Zero Touch Secure Device On-boarding

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Surabhi Kothiyal

(18MECE15)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

Zero Touch Secure Device On-boarding

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering (Embedded Systems)

Submitted By

Surabhi Kothiyal

(18MECE15)

Under the guidance of

External Project Guide:

Ritu Sethi

Engineering Manager

Divneil Wadhawan

Senior Software Developer,

Intel Technologies India Pvt. Ltd.

Internal Project Guide:

Dr. Sachin Gajjar

Asso. Professor, EC Dept,

Institute of Technology,

Nirma University, Ahmedabad

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Surabhi Kothiyal
18MECE15

Certificate

This is to certify that the major project entitled **"Zero Touch Secure Device Onboarding"** submitted by **Surabhi Kothiyal (Roll No : 18MECE15)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Electronics and Communication (Embedded Systems) of Nirma University, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this seminar, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Sachin Gajjar
Internal Guide,Asso.Professor,
EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. N. P Gajjar
Professor,
Coordinator M.Tech - EC (ES),
Institute of Technology,
Nirma University, Ahmedabad

Dr. Dhaval Pujara
Professor and Head,
EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Director,
Institute of Technology,
Nirma University, Ahmedabad

Certificate

This is to certify that the Major Project entitled **"Zero Touch Secure Device Onboarding "** submitted by **Surabhi Kothiyal (18MECE15)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance at **Intel Technologies India Pvt Ltd**. In our opinion, the submitted work has reached a level required for being accepted for examination.

Ritu Sethi
Engineering Manager,
Intel Technologies India Pvt Ltd,
Bangalore

Divneil Wadhawan
Senior Software Developer,
Intel Technologies India Pvt Ltd,
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr.N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems and **Dr.Sachin Gajjar**, Asso. Professor for their guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr.Sachin Gajjar**, guide of my internship project for his exemplary guidance, monitoring and constant encouragement.

I would like to express sincere gratitude to my manager **Ritu Sethi** Engineering Manager, Intel India Pvt. Ltd. and **Raghavendra Patil** Engineering Manager, Intel India Pvt. Ltd, for their encouragement and guidance. I would also like to thank my mentor **Divneil Wadhawan** Lead Engineer, Intel India Pvt. Ltd., for his valuable advices and support throughout the project.

Surabhi Kothiyal

Abstract

Internet of Things (IoT) has become the foundation of trending technology in this modern era. It has created many possibilities by integrating every object for interaction via embedded systems involving the transfer of data to and from servers. As IoT is becoming a essential part of human life, but with that there is an increase demand to provision devices and a need of assurance that the device is properly authenticated. In general, the process of on-boarding a device takes 20-50 minute, which involves co-ordination among installation technicians, network and security operations, that will configure the device and set the credentials to get the device on the network which can be costly and time consuming. The proposed technology scales IoT deployments to put more devices into service faster, by making it fully automated once on-boarded. It has a zero touch approach, that makes IoT devices to dynamically discover the customer's IoT platform at power. For testing purpose, the approach has been carried out with raspberry-pi which acts as device. When the device is first powered on it will connect to manufacturer server. The manufacturer server will insert the credentials and the address of the rendezvous server where it can connect next once it reaches the customer premises. Manufacturer server creates a digital ownership voucher which is then send to customer. At customer site when the device is powered on it is connected to rendezvous server which authenticates the device and it sends the address of the owner server to the device. The device is then connected to owner, the owner and device mutually authenticates themselves through secret messages and ownership voucher. Once the trust is made the device is owned by the owner.Thus zero touch secure on-boarding is achieved. For validation purpose, the testing has been carried out with different configurations and functionality of the code has been checked with the development of unit test case suit. The technology is using security framework which is using ARM Trust Zone.

List of Figures

Abbreviations

IoT	Internet of Things
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
SHA	Secure Hash Algorithm
AES	Advanced Encryption Standard
PKI	Public Key Infrastructure
RA	Registration Authority
CA	Certicate Authority
CRL	Certicate Revocation List
REST	REpresentational State Transfer
ECDSA	Elliptic Curve Digital Signature Algorithm
RV	Rendezvous
OV	Ownership Voucher
TEE	Trusted Execution Environment
TA	Trusted Application
UUID	Universally Unique IDentifier
CBC	Cipher-Block Chaining

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

The phrase “Internet of Things” was invented by Kevin Ashton in 1999.[1] He came up with the idea of communicating things with radio receiver using RFID tag.
Internet of Things(IoT) is a process of enabling the things to communicate with each other, by transferring and receiving information through network (wireless or wired) using technologies (e.g. Wifi, Bluetooth, Zigbee etc.) to transfer and receive the information, that requires less human intervention. The major components of IoT are: things, actuators, sensors and storage component.
The major components of IoT are: things, sensors, actuators and storage component-
Things here refers to smart things which can be devices or objects, equipped with electronics which are having the capability of computing and communicating. These smart things can be discovered, communicated, computed and controlled, over internet. **Sensors** forms an integral part of IoT, sensors collect data to perform the task as required by IoT application. It may vary according to the application used. The sensors can be proximity sensor, touch sensor, humidity sensor, temperature sensor, pressure sensor, accelerometer, etc. Micro-electro-mechanical systems (MEMS) are computer chips that includes both sensors and actuators. A sensor is a device that transforms the functional energy into electrical signals while, an **Actuators** is a device that transforms electrical signals into functional energy. Actuators analyzes the data which is collected and respond accordingly. [2] IoT connects a large number of devices that produces a large amount of data. **Storage** platform is required, so that data can be stored which can be used further

for analysing, storing and triggering the appropriate actions, which can be done using cloud based storage. One of the main objective of Internet of Things is to connect with all the existing different fields. Some of the applications of IoT in different sectors are as follows:

- IoT in industry : It can be used for monitoring temperature,toxic gas and oxygen levels inside industrial plants to ensure the safety of workers and goods.
- IoT in residence : IoT can turn the home into the smart home, by sensing and responding to the activities, by enabling objects/things to communicate with each other, with out human intervention. A smart home can be controlled easily with a smart phone or laptop.
- IoT in agricultural : It can be used to analyze crop environment and the method to improve the efficiency of decision making by analyzing harvest statistics.[3]
- IoT in health care : In health care IoT is used to check and keep a track upon the health conditions of patients,also when they are at remote location.

1.2 Motivation

IoT is becoming a vital part of human life but there have been many challenges that have started cropping up such as scaling, consistency, robustness and security. Scaling and security of Internet of Things is an important aspect that can not not be overlooked as typical scenarios.
Earlier reports of Cisco/IBM claimed that there will be 50 billion devices by 2020,[4] but according to the recent reports the reality is no where close to reaching those numbers. The recent research reports predicts around 30 billion devices in the same time frame. So, what might will be the reason of scale down?? The table 1.1 shows the report scale-down outlook.

Year	Source	Report
2010	IBM	”A world of 1 trillion connected devices” by 2015[5]
2011	Ericsson	”50 billion connected devices” by 2020[5]
2014	Intel Infographis	”31 billion devices connected to internet” by 2020 [5]
2018	Iot Analytics Research	It will reach ”22 billion devices connected to internet” by 2020[5]

Table 1.1: Scale Down Outlook of IoT Device

1.2.1 Reason of Scale Down Approach

- Real-time issues: includes provisioning and management of devices

  When the IoT device reaches the customer site, the on-boarding process for each device mostly takes 20 to 30 minutes which involves manual provisioning, coordination among the technicians, network and security operations. This is costly and time consuming , which is holding the industry back from the promise of more than tens billions of devices.

- Security issues: include data privacy, tracing ownership during distribution, secure communications from edge to cloud.

  There are chances that malware may enter into the IoT network as it connects a lot of devices. The integration of middleware, machine-to-machine communication, APIs etc. may lead to new security risks and complexity.

  An attacker can take the advantage of

  - weak, guessable or hard coded passwords

  - lack of secure update mechanism

  - insecure data transfer

  These malicious programs can perform a variety of different functions such as

  - deleting and stealing sensitive data

  - secretly monitoring users

  - altering the data or computing functions

  - can flood targeted resource, resulting the server to go down

1.3 Objectives

The objective of the thesis to make the on-boarding process of IoT device secure and automatic by cutting down the installation time to bring IoT devices online in seconds rather than hours. To simplify the installer's role, and eliminate poor security practices, such as shipping default passwords.

1.4 Problem Statement

Validation of security and network architecture used in zero-touch device on-boarding.

1.5 Thesis Outline

This report follows a standard outline which is divided into multiple chapters. Chapter-1, contains the introduction of the project which describes the background, motivation, problem statement and objective of the thesis. Chapter-2, contains literature review, which describes the need of security in IoT. It also describes some of the cryptography algorithm, PKI Infrastructure, working of TEE and brief about REST API. Chapter-3 describes the flow of the application with the block diagram and also it describes the working of trust zone in device. Chapter-4 presents the implementation and flow of the project done. Chapter-5 discusses the conclusion and future work.

Chapter 2

Literature Survey

2.1 Security in IoT

2.1.1 Need of Security

There are already billion of devices connected on the internet, with the technology being used in various sectors like industrial, agricultural, transportation, home automation, healthcare etc. but “behind the scenes” it is possible that data collected are used by the third party or enterprise as part of their managed services models. Most IoT devices can serve as entry points into a home or corporate network, exposing families and companies to significant data breach risk. For industrial IoT, those entry points can provide hackers with access to private servers. It’s not just corporate sensitive information at risk many of these business servers contain sensitive personal data of consumers, which could be jeopardized in attacks and leave unwitting customers open to theft. [6]

2.1.2 What is Cryptography?

Security in IoT is an major aspect in today’s era which can be achieved by cryptography. Cryptography is an art of hiding messages by communicating with codes, so the information can be read and processed only by the in-tenders and hackers can not decode it. In cryptography the pre-fix ”crypt” means ”hidden” and the suffix ”graphy” stands for ”writing.”[7] Encryption can be further divided into-

- Secret Key Cryptography / Symmetric Encryption:

  A single key is used for both encrypting and decrypting messages, the sender encrypts the clear/plain text with the key, and sends the encrypted text to the recipient. The receiver uses the same key to decrypt the message , a single key(shared by receiver and sender) is used to both encrypt and decrypt the message it is called as secret key encryption .[8]

- Public Key Cryptography / Asymmetric Encryption:[8]

  A pair of keys are used for encryption and decryption of messages. Sender and receiver can communicate securely through a communication channel link with no need to exchange the key. There are two keys, one of which is public key which is available to anyone and the other is private key. The sender(X) uses receiver (Y) public key to encrypt the message and only the receiver(Y) with its private key can decrypt the message.

- Hash function:[9]

  It is a mathematical function which converts a input value into other compressed numerical value. It does not uses keys for encryption and decryption of data. The input of the hash function can be of any length but its output is always of fixed in length. Hash value is also known as message digest. This algorithms are used to provide a digital fingerprint of a file's content, which ensures that the file has not been altered by an intruder or virus.

