基於實體層保密技術實現高安全無人機資料交換系統程式碼

作者：鄒穎麒 日期：2023/6/9

此文件整理了有關專本專題所使用的程式碼，為了方便使用者接下來自行更動，作者刪減掉程式碼中實際使用的部分，並重新挑整了部分的命名。請一定要熟悉class的使用。

程式碼模組清單：

- **datastream**：包含所有資料在不同介面之間交換的函式。ESP32的CSI資料透過serial的傳輸介面、樹梅派透過socket進行資料交換的傳輸介面。
- **IoD_UI**：專題呈現實作成果時所設計的使用者介面模組。透過Qtdesigner製作。
- **plkg**：實現plkg有關代碼及資料加密代碼。

非模組部分：

- 在demo資料夾中存放檔案皆為模組的基本功能展示，只需要將其拉到主資料夾中(也就是模組所在的資料夾中)就可以呈現其效果。
- 其餘沒有打包的外部程式為專題實作成果的展試檔。寫的時候比較隨興，建議未來使用者要建立成果於本研究之上時，使用所提供的三個模組重新構建自己的測試環境，以優化效能。

環境：

- python3.8
- 按裝包管理conda

datastream

chat.py

用於在裝置之見建立socket進行資料交換，必須連到同一個網路下，將裝置在該網路的IP/PORT輸入後方可進行資料交換。(細節請看網路概論課本)

需要模組：

- socket
- threading
- re

程式碼說明：

函式名稱	說明
`chat_manager(<string ip>,<int port>)`	兩台裝置皆輸入欲交換資料裝置的IP/PORT即可進行資料交換，預設PORT為5000。若想與同一台裝置建立兩條連線，請使用不同的PORT(網概)。

函式名稱	說明
`chat_manager.receive_task()`	持續進行接收資料的工作
`chat_manager.recv_init()`	初始化資料接收功能
`chat_manager.send_init()`	初始化資料傳輸功能
`chat_manager.chat_init()`	同時初始化接收與傳輸功能
`chat_manager.close_socket()`	同時關閉接收、發送socket
`chat_manager.pop()`	FIFO的方式return第一個字符，回傳格式為utf-8碼
`chat_manager.pop_line()`	FIFO的方式return透過`chat_manager.send_line()`傳輸的連續資料，請避免使用"-end"，此為判斷結束符號。回傳格式為string
`chat_manager.read_queue()`	一次性讀取記憶體內所有的資料
`chat_manager.queue_clear()`	清除記憶體內所有累積資料
`chat_manager.send(<string message>)`	傳輸資料給接收方，會自動進行utf-8編碼
`chat_manager.send_line(<string message>)`	傳輸資料給接收方，接收方可以使用`chat_manager.pop_line()`去取得整段資料
`chat_manager.send_original(<utf-8 message>)`	接收utf-8編碼後直接傳輸

基本功能展示:

```
from datastream import chat
import time
import threading

def show(chat):
    while True:
        time.sleep(0.3)
        message = chat.read_queue()
        if len(message)>0:
            print(message.decode("utf-8"))


Alice = chat.chat("192.168.0.143")#填彼此的IP
Alice.chat_init()
show_thread = threading.Thread(target=show,args=(Alice,))
show_thread.start()

while True:
    Alice.send(input('>>'))
```

上面程式碼實現了同時進行資料收發，可以以此做為參考更動為你的設計。

csi_interface.py

收集CSI資料時與ESP32進行資料交換的介面。同時在此處也實踐了Channel probing的介面，未來欲更動資料收集策略者請看此處。

需要模組：

- serial
- threading
- platform
- threading
- re
- csv

程式碼說明：

函式名稱	說明
`send(<string comport>,<string message>)`	透過serial傳輸資料給其他serial裝置
`read(<string comport>)`	指定特定serisl port接收資料
`savetocsv(<string filename>, <string data>)`	將`com_esp.aquire_csi()`中取得的資料儲存，先輸入filename, 然後輸入儲存的物件
`com_esp(<string comport>,<int baudrate>)`	控制資料交換的協議，與建置好的ESP32控制協議做交互，輸入comport與鮑率就好
`com_esp.set_port(<string comport>,<string baudrate>)`	重新設置comport與鮑率，調整接收裝置的對象
`com_esp.set_channel(<int channel>)`	設置CSI收集資料的channel
`com_esp.set_ping_f(<int frequency>)`	設置ESP32資料的收集頻率
`com_esp.set_timeout(<int t>)`	設置收集資料的時長
`com_esp.__monitor()`	優化過的收集資料核心
`com_esp.start_monitor()`	開始收集資料線程
`com_esp.stop_monitor()`	停止收機資料
`com_esp.clear_queue()`	清除收集資料的buffer
`com_esp.send(<string message>)`	傳輸資料給設定好的ESP32
`com_esp.send_command(<string command>)`	傳輸特定定的指令給ESP32：`ping`:開始ping CSI 給接收端收集，`recv`:開始收集資料CSI資料，`check`:確認ESP32端是否來活著 `restart`:重啟ESP32程序

