**Introducción**

El siguiente documento tiene como propósito reportar lo realizado en Implementación del protocolo criptográfico seguro y eficiente para manejo de información médica en dispositivos inteligentes, en conjunto con el sistema de autenticación de sensores en una red de área corporal basada en acelerómetro.

Se desglosa de forma detallada las complicaciones encontradas a lo largo de la implementación, así como las acciones realizadas para corregir su funcionamiento, también se incluyen los factores que debe ser tomados en cuenta a la hora de realizar pruebas, todo con el fin de facilitar futuras implementaciones.

**Propuesta**

La propuesta propone 4 elementos un Servidor A, un Nodo Coordinador B, un Sensor o Sensores C y un Actuador D.

* Servidor A.

Este elemento será una computadora, la cual tendrá montado un servidor en espera de peticiones.

* Nodo Coordinador B.

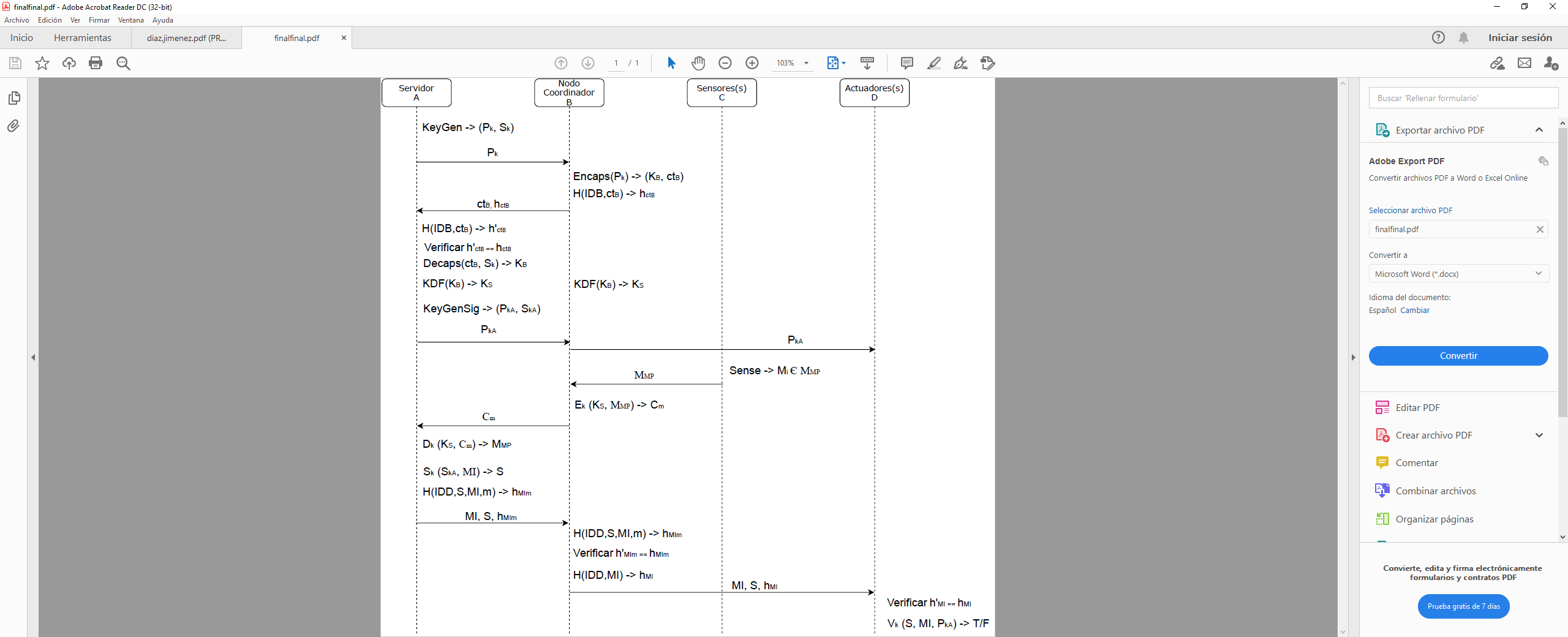
Este elemento será una Raspberry Pi 4 en conjunto con un sensor MPU-9250, la cual actuará de intermediario entre el Servidor A y el Actuador D, además de recibir los datos de los sensores C.

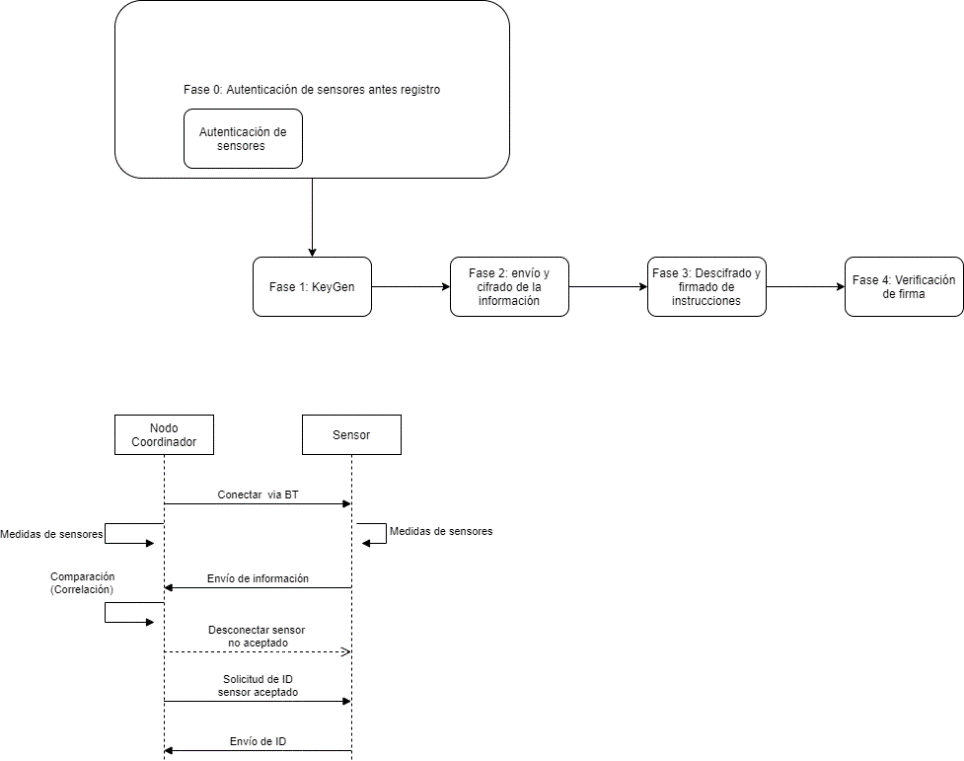
* Sensor o Sensores C.

Los sensores C dependerán del uso médico y para su autenticación es necesario la inclusión de un sensor MPU-9250.

* Actuador D.

El actuador D será un Arduino que recibirá comunicación desde el Nodo Coordinador B.

El siguiente esquema muestra detalladamente la comunicación y mensajes existentes entre los entes anteriormente listados:

Para el sistema de autenticación de sensores fue necesaria la inclusión de una fase 0 que se muestra en el siguiente diagrama:

Para la generación de llaves se hará uso de Kyber-512, para la función hash se hará uso de SHA-256, además se hará uso de una función KDF, para la encriptación se hará uso de AES en modo CBC, para la firma se hará uso de Dilithium 2.

**Desarrollo**

Del lado del Servidor A y Nodo Coordinador B, se decidió hacer uso de Python para el desarrollo de esta implementación.

Se hizo uso de tres bibliotecas externas “pycryptodome”, “pqcrypto” y “FaBo9Axis\_MPU9250” así que es necesario descargar e instalar estas bibliotecas para el funcionamiento correcto de esta implementación.

Al no saber la cantidad de sensores que serán agregados, ya que esto depende de la aplicación que se le quiera dar a la implementación, se decidió que toda la información que se desee enviar sea agregada en un documento, este puede ser .pdf, .txt, .doc e incluso imágenes es completamente libre.

Del lado del Actuador D se decidió usar una placa de desarrollo NodeMcu (ESP8266) esta utiliza el compilador de Arduino para cargar programas dentro de ella, se decidió utilizar esta placa de desarrollo por varias razones, una de ella es el precio ya que esta es muy barata, cuesta alrededor de 7 – 8 dlls, además esta placa ya tiene incluida un módulo de Wifi, además de que la documentación que existe es amplia, cuenta con diversas librerías y ejemplos que permiten realizar aplicaciones de forma sencilla y rápida, es necesario descargar la librería de ESP8266 y agregarla al compilador de Arduino para su correcto funcionamiento ya que por default este no la incluye.

* **Funcionamiento**

La aplicación se compone de un programa con sockets y un Web Service.

El Servidor A tendrá montado el servidor con un socket en blind, esperando peticiones del nodo Coordinador B

El Nodo Coordinador B hará función de cliente para el Servidor A realizará peticiones al servidor cada vez que lo necesite, para la autenticación de sensores, este elemento también tiene un servidor montado en espera de una petición de un sensor, además hace peticiones POST a un Web Service que estará montado en el Actuador D.

El Actuador D tendrá montado un Web Service el cual estará en espera de peticiones del nodo Coordinador B, se decidió realizarlo de esta forma para evitar abrir otro socket en el nodo coordinar B y aquí, aunque internamente si se realiza, a la hora de realizar la petición se automatiza todo este proceso. haciendo más ligero el código e igual de eficiente.

Los sensores realizaran peticiones al servidor montado en el Nodo Coordinador B en busca de autenticación, para esto se abre un socket y hace función de cliente.

* **Dificultades.**
  + **Problemas con Arduino**

A la hora del desarrollo de la aplicación en el Arduino, surgió una dificultad con la librería de Dilithium 2, esta al estar pensada en ser utilizada en una arquitectura de 64, 32 bits o ARM, existen algunos tipos de datos que utilizan una gran cantidad de memoria, el Arduino al ser un dispositivo de 8 bits estos datos no pueden ser utilizados ya que no pueden ser declarados, la cantidad de memoria utilizada es mayor a la que se tiene, la solución es: tener que reestructurar la librería para ser utilizada en una arquitectura de 8 bits, hacer uso de arreglos para lograr esto, sin embargo el tiempo de desarrollo de esta librería es amplio podría tardar algunos meses en completarse. Por tal razón se tuvo que reestructurar el protocolo en la fase de firma. Quedando de tal forma.

Ahora la verificación igual de ser realizada por el Actuador D será realizada por el Nodo Coordinador B, la verificación del hash seguirá siendo realizada en el Actuador D.

Nodo Coordinador B

Actuador D

Vk (S, MI, PkA) -> T/F

Verificar h’MI == hMI

hMI

* + **Problemas con la instalación de la librería “pqcripto” y solución.**

Se detecto el problema de que al instalar la librería de “pqcripto” en versiones iguales o superiores a Python 3.8, su instalación con pip falla, como recomendación primeramente el equipo debe estar completamente actualizado.

