PRÁCTICA 3: Protocolos

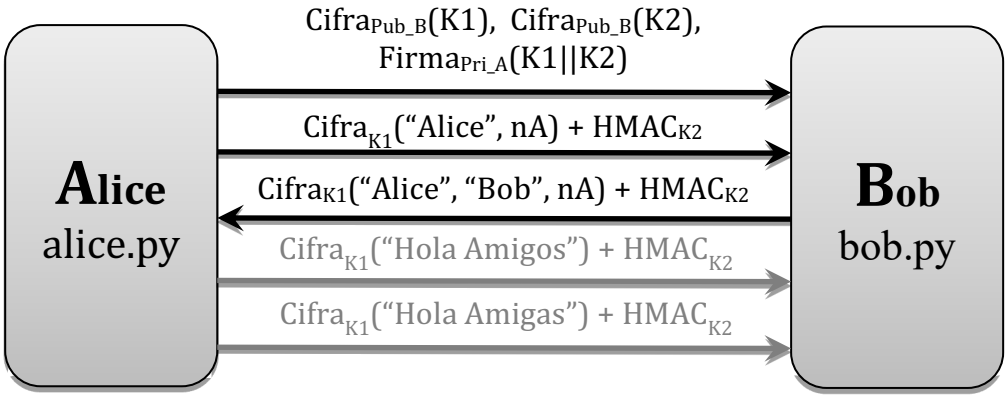
Seguridad en la Información
Curso 2021-2022

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

RELACIÓN DE EJERCICIOS:

1. Recordemos que en la prueba de laboratorio I se pedía implementar los ficheros `ca.py`, `alice.py`, y `bob.py`, los cuales permiten a dos entidades **Alice** y **Bob** comunicarse de forma segura. Para este ejercicio, se pide implementar un protocolo muy parecido, pero con algunas variaciones:
    a. La comunicación entre **Alice** y **Bob** no se realizará a través de ficheros, sino a través de sockets utilizando *la librería SOCKET_SIMPLE_TCP*, en donde **Bob** será un servidor y **Alice** un cliente (ver apéndice A).
    b. **Alice** le envía a **Bob** dos claves: K1 y K2. K1 se utilizará para cifrar el intercambio de mensajes entre Alice y Bob utilizando AES CTR 128, mientras que K2 se utilizará junto con un *HMAC* (basado en SHA256) para proporcionar integridad (ver apéndice A).
    c. **Alice** firma la concatenación entre K1 y K2.
    d. Se añade un nuevo paso antes del intercambio de información:
        i. **Alice** le envía a **Bob** su nombre ("Alice") y un nonce (valor aleatorio de 128 bits)
        ii. **Bob** le responde a **Alice** con su nombre ("Bob") y los campos enviados por Alice. En este punto **Alice** debe comprobar que los campos recibidos de **Bob** son los mismos que ella envió.

    Para la creación de un mensaje con varios campos (p.ej. {"Alice", nonce}) se debe utilizar *el formato JSON* (ver Apéndice A)



APENDICE A: Aspectos teóricos de Python a considerar

JSON

El formato JSON, o JavaScript Object Notation, es una forma de representar objetos en forma de diccionarios, donde **cada valor debe ser una cadena**. Para el envío de mensajes, es posible guardar cada campo dentro de un array, de la siguiente forma:

```
mensaje = [] # Array vacio
mensaje.append(alice)       # Donde alice == “Alice”
mensaje.append(nonce.hex()) # Conversion de Bytes a Hexadecimal
jStr = json.dumps(mensaje)  # Convertimos un Array Python a string
```

...y recuperarse de la siguiente forma:

```
mensaje = json.loads(jStr) # Recuperamos un Array Python de un string
alice, nonce_cadenaHEX = mensaje
nonce = bytearray.fromhex(nonce_cadenaHEX) # De Hexadecimal a Bytes
```

HMAC

Como hemos visto en la teoría, HMAC es una primitiva criptográfica que permite obtener un código de integridad, el cual conlleva autenticación implícita gracias al uso de una clave simétrica.

El uso de HMAC para PyCryptodome puede consultarse en:
https://pycryptodome.readthedocs.io/en/latest/src/hash/hmac.html

Múltiples variables devueltas por una función en Python

En Python, un método (función) puede proporcionar múltiples resultados. Por ejemplo, si un método “cifrarAES_GCM” devuelve:

```
return datos_cifrado, mac_cifrado
```

Entonces estas variables pueden recogerse de la siguiente forma:

```
c, mac = funciones_aes.cifrarAES_GCM(aes,json_ET.encode("utf-8"))
```

Librería “SOCKET_SIMPLE_TCP”

La librería “SOCKET_SIMPLE_TCP” es una librería simple[1] creada por los profesores de la asignatura que permiten enviar y recibir mensajes a través de sockets TCP. 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()
```

[1] No utilizable en entornos de producción

```
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). 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*.