2.2 Digital Signatures and Digital Certificates

2.2.1 Digital Signature

Digital Signature[10] ensures that contents of a message have not changed. In this process, data is hased using hash function and then encrypted with the private key of sender. The sender sends encrypted message and the data to receiver. The receiver decrypts it using the public key and received hash is matched with the sender data which is hashed by receiver. Figure 2.1 shows the flow of digital signature.

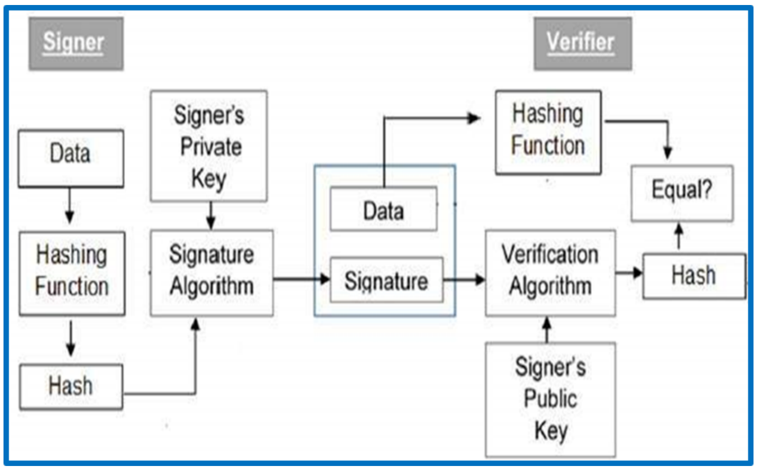


Figure 2.1: Flow of Digital Signature

2.2.2 Digital Certificate

Digital Certificate[10] is a digital document that proves the ownership of the public key, it ensures that during the transmission the content of message is not altered. It is also known as public key certificate. The digital certificate includes, certificate serial number, public key, validity of certificate, identity of the owner, the digital signature of the issuer that has verified the certificate's contents. Fig 2.2 shows how a digital certificate lokk like.

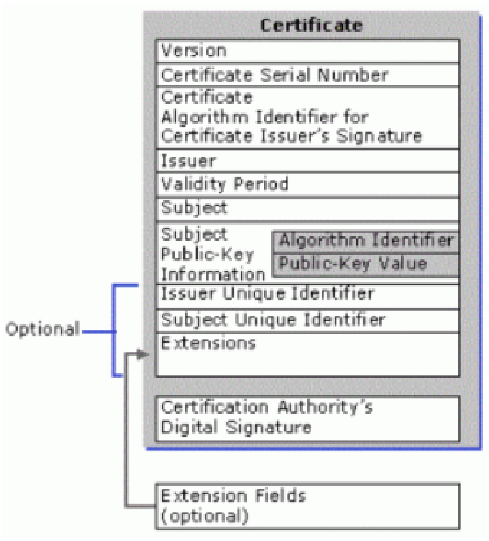

Certificate
Version
Certificate Serial Number
Certificate Algorithm Identifier for Certificate Issuer's Signature
Issuer
Validity Period
Subject
Subject Public-Key Information
Algorithm Identifier
Public-Key Value
Issuer Unique Identifier
Subject Unique Identifier
Extensions
Optional
Certification Authority's Digital Signature
Extension Fields (optional)

Figure 2.2: Digital Certificate

2.3 Cryptography Algorithms

In cryptography variety of conventional algorithms are available. They are RSA, AES, ECDSA, etc.[8] Some of the available cryptography algorithm are explained next-

Advanced Encryption Standard (AES) is widely adopted symmetric encryption algorithm, is based on different key sizes [128 bits (9 iterations), 192 bits (11 iterations), and 256 bits (13 iterations)].

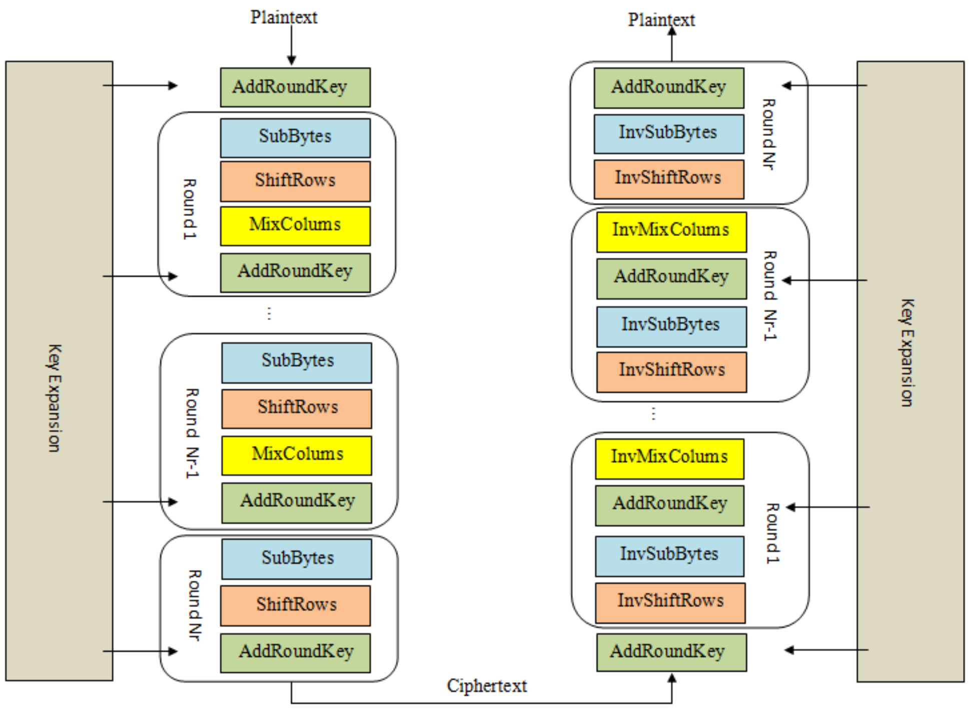


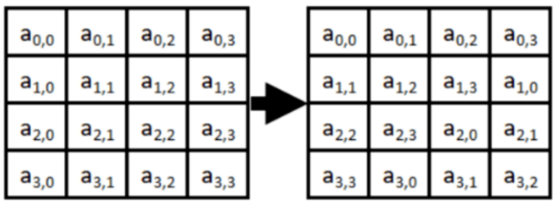
Figure 2.3: Encryption and Decryption in AES

As shown in figure 2.3 AES encryption and decryption consists of 3 round : the initial round , the main round and the final round.

-Add Round key - In this step, the initial key and the message are XOR (Exclusive-ored).

-Sub bytes - As per the pre-determined table each byte is substituted.

-Shift Rows - In this stage each row is shifted to left.The first row remains as it is, it is not shifted, the second row is shifted by one and so on. This is illustrated in the Figure 2.4.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 2.4: Shift Rows in AES

-Mix Columns - In this stage the matrix is split-ted by columns, it performs matrix multiplication.

ECDSA and RSA are algorithms used by public key cryptography systems and are collectively known as digital signature algorithms. RSA algorithm, stands for Ron Rivest, Adi Shamir and Leonard Adleman, it consists of a public key and private key and consists of large prime numbers. It is a three step process that involves key generation, message encryption and message decryption. The algorithm is as follow [11]

- Key generation:
  The block diagram 2.5 illustrates the RSA Key Generation
  (1)Firstly, generate two random prime numbers p and q
  (2)Calculate n = p*q and z = (p-1) (q-1)
  (3) Choose the number e such that, e is less than n, which has no common factor(other than 1) with z .
  (4)Find the number d such that (ed - 1) is exactly divisible by z.
  (5) Keys generated using n, d and e : Public Key is (n,e) and Private key is (n,d).

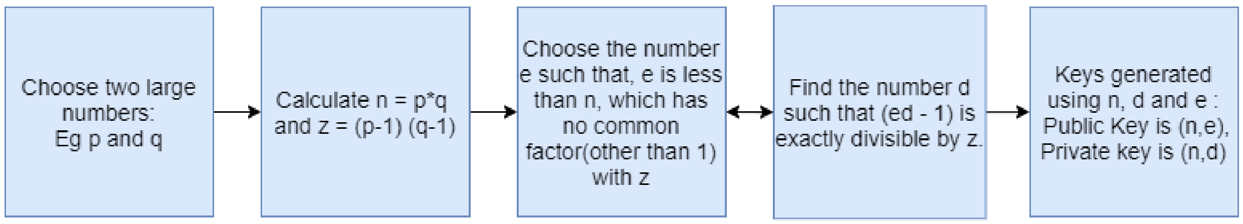

Choose two large numbers: Eg p and q
Calculate n = p*q and z = (p-1) (q-1)
Choose the number e such that, e is less than n, which has no common factor(other than 1) with z
Find the number d such that (ed - 1) is exactly divisible by z.
Keys generated using n, d and e : Public Key is (n,e), Private key is (n,d)

Figure 2.5: Block Diagram Illustrating the Key Generation of RSA Algorithm

- Message Encryption : The sender uses the following method to encipher the message M. Cipher text C=M^e Mod(n) where C is the cipher text generated after encryption.[11] For Example
  C=3^5 Mod(35)
  =243 Mod(35)
  =33
- Message Decryption : The receiver uses the following method to decipher the cipher text C. The original message M = C^dMod(n).[11]
  The other method to create digital signature is through **ECDSA**. Elliptic Curve Digital Signature Algorithm (ECDSA), is used to create a digital signature of data. It is used for verifying its authenticity by making sure that the data was not tampered.
  For signing a file, a private key and the hash of the file is combined together to give the signature. A hash is a mathematical equation that is applied on the data to give a fixed length of unique data as an output. So if the message (the file) is tampered in the transmission then the hash also be changed.