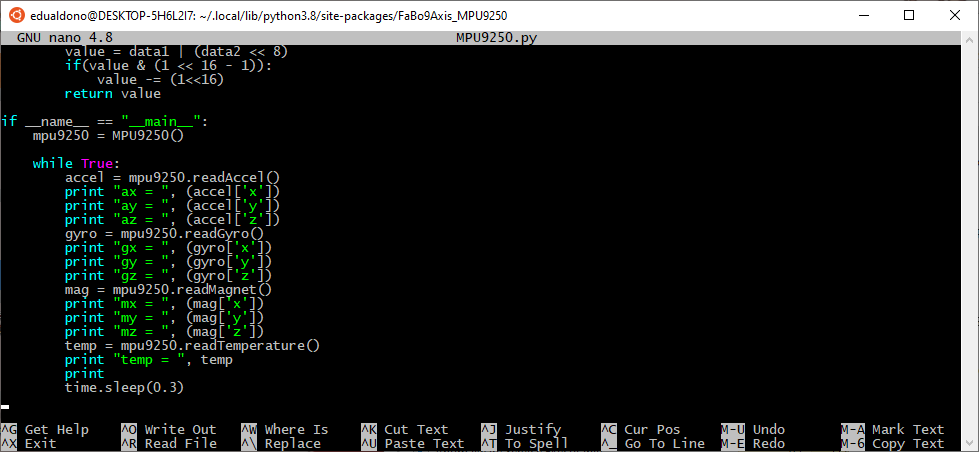
El error detectado es uno que está relacionado con el archivo “setup.py”, específicamente indicando que la librería “setuptools” no existe, aun cuando esta si se encuentra dentro del sistema. Este error es causado por un conflicto con los archivos de instalación.

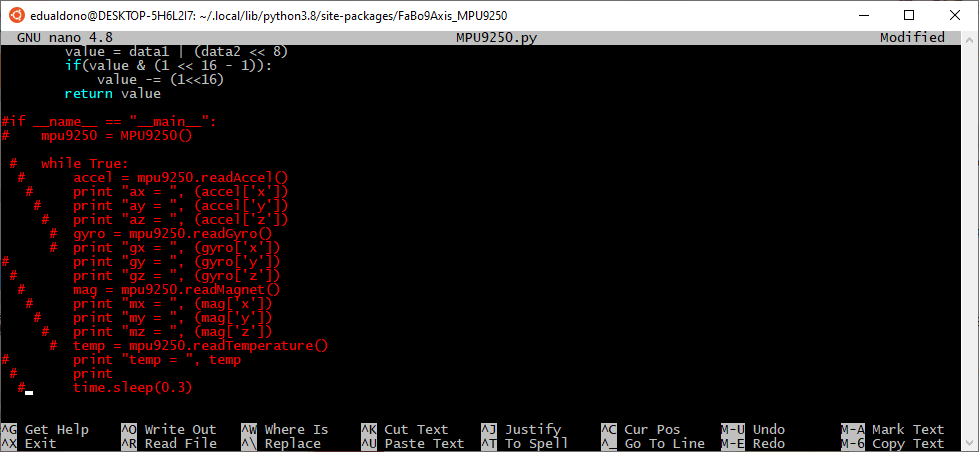
Para su solución es necesario descargar los archivos de instalación en <https://pypi.org/project/pqcrypto/>, se descargará un archivo con extensión “.tar.gz” se debe descomprimir dicho archivo, dentro de la carpeta obtenida se debe eliminar el archivo “pyproject.toml” posteriormente se vuelve a comprimir esta carpeta, la extensión es indiferente puede ser “.tar”, “.zip” o “.rar”. En la ubicación de este archivo se debe correr el comando “pip install Archhivo\_comprimido”, de esta forma la librería debería ser instalada sin ningún problema.

* + **Problemas con el uso de la librería “FaBo9Axis\_MPU9250” con Python 3 o superior y solución.**

Un problema detectado con esta librería es que dicha librería esta escrita para Python 2.7.

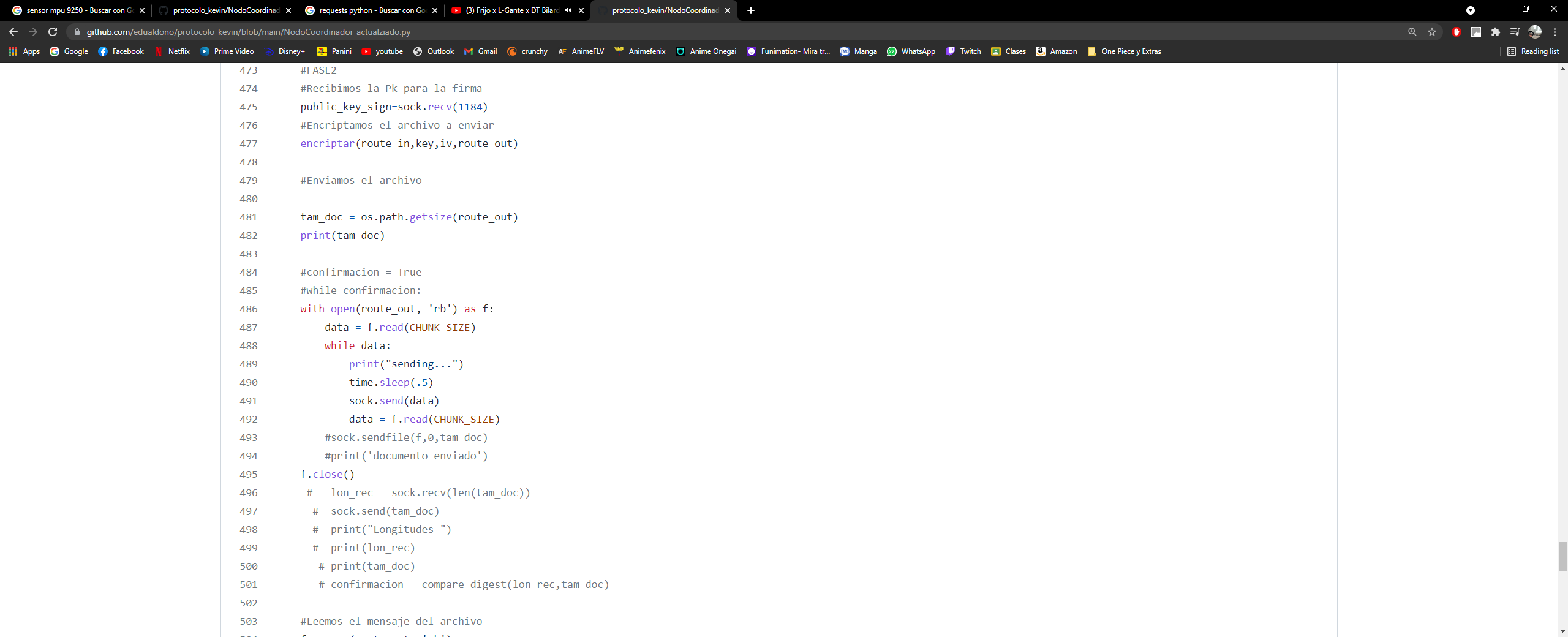
Para resolver el problema debemos dirigirnos a la carpeta donde fue instalada la librería normalmente la ruta es “.local/lib/python3.8/site-packages/ FaBo9Axis\_MPU9250” dentro de esta carpeta hay un archivo con nombre MPU9250.py este debe ser editado.

Al final del archivo existe un “if” que se muestra en la siguiente imagen todo lo que se encuentra dentro de ese “if” debe ser comentado.

El código terminara de esta forma y debe ser guardado, posteriormente a esto la librería podría ser usada sin ningún problema.

* + **Problemas con la recepción de archivos incompletos y solución.**

Un problema observado realizando pruebas ya con los 4 elementos funcionando, es en la recepción de documentos, las primeras pruebas realizadas en local host no surgió ningún problema sin embargo, en el escenario donde todos los dispositivos ya están funcionando con WIFI se detectaron algunos problemas, en distancias cortas entre los dispositivos y el punto de acceso a internet las pruebas fueron exitosas, por otra parte en el caso de existir distancias lejanas del punto de acceso a internet, se observó que algunos datos estaban llegando incompletos, especialmente el documento encriptado y el bloque de firma. Por alguna razón al existir mucha distancia, bloques de datos arriba de los 1024 bytes son susceptibles a perderse, además si estos se envían de forma muy continua de igual forma tiende a perderse información. Este es un problema grave sobre todo en el documento encriptado ya que se desencripta en bloques de 16 bytes, al estar incompleto el archivo el sistema completo falla.

La solución que se dio fue que todos los bloques enviados no sean superiores a 1024 bytes, además de darle pausas al envió de datos, como se observa en la siguiente figura.

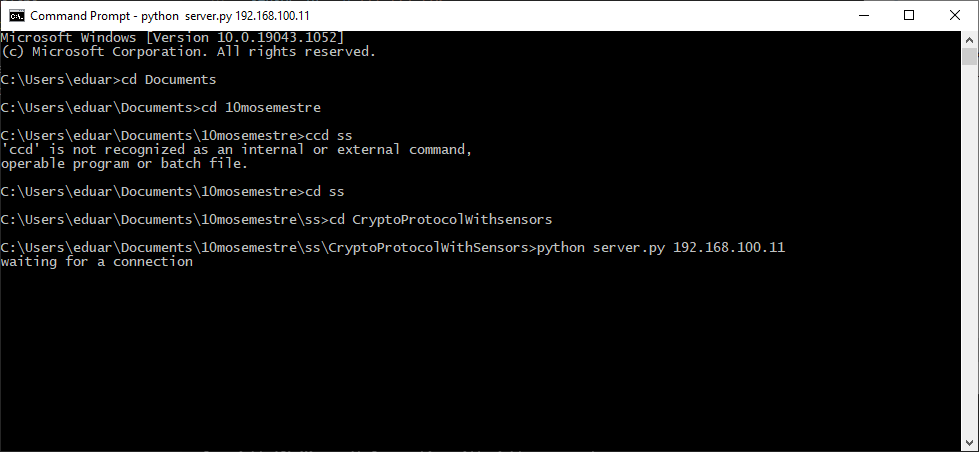
En la línea 490 del código destinado al nodo coordinador B, se tiene un sleep de .5 para prevenir fallas a la hora de la recepción de documentos. El problema de esta solución es que aumenta el tiempo de ejecución, por tal razón se recomienda que dependiendo el escenario en el que se use este sistema se realicen pruebas para que este tiempo sea el mínimo.

