**Introducción**

El siguiente documento tiene como propósito reportar lo realizado en la implementación del protocolo criptográfico seguro y eficiente para manejo de información médica en dispositivos inteligentes.

De igual forma desglosar detalladamente y explicar las decisiones tomadas a lo largo de este trabajo.

**Propuesta Inicial**

La propuesta inicial 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.

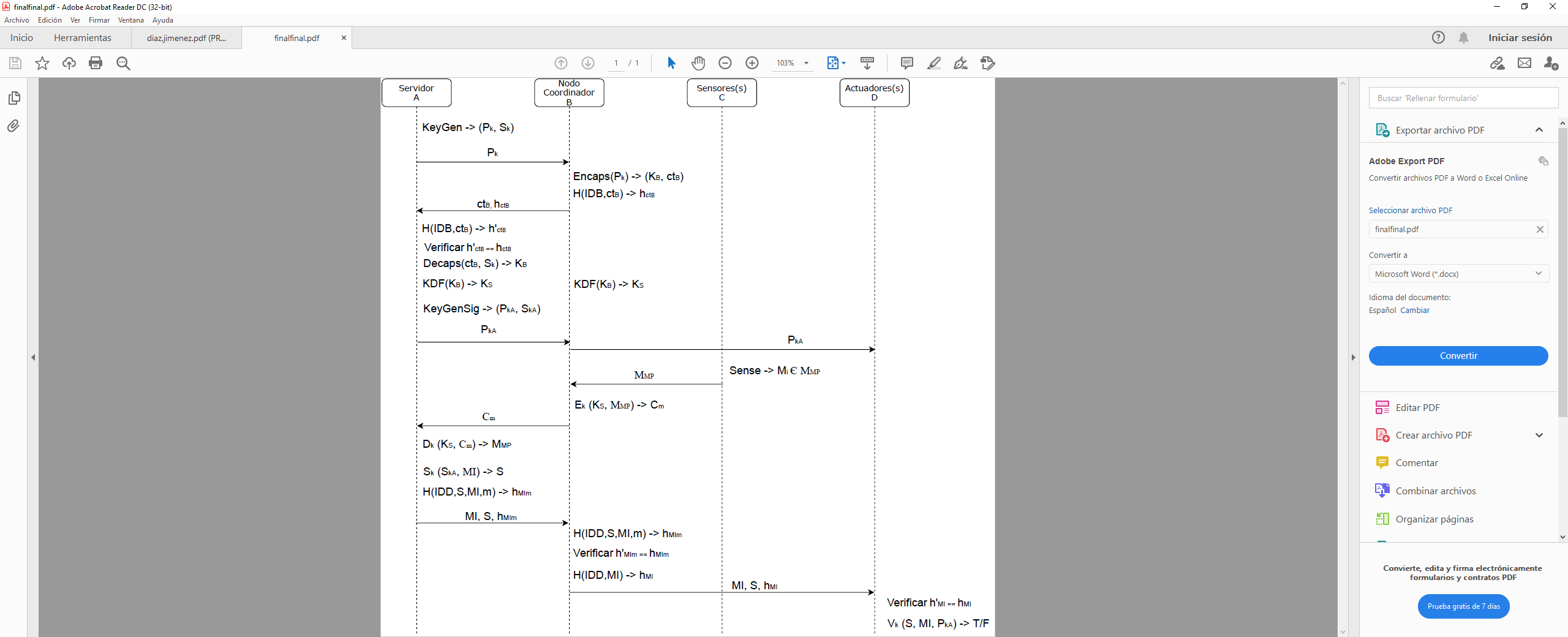
Este elemento será una Raspberry Pi 4, la cual actuará de intermediario entre el Servidor A y el Actuador D, además de recibir los datos de los sensores C.

* Sensor o Sensores C.

Los sensores C dependerán del uso médico.

* 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 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, esta decisión fue tomada para acelerar el tiempo de desarrollo.

Se hará uso de dos bibliotecas “pycryptodome” y “pqcrypto” así que es necesario descargar 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 modulo 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 realizará peticiones al servidor cada vez que lo necesite, además de hacer peticiones POST al 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 ya que así evitábamos abrir otro socket desde el nodo coordinar B y desde aquí, aunque internamente si se realiza, a la hora de realizar la petición se automatiza toda esta parte. haciendo más ligero el código e igual de eficiente.

