKET_SIMPLE_TCP('127.0.0.1',3333)
Ejemplo de envío (socket cliente): s.conectar() / s.enviar(dato) / s.cerrar()
Ejemplo de recepción (socket servidor): s.escuchar() / dato = s.recibir() / s.cerrar()
"""

```