Our file/executable program integrity authorization program used the OP-TEE technology provide by OP-TEE c/o Linaro which is a Trusted Execution Environment (TEE) designed as companion to a non-secure Linux kernel running on Arm. Our program will do the file SWATT signature/check-sum calculation in the embedded computer’s secure word, encrypt and send the result to server through TCP. The sever will compare the data with its own calculated result to do the integrity authorization. The software contains 3 parts of program:

1. Trust-Application [Raspberry PI secure world]: To do the AES-256 Key selection, message encryption/decryption and file SWATT signature/check-sum calculation.

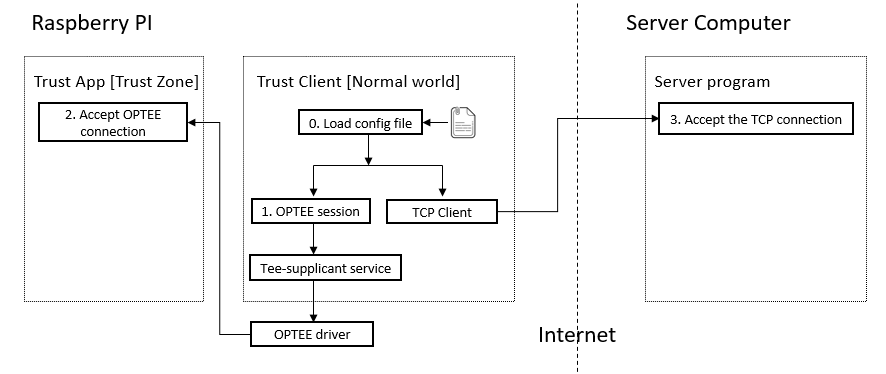
2. Trust-Client [Raspberry PI normal world]: The client program to load configure file, connect to the trust application, fetch the file need to check and connect to the server through TCP.

3. Trust-Server [Server computer]: The server program to generate communication session key, authorize the Integrity of the file and connect to database to save/load related data.

The program will do four steps to authorize the integrity of a specified file

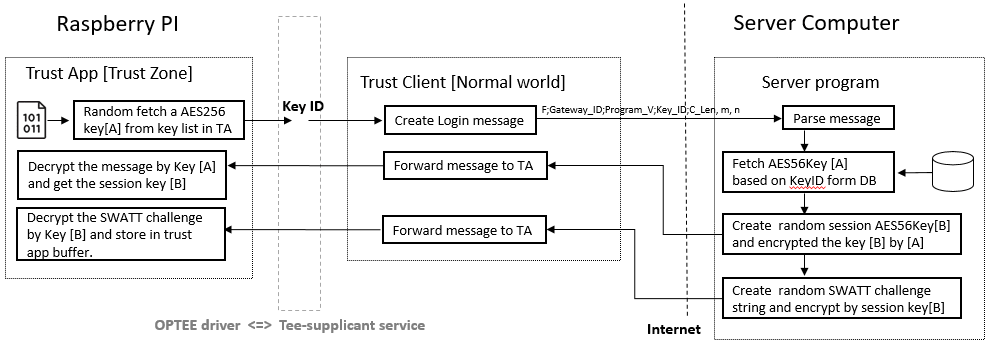
Step 1: System initialization

In the initialization step, the Trust-Client will load the setting information (IP, port, check program, version, SWATT-Challenge str length, SWATT-iteration time) from the gateway's local configure file, then start the "Tee-supplicant service process" to connect to the OPTEE driver. After the OPTEE driver has connected, it will start a OPTEE session to connect to the Trust-Application in the secure world and establish TCP connection to the Trust-Server. (The program execution flow is shown in the Figure 2)



