# Introduccion

El presente documento tiene como propósito reportar los cambios, actualizaciones, así como las pruebas realizadas sobre la implementación del protocolo criptográfico seguro para manejo de información médica en dispositivos inteligentes.

Las pruebas se realizaron bajo los dispositivos con las siguientes características:

* Servidor A: Desktop Fedora Linux 37
* Coordinador B y Sensores C: Raspberry pi model 3 B raspbian Bullseye
* Actuador D: Devkit ESP32

# Versiones de paquetes

A continuación se muestran las versiones utilizadas durante las pruebas para los distintos paquetes para python 3.9

* pycroptodome v 3.16.0
* pqcrytpo 0.1.3
* numpy 1.23.5
* keras 2.11.0
* Tensorflow 2.11.0
* FaBoaxis-MPU9250 1.0.0
* FaBo9Axis-MPU9250-Python3 1.0.3

Problemas

Debido a la integración pausada de paquetes a la rama principal de Debian y por consiguiente a Raspbian, es posible que existan problemas de compatibilidad al realizar la instalación de paquetes. En lo particular para la instalación del paquete pqcrypto se requiere forzar la actualización del paquete setuptools, nativo de la instalación de python 3.9. El cambio se da de la versión 50.2.0 a 62.6.0, aunque este problema es recurrente con distintos paquetes en actualizaciones de python, por lo que en caso de presentarse problemas es necesario con probar una versión actualizada del paquete.

Se puede forzar la actualización mediante el comando

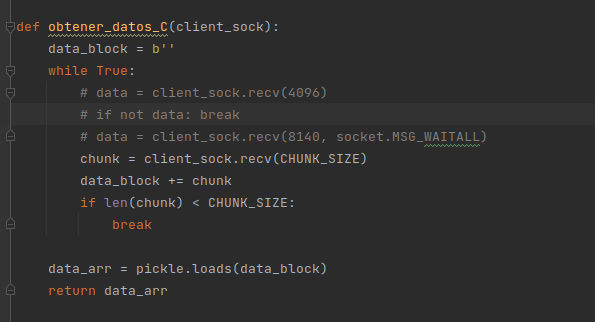
pip install setuptools==62.6.0

Como se muestra en el listado de paquetes, ya no es necesaria la edición del paquete FaBo9Axis-MPU9250 como se proponía en reportes anteriores, en su lugar se instala la rama del paquete para python3 que hace las modificaciones sintácticas necesarias sin alterar el funcionamiento.

# Modificaciones

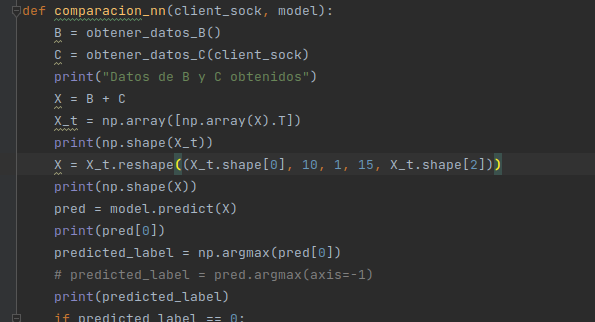
### Cambios en la implementación del nodo B

La recepción de los datos sensados por C fue modificada para evitar problemas de bloqueo al no vaciarse por completo la cola del socket respectivo.



El cambio obtiene los datos por bloques del tamaño hasta vaciar el socket, en lugar de leer bytes fijos de la cola.

La actualización de los paquetes Tensorflow y Keras cambia ligeramente, pasa de entregar la clasificación de datos a entregar el vector de probabilidad de clasificación bajo las distintas clases.



#### Mediciones

Se introdujo una nueva clase para realizar la medición de tiempo, ciclos de reloj y memoria utilizada durante la ejecución de las distintas fases del protocolo.

Esta clase permite la medición de bloques de código una vez inicializada una instancia. Crea reporte por iteración sobre todas las fases especificadas, que puede ser en formato documento txt o csv especificando en la declaración de la instancia (Recomendado txt para una sola iteración y csv para múltiples iteraciones). El archivo generado lleva la nomenclatura ‘StatResultsdd/mm/AAAA (HH:mm:ss en el caso de txt).csv/txt’

Para reportar las mediciones se hace uso de las bibliotecas psutil y time. Mediante el uso de psutil es posible determinar el uso de memoria a nivel sistema, lo cual al medirla en un instante de uso nos permite calcular la memoria usada al restarla a la memoria total (esta memoria no cuenta swap). De manera similar nos permite consultar el frecuencia estimada de los nucleos del procesador, así como el promedio entre ellos, esto en conjunto con la biblioteca time, permite realizar un estimado de los cliclos de reloj por fase.