* **Dificultades.**

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

**Código**

* Servidor A (python)

|  |
| --- |
| '''  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')      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 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 = '127.0.0.1'  PORT = 65432  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          with open(route\_in, "wb") as f:              chunk = connection.recv(CHUNK\_SIZE)              while chunk:                  f.write(chunk)                  if len\_chunk < CHUNK\_SIZE:                      break                  else:                      #print('recibiendo')                      #print(len\_chunk)                      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(t)          #enviamos el mensaje          connection.sendall(mi)          #Enviamos el bloque de firma          connection.sendall(signature)          #Enviamos el hasheo          connection.sendall(msg\_hash)          #FINFASE2          time.sleep(t)        finally:          connection.close() |

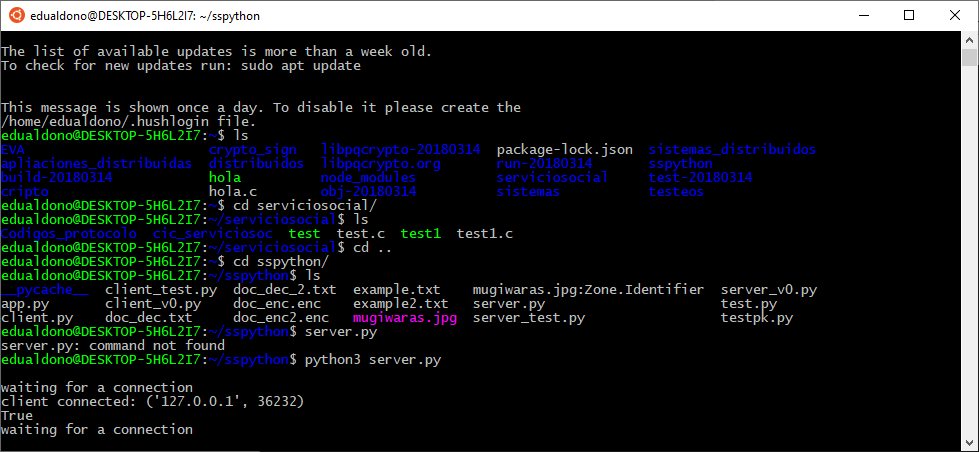
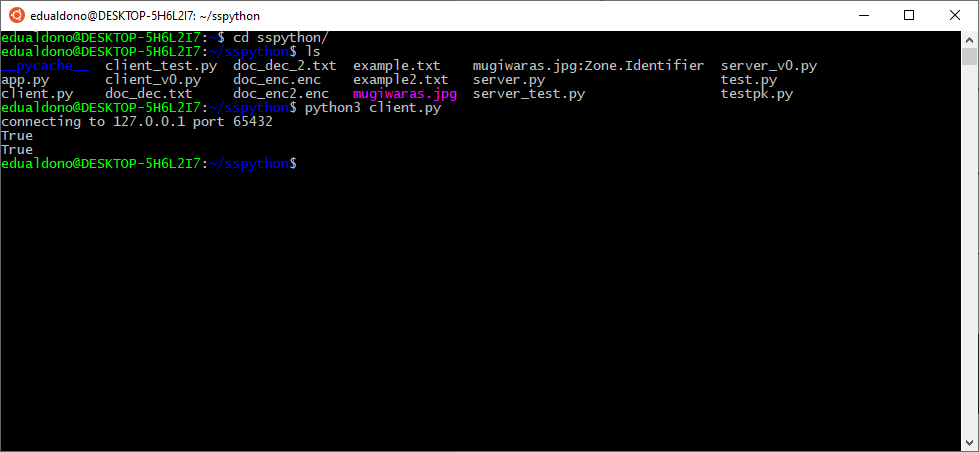
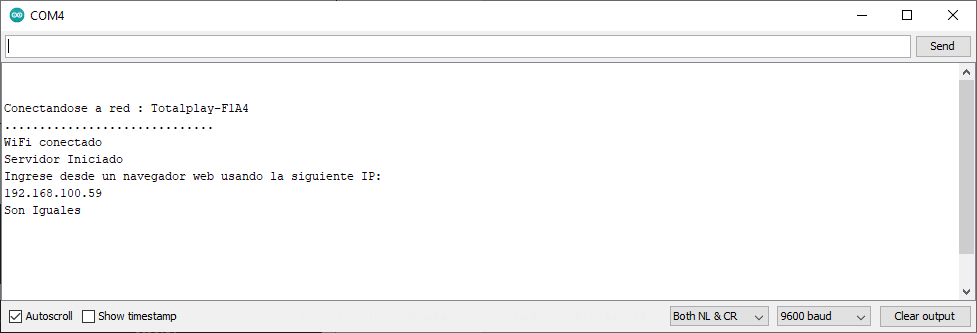
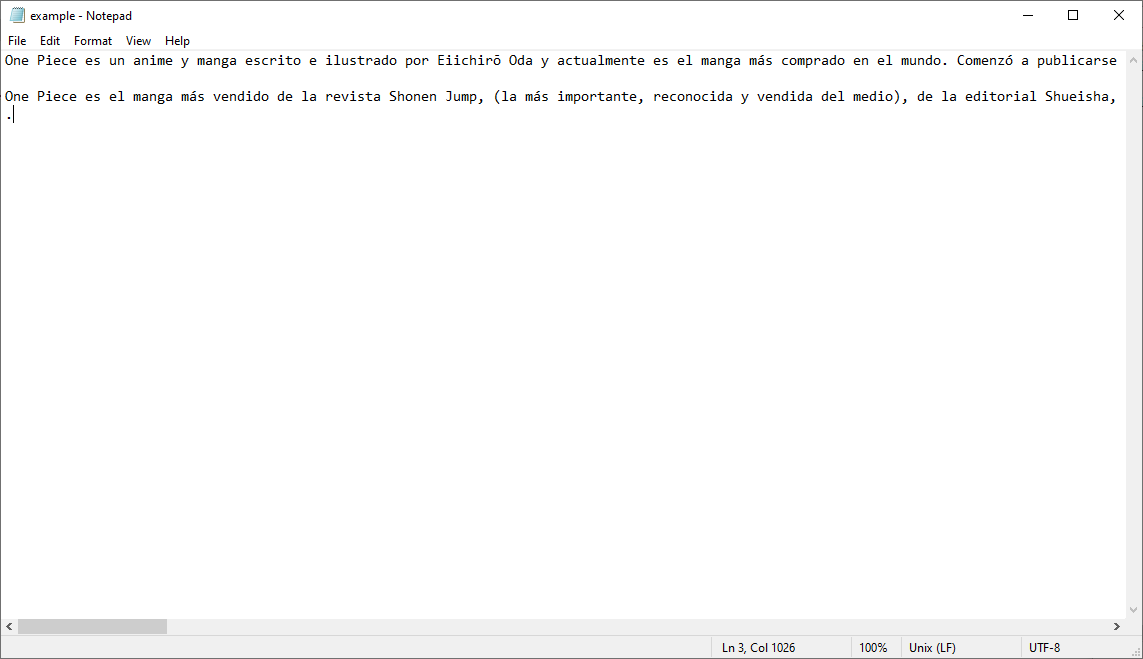
* Nodo Coordinador B (python)

|  |
| --- |
| '''  El siguiente programa sera corrido en el Nodo Coordinador B  Se deben descargar las bibliotecas pycryptodome y pqcrypto  '''  import socket  import sys  import os  import pqcrypto  import time  import requests  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:     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  server\_address = ('127.0.0.1', 65432)  print('connecting to {} port {}'.format(\*server\_address))  sock.connect(server\_address)  #declaramos rutas a utilizar  route\_in = 'example.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:      #FASE1      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      with open(route\_out, 'rb') as f:          sock.sendfile(f)          #print('documento enviado')      f.close()      #Leemos el mensaje del archivo      f = open(route\_out, 'rb')      mensaje = f.read()      f.close()      #recibimos el mensaje unico      mi = sock.recv(32)      #recibimos el bloque de firma      signature = sock.recv(2044)      #recibimos el mensaje hasheado      msg\_hash = sock.recv(256)      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()        #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() |

* 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);**  **}** |

**Resultados**

* Servidor A
* Nodo Coordinador B
* Actuador D
* Archivo Enviado
* Archivo Recuperado