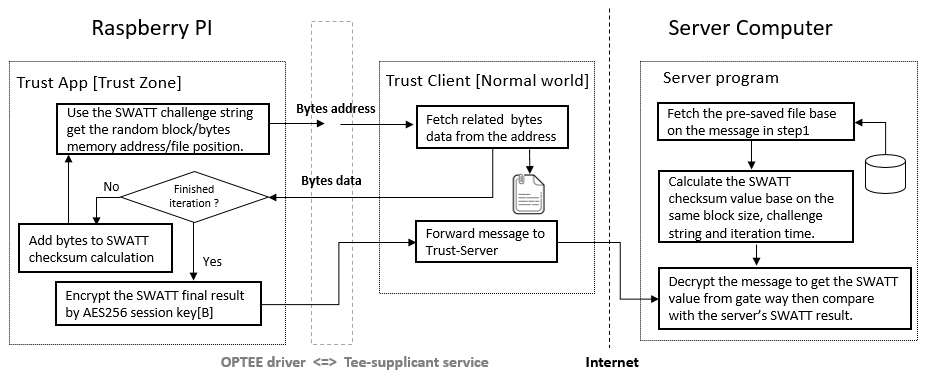
Step2: AES-256 session key exchange

In the key exchange step, the Trust-Application will randomly pick one AES-256 key(key[A]) from its own key notebook and send that key’s ID with other configuration data to the server. The server will fetch the same key notebook from database based on the IOT gateway ID. Then the server will use key ID to get the Key[A]. After that server will create a random session key[B] for all the messages encryption/decryption in the next steps. The key[B] will be encrypted by key[A] and send back to Trust-Client. Then the Trust-Application will decrypt the message to get key[B] and store it in trust buffer. After that the server will create the random SWATT check-sum challenge string and encrypt the string by key[B]. Then send the encrypted challenge string to the Trust-Application. (The program execution flow is shown in the Figure 3)



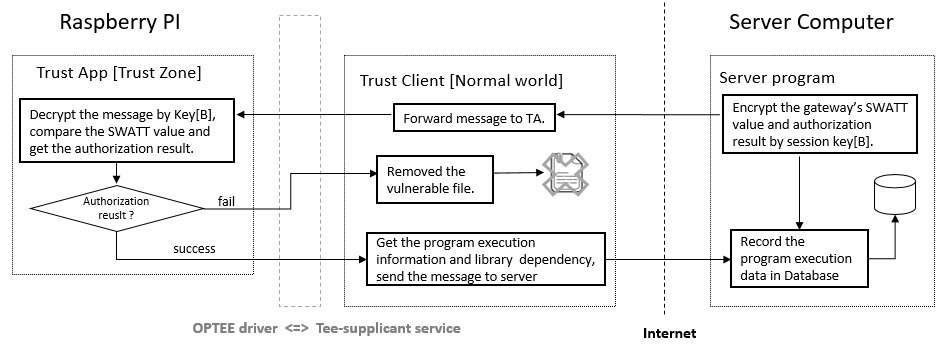
Step3: File integrity authorization

After the session key[B] and SWATT challenge string have been set in the Trust-Application, we will do the file integrity check step: The Trust-Client will load the file need to check from gateway’s local file system and the Trust-Server will load the same pre-saved file from its data base. The Trust-Application will do the SWATT calculation to get the gateway’s local file's SWATT check sum and Trust-Server will also do the SWATT calculation for the server file. Then the Trust-Application will encrypt the gateway's result by session key and send to Trust-Server for comparison.(The program execution flow is shown in the Figure 4)



Step 4: Authorization result feedback.

After finished the SWATT value comparison, Trust-Server will encrypt the gateway’s SWATT value and the authorization result by session key[B], then feed back the message to Trust-Client. The Trust-Application will decrypt the result, verify the correction of feedback (the SWATT value is the one Trust-Application calculated just now). If the authorization result from server shows success, Trust-Client will get the checked program's process ID, execution user, related file descriptor, memory usage size and offset, node information, system lib file name and send all the information to the Trust-Server. The Trust-Client will remove the checked file if the authorization result is fail.(The program execution flow is shown in the Figure 5)



Add the firmware registration part here: