**Specification and Test Plan Development**

It is important to develop a specification and test plan, and to identify the minimum viable product (**MVP**), prior to initiating product code development. Often, some experimentation is required ahead of these tasks, to determine the viability of certain architectural approaches or the use of new peripherals. However, the specification, test plan, and MVP should be used to guide code development and to help provide metrics of progress.

A test plan is based on the specification and should include the following:

* A requirement or function
* Details on how that requirement or function will be tested
* The acceptable results for the test
* The actual results for the test
* A pass/fail indication for the test

**The TCP/IP Network Stack & Protocols**

There are five layers to the **TCP/IP (Transmission Control Protocol/Internet Protocol)** network stack. Each layer takes the input of the layer above it and embeds that information into the **Protocol Data Units (PDUs)** of that layer, which is an atomic unit of data for that layer.

**Layer 5 – Application**

Protocols: DHCP, DNS, HTTP, MQTT,etc. PDU: Data

**Layer 4 – Transport**

Protocol: TCP (Transmission Control Protocol) PDU: Segment

Protocol: UDP (User Datagram Protocol) PDU: Datagram

**Layer 3 – Network**

Protocols: IP (Internet Protocol) PDU: Packet

**Layer 2 – Data-Link**

Protocols: 802.11 MAC PDU: Frame

**Layer 1 - Physical**

Protocols: 802.11a,b,g,n,ac PDU: Bits

**Checksum, CRC (Cyclic Redundancy Check)**

Error detection methods. Checksums can remain the same even if several bytes have been corrupted, whereas CRC’s are less prone to this weakness.

**ARP (Address Resolution Protocol)**

Provides the MAC address for a given IP address

**DCHP (Dynamic Host Configuration Protocol)**

A DHCP server dynamically assigns an IP address to a device on the network

**DNS (Domain Name System)**

Resolves a domain name (“sandiego.edu”) to an IP address

**HTTP (Hypertext Transfer Protocol)**

Provides a standard way for web browsers and servers to communicate

**MQTT (MQ Telemetry Protocol)**

Simple, lightweight messaging protocol designed for IoT devices with constrained resources

From the mqtt.org FAQ:

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

**NTP (Network Time Protocol)**

Synchronizes the time of a device on the network to a reference time server

**SSID (Service Set Identifier)**

Primary name associated with a wireless network

**WiFi Encryption Standards**

* Open – no encryption
* WEP (Wired Equivalent Privacy) – uses a password, not very secure
* WPA (WiFi Protected Access) – uses a password, was compromised
* WPA2 (WiFi Protected Access 2) – uses a password, WPA2 is the common standard now

**Sockets**

A TCP/IP **socket** is a reliable connection between a client and a server on a network.

A socket is specified by two pairs of values, one for the client and one for the server:

* **IP Address:** 192.168.16.1 (4 bytes or 32 bits)
* **Port**: 0 – 65535 (16 bits)

These are combined in the format: 192.168.16.1:1883.

To establish a socket connection, the client performs the following steps …

* **Create** a socket
* **Bind** the socket to a local port
* **Connect** to the server

… and the server performs the following steps:

* **Create** a socket
* **Listen** on a specific port
* **Accept** connections from clients