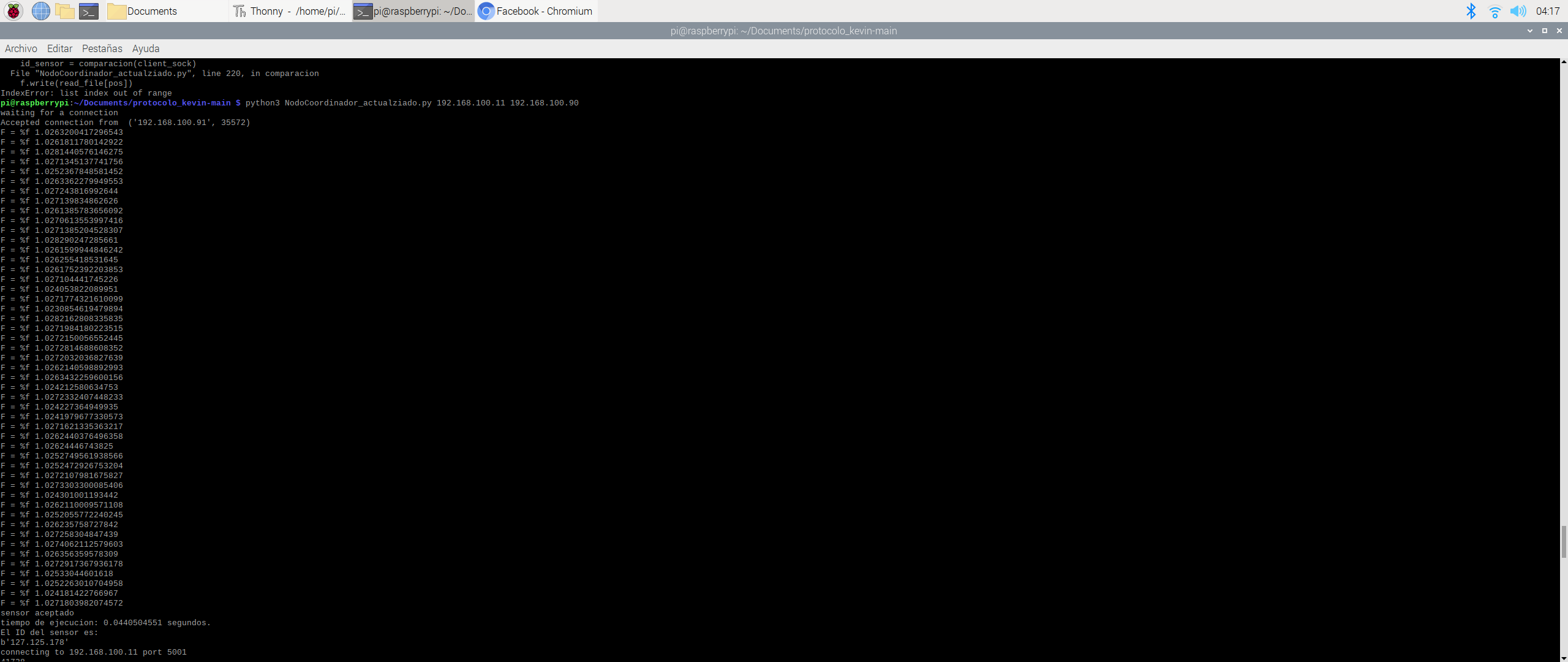
**Pruebas realizadas**

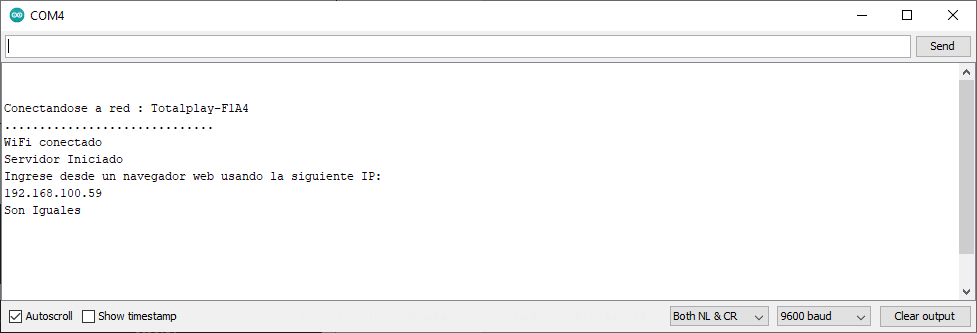
Para estas pruebas se hizo uso de 4 dispositivos todos conectados en la misma red mediante Wifi, existe una distancia aproximada de 10 m con paredes entre los dispositivos y el modem.

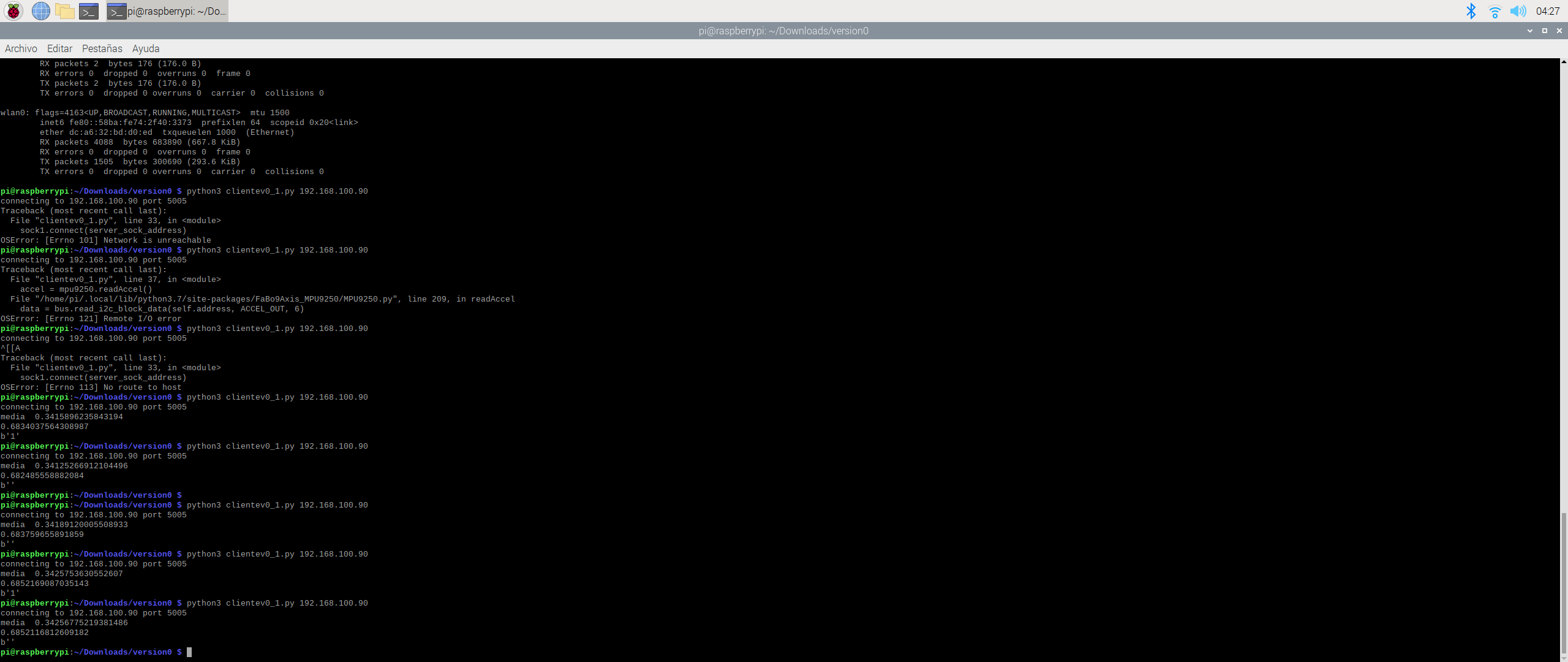
* El servidor A es un sistema Ubuntu 20.04 con Python 3.8 su dirección IP es 192.168.100.11.
* El nodo Coordinador B es una Raspberry Pi 4 B con Raspbian y Python 3.7, además tiene conectado un sensor MPU-9250 y su dirección IP es 192.168.100.90.
* El sensor MPU-9250 que solicita autenticación esta conectado a una Raspberry Pi 4 B con Raspbian, Python 3.7 y su dirección IP es 192.168.100.91.
* Como Actuador D se tiene una placa de desarrollo NodeMcu (ESP8266) con dirección IP 192.168.100.59.

Se realizaron 2 pruebas una donde el sensor es rechazado y otra donde es aceptado.

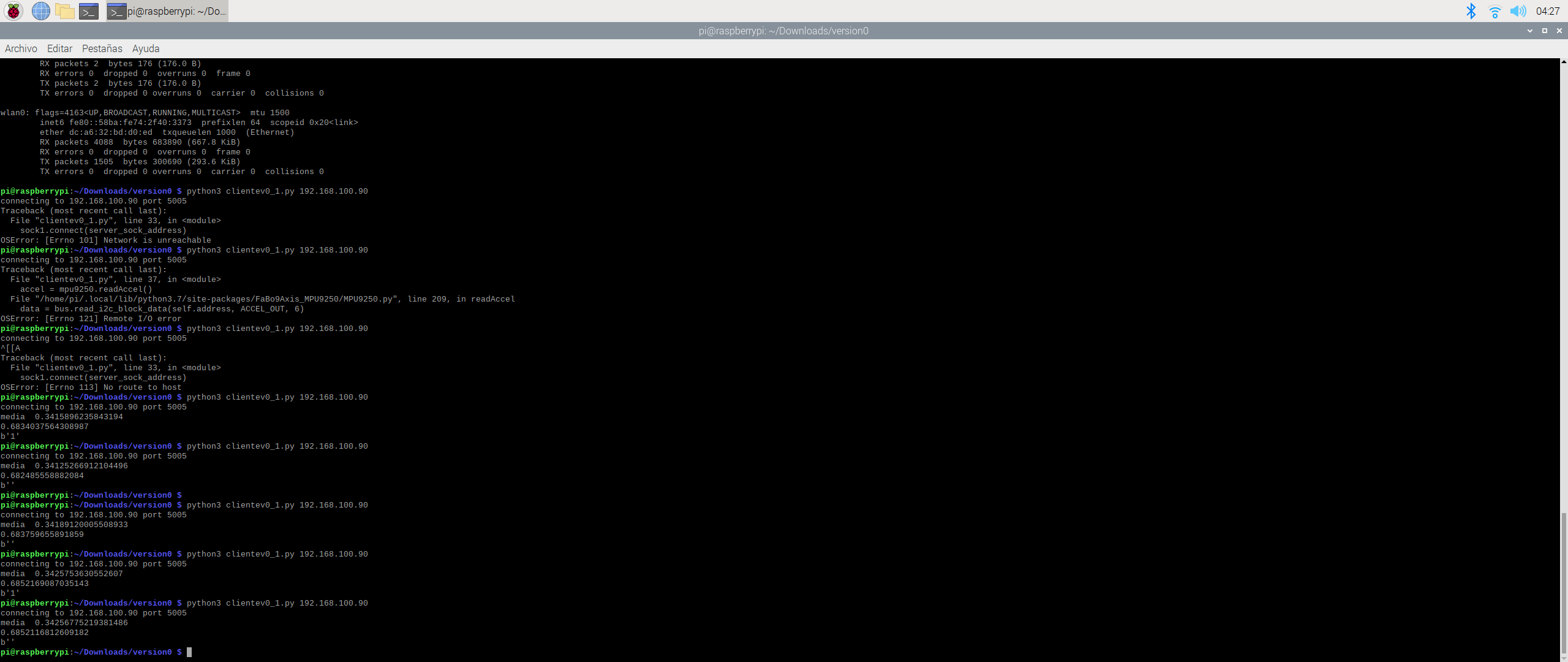
El primer paso fue correr el Servidor A en su respectiva dirección IP.