2.4 Public Key Infrastructure

The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke PKCs based on public-key cryptography.
In PKI trust comes from third party that is certificate authority.[12]

2.4.1 Components of PKI

The user generates public and private keys and sends the public key to the Certificate Authority. Figure 2.6 shows Public Key Infrastructure. PKI consists of 4 elements-

- Registration Authority (RA) :Validates the registration of a digital certificate and verifies that requester has a private key which is associated with the public key and submits the certificate request to the CA.[12]
- Certificate Authority (CA) : Verifies the identity of entities and issues and revokes the digital certificates. CA inserts the issuer public key into the certificate.
- Certificate Repository :It stores keys, certificates and Certificate Revocation Lists (CRLs). Key recovery is an advanced function required to recover data or messages when a key is lost .[12]

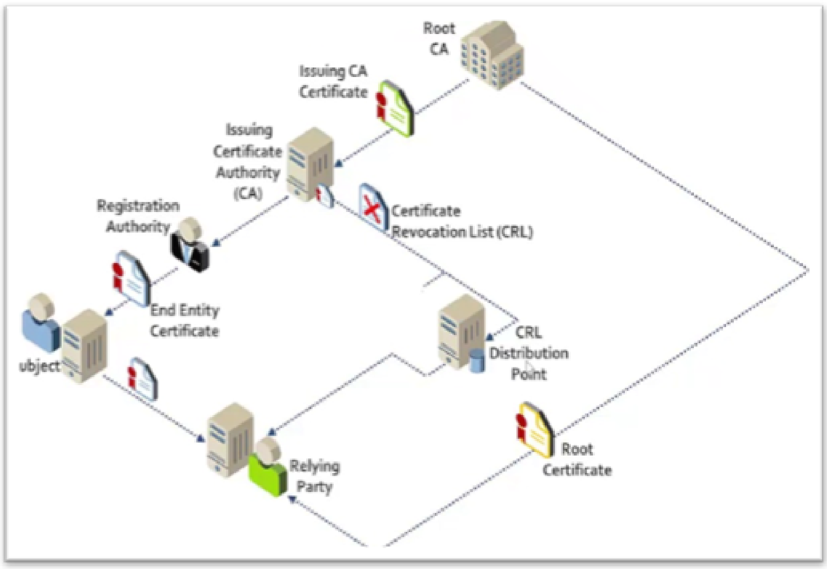


Figure 2.6: Public Key Infrastructure

2.5 Security Model

The Security Model is based on -

- Confidentiality: This can be defined as the secure method for keeping data secret and safe during transmission of information over networks so that unauthorised party can not access the data. Confidentiality can be ensured by encryption algorithms.

- Integrity: The concept of data integrity is that data must be accurate and should not be modified while transmitting.It makes sure that data is authentic.[13] Integrity of data is ensured by message hashing.

- Availability: It is the ability of a user to access information from a specific location in the correct format.[13]

- Non Repudiation: The basis of non-repudiation is that the sender cannot disown any information sent at a later time. It offers non-repudiation through digital signatures.

2.6 Trusted Execution Environment (TEE)

It is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the run time states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and run time states stored on a persistent memory. In addition, it shall also be able to provide remote attestation that proves its trustworthiness for third-parties. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system.[14]

2.6.1 Open-source Portable Trusted Execution Environment(OP-TEE)

It is an open source implementation of Trusted Execution Environment, it consist of secure world and normal word. The client application is running on Linux and trusted application is running on TEE. It uses shared memory to send and receive messages between client application and trusted application.OP-TEE is basically using Arm Trust Zone technology. The main design goals for OP-TEE are: [15]

- Isolation - It must provides isolation from the non-secure world and should ensure data and code confidentiality and integrity .Its main objective is that it to execute the code in complete isolation with respect to other applications which are executing in the same device.

- Small footprint - the size of TEE should be small so that it can reside easily inside on-chip memory as in Arm based systems.

- Portability - TEE can be easily plug-gable to different hardware and architecture like multiple client OS or multiple TEEs.
  An interrupt occurs whenever there is switching from normal world to secure world or vice-versa .It switches through Secure Monitor Call (SMC).An interrupt is signaled by the ARM GIC (Generic Interrupt Controller).

There are two types of interrupt :
(a) Native interrupt - The interrupt is to enter the secure world (FIQ)
(b) Foreign interrupt - The interrupt is to enter non-secure world (IRQ)
Each world has own interrupt exception vector. When an interrupt is in same world, it directly handles the interrupt.When an interrupt is in different world, it switches a context (by Monitor vector) to the corresponding world first and then handles the interrupt. Figure 2.7 shows the flow of IRQ and FIQ

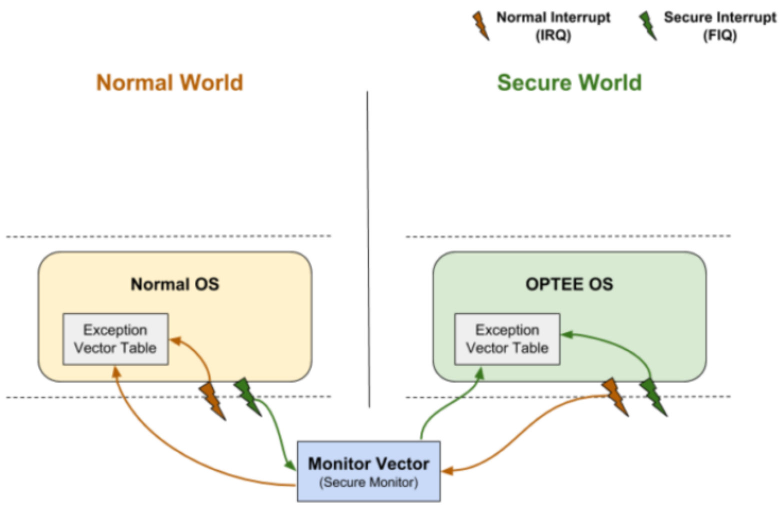


Figure 2.7: Overview of Interrupt Handling

Entering the Secure Monitor :

On every context switching, it saves a state of current world and restores a previous state of another world. On entry for fast SMC it blocks all IRQ/FIQ exception until it returns back to normal world and for Standard SMC will execute the requested service with interrupts unblocked, it assigns a trusted thread to the SMC request. The trusted thread stores the execution context of the requested service. This context can be suspended and resumed as the requested service executes and is interrupted. The trusted thread is released only once the service execution returns with a completion status. The flow diagram 2.8 shows SMC entry to secure world .

The monitor manages all entries and exits of secure world. To enter secure world from normal world the monitor saves the state of normal world and restores the previous state of secure world. Then return from exception is performed and the restored secure state is resumed.

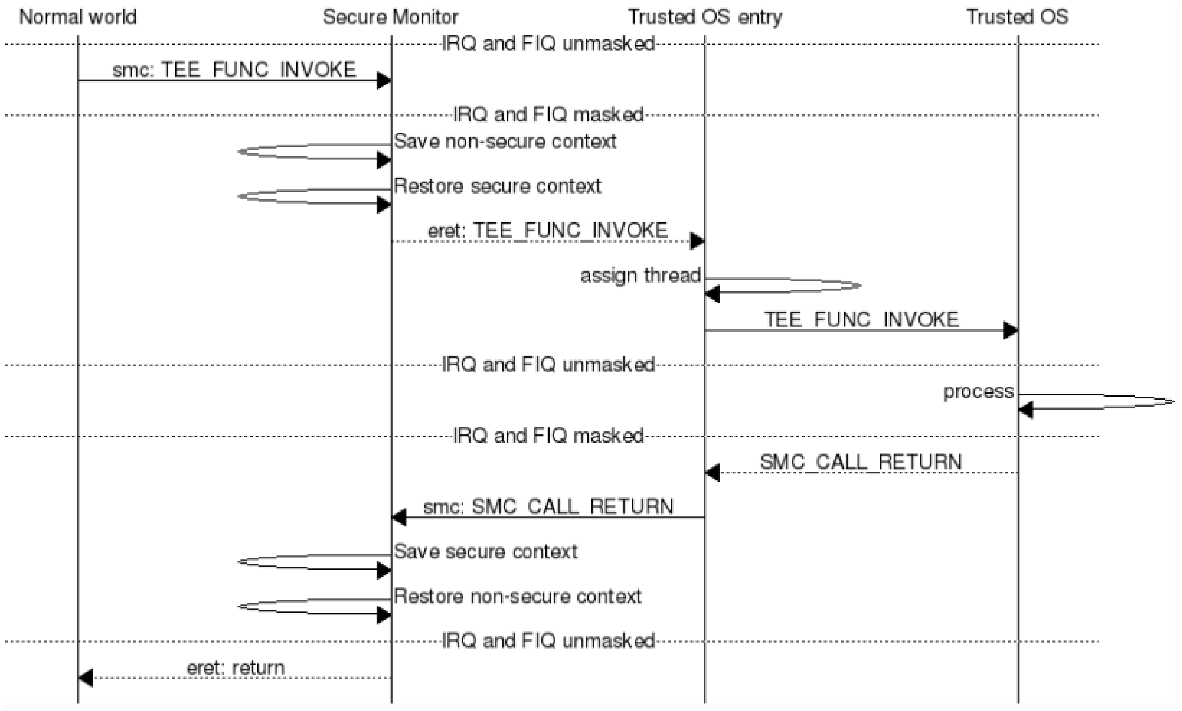


Figure 2.8: SMC Entry to Secure World

2.7 RESTful API

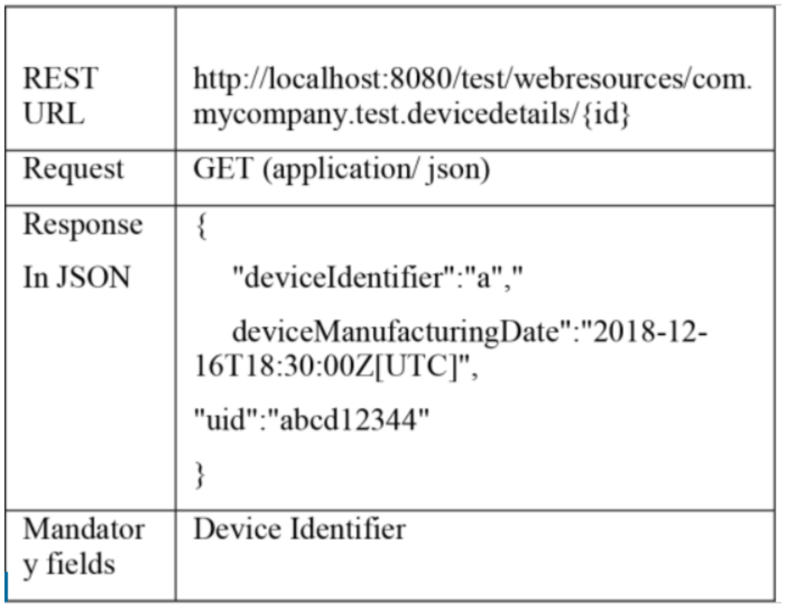
The security in devices/things is a major concern in IoT, which requires an end to end encryption of data to secure data in transit.

API allows to expose the IoT connected device to client or owner in a secure manner which can be accomplished by using RESTful APIs. REST stands for REpresentational State Transfer. It was first introduced by Roy Fielding in year 2000. It acts as a medium between the system which is used to get the data and to apply operations on that data using HTTP formats like XML and JSON.

Each URL is request and the data which is send back to request is an response There are four important data transactions in any REST system and HTTP specification: POST (create), GET (read), PUT (edit) and DELETE. **GET** is a read-only operation, it can be repeated without affecting the state of the resource and can be cached. **PUT** is used to replace/update a resource.(create if doesn't exist) **DELETE** is used to remove a resource. **POST** is used to create a new resource.(e.g. Twitter post)[16]

The data through REST API can be send in multiple formats such as HTML, XML,plain text, YAML, and JSON. Java Script Object Notation(JSON) is lightweight, understandable data-interchange format. It is compatible to any programming language, and easy to read and write .

The figure 2.9 shows and example of REST API using JSON format.



