Introduction

This document introduces the smart plug firmware design. It explains smart plug firmware architecture, function blocks, source project structure, and some main APIs.

Features

- MCU + Connectivity + Security + Sensing
- Comprehensive system solution working with an Android™ app
- ATSAMW25 – Fully-integrated MCU + IEEE® 802.11 b/g/n Wi-Fi® solution
- ATECC508A – Elliptic Curve Cryptography (ECC) based crypto device
- ATM90E26 – High-performance wide-span metering AFEs
- US, CN, and EU plug standard compliant

Figure 1. Atmel Smart Plug – US, CN, and EU Standard
# Table of Contents

1 Overview............................................................................................................. 4  
1.1 Smart Plug Block Diagram ........................................................................... 4  
1.2 Smart Plug Functions ................................................................................... 5  

2 Development Tools.......................................................................................... 6  

3 Firmware Architecture...................................................................................... 7  
3.1 Wi-Fi Connectivity ...................................................................................... 7  
3.2 Security .......................................................................................................... 8  
3.2.1 Hardware Crypto Engine ........................................................................... 8  
3.2.2 Software Crypto Library .......................................................................... 8  
3.3 Sensor Control ............................................................................................... 8  
3.4 QTouch PTC Library .................................................................................... 8  

4 Function Blocks................................................................................................9  
4.1 Wi-Fi Function .............................................................................................. 9  
4.1.1 Wi-Fi Provision ......................................................................................... 9  
4.1.2 Node Discovery ......................................................................................... 9  
4.2 Security Features ........................................................................................... 9  
4.2.1 Security Terminology ............................................................................. 10  
4.2.2 Main Security Flow .................................................................................. 10  
4.2.2.1 Authenticate Smart Plug ..................................................................... 10  
4.2.2.2 Authenticate User .............................................................................. 12  
4.2.2.3 Encrypted Communication ................................................................. 13  
4.2.2.4 Integrity Check .................................................................................... 13  
4.2.3 Smart Plug Sharing .................................................................................. 13  
4.3 Communication Protocol ............................................................................ 15  
4.3.1 Smart Plug Commands ........................................................................... 15  
4.3.1.1 Network Entry Command .................................................................. 15  
4.3.1.2 Node Authentication Command ............................................................ 16  
4.3.1.3 Query Command ................................................................................ 17  
4.3.1.4 Control Command ............................................................................. 18  
4.3.1.5 Report Command ............................................................................... 18  
4.3.2 Smart Plug Function Cluster .................................................................. 20  
4.3.2.1 On/Off Cluster .................................................................................. 20  
4.3.2.2 Device Temperature Cluster ................................................................. 20  
4.3.2.3 Time/Calendar Cluster ....................................................................... 21  
4.3.2.4 Schedule Cluster ............................................................................... 21  
4.3.2.5 Historic Log Cluster ......................................................................... 22  
4.3.2.6 System Status Cluster ....................................................................... 23  
4.3.2.7 MAC Address Cluster ....................................................................... 23  
4.3.2.8 Product Information Cluster ................................................................. 24  
4.3.2.9 Energy Measurement Cluster ............................................................... 24  
4.3.2.10 Plug Socket Standard Cluster ............................................................... 25  
4.4 Sensor Reading .............................................................................................. 25  
4.4.1 Internal Temperature Sensor ................................................................ 25  
4.4.2 Energy Measurement ............................................................................. 25  

4.5 User Interface ............................................................................................... 25  
4.5.1 Touch Button ........................................................................................... 25
4.5.2 LED Indicator................................................................. 25
4.6 OTAU .............................................................................. 26
  4.6.1 MCU Memory Map ...................................................... 26
  4.6.2 OTAU Flow ................................................................. 27
  4.6.3 Bootloader ................................................................. 27

5 Source Project Structure .................................................... 28

6 Main API Introduction ....................................................... 30
  6.1 Wi-Fi Connectivity API .................................................. 30
  6.2 Security APIs .................................................................. 30
  6.3 Communication Protocol APIs ................................. 30
  6.4 Sensor Reading APIs ................................................. 31
  6.5 UI APIs ...................................................................... 31

7 Memory Footprint ............................................................. 33

8 Revision History ............................................................... 34
1 Overview

The smart plug showcases a typical Internet of Things (IoT) application by integrating an Atmel® MCU, Wi-Fi connectivity, security as well as sensing devices. Besides the smart plug hardware, Atmel also provides an Android app to work with it.