函式名稱	說明
`com_esp.aquire_csi()`	將接收的資料整理程一連串的字串並回傳，格式為"string","string"..."string","string"
`com_esp.run_collection(<bool priority>,<int frequency>,<int timeout>)`	執行一個設計過的資料收集協議，priority決定資料的收集資料順序，frequency決定收集速率，timeout保證系統的計算時間夠長可以保證系統穩定性

load.py

將從`csi_interface.py`資料轉換成陣列方便使用，會去除掉無用的CSI subcarrier。

需要模組：

- csv
- re
- numpy
- math

函式名稱	說明
`amplitude(<float 實部>,<float 虛部>)`	計算實部和虛部資料相加的大小
`transform(<CSI資料from com_esp.aquire_csi()>)`	將資料轉換成可用的格式[CSIdata],[CSIdata],[CSIdata]...[CSIdata]回傳
`load(<string filename>)`	將過往儲存的資料取出，重現過往的實驗結果

plkg

aes.py

用作AES加密，將密鑰和訊息輸入進行加解密。

需要模組：

- Crypto
- binascii

函式名稱	說明
`encrypt(<string plain_text>,<string secret_key>)`	進行資料加密
`decrypt(<utf-8 encrypted_text>,<string secret_key>)`	將接收到的utf-8加密資料進行解密
`byte_check(<unKnownTypeText>)`	檢查字串的型別

ecc.py

位元糾錯模組。

需要模組：

- bchlib
- random
- math

函式名稱	說明
`bit_flip(<string binary>)`	隨機轉換二元資料的0/1，以10%的機率進行
`rand_sequence(<int num>)`	隨機生成二元編碼
`binary_byte_convertor(<string data>)`	轉換string的二元資料成為byte
`byte_binary_convertor(<string data>)`	轉換byte的二元資料成為string
`run_xor(<string binary_sequence1>,<string binary_sequence1>)`	進行binary string之間的xor
`check_same(<string binary_sequence1>,<string binary_sequence1>)`	確認兩格binary 資料是否完全相同
`BCH_gen()`	生成具備位元糾錯碼的隨機編碼
`reconciliation_encode(<string 量化結果>)`	將量化結果編碼
`reconciliation_decode(<string 量化結果>)`	將量化結果解碼

greycode_quantization.py

量化策略集

需要模組：

- copy

函式名稱	說明
`average(<transform的資料>)`	將CSI資料累積成平均
`swap(bit,x)`	greycode算法
`gray_code_gen(<int number_of_bits>)`	生成greycode的函式，會回傳nbit的grey code
`quantization_1(<string probing_result>,<int number_of_bit_in_greycode>,<int how_many_bits_for_one_greycode>)`	進行量化的策略會回傳量化結果

sha256.py

進行隱私放大

需要模組：

- hashlib

函式名稱	說明
`sha_byte(<string quantization_result>)`	將量化結果通過sha256進行隱私放大

plkg.py

集成子模組，可運行完整的PLKG protocol， 。

需要模組：

- time

函式名稱	說明
`end_device(<string device_tag>)`	設置獨立的資料收集裝置，需要設置裝置名稱 UAV"U"， IoT device"I"
`end_device.set_chatmanager(<class chatmanager>)`	設置資料傳輸的chat_manager
`end_device.save_probing_result(<string filename>)`	設置是否要存檔量化結果
`end_device.time_synchronize()`	進行資料收集與交換時，先進行時間同步以確保穩定
`end_device.channel_probing()`	執行通道探測
`end_device.quantization()`	執行量化
`end_device.information_reconciliation()`	執行訊息糾錯
`end_device.privacy_amplification()`	進行隱私放大
`end_device.plkg()`	執行完整的plkg
`end_device.key`	plkg的key
`end_device.quantization_result`	量化結果