REST URL	http://localhost:8080/test/webresources/com.mycompany.test.devicedetails/{id}
Request	GET (application/ json)
Response In JSON	{ "deviceIdentifier":"a"," deviceManufacturingDate":"2018-12-16T18:30:00Z[UTC]", "uid":"abcd12344" }
Mandatory fields	Device Identifier

Figure 2.9: REST API using JSON Format

Chapter 3

Zero Touch Secure Device On-boarding

The project aims to connect the device securely, to the owner's platform without the human intervention. The flow consist of exchange of messages between device, RV server and owner(client) server.

3.1 Working of Application

The manufacturer of the device inserts keys and credentials of the device. When the device is powered on the customer's site the device automatically connects with the RV server. The RV Server has the information of the owner's address and passes the information to the device. The device connects to owner server. The owner and device verifies itself, creates an encrypted channel and then the information is transferred to device. The main entities of the application are as follows: [17]

- **Manufacturer** :
  It is the manufacturer of device which has details of the devices. It sends the details of the device once, it gets invoked.

- **Device** :
  The device consists of hardware and software.It also includes Trusted Execution Environment TEE which is a secure area inside a main processor. It runs in parallel of the operating system, in an isolated environment. It guarantees that the code and data loaded in the TEE are protected with respect to confidentiality and integrity.

- **Rendezvous Server** :
  It is a service on internet or centralized medium between the device and owner's cloud
- **Owner Client/Cloud** :
  It is a service that is able to prove ownership to the device using an Ownership Voucher and a private key.

3.2 Work Flow of Device

As shown in figure 3.1, the work flow of the device can be described in 3 phases:

3.2.1 Phase 1 : At the Time of Device Manufacturing

- The processor contains a unique hardware root of trust key (ECDSA)
- The Original design manufacturer inserts credentials into the device.
- The Manufacturing Toolkit creates a digital Ownership Voucher (OV) that is sent to the new device owner

3.2.2 Phase 2 : Before Installation

- Step 1: The OV is passed to target IoT platform
  Ownership Voucher is used to establish trust, it is a digital document which is used to transfer digital ownership credentials from owner to owner without the need to power on the device.
- Step 2: The new device is registered in the Rendezvous Service together with the URL for the target IoT platform

3.2.3 Phase 3 : During Installation

- Step 3 : When the device is powered on, it first connects to RV server and it authenticates the device and provides the address(URL) of the target owner.
- Step 4 : The target owner and the device verifies themselves using the root of trust and Ownership Voucher.

– Step 5 : The provisioning payload for the target is transferred to the device. This can be credentials such as passwords or can be a complete platform agent.

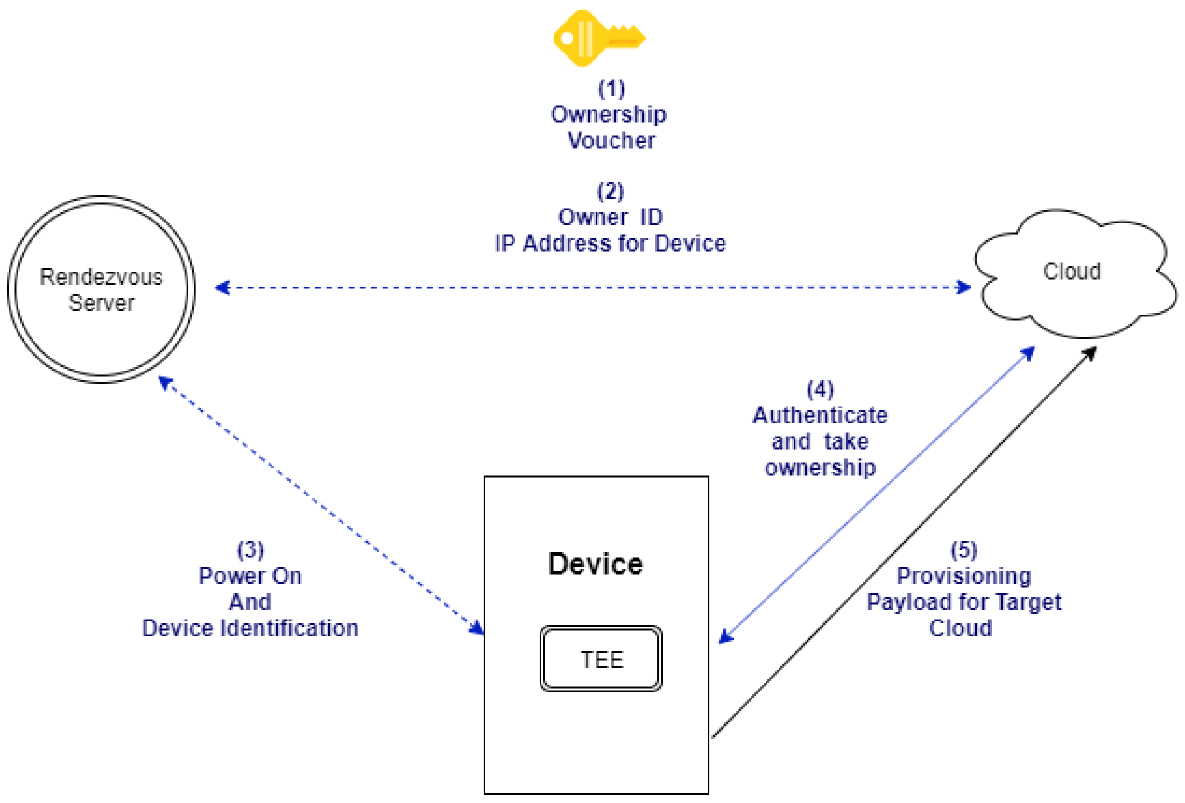


Figure 3.1: Block Diagram

3.3 Flow of TEE

TEE is the core of the device, it provides confidentiality of application and data which has isolated execution environment. Sensitive data is stored in the TEE ARM released trust zone technology which provides a system level security .[15] Figure 3.2 shows the flow of TEE.

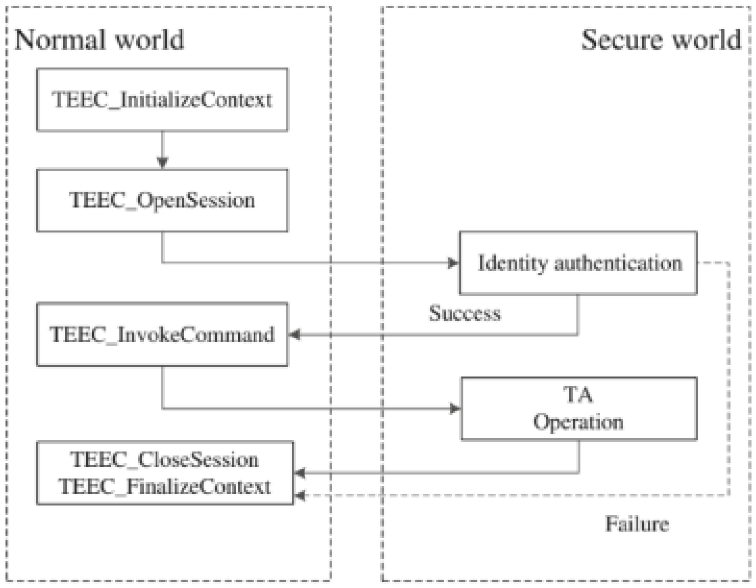


Figure 3.2: TEE Flow

- Client Application(linux) invokes the TEEC InitalizeContext API to initialize TEEC context variable.
- After initialisation, TEEC OpenSession API opens a session with Trust Application(TA).The session is used to transfer parameter between Client Application(CA) and TA. The trust zone finds and load TA which is determined by Universally Unique IDentifier(UUID).
- When TEEC InvokeCommand is called, a context switch occurs and the TA for which the session was opened will receives the data in its corresponding function named TA InvokeCommandEntryPoint. The parameter params contains all information sent by the CA.
- TEEC Close Session is used to release a session.

3.4 Test-Bed Setup

3.4.1 Software

Operating system used is Ubuntu 18. ARM based tool-chain has been used, as supported by OP-TEE .The compiler version used for testing is gcc-arm-8.2-2018.08-x86_64-aarch64-linux-gnu. For validation purpose programming language C has been used. To communicate with TEE GlobalPlatform API are used. To build the Trusted Application in secure world it should include Makefile, sub.mk and header file. Make file consists of configurations and cross compiler information. sub.mk consists of list of the source files required to build. Header file is specific ANSI-C header file.[15]

3.4.2 Hardware

Hardware used is Raspberry Pi 3 Model B,[18] the reason of choosing this hardware is TEE can be easily ported to it. Broadcom BCM2737 SoC which is present in Raspberry Pi 3 board has a TrustZone isolation and protection for sensitive information such as cryptographic keys and data. To access the serial console, connect a USB to TTL Serial Cable to the device UART pins as shown below. Figure 3.3 shows the connection of UART with Rpi-3. 3 pins of Rpi3 are required for using UART:[15]

GND pin: connect to ground of the circuit.

RX pin: to receive the incoming messages from the system.

TX pin: to transmit messages to the system

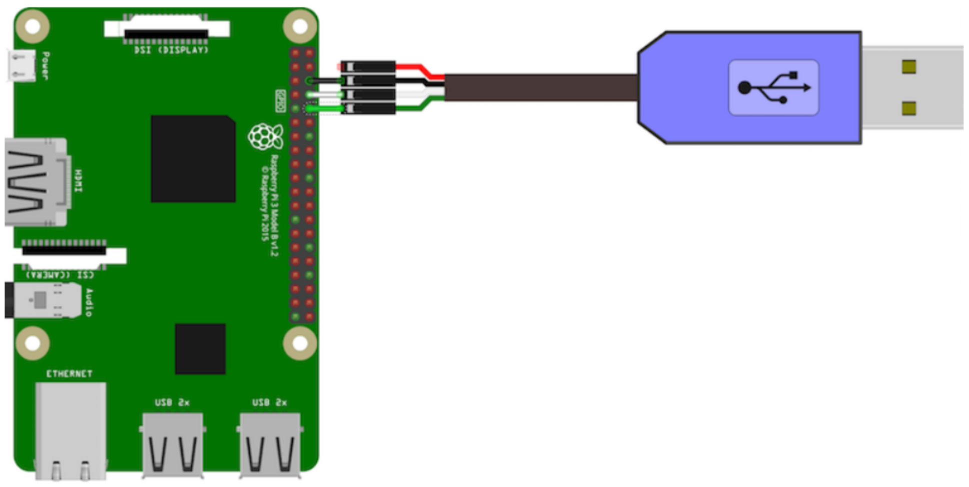


Figure 3.3: Connection of UART with Raspberry pi3

UART pin	Signal	Rpi3 pin
Black (GND)	GND	6
White (RXD)	TXD	8
Green (TXD)	RXD	10

Table 3.1: Pin Connection of UART with Raspberry pi3

Table 3.1 shows connection of UART with Rpi3. On the system we can open up connection to the USB serial device through a terminal program, such as gtk-terminal. The serial port parameters for the console are as follows:

Port :/dev/ttyUSB0

Baud Rate : 115200

Data Bits : 8

Parity : None

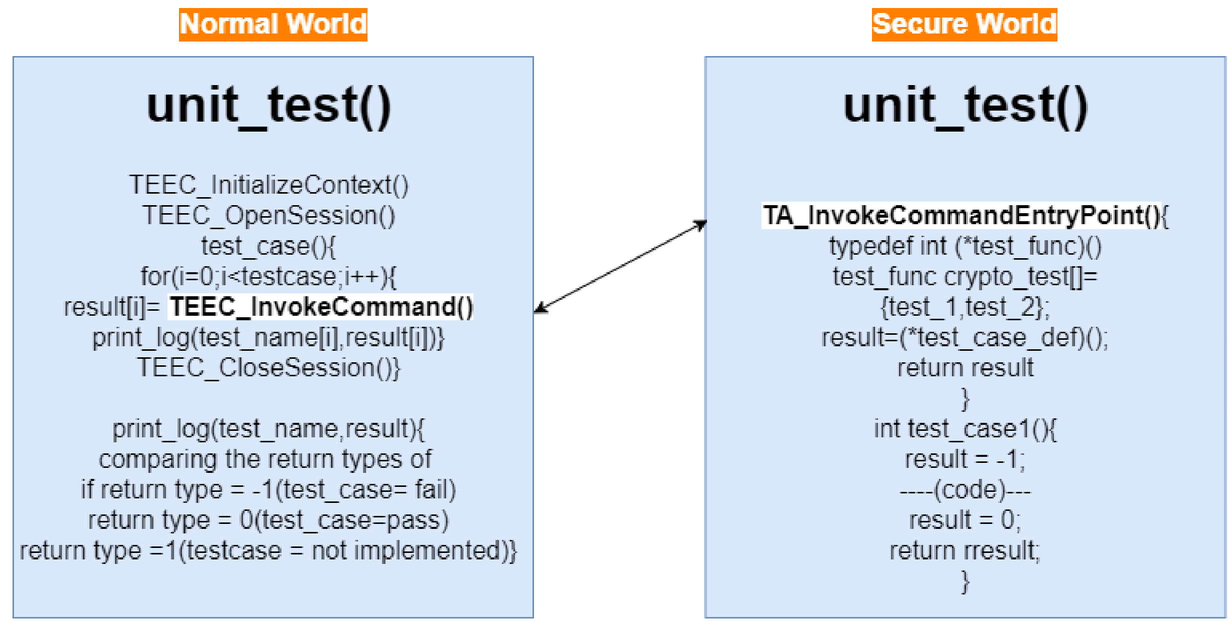
Stop Bits : 1

Chapter 4

Validation of Unit Test Case

4.1 Architecture of Unit-Test Case

A unit test is used to verify the sub-functionality of the application which is independent from other parts, it ensures that every single part of the application works correctly. TEE based architecture is used implement unit test case. OP-TEE requires both hardware and software support. For hardware purpose raspberry pi (ARM based processor) is used to achieve TEE using Trust-Zone technology. The below figure 4.1 shows the flow of development of unit test case:



```
TEEC_InitializeContext()
TEEC_OpenSession()
test_case(){
for(i=0;i<testcase;i++){
result[i]= TEEC_InvokeCommand()
print_log(test_name[i],result[i])}
TEEC_CloseSession()}

print_log(test_name,result){
comparing the return types of
if return type = -1(test_case= fail)
return type = 0(test_case=pass)
return type =1(testcase = not implemented)}
```

```
TA_InvokeCommandEntryPoint(){
typedef int (*test_func)()
test_func crypto_test[]=
{test_1,test_2};
result=(*test_case_def)();
return result
}
int test_case1(){
result = -1;
----(code)---
result = 0;
return rresult;
}
```

Figure 4.1: Flow of Development of Unit Test Case

The flow of unit test case of the op-tee environment starts with TEEC_InitializeContext()

which initialize a context between client to TEE. The normal world (client) creates a session with Trusted Application with TEE_OpenSession() Global Platform API. Once a session is created function test_case() is called, for each test case TEE_InvokeCommand() is called, which triggers TA_InvokeCommandEntryPoint() in host side(secure world) that is trust zone. This Global Platform API calls test_case() in host side which executes the test case. In unit test case suite valid and invalid parameters are send to function and then its return type are compared . That is expected value is compared with the return value if both are same test is passed else failed. The value is returned to the client side(normal world). Print_log() function compares the return value received from the host side and according to which log is generated with the unit-test case name and its status. If the return value is 0, test case status is passed, if the return value is -1, test case status is failed. The normal world will print unit test name with corresponding result.

4.1.1 Sequence of Unit Test Case

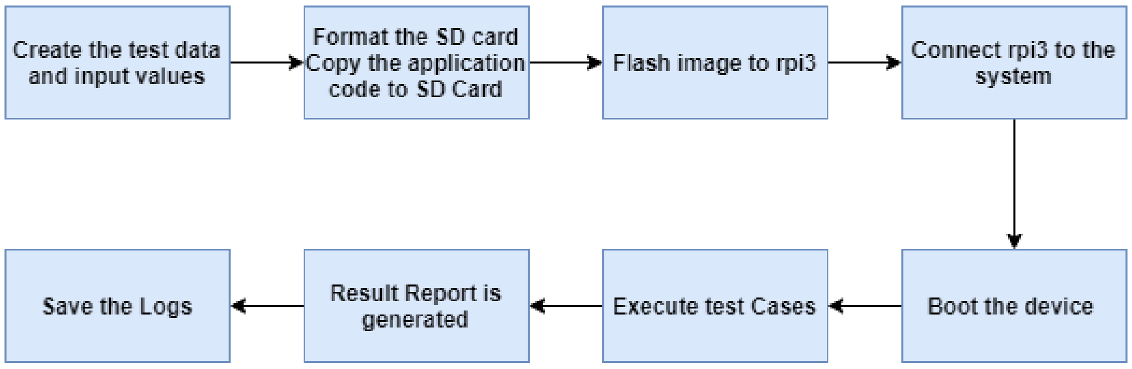

Create the test data and input values
Format the SD card Copy the application code to SD Card
Flash image to rpi3
Connect rpi3 to the system
Boot the device
Execute test Cases
Result Report is generated
Save the Logs

Figure 4.2: Flow of Unit Test Case

Figure 4.2 shows flow of unit test case. The unit test case suite will validate the function by sending valid and invalid parameter to the function .

In first phase it create a test data or input values for an application it wants to test, known as System Under Test(SUT), once the code is build. It is copied to SD card through SD card reader. The image is flashed to Raspberry Pi3. The test case in executed in the serial port terminal which is connected to the device (Raspberry Pi3). The unit test case is passed if the observed behaviour is same as the expected else it fails.

4.2 Implementation of Unit Test Case

4.2.1 Cryptography Unit Test Case Suite

4.2.1.1 Advanced Encryption Standard

It is one of the most popular and widely adopted symmetric encryption algorithm. The test case is executed for both mode of operations that is CBC and CTR.

- CBC (Cipher-Block Chaining) Mode : In CBC mode of encryption, plain text is XOR with the cipher text block that was previously generated. The result is encrypted using the AES algorithm. So, every next block depends upon the previous one. The initial plaintext block is XOR with random initialization vector(iv).The size of iv is same as plain text block. For decryption it is just the opposite of encryption. Figure 4.3 shows the flow of CBC Encryption and Decryption.

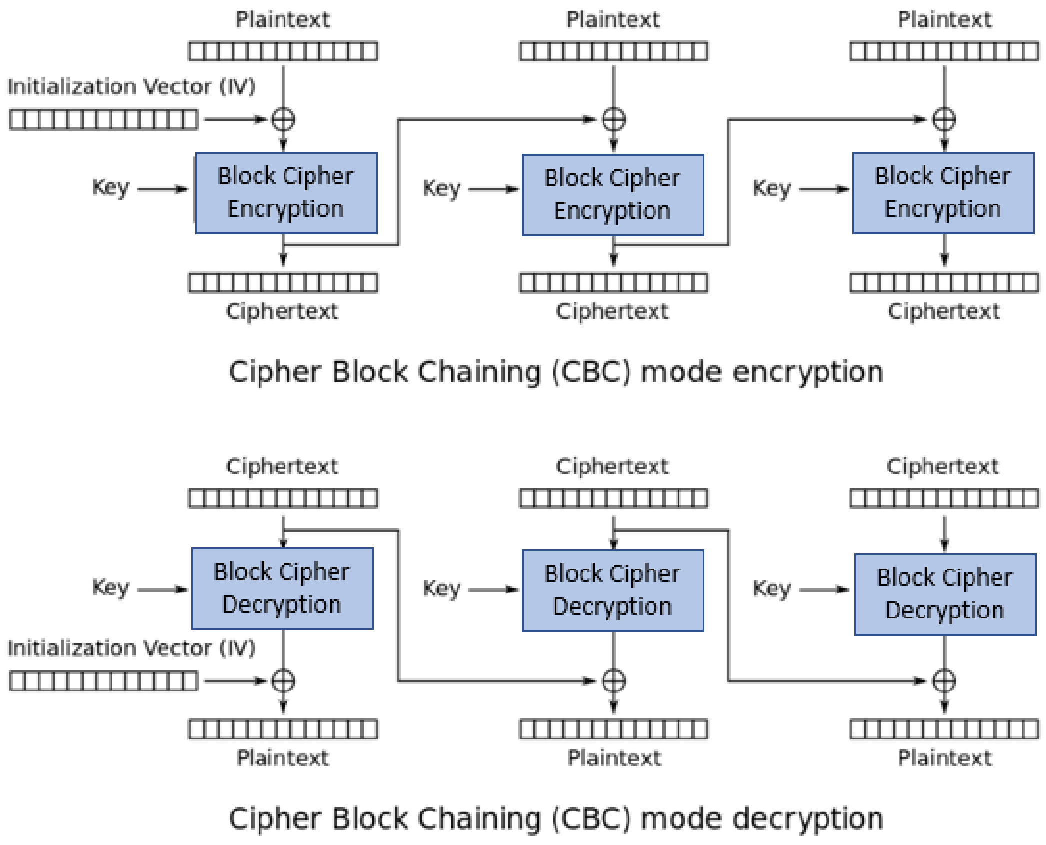


Figure 4.3: CBC Encryption and Decryption

- CTR(Counter) Mode : In CTR mode of encryption the increasing subsequent values of the counter are added to a nonce value (number used once) and the results is encrypted and further XOR with the block of plain text. There is no dependency as there is no feedback. Figure 4.4 shows CTR encryption and decryption