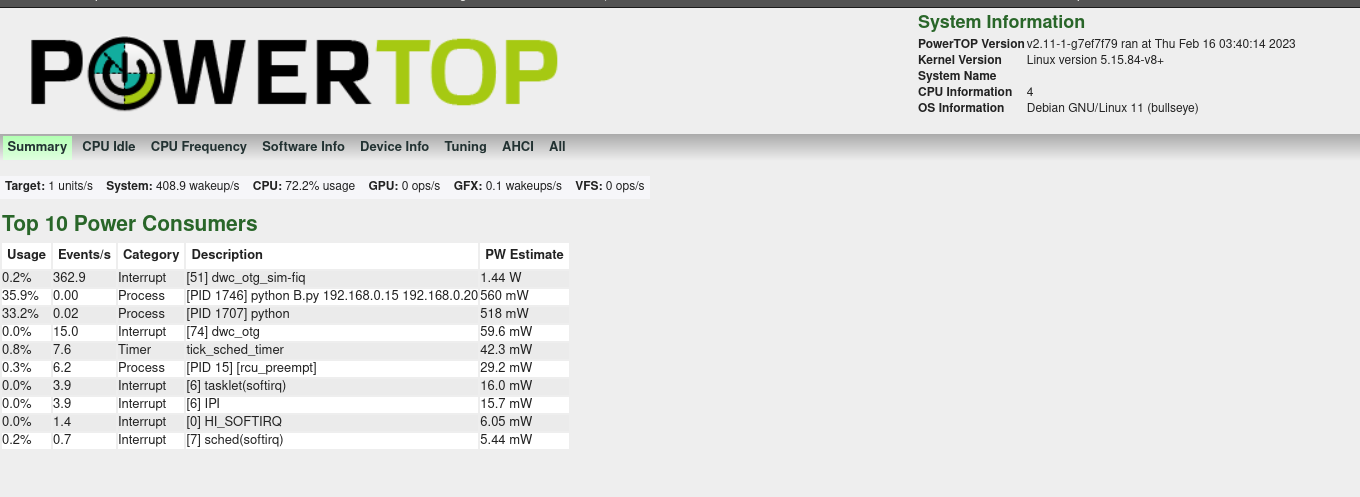
El timer utlizado es perf\_counter, el cual utiliza el timer de precisión del sistema, midiendo tiempo total del proceso incluyendo llamadas a sistema y tiempo en sleep. Es el timer utilizado en paquetes de mediciones de rendimiento como timeit.

Las mediciones toman como referencia el estado inicial del tracker de estadísticas y en el caso de tiempo y ciclos de reloj, se trata de tiempo y ciclos transcurridos desde la fase anterior. Debido a este comportamiento es necesario hacer la llamada de reset\_clock en el caso en que se requieran multiples iteraciones a medir.

El registro de las estadísticas se hace de acuerdo a la llamada, la cual indica el fin de la fase especificada. Las fases se encuentran listadas en un enum, permitiendo que se agreguen o eliminen del listado, o se agreguen comportamientos especiales por fase, sin tener que depender de cadenas de texto.

#### Mediciones de energía

Se hizo uso de la herramienta Powertop, monitorea procesos a nivel sistema y arroja información como estimados de tiempo promedio en procesador y un estimado de consumo energético en watts.

Ejemplo de reporte generado por Powertop

Posteriormente se realizaron mediciones en paralelo a la ejecución del protocolo, llevando interrupciones como control. Se obtuvieron las siguientes mediciones como resultado:

|  |  |
| --- | --- |
| Fase | Consumo |
| Fase 0 | 742 mW |
| Fase 1 | 2.11 mW |
| Fase 2 | 8.17 mW |
| Fase 3 | 2.17 mW |
| Fase 4 | 6.18 mW |

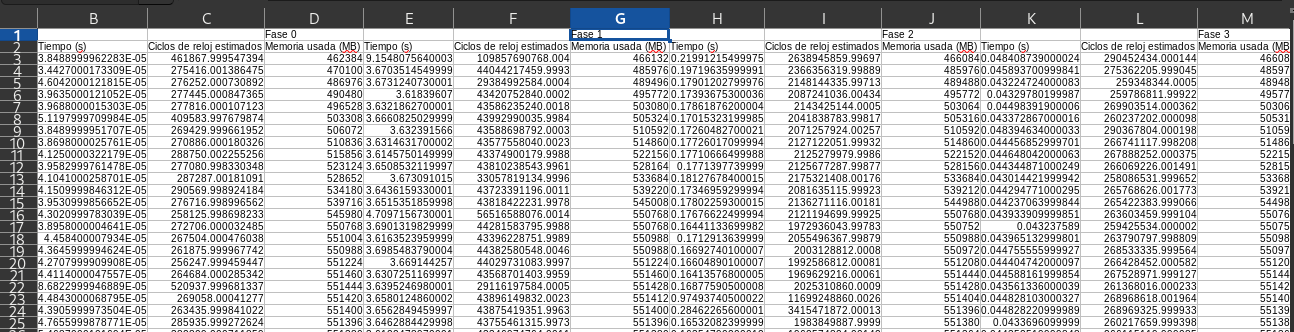
### Actuador D

El devkit ESP8266 probó ser intercambiable por el devkit ESP32 con unicamente modificaciones a la verificación de firma. Dichos cambios incluyen el cambio de bibliotecas integradas para el manejo de tipos y directivas criptográficas.



De igual manera se requiere la inicialización de registros para la aceleración de hardware y procesamiento del hash bajo SHA256.

# Funcionamiento

Ejemplo de mediciones obtenidas

# Código tracker de estadísticas

class Stage(Enum):  
 INICIAL = 0  
 FASE\_0 = 1  
 FASE\_1 = 2  
 FASE\_2 = 3  
 FASE\_3 = 4  
 FASE\_4 = 5  
  
  
class StatTracker:  
 output\_file\_template = 'StatResults{0}.{1}'  
 estimated\_cpu\_cycles = 0  
 previous\_timer = 0  
 delay = 0  
 number\_of\_delays = 0  
 header\_spacer = [None] \* 2  
 csv\_header1 = (["Inicial"] + header\_spacer + ["Fase 0"] + header\_spacer + ["Fase 1"] + header\_spacer  
 + ["Fase 2"] + header\_spacer + ["Fase 3"] + header\_spacer + ["Fase 4"] + header\_spacer)  
 csv\_header2 = ["Memoria usada (MB)", "Tiempo (s)", "Ciclos de reloj estimados"]  
  
 def reset\_clock(self):  
 self.previous\_timer = self.initialization\_time = time.perf\_counter()  
 self.final\_time = 0  
  
 def set\_stats(self, stage: Stage):  
 # Se usa perf\_counter al ser de precision, es el timer default usado por timeit  
 # No se usa timeit en esta clase dado que será una medición única y con tareas bloqueantes  
 if stage == Stage.FASE\_0:  
 self.number\_of\_delays = 150  
 self.delay = 0.01  
 if stage == Stage.FASE\_2:  
 self.delay = 0.5  
 self.final\_time -= (self.delay \* self.number\_of\_delays)  
 self.elapsed\_time = time.perf\_counter() - self.previous\_timer - (self.delay \* self.number\_of\_delays)  
 self.estimated\_cpu\_cycles = psutil.cpu\_freq().current \* self.elapsed\_time \* 10e6  
 self.used\_mem: int = self.total\_memory - psutil.virtual\_memory().available  
 if stage == Stage.FASE\_4:  
 self.final\_time += time.perf\_counter() - self.initialization\_time  
 self.number\_of\_delays = 0  
 print(f"Tiempo {stage.name}\n" + str(time.perf\_counter() - self.previous\_timer))  
 option = ""  
 while option.strip().lower() != "y":  
 option = input("Continuar? [y/n]")  
 if option.strip().lower() != "n" and option.strip().lower() != "y":  
 option = input("Seleccionar [y/n]")  
 self.previous\_timer = time.perf\_counter()  
  
 def write\_stats(self, stage: Stage):  
 self.set\_stats(stage)  
 if self.csv\_file:  
 with open(self.output\_file, 'a') as file:  
 file.write(  
 str(self.used\_mem / 1024) + ','  
 + str(self.elapsed\_time) + ','  
 + str(self.estimated\_cpu\_cycles)  
 + ',')  
 if stage == Stage.FASE\_4:  
 file.write(str(self.final\_time) + '\n')  
  
 else:  
 with open(self.output\_file, 'a') as file:  
 file.write("Uso de memoria {0}\n".format(stage.name))  
 file.write(str(self.used\_mem / 1024) + " MB\n")  
 file.write("Tiempo transcurrido {0} (s)\n".format(stage.name))  
 file.write(str(self.elapsed\_time) + "\n")  
 file.write("Ciclos de reloj estimados {0}\n".format(stage.name))  
 file.write(str(self.estimated\_cpu\_cycles) + "\n")  
  
 def \_\_init\_\_(self, csv\_file=False):  
 self.final\_time = None  
 self.total\_memory = psutil.virtual\_memory().total  
 self.used\_mem = self.total\_memory - psutil.virtual\_memory().available  
 self.initialization\_time = time.perf\_counter()  
 self.elapsed\_time = 0  
 self.csv\_file = csv\_file  
 now = datetime.datetime.now()  
 if csv\_file:  
 self.output\_file = self.output\_file\_template.format(now.strftime(f"%d %m %Y"), "csv")  
 with open(self.output\_file, 'w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(self.csv\_header1)  
 writer.writerow(self.csv\_header2 \* 6 + ["Tiempo total"])  
 else:  
 self.output\_file = self.output\_file\_template.format(now.strftime(f"%d %m %Y %H:%M:%S"), "txt")  
 with open(self.output\_file, 'w') as file:  
 f.write("Estadísticas del {0}".format(now.strftime(f"%d %m %Y %H:%M:%S")) + "\n")  
 # self.write\_stats(Stage.INICIAL)