

p-b.py.pdf



piedad_pg



Seguridad de la Información



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga



**Que no te escriban poemas de amor
cuando terminen la carrera**



*(a nosotros por
suerte nos pasa)*

WUOLAH

Que no te escriban poemas de amor
cuando terminen la carrera ▶▶▶▶▶▶▶▶



WUOLAH

(a nosotros por suerte nos pasa)

No si antes decirte
Lo mucho que te voy a recordar

Pero me voy a graduar.
Mañana mi diploma y titulo he de
pagar

Llegó mi momento de despedirte
Tras años en los que has estado mi
lado.

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Oh Wuolah wuolilah
Tu que eres tan bonita

17/10/23, 14:39

p-b.py

```
1
2
3 from Crypto.Hash import SHA256, HMAC
4 import base64
5 import json
6 import sys
7 from socket_class import SOCKET_SIMPLE_TCP
8 import funciones_aes
9 from Crypto.Random import get_random_bytes
10
11 # Paso 0: Inicializacion
12 #####
13
14 # Lee clave KBT
15 KBT = open("KBT.bin", "rb").read()
16
17 # Paso 1) B->T: KBT(Bob, Nb) en AES-GCM
18 #####
19
20 # Crear el socket de conexion con T (5551)
21 print("Creando conexion con T...")
22 socket = SOCKET_SIMPLE_TCP('127.0.0.1', 5551)
23 socket.conectar()
24
25 # Crea los campos del mensaje
26 t_n_origen = get_random_bytes(16)
27
28 # Codifica el contenido (los campos binarios en una cadena) y contruyo el mensaje
29 # JSON
30 msg_TE = []
31 msg_TE.append("Bob")
32 msg_TE.append(t_n_origen.hex())
33 json_ET = json.dumps(msg_TE)
34 print("B -> T (descifrado): " + json_ET)
35
36 # Cifra los datos con AES GCM
37 aes_engine = funciones_aes.iniciarAES_GCM(KBT)
38 cifrado, cifrado_mac, cifrado_nonce =
39 funciones_aes.cifrarAES_GCM(aes_engine,json_ET.encode("utf-8"))
40
41 # Envia los datos
42 socket.enviar(cifrado)
43 socket.enviar(cifrado_mac)
44 socket.enviar(cifrado_nonce)
45
46 # Paso 2) T->B: KBT(K1, K2, Nb) en AES-GCM
47 #####
48 cifradoB = socket.recibir()
49 cifrado_macB = socket.recibir()
50 cifrado_nonceB = socket.recibir()
51 datos_claro=funciones_aes.descifrarAES_GCM(KBT, cifrado_nonceB,
52 cifradoB,cifrado_macB)
53
54 # Decodifica el contenido: Bob, Nb
55 json_BT = datos_claro.decode("utf-8" ,"ignore")
56 print("B->T (descifrado): " + json_BT)
57 msg_BT = json.loads(json_BT)
58
59 # Extraigo el contenido
60 K1,K2, t_nb = msg_BT
```

localhost:4649/?mode=python

WUOLAH 1/3

```

58 K1 = bytearray.fromhex(K1)
59 K2 = bytearray.fromhex(K2)
60 t_nb = bytearray.fromhex(t_nb)
61
62 if(t_nb==t_n_origen):
63     print("El nonce es el mismo")
64 else:
65     print("El nonce no es el mismo")
66     exit
67
68
69
70
71 # Cerramos el socket entre B y T, no lo utilizaremos mas
72 socket.cerrar()
73
74 # Paso 5) A->B: KAB(Nombre) en AES-CTR con HMAC
75 #####
76 print("Esperando a Alice...")
77 socket = SOCKET_SIMPLE_TCP('127.0.0.1', 5553)
78 socket.escuchar()
79 #recibe el nombre
80 paquete = socket.recibir()
81 # Decodifica el contenido:
82 json_BT = paquete.decode("utf-8" ,"ignore")
83 print("A->B (descifrado): " + json_BT)
84 msg_BT = json.loads(json_BT)
85
86 # Extraigo el contenido
87 datos_cifrado, nonce, mac = msg_BT
88 datos_cifrado= bytearray.fromhex(datos_cifrado)
89 nonce = bytearray.fromhex(nonce)
90
91 #lo descifro
92 aes_descifrado=funciones_aes.iniciarAES_CTR_descifrado(K1, nonce)
93 datos_claro=funciones_aes.descifrarAES_CTR(aes_descifrado, datos_cifrado)
94 mensaje_claro_json = datos_claro.decode("utf-8")
95 print("A -> B (descifrado): " + mensaje_claro_json)
96
97
98 hmacB = HMAC.new(K2, digestmod=SHA256)
99 hmacB.update(mensaje_claro_json .encode("utf-8"))
100 try:
101     hmacB.hexverify(mac)
102     print("Mensaje correcto")
103 except ValueError:
104     print("Mensaje manipulado")
105     socket.cerrar()
106     exit()
107
108
109 # Paso 6) B->A: KAB(Apellido) en AES-CTR con HMAC
110 #####
111 apellido="Grillo"
112 aes_cifrado, nonce_16_ini = funciones_aes.iniciarAES_CTR_cifrado(K1)
113 datos_cifrado = funciones_aes.cifrarAES_CTR(aes_cifrado, apellido.encode("utf-8"))
114 #creo el hmac
115 hsend = HMAC.new(K2, msg=apellido.encode("utf-8"), digestmod=SHA256)
116 mac = hsend.digest()
117

```

```
118 mensaje = []
119 mensaje.append(datos_cifrado.hex())
120 mensaje.append(nonce_16_ini.hex())
121 mensaje.append(mac.hex())
122 json_paquete = json.dumps(mensaje)
123 socket.enviar(json_paquete.encode("utf-8"))
124
125
126
127 # Paso 7) A->B: KAB(END) en AES-CTR con HMAC
128 #####
129 #recibe el end
130 paquete = socket.recibir()
131 # Decodifica el contenido:
132 json_BT = paquete.decode("utf-8", "ignore")
133 print("A->B (descifrado): " + json_BT)
134 msg_BT = json.loads(json_BT)
135
136 # Extraigo el contenido
137 datos_cifrado, nonce, mac = msg_BT
138 datos_cifrado= bytearray.fromhex(datos_cifrado)
139 nonce = bytearray.fromhex(nonce)
140
141 #lo descifro
142 aes_descifrado=funciones_aes.iniciarAES_CTR_descifrado(K1, nonce)
143 datos_claro=funciones_aes.descifrarAES_CTR(aes_descifrado, datos_cifrado)
144 mensaje_claro_json = datos_claro.decode("utf-8")
145 print("A -> B (descifrado): " + mensaje_claro_json)
146
147
148 hmacB = HMAC.new(K2, digestmod=SHA256)
149 hmacB.update(mensaje_claro_json .encode("utf-8"))
150 try:
151     hmacB.hexverify(mac)
152     print("Mensaje correcto")
153 except ValueError:
154     print("Mensaje manipulado")
155     socket.cerrar()
156     exit()
157
158 print("tras enviar el end se cierran las conexiones")
159 socket.cerrar()
160 # (A realizar por el alumno/a...)
161
162
```