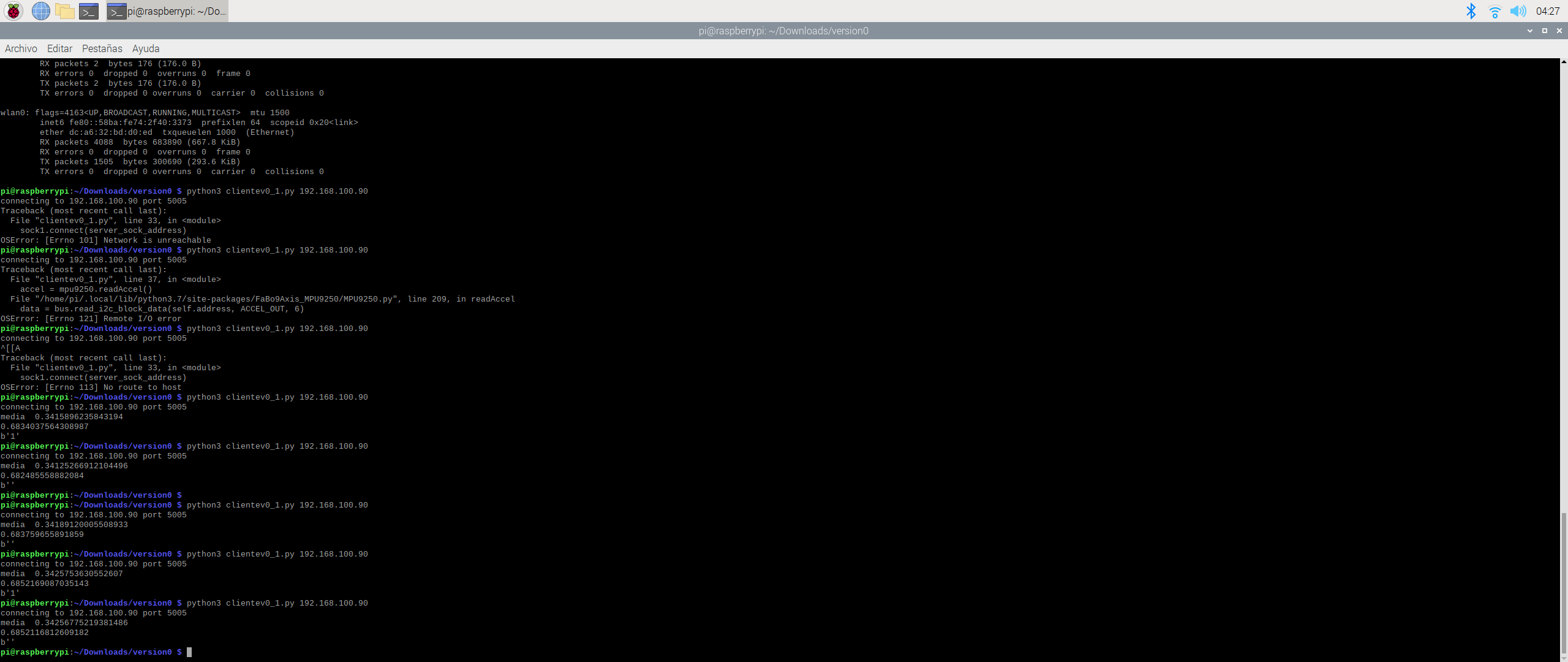
Después se corrió él Nodo Coordinador B en su dirección IP y apuntando a la dirección IP del Servidor A.

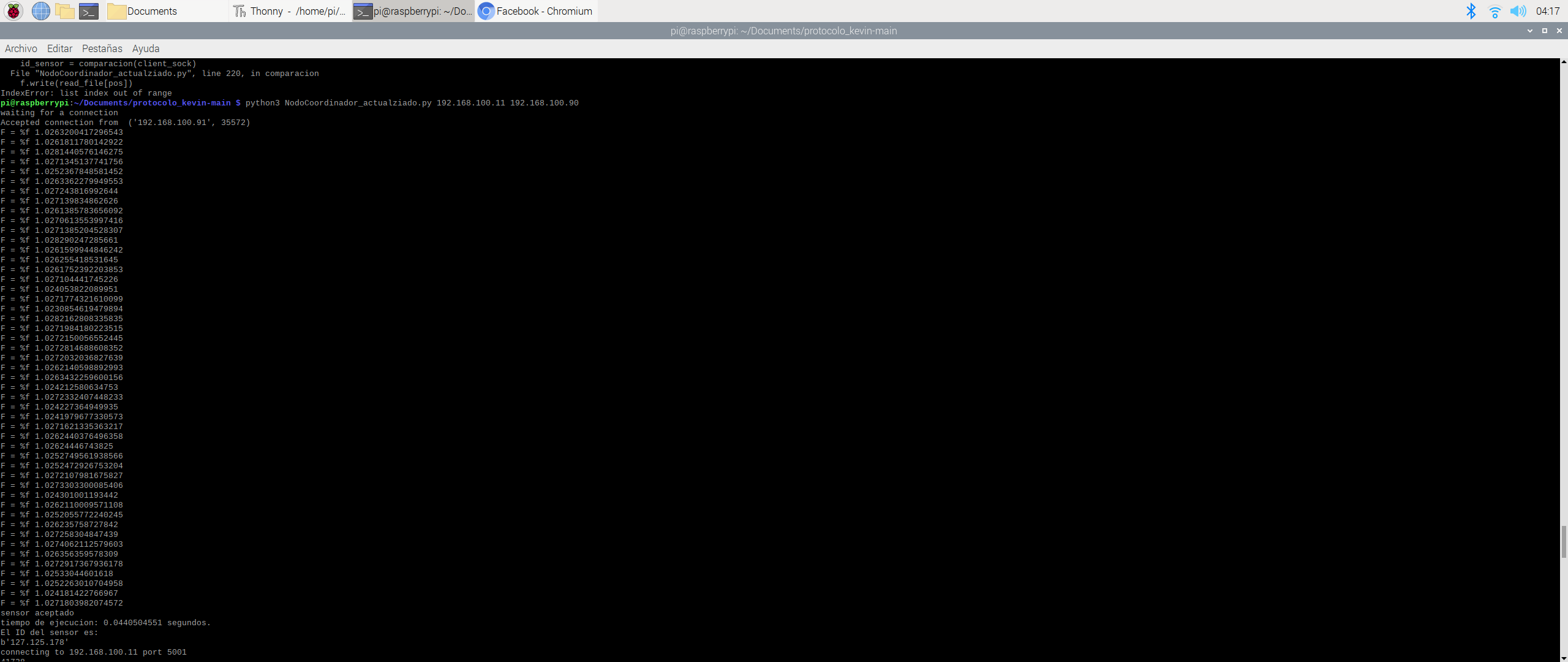
En conjunto se corre el Actuador D.

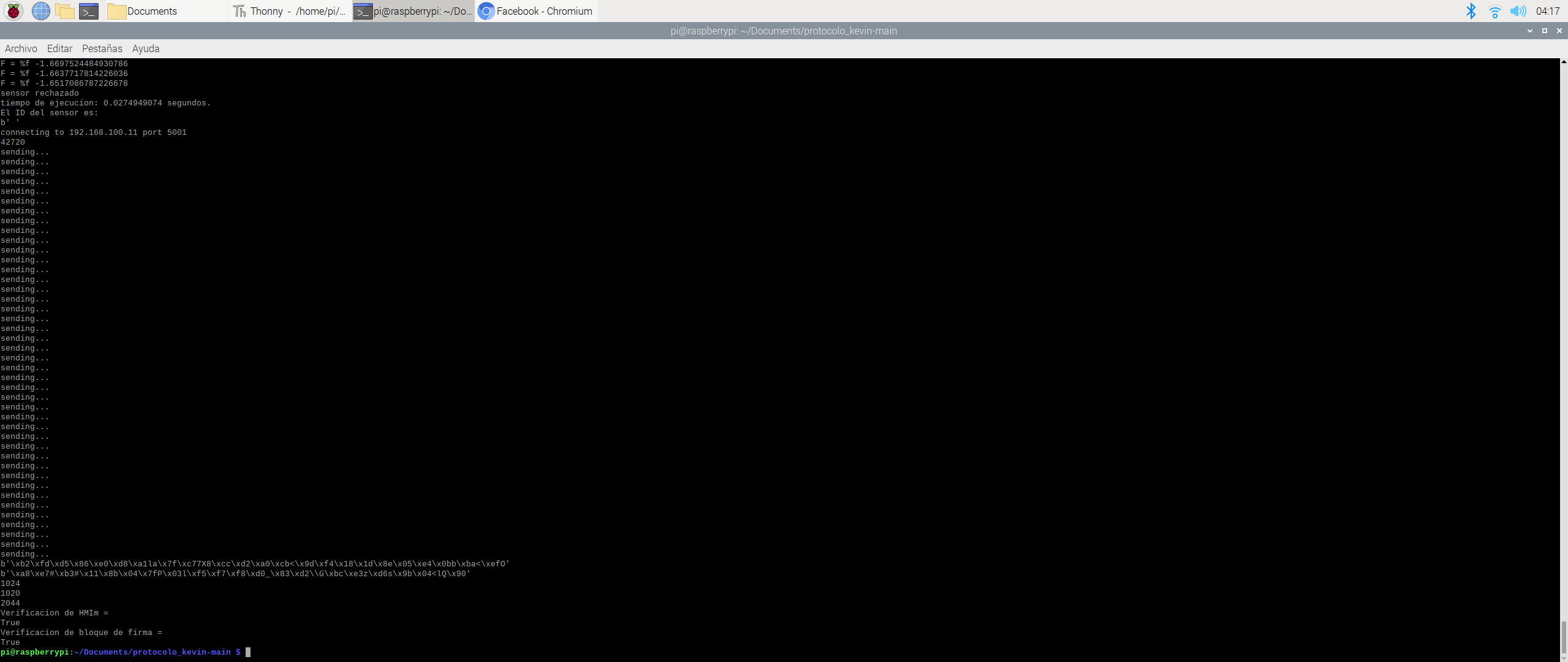
Y por último se solicita la autenticación de un sensor.