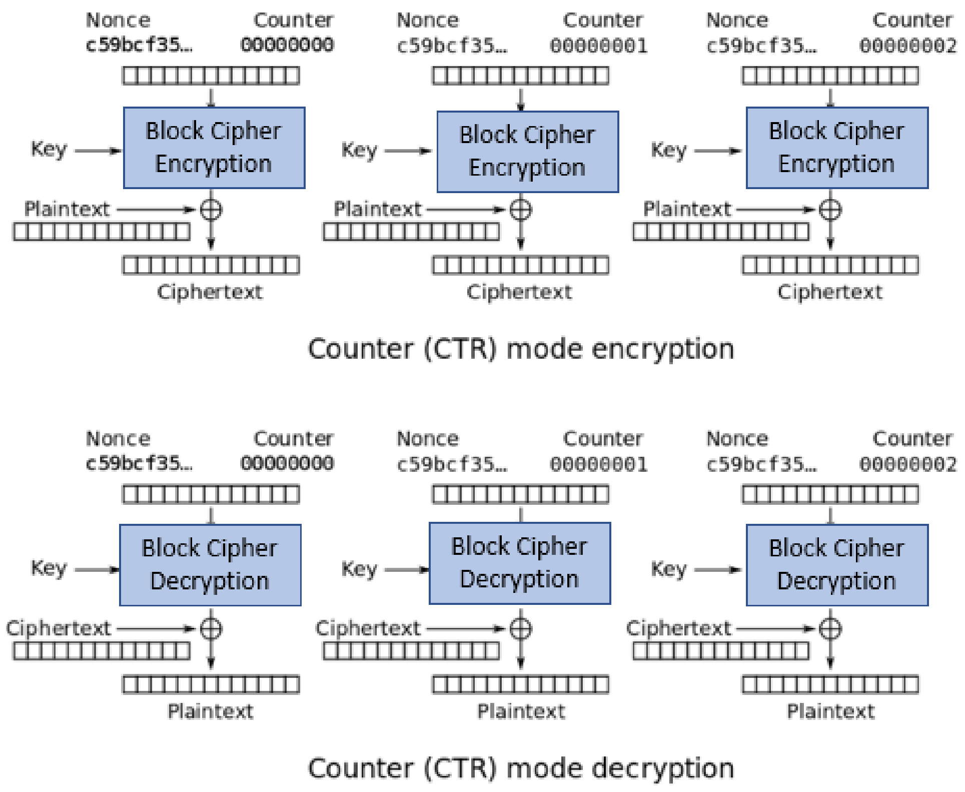


Figure 4.4: CTR Encryption And Decryption

AES Unit test Case Flow

[1] AES Encryption Flow

AES encryption takes clear text (original text) and based on the mode it calculates the cipher length. If the mode is CTR the length of cipher is equal to the clear text length. If the mode is CBC cipher length is equal to [(clear text length/block size of AES) * block size] of AES.Once cipher length is calculated the AES encrypt API will encrypt the clear text to cipher text. Figure 4.5 shows Unit Test Case Flow of AES Encryption. Figure 4.5 shows AES encryption.

[2] AES Decryption Flow

AES decrypt API takes the cipher text and based on the mode it calculates the clear text

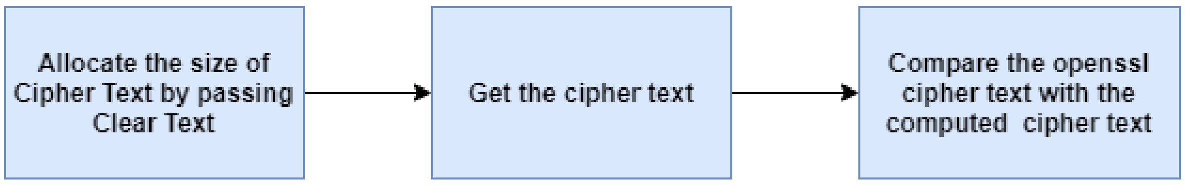

Allocate the size of Cipher Text by passing Clear Text
Get the cipher text
Compare the openssl cipher text with the computed cipher text

Figure 4.5: Unit Test Case Flow of AES Encryption

length. If the mode CBC or CTR the length of cipher is equal to the cipher text length. Once clear text length is calculated the AES Decrypt API will decrypt the clear text to cipher text. Figure 4.6 shows AES decryption.

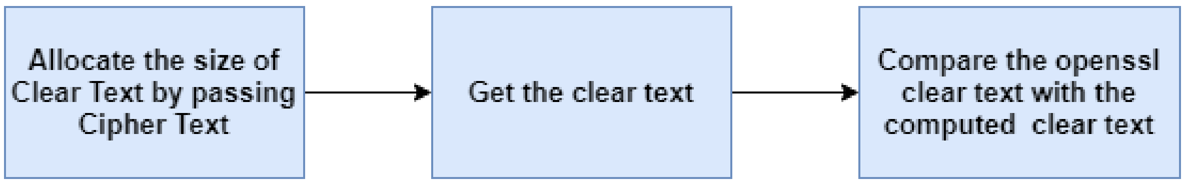
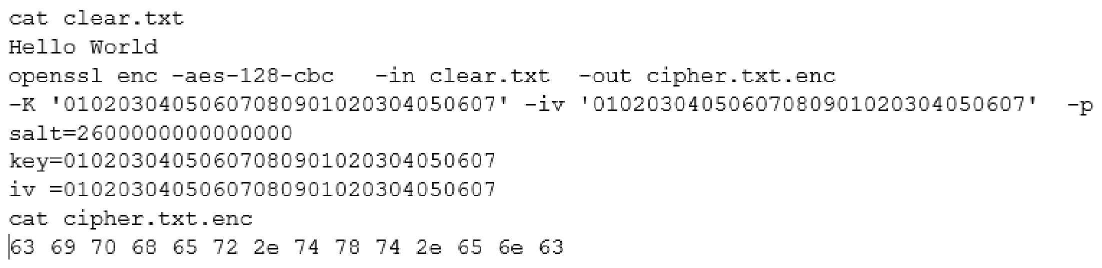

Allocate the size of Clear Text by passing Cipher Text
Get the clear text
Compare the openssl clear text with the computed clear text

Figure 4.6: Unit Test Case Flow of AES Decryption

Verifying using OpenSSL

OpenSSL is a powerful cryptography toolkit that can be used for encryption and decryption of messages and files. For the validation purpose the results generated from the aes encryption and decryption unit test case suit was compared with the openSSL. Figure 4.7 shows encrypted text using openSSL.

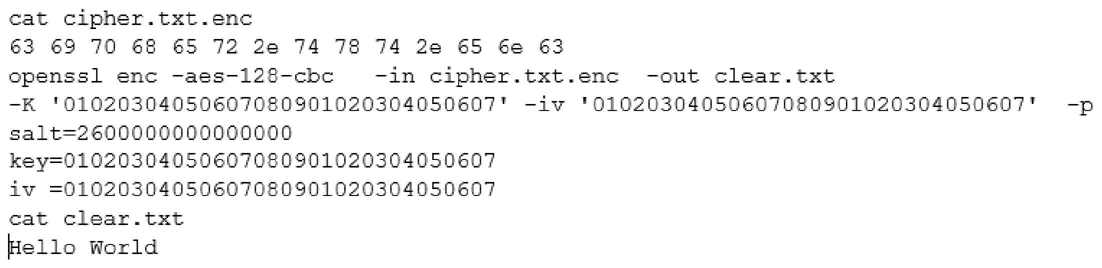


```
cat clear.txt
Hello World
openssl enc -aes-128-cbc   -in clear.txt  -out cipher.txt.enc
-K '01020304050607080901020304050607' -iv '01020304050607080901020304050607'  -p
salt=2600000000000000
key=01020304050607080901020304050607
iv =01020304050607080901020304050607
cat cipher.txt.enc
63 69 70 68 65 72 2e 74 78 74 2e 65 6e 63
```

Figure 4.7: Encrypted Text using OpenSSL

The result generated by decryption API is compared with openSSL command by providing same parameters in both the case . Figure 4.8 shows decrypted text using openssl.

The table 4.1 shows an example of AES Encrypt Unit Test case suite in which valid and invalid parameter are passed for validation purpose.



```
cat cipher.txt.enc
63 69 70 68 65 72 2e 74 78 74 2e 65 6e 63
openssl enc -aes-128-cbc   -in cipher.txt.enc  -out clear.txt
-K '01020304050607080901020304050607' -iv '01020304050607080901020304050607'  -p
salt=2600000000000000
key=01020304050607080901020304050607
iv =01020304050607080901020304050607
cat clear.txt
Hello World
```

Figure 4.8: Decrypted Text using OpenSSL

Parameter/ Function Name (AES encrypt)	Plain Text	Plain Text Length	Cipher Text	Cipher Text Length	Key	Key Length	Block Size	IV	Mode
all valid	valid	valid	-	valid	valid	valid	valid	valid	valid
Clear txt (invalid)	invalid	valid	-	valid	valid	valid	valid	valid	valid
Clear txt length (invalid)	valid	invalid	-	valid	valid	valid	valid	valid	valid
Ciphertext (invalid)	-	-	-	-	-	-	-	-	-
Cipher length (invalid)	valid	valid	-	invalid	valid	valid	valid	valid	valid
Key (invalid)	valid	valid	-	valid	invalid	valid	valid	valid	valid
Key length (invalid)	valid	valid	-	valid	valid	invalid	valid	valid	valid
Block size (invalid)	valid	valid	-	valid	valid	valid	invalid	valid	valid
IV (invalid)	valid	valid	-	valid	valid	valid	valid	invalid	valid
Mode (invalid)	valid	valid	-	valid	valid	valid	valid	valid	invalid
all invalid	invalid	invalid	-	invalid	in valid	invalid	invalid	invalid	invalid

Table 4.1: Overview of AES Encryption Test Case

4.2.2 Signature Verification Test Case

The signature verification test case validates the authenticity of digital messages. The private key with the message hash is used to generate signature, which can be verified by its corresponding public key.

4.2.2.1 Signature Verification using RSA

RSA is one of the most popular and secure public-key encryption methods. Figure 4.9 shows flow of RSA test. The step of the RSA verification test case are the following :

1. The first step includes to get a clear text, for validation purpose,a random text is generated using the Global Trusted Application.
2. For verification purpose RSA key pair (public and private key) are generated.
3. The next step returns the message digest according the configuration selected(Secure Hash Algorithm-256 or Secure Hash Algorithm-384) after that it calculates the message-digest of the the text,with respect to the above the message digest returned.
4. In the fourth step message digest is encrypted with the private RSA key which is generated in step[2].
5. For verification purpose : (a) message digest which is used in step [3] is applied in the text message and (b) the generated signature is de-crypted using using RSA public key which was generated in step[2]. If both the message (a) and (b) results are same then the signature is verified.

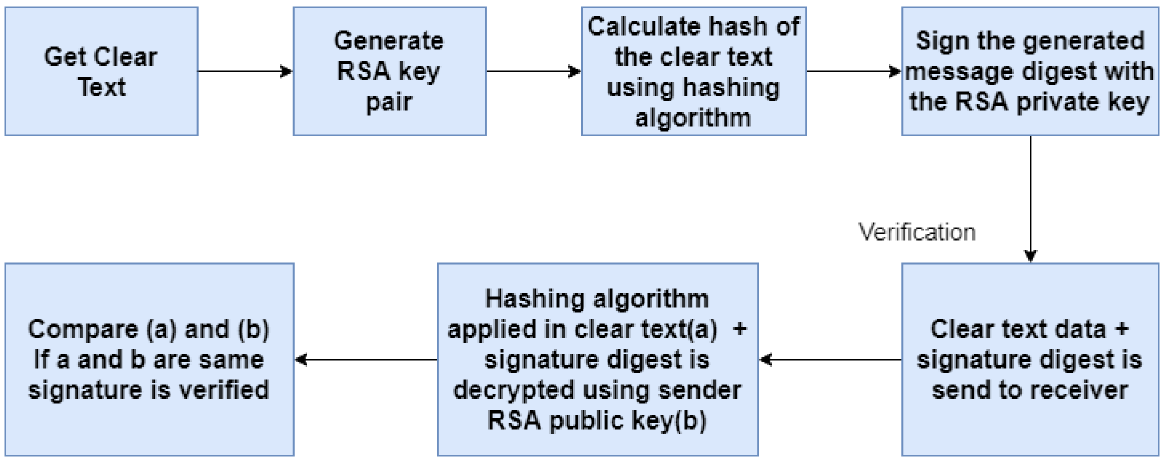

Get Clear Text
Generate RSA key pair
Calculate hash of the clear text using hashing algorithm
Sign the generated message digest with the RSA private key
Verification
Clear text data + signature digest is send to receiver
Hashing algorithm applied in clear text(a) + signature digest is decrypted using sender RSA public key(b)
Compare (a) and (b) If a and b are same signature is verified

Figure 4.9: Flow of RSA Test

The table 4.2 shows an example of RSA Verification Test case suite in which valid and invalid parameter are passed for validation purpose. Here key encoding used is RSA, parameter message is encoded message.

Parameter/ Function Name (Sign Verification)	Key Encoding	Message	Message Length	Message Sign	Sign Length	Key	Key Length
all valid	valid	valid	valid	valid	valid	valid	valid
Key Encoding (invalid)	invalid	valid	valid	valid	valid	valid	valid
Protocol Version (invalid)	valid	invalid	valid	valid	valid	valid	valid
Message (invalid)	valid	valid	invalid	valid	valid	valid	valid
Message Length (invalid)	valid	valid	valid	invalid	valid	valid	valid
Message Sign (invalid)	valid	valid	valid	valid	invalid	valid	valid
Sign Length (invalid)	valid	valid	valid	valid	valid	invalid	valid
Key (invalid)	valid	valid	valid	valid	valid	valid	invalid
Key Length (invalid)	valid	valid	valid	valid	valid	valid	invalid
(all invalid)	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Table 4.2: Overview of RSA Verification Test Case

4.2.3 Network Test Case Suite

Network Test Case is one of the crucial validation test, as it is used to build a secure connection between device and server. Here, RESTful API are used to transmit message and generate operations on those data using HTTP in JSON format. There are four major components in any REST system : POST (create), GET (read), PUT (edit) and DELETE.

4.2.3.1 Network Unit Test Case Flow

The network setup initialise the REST Context, then it will look for the particular host through DNS lookup table and POST the IP-address and port number to REST. It will further connect to the network socket and the connection socket API will return the socket handle. The messages can be send and received using send() and receive() network API. After the transmission of message is done. The network connection is closed. Figure 4.10 shows flow of network test.

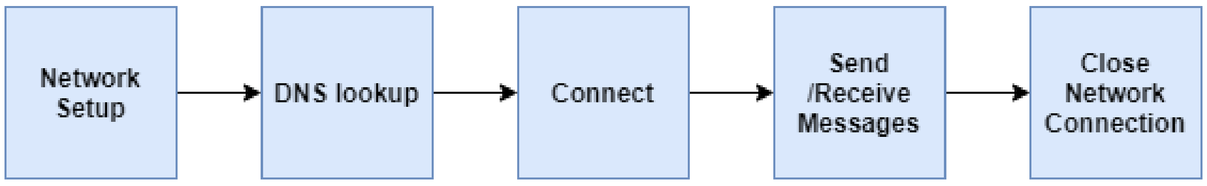


Figure 4.10: Flow of Network Test

The table 4.3 shows an example of Network Unit Test case suite in which valid and invalid parameter are passed for validation purpose.

Parameter/ Function Name (Send Message)	Socket Handler	Protocol Version	Message Type	Buffer	Buffer Size	Key Length	Block Size
all valid	valid	valid	valid	valid	valid	valid	valid
Socket Handler (invalid)	invalid	valid	valid	valid	valid	valid	valid
Protocol Version (invalid)	valid	invalid	valid	valid	valid	valid	valid
Message Type (invalid)	valid	valid	invalid	valid	valid	valid	valid
Buffer (invalid)	valid	valid	valid	invalid	valid	valid	valid
Buffer Size (invalid)	valid	valid	valid	valid	invalid	valid	valid
Key (invalid)	valid	valid	valid	valid	valid	invalid	valid
Key length (invalid)	valid	valid	valid	valid	valid	valid	invalid
(all invalid)	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Table 4.3: Overview of Network Test Case

4.2.4 Implementation of Wrapper Function Test

Wrapper function is used to call a 2nd sub-routine or a system call. The main purpose of wrapper function is to check errors during function calls. The figure 4.11 shows the demo code used to check the dynamic memory location: malloc. The malloc wrapper function[__wrap_malloc()] will be called in place of malloc()function. The following have been performed in the given below code:

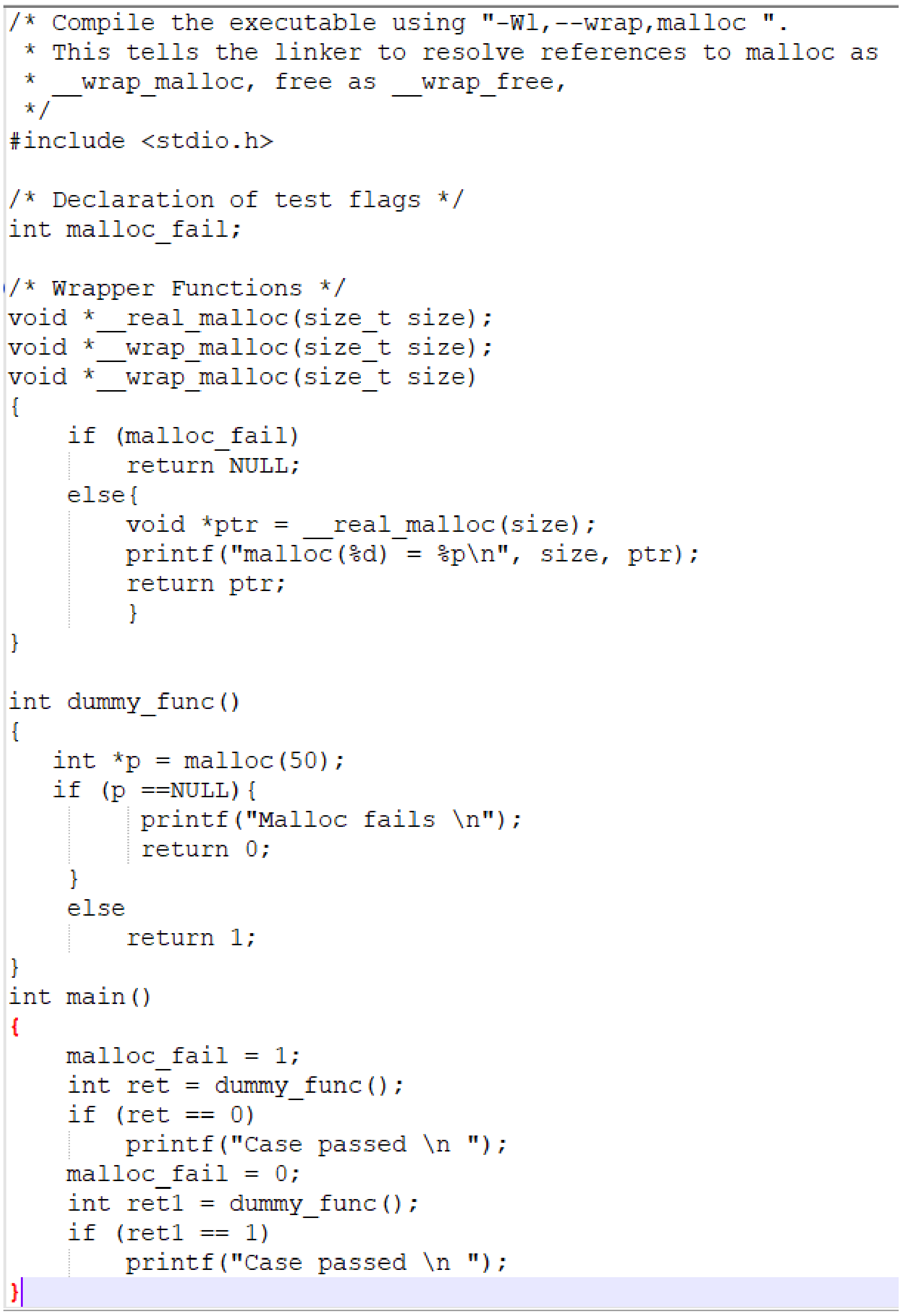
- Forcing the second sub-routing malloc to fail.

  For this we are setting the test flag to 1. The malloc will return NULL, which will force the malloc to fail.

- Calling the real malloc function

  The flag is set to 0, so real_malloc will be called which will return size and location of memory allocated.

Note: after after setting the malloc flag we reset it so malloc function works correctly.



