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 signature/checksum 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 authorization. The program contains 3 part of program:

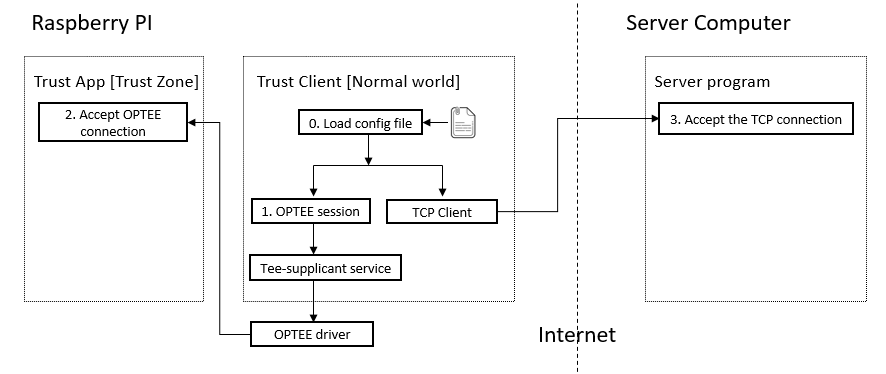
1. Trust Application [Raspberry PI trust world]: To do the AES-Key selection, file signature/checksum calculation and message encryption.
2. Trust Client [Raspberry PI normal world]: The client to connect the trust application, to fetch the file need to check and connect to the server by TCP.
3. Server program [Server computer]: A server program to authorize the Integrity of the file running in the raspberry PI.

--

The program will do 5 steps to authorize the integrity of a pacified file:

Step1: Program initialization

During initialization state, the Trust-Client will start to load the setting from the gateway configure file (IP, port, check program, version, SWATT-Challenge str length, SWATT-iteration time), then start the Tee-supplicant service process to connect to the OPTEE dirver. Then it will start a OPTEE session to connect to the Trust-Application in the secure work and build TCP connection to the Trust-Server.(The program execution flow is shown in the Figure 2)



Step2: AES256 session key exchange.

During the key change step, the trust Application will random pick one AES256 key(key[A]) from its own key notebook and send the key’s ID with other configuration data to the server. The server will fetch the same key notebook from database based on the gateway ID. They the server will get the Key[A]. After that server will create a random session key[B] for all the message encryption/decryption in the next steps. The key[B] will be encrypted by key[A] and the send back to trust-Client then store in the trust-Application’s trust buffer. After that the server will create the random SWATT checksum challenge string and encrypt the string by key[B]. Then send the encrypted challenge string to the Trust-Application.