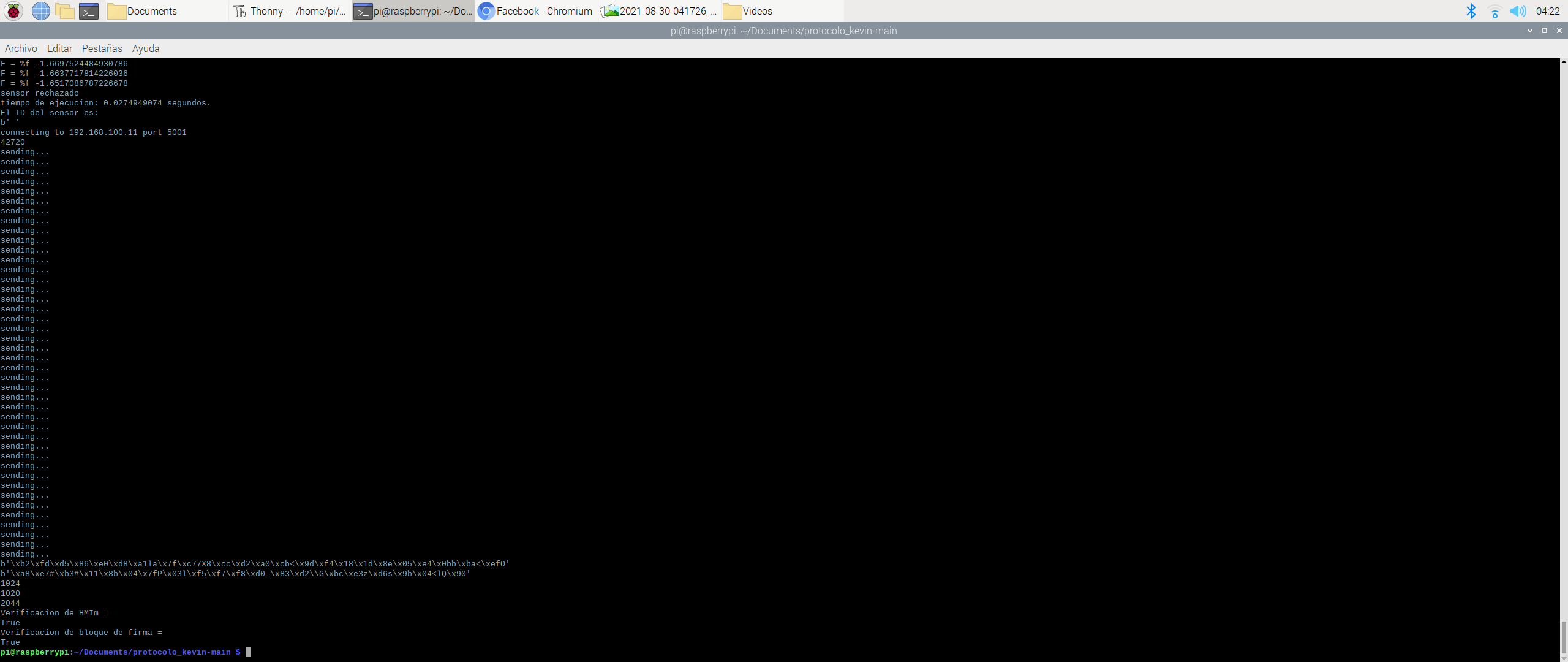
Aquí suceden 2 casos el sensor puede ser rechazado o aceptado, en caso de ser aceptado este mandará su ID, en caso contrario se desconectará automáticamente.

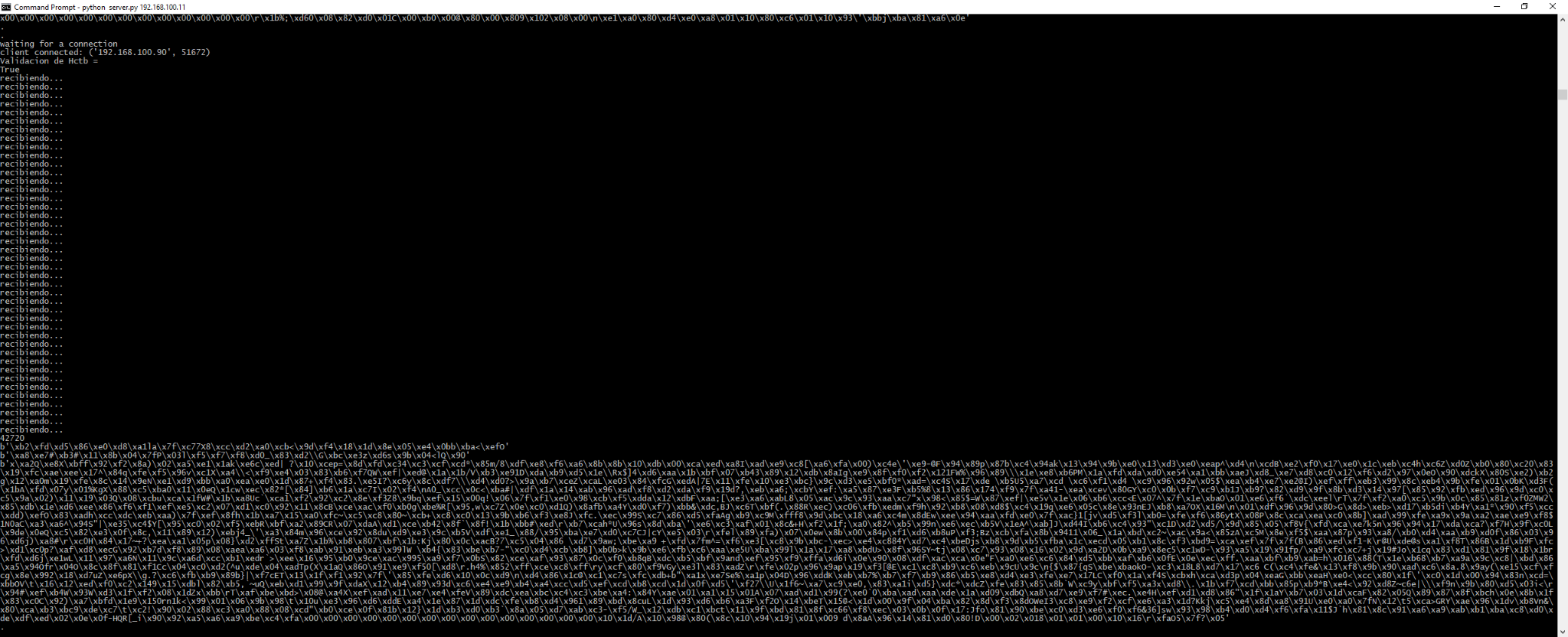
Si el sensor es aceptado recibe una confirmación con un 1:

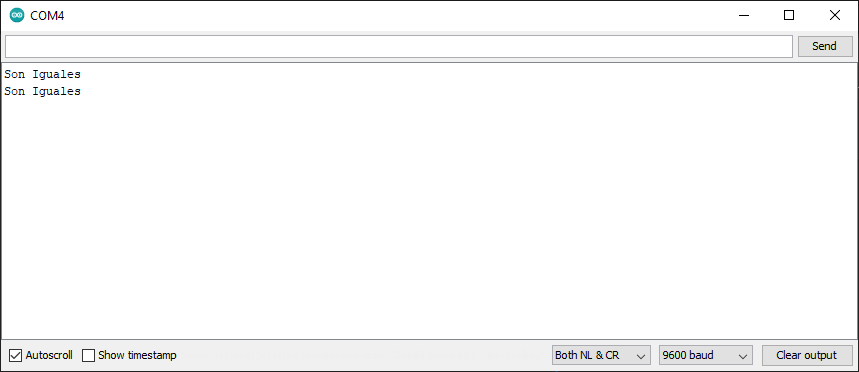
En caso contrario recibe un empty:

Del lado del Nodo Coordinador B en caso de ser aceptado se tiene lo siguiente:

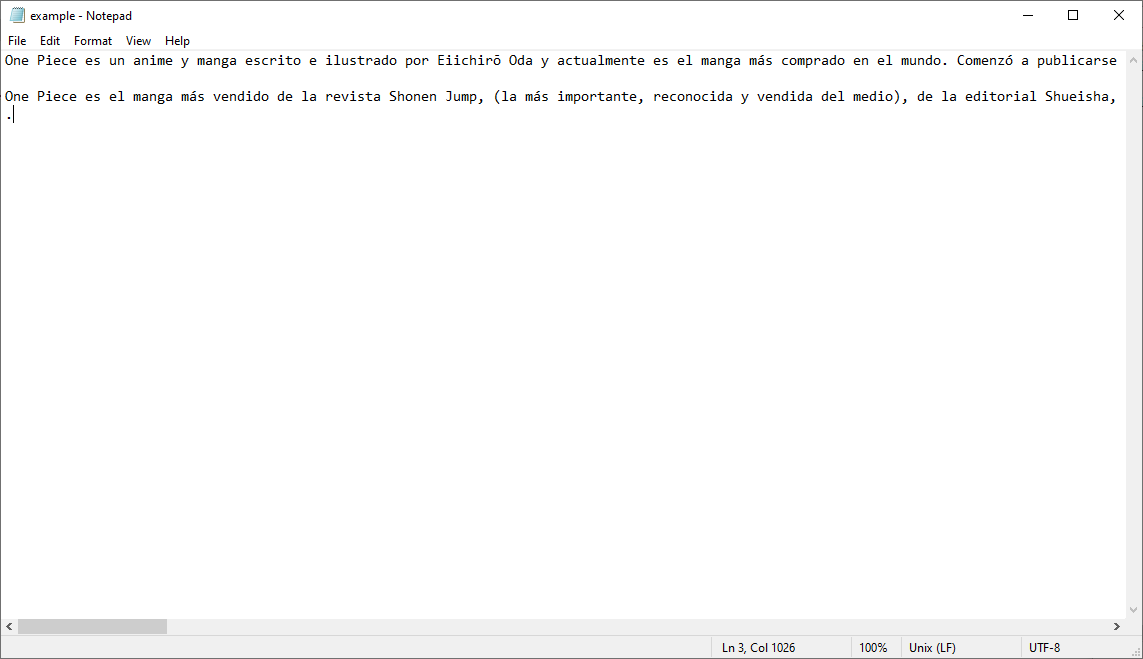
****En caso de ser rechazado el resultado es:

****Posteriormente el Nodo Coordinador hace el intercambio de claves, las validaciones correspondientes y el envío de archivo entre él y el Servidor.

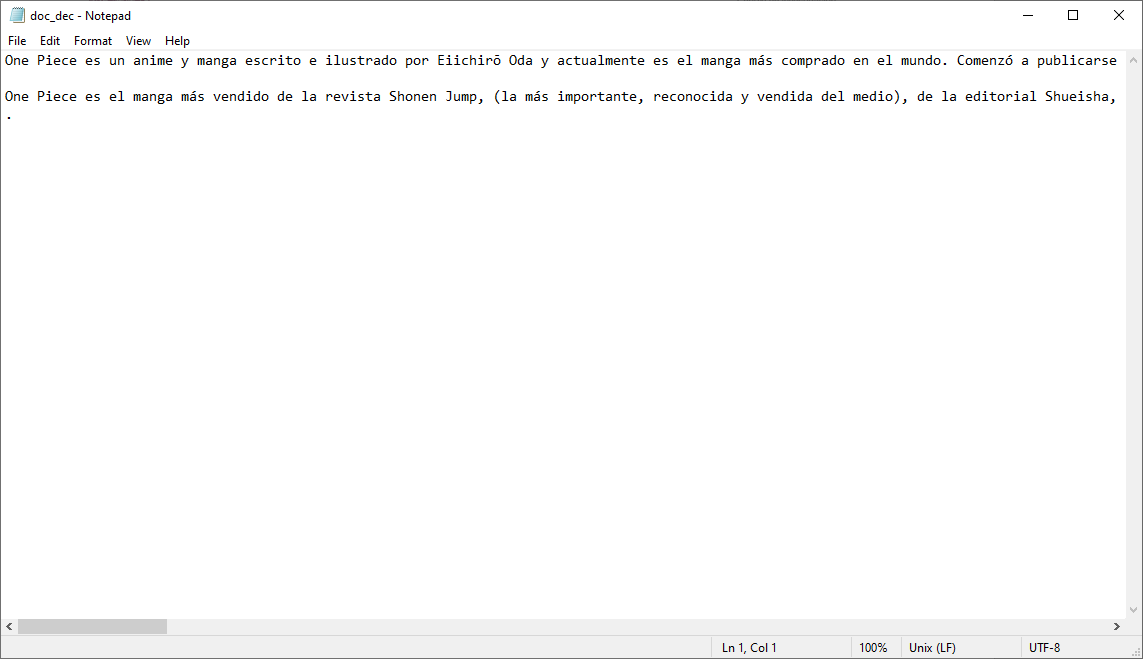
****De parte del servidor se obtiene lo siguiente:

Por último, el Actuador D hace su validación y lanza su mensaje correspondiente

Por parte del archivo encriptado enviado a continuación, se puede observar que es el mismo en el nodo coordinador B y en el Servidor A.

Archivo enviado:

Archivo recuperado:



**Códigos**

* Servidor A (Python)

|  |
| --- |
| '''  Para corre el siguiente programa debe usarse versiones posteriores a python 3  y al llamarse debe incluirse un argurmento, dicho argumento es la direccion IP  dodne esta montado el servidorr.  Ejemplo:  "python3 server.py 192.168.100.11"  El siguiente programa sera corrido en el Servidor A  Se deben descargar las bibliotecas pycryptodome y pqcrypto  '''  import socket  import sys  import os  import pqcrypto  import time  from Crypto.Cipher import AES  from Crypto.Hash import SHA256  from secrets import compare\_digest  from pqcrypto.sign.dilithium2 import generate\_keypair, sign, verify  from Crypto.Protocol.KDF import scrypt  from Crypto.Random import get\_random\_bytes  '''  Nombre:     desencriptar  Descripcion: Desencripta un arcvhivo .enc en el original, se debe agregar la terminacion del archivo que se desea recuperar               Ejemplo: Si el archivo original es un txt, se debe ingresar un archivo a la direccion de salida del mismo tipo txt.  Argumentos      --dire: Direccion donde se encuentra el archivo .enc                  --key: llave con la que se encripto el archivo                  --iv: vector iv que se utilizo en la encriptacion                  --dirout: direccion del archivo de salida que se dese obtener  '''  def desencriptar(dire, key, iv, dirout):      archivo = open(dire, "rb")      tam = os.path.getsize(dire)      encriptador = AES.new(key, AES.MODE\_CBC, iv)      archivo\_desencriptado = open(dirout, 'wb')      print(tam)      while True:          data = archivo.read(256)          #print(data)          n = len(data)          if n == 0:              break          #print(n)          decd = encriptador.decrypt(data)          n = len(decd)          if tam > n:              archivo\_desencriptado.write(decd)          else:              archivo\_desencriptado.write(decd[:tam])          tam -= n      archivo.close()      archivo\_desencriptado.close()  '''  Nombre desencapsulado  Descripcion: Desencapsula el Ct obtenido desde del cliente, genera el par de llave con kyber y  realiza la validacion del hash creado co nel Ct y el ID del cliente  Argumentos:     --id ingresa el ID del cliente  Returns:        --regresa la llave generada despues de pasarla por un KDF                  --regresa los primeros 16 digitos del hasheo generado a traves del ID y el ct  '''  def desencapsulado(id):      #declaramos la libreria de kyber      from pqcrypto.kem.kyber512 import generate\_keypair, encrypt, decrypt      #generamos la llave publica y secreta      public\_key, secret\_key = generate\_keypair()      #enviamos al cliente la llave publica      connection.sendall(public\_key)      #recibimos el Ct del cliente      ciphertext = connection.recv(768)      #recibimos el hash generado en el cliente      ct\_hash = connection.recv(256)      #Se concatena el Ct con el ID del cliente      ct\_hash\_esperado = ciphertext + id      #declaramos la funcion hash y hasheamos el Ct con el ID      hash = SHA256.new()      hash.update(ct\_hash\_esperado)      ct\_hash\_esperado = hash.digest()      #Se valida si el hash recibido es igual al hash generado en estta funcion      print('Validacion de Hctb =')      print(compare\_digest(ct\_hash,ct\_hash\_esperado))      #se desencapsula el Ct con la Sk y obtenemos el Pt      plaintext\_recovered = decrypt(secret\_key, ciphertext)      #connection.sendall(plaintext\_recovered)      #Se realiza la funcion KDF con el Pt y hash      key = scrypt(plaintext\_recovered, ct\_hash, 16, N=2\*\*14, r=8, p=1)      return key, ct\_hash[0:16]  # Create a TCP/IP socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # Inicializamos el socket y lo ponemos en Bind  HOST = sys.argv[1]  PORT = 5001  server\_address = (HOST, PORT)  sock.bind(server\_address)  sock.listen(1)  #Declaramos las rutas de los archivos  route\_in='doc\_enc2.enc'  route\_out='doc\_dec.txt'  #Estos valores son utlizados para recibir el documento  CHUNK\_SIZE = 1024  len\_chunk = CHUNK\_SIZE  #Se declaran los ID  id\_b = b'12.34.56.78'  id\_d = b'87.65.43.21'  #Se declara el mensaje que es utilizado en la fase 2  mi = get\_random\_bytes(32)  while True:      print('waiting for a connection')      connection, client\_address = sock.accept()      amount\_expected = 1088      amount\_received = 0      try:          #Imprimimos que se ha conectado el cliente          print('client connected:', client\_address)          #FASE1          key, iv= desencapsulado(id\_b)          #FASE2          #Generamos la Pk para la firma con Dilithium          public\_key\_sign, secret\_key\_sign = generate\_keypair()          #Enviamos al Cliente la llave publica          connection.sendall(public\_key\_sign)          #Recibimos el documento enmcriptado enviado por el cliente          CHUNK\_SIZE = 1024          len\_chunk = CHUNK\_SIZE          with open(route\_in, "wb") as f:              chunk = connection.recv(CHUNK\_SIZE)              while chunk:                  print("recibiendo...")                  f.write(chunk)                  if len\_chunk < CHUNK\_SIZE:                      break                  else:                      chunk = connection.recv(CHUNK\_SIZE)                      len\_chunk = len(chunk)          f.close()          #Desencriptamos el documento recibido          desencriptar(route\_in, key, iv, route\_out)          #Tomamos el mensaje recibido          f = open(route\_in, 'rb')          mensaje = f.read()          f.close()          #Creamos el bloque de firma          signature = sign(secret\_key\_sign,mi)          #Concatenamos el ID\_C, el bloque de firma, el mensaje delcarado anteriormente y el mensaje          msg\_hash = id\_d + signature + mi + mensaje          #Hasheamos el mensaje anteriormente concatenado          hash = SHA256.new()          hash.update(msg\_hash)          msg\_hash = hash.digest()          t=1          time.sleep(.5)          #enviamos el mensaje          #print(mi)          connection.send(mi)          #Enviamos el hasheo          #print(msg\_hash)          connection.send(msg\_hash)          #Enviamos el bloque de firma          #print(signature)          connection.send(signature[0:1024])          #print(".")          time.sleep(.5)          #print(".")          connection.send(signature[1024:2044])          #FINFASE2          #time.sleep(t)      finally:          connection.close() |

* Nodo Coordinador B (Python)

|  |
| --- |
| '''  Para correr el siguiente programa debe usarse versiones posteriores a python 3  y al llamarse debe incluirse dos argurmentos, el primer argumento es la  direccion IP a la que desea conectarse el nodo coordinador, en otras palabras la  direccion IP del servidor A. El segundo argumento es la direccion con la que  cuenta el nodo coordinador  Ejemplo:  "python3 server.py 192.168.100.11 192.168.100.90"  El siguiente programa sera corrido en el Nodo Coordinador B  Se deben descargar las bibliotecas pycryptodome, pqcrypto y FaBo9Axis\_MPU9250  '''  import socket  import sys, getopt  import functools  sys.path.append('.')  import os  import pqcrypto  import time  import requests  import datetime  import math  import operator  import FaBo9Axis\_MPU9250  import os.path  from Crypto.Cipher import AES  from Crypto.Hash import SHA256  from secrets import compare\_digest  #from pqcrypto.kem.kyber768 import generate\_keypair, encrypt, decrypt  from pqcrypto.sign.dilithium2 import generate\_keypair, sign, verify  from Crypto.Protocol.KDF import scrypt  from Crypto.Random import get\_random\_bytes    '''  Nombre corre\_imu  Descripcion:    Lee datos del sensor y recibe los obtenidos del sensor externo                  ademas de llamar a la funcion comparacion()  Argumentos:     --server\_sock es la informacion del socket  Returns:        --regresa el id del sensor  '''  def corre\_imu(server\_sock):      magnitud = 0.0      magnitudG = 0.0      media = 0.0      media\_rec = 0.0      d\_e = 0.0      suma = 0.0      ayuda = 0.0      magnitud\_t = 0.0      magnitudG\_t = 0.0      j=0      print('waiting for a connection')      client\_sock, client\_info = server\_sock.accept()      print("Accepted connection from ", client\_info)      mpu9250 = FaBo9Axis\_MPU9250.MPU9250()      while True:          tiempo=datetime.datetime.now()          accel = mpu9250.readAccel()          magnitud = math.sqrt(((accel['x'])\*\*2)+((accel['y'])\*\*2)+((accel['z'])\*\*2))          gyro = mpu9250.readGyro()          magnitudG = math.sqrt(((gyro['x'])\*\*2)+((gyro['y'])\*\*2)+((gyro['z'])\*\*2))          if j==0:              f=open('nuevo.txt','w')              f.write(str(magnitudG) + '\n' + str(magnitud))              f.close              j=j+1          else:              f=open('nuevo.txt','a')              f.write('\n' + str(magnitudG) + '\n')              f.write(str(magnitud))              f.close              j= j+1          time.sleep(0.01)          if j==200: break      start\_time = time.time()      j=0      f=open('nuevo.txt','r')      read\_file=f.readlines()      f.close      pos=0      pos1=1      aux=0      n = 0      m = 90      pos=1      apuntador = 0      apuntado = 1      ### obteniendo valor para covarianza de vectores      while  True:        for i in range (1,100):            magnitud\_t = magnitud\_t + float(read\_file[pos])            magnitudG\_t = magnitudG\_t + float(read\_file[n])            n=n+2            pos=pos+2            #print(n)        med=magnitud\_t/100        medG = magnitudG\_t/100        sumdes=0.0        sumdesG=0.0        n = 0        pos = 1        for i in range (1,100):            sumdes= sumdes + (float(read\_file[pos])-med)\*\*2            sumdesG= sumdesG + (float(read\_file[n])-med)\*\*2            n=n+2            pos=pos+2        des = math.sqrt(sumdes/100)        desG = math.sqrt(sumdesG/100)        aux = 0.0        auxG = 0.0        pos=1        n=0        for i in range(1,100):            aux = aux + ((float(read\_file[pos])-med)/des)            auxG = auxG + ((float(read\_file[pos])-medG)/desG)            n=n+2            pos=pos+2        cov = aux/99        covG = auxG/99        if m == 140:            #print(m)            break        if apuntador == 0:            f=open('giroscopio.txt','w')            f.write(str(covG) + '\n')            f.close            f=open('acelerometro.txt','w')            f.write(str(cov) + '\n')            f.close            magnitudG\_t = 0.0            magnitud\_t = 0.0            apuntador = apuntador + 2            apuntado = apuntado + 2            n=apuntador            pos=apuntado        else:            f=open('giroscopio.txt','a')            f.write(str(covG) + '\n')            f.close            f=open('acelerometro.txt','a')            f.write(str(cov) + '\n')            f.close            magnitudG\_t = 0.0            magnitud\_t = 0.0            apuntador = apuntador + 2            apuntado = apuntado + 2            n=apuntador            pos=apuntado        m=m+1      try:          f=open('acc\_ext.txt','w')          f.write("")          f.close()          f=open('giro\_ext.txt','w')          f.write("")          f.close()          CHUNK\_SIZE = 1024          len\_chunk = CHUNK\_SIZE          with open('conexion1.txt', "wb") as f:              chunk = client\_sock.recv(CHUNK\_SIZE)              while chunk:                  f.write(chunk)                  if len\_chunk < CHUNK\_SIZE:                      break                  else:                      chunk = client\_sock.recv(CHUNK\_SIZE)                      len\_chunk = len(chunk)          f.close()          id\_sensor = comparacion(client\_sock)      except IOError:          pass      finally:          client\_sock.close()          #server\_sock.close()      return id\_sensor    '''  Nombre comparacion  Descripcion:    Realiza la comparacion de los datos obtenidos y recibidos para                  posteriormente identificar si pertenece a la red o no, ademas de                  enviarle una confirmacion al sensor de que pertenece a la red                  y posteriormente recibir su ID  Argumentos:     --client\_sock es la informacion del socket del sensor  Returns:        --regresa el id del sensor  '''  def comparacion(client\_sock):      start\_time = time.time()      f=open('conexion1.txt','r')      read\_file=f.readlines()      f.close      pos=0      pos1=1      for i in range(0,49):          f=open('giro\_ext.txt','a')          f.write(read\_file[pos])          f.close          pos=pos+2          f=open('acc\_ext.txt','a')          f.write(read\_file[pos1])          f.close          pos1=pos1+2          #j=j+1      media = 0.0      media\_rec = 0.0      d\_e = 0.0      d\_e\_r = 0.0      suma = 0.0      ayuda = 0.0      ayuda\_rec = 0.0      j = 100      suma\_rec = 0.0      auxiliar = 0.0      cov=0.0      mediaG = 0.0      media\_recG = 0.0      d\_eG = 0.0      d\_e\_rG = 0.0      sumaG = 0.0      ayudaG = 0.0      ayuda\_recG = 0.0      j = 100      suma\_recG = 0.0      auxiliarG = 0.0      covG=0.0      f=open('giroscopio.txt','r')      read\_file = f.readlines()      f.close      g=open('giro\_ext.txt', 'r')      read\_files = g.readlines()      g.close      f=open('acelerometro.txt','r')      read\_fil = f.readlines()      g=open('acc\_ext.txt', 'r')      read\_fils = g.readlines()      g.close      pos=0      pos1=0      tiempo=datetime.datetime.now()         #####covarianza y fusion      f=open('resultados.txt','a')      f.write(str(tiempo) + '\n')      g = open('ayuda.txt', 'a')      for i in range(1,50):          auxiliar= ((float(read\_fil[pos]))\*(float(read\_fils[pos])))          pos=pos+1          auxiliarG= ((float(read\_file[pos1]))\*(float(read\_files[pos1])))          pos1=pos1+1          cov = auxiliar          covG = auxiliarG          fusion=0.0          if cov >= 0.2:              fusion = cov          else:              fusion=(cov+covG)/2          print ('F = %f', fusion)          f.write(str(fusion) + '\n')          g.write(str(fusion) + '\n')          suma = suma + fusion      f.close()      g.close()      media = suma/50      f = open('ayuda.txt','r')      read = f.readlines()      f.close()      for i in range (0,49):          ayuda = ayuda + ((float(read[i]) - media )\*\*2)      d\_e = math.sqrt(ayuda/49)      t = (media - 0.2)/(math.sqrt(d\_e/50))      if t>= 1.296:          print ("sensor aceptado")          confirmacion = b'1'          client\_sock.send(confirmacion)          id\_sensor = client\_sock.recv(1024)      else:          print ("sensor rechazado")          id\_sensor = b' '      elapsed\_time = time.time() - start\_time      print("tiempo de ejecucion: %.10f segundos." %elapsed\_time)      return id\_sensor    '''  Nombre:     encriptar  Descripcion: encripta cualquier tipo de arcvhivo y regresa un archivo .enc  Argumentos      --dire: Direccion donde se encuentra el archivo a ecriptar                  --key: llave con la que se desea encriptar el archivo                  --iv: vector iv que se utilizo en la encriptacion                  --dirout: direccion del archivo de salida .enc que se obtendra  '''  def encriptar(dire, key, iv, dirout):      encriptador = AES.new(key, AES.MODE\_CBC, iv)      archivo = open(dire, "rb")      archivo\_encriptado = open(dirout, "wb")      while True:          data = archivo.read(16)          n = len(data)          if n == 0:              break          elif n % 16 != 0:              data += b' ' \* (16 - n % 16)          enc = encriptador.encrypt(data)          archivo\_encriptado.write(enc)      archivo.close()      archivo\_encriptado.close()  '''  Nombre:     desencriptar  Descripcion: Desencripta un arcvhivo .enc en el original, se debe agregar la terminacion del archivo que se desea recuperar               Ejemplo: Si el archivo original es un txt, se debe ingresar un archivo a la direccion de salida del mismo tipo txt.  Argumentos      --dire: Direccion donde se encuentra el archivo .enc                  --key: llave con la que se encripto el archivo                  --iv: vector iv que se utilizo en la encriptacion                  --dirout: direccion del archivo de salida que se dese obtener  '''  def desencriptar(dire, key, iv, dirout):      archivo = open(dire, "rb")      tam = os.path.getsize(dire)      encriptador = AES.new(key, AES.MODE\_CBC, iv)      archivo\_desencriptado = open(dirout, 'wb')      while True:          data = archivo.read(256)          #print(data)          n = len(data)          if n == 0:              break          decd = encriptador.decrypt(data)          n = len(decd)          if tam > n:              archivo\_desencriptado.write(decd)          else:              archivo\_desencriptado.write(decd[:tam])          tam -= n      archivo.close()      archivo\_desencriptado.close()  '''  Nombre encapsulado  Descripcion: Encapsula con Kyber obteniendo el Ct con la public key recibida desde el servidor,  Argumentos:     --id ingresa el ID del cliente  Returns:        --regresa la llave generada despues de pasarla por un KDF                  --regresa los primeros 16 digitos del hasheo generado a traves del ID y el ct  '''  def encapsulado(id):      #Importamos la libreria de kyber      from pqcrypto.kem.kyber512 import generate\_keypair, encrypt, decrypt      #recibimos la public key      public\_key = sock.recv(800)      #Se realiza el encapsulado y se obtenemos el CT y el PT      ciphertext, plaintext\_original = encrypt(public\_key)      #Enviamos el CT      sock.sendall(ciphertext)      #Concatenamos el CT con el ID del cliente      ct\_hash = ciphertext + id      #Realizamos el hash al CT con el ID      hash = SHA256.new()      hash.update(ct\_hash)      ct\_hash = hash.digest()      #Enviamos el hash      sock.sendall(ct\_hash)      #data = sock.recv(1024)      #print(len(data))      #print(compare\_digest(plaintext\_original, data))      #time.sleep(1)      #salt = b'1234567891123456'      #con una funcion KDF se obtiene una llave a traves del PT y el hash      key = scrypt(plaintext\_original, ct\_hash, 16, N=2\*\*14, r=8, p=1)      return key, ct\_hash[0:16]  # Create a TCP/IP socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # Connect the socket to the port on the server  # given by the caller  #este socket es para utilizar con el Servidor  HOST=sys.argv[1]  PORT=5001  server\_address = (HOST, PORT)  #print('connecting to {} port {}'.format(\*server\_address))  #sock.connect(server\_address)  #se declara un segundo socket para la autenticacion de sensores  #WARNING: la direccion debe ser diferente al anterior socket  start\_time = time.time()  server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  HOST\_SOCK = sys.argv[2]  PORT\_SOCK = 5005  server\_sock\_address = (HOST\_SOCK, PORT\_SOCK)  server\_sock.bind(server\_sock\_address)  server\_sock.listen(1)  #declaramos rutas a utilizar  route\_in = 'resultados.txt'  route\_out = 'doc\_enc.enc'  route\_out2 = 'doc\_dec\_2.txt'  CHUNK\_SIZE = 1024  #declaramos IDs  id\_b =b'12.34.56.78'  id\_d = b'87.65.43.21'  ip\_c = '192.168.100.59'  try:      #FASE 0      id\_sensor = corre\_imu(server\_sock)      print("El ID del sensor es:")      print(id\_sensor)      #FASE1      print('connecting to {} port {}'.format(\*server\_address))      sock.connect(server\_address)      amount\_expected = 1184      amount\_received = 0      key , iv= encapsulado(id\_b)      #FINFASE1      #FASE2      #Recibimos la Pk para la firma      public\_key\_sign=sock.recv(1184)      #Encriptamos el archivo a enviar      encriptar(route\_in,key,iv,route\_out)      #Enviamos el archivo      tam\_doc = os.path.getsize(route\_out)      print(tam\_doc)      #confirmacion = True      #while confirmacion:      with open(route\_out, 'rb') as f:          data = f.read(CHUNK\_SIZE)          while data:              print("sending...")              time.sleep(.5)              sock.send(data)              data = f.read(CHUNK\_SIZE)          #sock.sendfile(f,0,tam\_doc)          #print('documento enviado')      f.close()       #   lon\_rec = sock.recv(len(tam\_doc))        #  sock.send(tam\_doc)        #  print("Longitudes ")        #  print(lon\_rec)         # print(tam\_doc)         # confirmacion = compare\_digest(lon\_rec,tam\_doc)      #Leemos el mensaje del archivo      f = open(route\_out, 'rb')      mensaje = f.read()      f.close()      #recibimos el mensaje unico      mi = sock.recv(32)      print(mi)      #recibimos el mensaje hasheado      msg\_hash = sock.recv(32)      print(msg\_hash)      #recibimos el bloque de firma      signature2 = sock.recv(1024)      signature3 = sock.recv(1024)      signature = signature2 + signature3      print(len(signature2))      print(len(signature3))      print(len(signature))      msg\_hash\_esperado = id\_d + signature + mi + mensaje      #print(msg\_hash\_esperado)      hash = SHA256.new()      hash.update(msg\_hash\_esperado)      msg\_hash\_esperado = hash.digest()      #print(msg\_hash\_esperado)      #verificamos que el hasheo que recibimos sea el mismo      print("Verificacion de HMIm =")      print(compare\_digest(msg\_hash\_esperado,msg\_hash))      #verificamos quie la firma sea autentica      print("Verificacion de bloque de firma = ")      print(verify(public\_key\_sign,mi,signature))      #FINFASE2      #INICIOFASE3      #concatenamos el idd con mi      msg\_hash\_mi=id\_d+mi      #hasheamso el mensaje concatenado      hash = SHA256.new()      hash.update(msg\_hash\_mi)      msg\_hash\_mi = hash.digest()      #adjuntamos los datos que mandaremos al arduino      data = {'mi':mi,'hash\_mi':msg\_hash\_mi}      #      direccion = 'http://192.168.100.59/hash'      #hacemos una peticioon para que el arduino haga la validacion      requests.post(direccion,data)      #FINFASE3  finally:      sock.close() |

* Sensores (Python)

|  |
| --- |
| import sys, getopt  sys.path.append('.')  import FaBo9Axis\_MPU9250  import os.path  import time  import datetime  import math  import operator  import socket  from numpy import \*  import os  #import paho.mqtt.client as mqttClient  import json  import random  sensor\_id = b'127.125.178'  magnitud=0.0  magnitudG = 0.0  media=0.0  d\_e=0.0  suma=0.0  magnitud\_t = 0.0  magnitudG\_t = 0.0  j = 0  mpu9250 = FaBo9Axis\_MPU9250.MPU9250()  sock1 = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  HOST\_SOCK = sys.argv[1]  PORT\_SOCK = 5005  server\_sock\_address = (HOST\_SOCK, PORT\_SOCK)  print('connecting to {} port {}'.format(\*server\_sock\_address))  sock1.connect(server\_sock\_address)  while True:      accel = mpu9250.readAccel()      magnitud = math.sqrt(((accel['x'])\*\*2)+((accel['y'])\*\*2)+((accel['z'])\*\*2))      gyro = mpu9250.readGyro()      magnitudg = math.sqrt(((gyro['x'])\*\*2)+((gyro['y'])\*\*2)+((gyro['z'])\*\*2))      if j==0:              f = open('magnitudes.txt','w')              f.write(str(magnitudg) + '\n' + str(magnitud))              f.close              j = j + 1      else:              f = open('magnitudes.txt','a')              f.write('\n'+str(magnitudg) + '\n' + str(magnitud))              f.close              j = j + 1      time.sleep(0.01)      if j == 150: break  f = open('magnitudes.txt','r')  read\_file = f.readlines()  f.close  with open('magnitudes.txt', 'rb') as f:      sock1.sendfile(f)      #print('documento enviado')  f.close()  pos = 0  pos1 = 1  aux = 0  n = 0  m = 90  apuntador = 0  apuntado = 1  magnitud\_t = 0.0  magnitudg\_t = 0.0  while True:          for i in range (1,90):                  magnitud\_t = magnitud\_t + float(read\_file[pos1])                  magnitudg\_t = magnitudg\_t + float(read\_file[n])                  n = n + 2                  pos1 = pos1 + 2                  #print n          if m == 140:                  break          if apuntador == 0:                  f=open('giroscopio.txt','w')                  f.write(str(magnitudg\_t) + '\n')                  f.close                  f=open('acelerometro.txt','w')                  f.write(str(magnitud\_t) + '\n')                  f.close                  magnitudG\_t = 0.0                  magnitud\_t = 0.0                  apuntador = apuntador + 2                  apuntado = apuntado + 2                  n = apuntador                  pos1 = apuntado          else:                  f=open('giroscopio.txt','a')                  f.write(str(magnitudg\_t) + '\n')                  f.close                  f=open('acelerometro.txt','a')                  f.write(str(magnitud\_t) + '\n')                  f.close                  magnitudG\_t = 0.0                  magnitud\_t = 0.0                  apuntador = apuntador + 2                  apuntado = apuntado + 2                  n = apuntador                  pos1 = apuntado          m = m + 1  pos=0  pos1=1  for i in range(0, 50):      #sock1.send(read\_file[pos])      #time.sleep(0.01)      #sock1.send(read\_file[pos1])      suma=suma + float(read\_file[pos1])      pos=pos+1      pos1=pos1+2  media=suma/j  ayuda=0.0  pos1=1  for i in range(0,j):      ayuda= (float(read\_file[pos1])-media)\*\*2 + ayuda      pos1=pos1+2  d\_e=math.sqrt(ayuda/j)  print('media ', media)  print(d\_e)  data = sock1.recv(1)  print(data)  if data==b'1':          #os.system("python senseUbi.py")  #        print ('data es 1')          sock1.send(sensor\_id)  else:          sock1.close() |

* Actuador D (Arduino)

|  |
| --- |
| **#include <ESP8266WiFi.h>**  **#include <WiFiClient.h>**  **#include <ESP8266WebServer.h>**  **#include <ESP8266mDNS.h>**  **#include <ArduinoJson.h>**  **#include <TypeConversion.h>**  **#include <Crypto.h>**  **extern "C"{**  **#include "sign.h"**  **};**  **int ret;**  **namespace TypeCast = experimental::TypeConversion;**  **const char\* ssid = "Totalplay-F1A4";**  **const char\* password = "F1A4BB1AWMA8xJ5X";**  **//WiFiServer server(80);**  **ESP8266WebServer server(80);**  **const String postForms = "<html>\**  **<head>\**  **<title>ESP8266 Web Server POST Crypto</title>\**  **<style>\**  **body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color: #000088; }\**  **</style>\**  **</head>\**  **<body>\**  **<h1>POST plain text to /postplain/</h1><br>\**  **</body>\**  **</html>";**  **void setup() {**  **Serial.begin(9600);**  **delay(100);**  **//Configuración del GPIO2**  **pinMode(2, OUTPUT);**  **digitalWrite(2,LOW);**      **Serial.println();**  **Serial.println();**  **Serial.print("Conectandose a red : ");**  **Serial.println(ssid);**    **WiFi.begin(ssid, password); //Conexión a la red**    **while (WiFi.status() != WL\_CONNECTED)**  **{**  **delay(500);**  **Serial.print(".");**  **}**  **Serial.println("");**  **Serial.println("WiFi conectado");**  **server.on("/", handleRoot);**  **server.on("/hash",set\_hash);**  **//server.on("/pk",setPk);**  **server.begin(); //Iniciamos el servidor**  **Serial.println("Servidor Iniciado");**  **Serial.println("Ingrese desde un navegador web usando la siguiente IP:");**  **Serial.println(WiFi.localIP()); //Obtenemos la IP**  **}**  **void loop() {**    **server.handleClient();**  **}**  **/\***  **void setPk()**  **{**  **if (server.method() != HTTP\_POST) {**  **//digitalWrite(led, 1);**  **server.send(405, "text/plain", "Method Not Allowed");**  **//digitalWrite(led, 0);**  **} else {**  **//digitalWrite(led, 1);**  **unsigned char pk[CRYPTO\_PUBLICKEYBYTES];**  **char pk2[CRYPTO\_PUBLICKEYBYTES];**  **String pk\_r = server.arg(0);**  **String sm\_r = server.arg(1);**  **String m\_r = server.arg(2);**  **int i = pk\_r.length();**  **int i2 = sm\_r.length();**  **int MLEN = m\_r.length();**  **unsigned char m[MLEN];**  **unsigned char sm[MLEN + CRYPTO\_BYTES];**  **char m2[MLEN];**  **char sm2[MLEN + CRYPTO\_BYTES];**  **//pk\_r.toCharArray(pk2,i);**  **//sm\_r.toCharArray(sm2, i2);**  **//m\_r.toCharArray(m2, MLEN);**  **unsigned long long smlen = i2;**  **unsigned long long m\_len = MLEN;**  **//strcpy((char\*)pk,pk2);**  **//strcpy((char\*)sm,sm2);**  **//strcpy((char\*)m,m2);**  **for (int x= 0; x < i2; x++) {**  **sm[x]=sm\_r.charAt(x);**  **//Serial.println(pk[x]);**  **if(x<i){**  **pk[x] = pk\_r.charAt(x);**  **}**  **//if(x<MLEN){**  **//m[x]=m\_r.charAt(x);**  **//}**  **}**  **Serial.println(pk[0]);**  **Serial.println(sm[0]);**  **// Serial.println(m[0]);**  **Serial.println(pk[i-1]);**  **Serial.println(sm[i2-1]);**  **// Serial.println(m[MLEN-1]);**  **int ret;**  **ret = crypto\_sign\_open(m, &m\_len, sm, smlen, pk);**  **Serial.println("La verificacion es:");**  **Serial.println(ret);**  **server.send(200, "text/plain", "POST body was:\n" + server.arg("plain"));**  **//digitalWrite(led, 0);**  **}**  **}**  **\*/**  **//Esta funcion se encargara de recibir el hasheo y validarlo**  **void set\_hash(){**  **using namespace experimental::crypto;**  **uint8\_t resultArray[SHA256::NATURAL\_LENGTH] { 0 };**    **if (server.method() != HTTP\_POST) {**  **//digitalWrite(led, 1);**  **server.send(405, "text/plain", "Method Not Allowed");**  **//digitalWrite(led, 0);**  **} else {**  **//digitalWrite(led, 1);**  **String mi = server.arg(0);**  **String msg\_hash = server.arg(1);**  **String IDD = "87.65.43.21";**  **IDD.concat(mi);**  **SHA256::hash(IDD.c\_str(), IDD.length(), resultArray);**  **String stringOne = (char\*)resultArray;**  **String stringTwo = stringOne.substring(0,32);**  **if (stringTwo.equals(msg\_hash)) {**  **Serial.println("Son Iguales");**  **} else {**  **Serial.println("Son Diferentes");**  **}**  **server.send(200, "text/plain", "POST body was:\n" + server.arg("plain"));**  **//digitalWrite(led, 0);**  **}**  **}**  **void handleRoot() {**  **//digitalWrite(led, 1);**  **server.send(200, "text/html", postForms);**  **//digitalWrite(led, 0);**  **}**  **void handleNotFound() {**  **//igitalWrite(led, 1);**  **String message = "File Not Found\n\n";**  **message += "URI: ";**  **message += server.uri();**  **message += "\nMethod: ";**  **message += (server.method() == HTTP\_GET) ? "GET" : "POST";**  **message += "\nArguments: ";**  **message += server.args();**  **message += "\n";**  **for (uint8\_t i = 0; i < server.args(); i++) {**  **message += " " + server.argName(i) + ": " + server.arg(i) + "\n";**  **}**  **server.send(404, "text/plain", message);**  **//digitalWrite(led, 0);**  **}** |