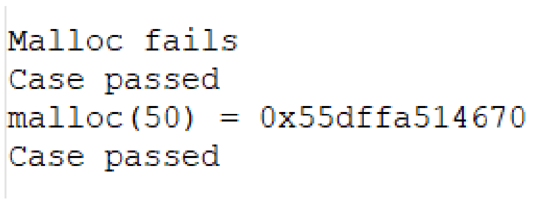
```
/* Compile the executable using "-Wl,--wrap,malloc ".
 * This tells the linker to resolve references to malloc as
 * __wrap_malloc, free as __wrap_free,
 */
#include <stdio.h>

/* Declaration of test flags */
int malloc_fail;

/* Wrapper Functions */
void *__real_malloc(size_t size);
void *__wrap_malloc(size_t size);
void *__wrap_malloc(size_t size)
{
    if (malloc_fail)
        return NULL;
    else{
        void *ptr = __real_malloc(size);
        printf("malloc(%d) = %p\n", size, ptr);
        return ptr;
        }
}

int dummy_func()
{
   int *p = malloc(50);
   if (p ==NULL){
        printf("Malloc fails \n");
        return 0;
    }
    else
        return 1;
}
int main()
{
    malloc_fail = 1;
    int ret = dummy_func();
    if (ret == 0)
        printf("Case passed \n ");
    malloc_fail = 0;
    int ret1 = dummy_func();
    if (ret1 == 1)
        printf("Case passed \n ");
}
```

Figure 4.11: Demo Code for Wrapper Function



```
Malloc fails
Case passed
malloc(50) = 0x55dffa514670
Case passed
```

Figure 4.12: Output of Wrapper Function

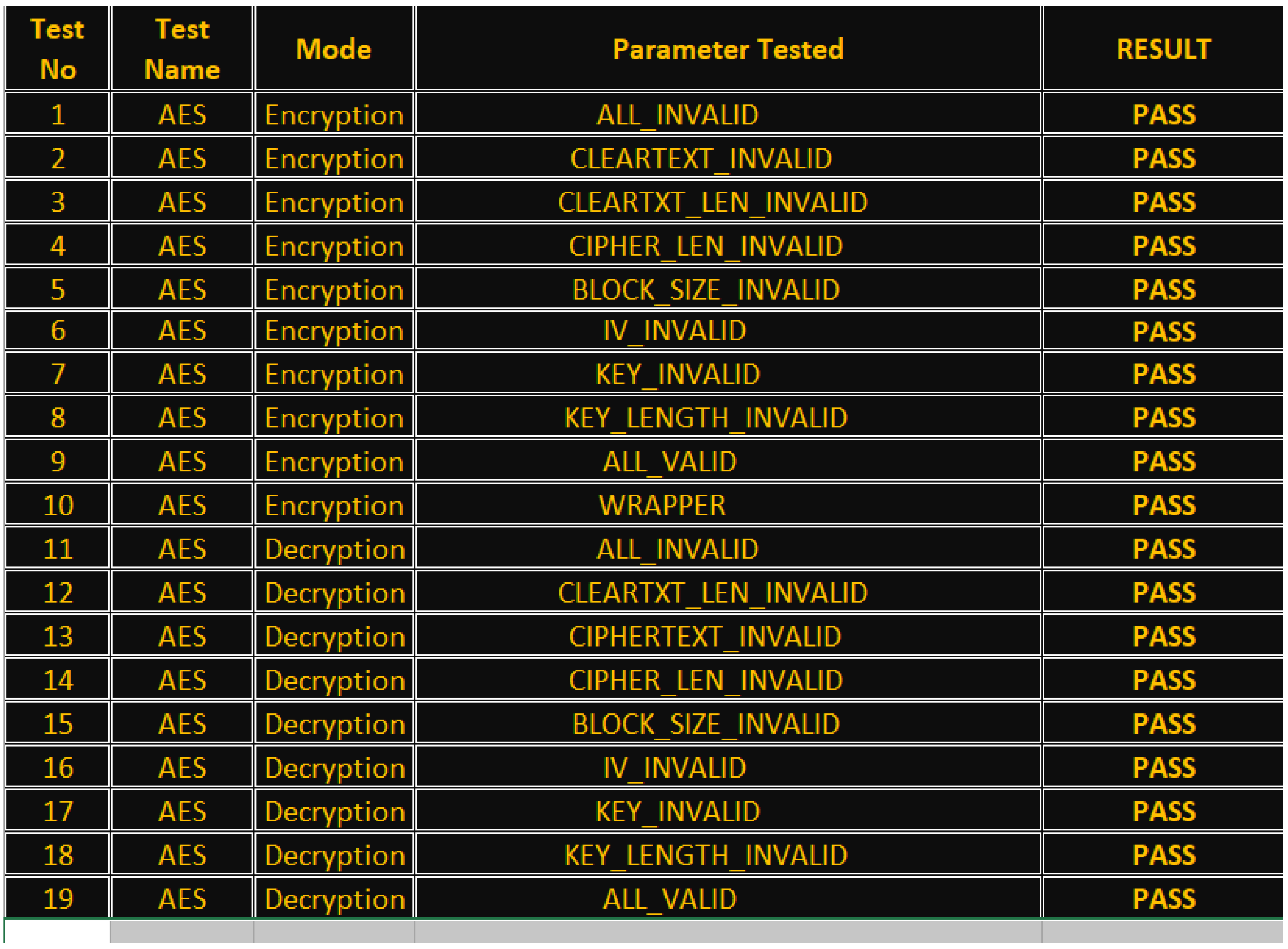
The output of the above program is show in figure 4.12

4.3 Results

The results are obtained by building the C code using ARM based cross compiler in Linux environment. Here the cross compiler is used because OP-TEE supports Arm Trust-Zone technology. After building the code, the image is flashed in rpi-3. To access the serial terminal USB to TTL Serial Cable is used. The test command is written in gtk-terminal and all the unit test case are executed in one go. The log file is generated. The log files shows the name of unit-test case suit with its results.
If the unit-test case returns 0, then the status shown is PASSED, if the unit-test case returns -1, the status shown in FAILED. A unit-test suite consist of different functions that are checked by passing valid and invalid parameter, if any of the functions fails -1 is returned. Test case may fail because of the incorrect error handling in the source code, if the flow of the unit test case is not correct or if there is a bug in source code. One of the way of debugging the error in OP-TEE based environment is is to print messages and its value. The total number of test case implemented is 200 and all of them were passed. The figure 4.13 shows the example log file generated in serial port terminal.

- Development of various unit test case suite for the application. Results :
  Passed : 200
  Failed : 0
  Not Implement : 0

Validation of the full flow of zero touch secure device on-boarding is carried out with different configuration, here configuration means with different public key encryption,device attestation techniques and AES modes etc. Different combinations are used to obtain the result. Java services of manufacturer, owner and RV server must be running in background for the transfer of messages with the device and with each other. For every new



Test No	Test Name	Mode	Parameter Tested	RESULT
1	AES	Encryption	ALL_INVALID	PASS
2	AES	Encryption	CLEARTEXT_INVALID	PASS
3	AES	Encryption	CLEARTXT_LEN_INVALID	PASS
4	AES	Encryption	CIPHER_LEN_INVALID	PASS
5	AES	Encryption	BLOCK_SIZE_INVALID	PASS
6	AES	Encryption	IV_INVALID	PASS
7	AES	Encryption	KEY_INVALID	PASS
8	AES	Encryption	KEY_LENGTH_INVALID	PASS
9	AES	Encryption	ALL_VALID	PASS
10	AES	Encryption	WRAPPER	PASS
11	AES	Decryption	ALL_INVALID	PASS
12	AES	Decryption	CLEARTXT_LEN_INVALID	PASS
13	AES	Decryption	CIPHERTEXT_INVALID	PASS
14	AES	Decryption	CIPHER_LEN_INVALID	PASS
15	AES	Decryption	BLOCK_SIZE_INVALID	PASS
16	AES	Decryption	IV_INVALID	PASS
17	AES	Decryption	KEY_INVALID	PASS
18	AES	Decryption	KEY_LENGTH_INVALID	PASS
19	AES	Decryption	ALL_VALID	PASS

Figure 4.13: Log Results

patch of code, the testing is carried out. The implementation of tests takes around 2 hours.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In today's era billion of devices are connected to internet, which requires provisioned resulting in problem of time and security. In general, the process of on-boarding a device takes 20-50 minute, which involves co-ordination among installation technicians, network and security operations, that will configure the device and set the credentials to get the device on the network which can be costly and time consuming.
The technology Zero Touch Secure Device On-board enables IoT on-boarding in seconds, eliminating the time of provisioning. It goes through hard security checks between devices, servers/clouds, and through the full supply chain to end customers. It quickly attests the device and rendezvous the device to the owners IoT platform.
Validation with different configurations has been carried out to check the whole cycle and to check the functionality of the code development of unit test case suit has been carried out. Test case may fail because of the incorrect error handling in the source code, if the flow of the unit test case is not correct or if there is a bug in source code. Sometimes environment issues may also cause problem like if the package version are not as required. Once all the test cases are passed the validation part is done, making the the application more secure.
With the proper knowledge of the problems and solutions and little bit of experience a success-full release can be achieved.

5.2 Future Work

The technology can be used with any IoT device and can be deployed anywhere. It can be used in smart cities, it can on-board more senors for smart city management. More over it can also be used in smart parking, reducing installation time to enable smart remotely managed parking.

Bibliography

[1] P. Pande and A. R. Padwalkar, “Internet of things–a future of internet: A survey,” *International Journal of Advance Research in Computer Science and Management Studies*, vol. 2, no. 2, 2014.

[2] P. Matta, B. Pant, and M. Arora, “All you want to know about internet of things (iot),” in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, IEEE, 2017, pp. 1306–1311.

[3] P. Dudhe, N. Kadam, R. Hushangabade, and M. Deshmukh, “Internet of things (iot): An overview and its applications,” in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, IEEE, 2017, pp. 2650–2653.

[4] I. Bojanova, G. Hurlburt, and J. Voas, “Imagineering an internet of anything,” *Computer*, vol. 47, no. 6, pp. 72–77, 2014.

[5] S. Z. Hosain, *Reality check: 50b iot devices connected by 2020 – beyond the hype and into reality*, Last accessed 4 April 2020, 2016. [Online]. Available: https://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected-2020-beyond-hype-reality-tag10.

[6] T. Sherwood, *The importance of securing the internet of things*, Last accessed 12 March 2020, 2018. [Online]. Available: https://www.tatacommunications.com/blog/2018/10/the-importance-of-securing-the-internet-of-things/.

[7] M. N. Dhivya and M. S. Banupriya, “Network security with cryptography and steganography,”

[8] A. Eskicioglu and L. Litwin, “Cryptography,” *IEEE Potentials*, vol. 20, no. 1, pp. 36–38, 2001.

[9] B. Preneel, “Cryptographic hash functions,” *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 431–448, 1994.

[10] T. Moses and A. O. Mancini, *Method and system for notarizing digital signature data in a system employing cryptography based security*, US Patent 6,314,517, Nov. 2001.

[11] R. Minni, K. Sultania, S. Mishra, and D. R. Vincent, “An algorithm to enhance security in rsa,” in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, IEEE, 2013, pp. 1–4.

[12] S. Xenitellis, “The open–source pki book,” *Open CA Team*, pp. 34–35, 2000.

[13] A. A. Mishra, K. Surve, U. Patidar, and R. K. Rambola, “Effectiveness of confidentiality, integrity and availability in the security of cloud computing: A review,” in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, IEEE, 2018, pp. 1–5.

[14] M. Sabt, M. Achemlal, and A. Bouabdallah, “Trusted execution environment: What it is, and what it is not,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 57–64.

[15] *Op-tee documentation*, Last accessed 24 March 2020. [Online]. Available: https://optee.readthedocs.io/en/latest/general/about.html#op-tee-components.

[16] R. Cummings, *Restful apis and 802.1qcc*, Last accessed 7 March 2020, 2016. [Online]. Available: http://www.ieee802.org/1/files/public/docs2016/cc-cummings-REST-0516-v00.pdf.

[17] Intel, *Iot device provosioning*, Last accessed 25 March 2020. [Online]. Available: https://www.intel.in/content/www/in/en/internet-of-things/secure-device-onboard.html.

[18] *Raspberry pi 3 model b specs*, Last accessed 3 May 2020. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.