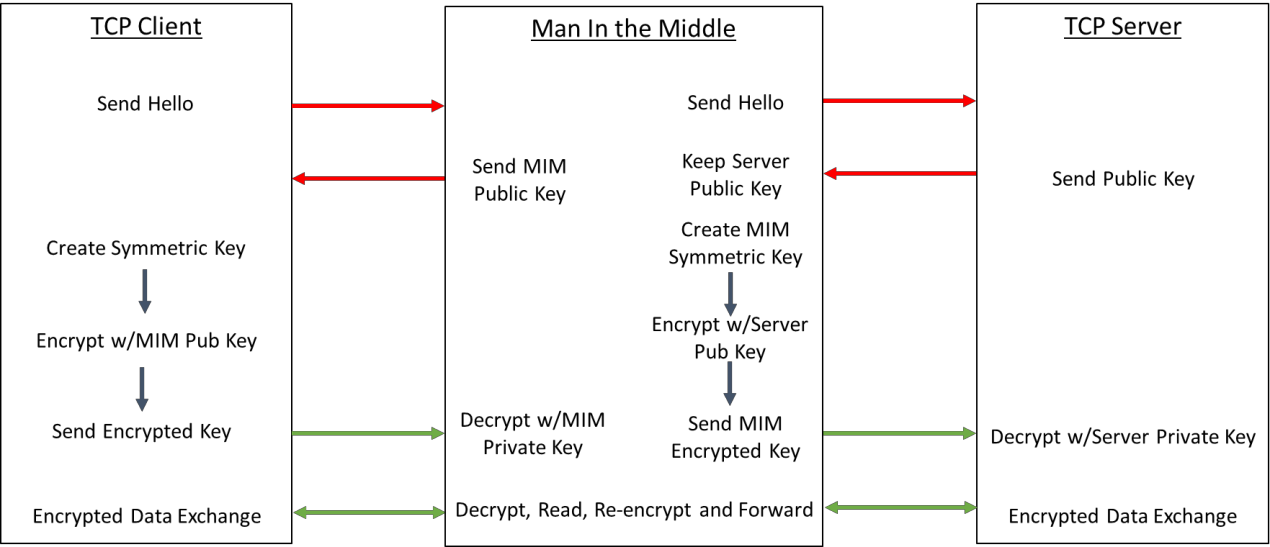
Communication takes the form of a stream in which you write and read a series of bytes.

TCP/IP sockets are reliable, but they are not encrypted. You are familiar with seeing *https* when you browse a secure web site – the *s* stands for **secure**. To provide secure communications requires **encryption** of the data. There are two types of encryption schemes:

* **Symmetric**: Both sides use the same key. This is very fast and secure, but it involves distributing keys.
* **Asymmetric**: This is also known as **Public Key Encryption**. It involves two keys that are derived mathematically at the same time – the **public key** and the **private key**. You hand out your public key to anyone who wants to send you a message and they encrypt the message with your public key, then you can decrypt it with your private key. The private key cannot be ascertained from the public key, but asymmetric encryption is slow.

The way to get the best of both worlds is to use asymmetric encryption to pass a private symmetric key, which is then used for the rest of the communication. This provides secure communication against someone sniffing the communication channel, but it is still prone to a…

**Man in the Middle (MITM) Attack**



**JSON – JavaScript Object Notation**

From the Cypress WICED WW101 Lab Manual:

JSON is an open-standard format that uses human-readable text to transmit data. It is the de facto standard for communicating data to/from the cloud. JSON supports the following data types:

* Double precision floating point
* Strings
* Boolean (true or false) and null
* Arrays (use "[]" to specify the array with values separated by ",")
* Key/Value (keymap) pairs as "key" : value (use "{}" to specify the keymap) with "," separating the pairs

Key/Value pair values can be arrays or can be other key/value pairs

Arrays can hold Key/Value pairs

For example, a legal JSON file looks like this:

{  
 "name" : "alan",  
 "age" : 49,  
 "parent" : true,  
 "children" : ["Anna","Nicholas"],  
 "address" :  
 {  
 "number" : 201,  
 "street" : "East Main Street",  
 "city" : "Lexington",  
 "state" : "Kentucky",  
 "zipcode" : 40507   
 }  
}

Note that carriage returns and spaces (except within the strings themselves) don't matter. For example, the above JSON code could be written as:

{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],  
"address":{"number":201,"street":"East Main Street","city":"Lexington","state":"Kentucky",  
"zipcode":40507}}

While this is more difficult for a person to read, it is easier to create such a string in the firmware when you need to send JSON documents.

Unfortunately, quotes mean something to the C compiler so if you are including a JSON string inside a C program you need to escape the quotes that are inside the JSON with a backslash (\). The above JSON would be represented like this inside a C program:

{\"name\":\"alan\",\"age\":49,\"badass\":true,\"children\":[\"Anna\",\"Nicholas\"],  
\"address\":{\"number\":201,\"street\": \"East Main Street\",\"city\":\"Lexington\",  
\"state\":\"Kentucky\",\"zipcode\":40507}}

JSON is described fully and succinctly at [www.json.org](http://www.json.org).

**Class Notes**