Here is the smart plug work flow which is quite common for any IoT system designs.

**Figure 1-1. Smart Plug Work Flow**

As shown in Figure 1-1, there two working stages. The first one is adding new plug stage, which is also known as provisioning stage. The second one is data exchange stage, in which the plug has been added into existing Wi-Fi network and can be operated via the Android app.

1.1 Smart Plug Block Diagram

Smart plug usually works as an edge node in a typical IoT scenario. The block diagram of the smart plug is shown in Figure 1-2.

**Figure 1-2. Smart Plug Block Diagram**
The key part of this reference design is the ATSAMW25-MR510PB – a fully-integrated MCU + IEEE 802.11 b/g/n Wi-Fi module. In this compact-sized module, Atmel provides a value added system solution with industry leading MCU (ATSAMD21G18) + Connectivity (ATWINC1500) + Security (ATECC508A) devices.

Besides the module, a hardware metering chip (ATM90E26) is used to measure the electrical output by the plug. Functions like the LED indicators, touch button, output control and temperature sensor are fully taken care of by the MCU embedded in the module.

For more details about the hardware design, refer to Atmel AT16225: Atmel Smart Plug Hardware User Guide.

1.2 Smart Plug Functions

The smart plug has output On/Off control as well as status LEDs. Furthermore the Atmel smart plug also showcases some more advanced functions:

- On/Off control by Android app
- Connection status indication (online/offline)
- Touch button control
- Programmable 7-day/week schedules (day/hour/minute)
- Energy/Voltage/Current/Power measurement
- Device temperature monitor and alarm
- Historic record of output energy and output state

As an IoT application, the following functions are quite necessary and seamlessly implemented on the Atmel smart plug:

- One-step Wi-Fi configuration to add new smart plug
- Share smart plug to multiple users
- Smart Plug firmware Over-The-Air-Update (OTAU)
- Elliptic Curve Cryptography (ECC) based security

For instructions on using the above features (both hardware usage as well and the Android app), refer to Atmel AT15736: Atmel Smart Plug Getting Started Guide.
2 Development Tools

To download or debug the firmware, the following development toolchain is used:

- **Atmel Studio 7.** Version: 7.0.634 - or above.
- **Atmel Software Framework.** Version: 3.29.0 or above.
- Programming and debugging tool: **Atmel SAM-ICE™.**


⚠️ **WARNING**  
Atmel smart plug is designed as a real product reference design. To avoid electrical hazard, open the case when plug is powered on by AC input is strictly prohibited. Make sure you understand the connections are safe before connecting any development tools to the plug.
3 Firmware Architecture

In the smart plug reference design, the main program runs in the ATSAMD21G18 MCU (embedded inside ATSAMW25 module). The main function blocks handled by MCU are:

- Wi-Fi connectivity – ATWINC1500 driver
- Security – ATECC508A driver and software AES
- Communication protocol and logical control
- Sensor control:
  - Energy measurement – ATM90E26 driver
  - MCU internal temperature sensor
- Touch Button - QTouch® PTC library
- LED indicator – Wi-Fi status and On/Off status

As the firmware does not use an RTOS, all these function blocks are running in an endless loop after the initialization functions have been called. Depending on different requirement, the function blocks are called at predefined intervals.

The application layer timer is based on the ARM® Cortex®-M0+ SysTick timer. It's initialized to generate a 1ms interrupt and a millisecond tick is incremented in its interrupt handler. The other application layer timers are derived from this tick.

The low level drivers are based on Atmel Software Framework (ASF).

Figure 3-1. Smart Plug Firmware Structure

3.1 Wi-Fi Connectivity

The smart plug Wi-Fi function is based on the Atmel industry-leading low-power 2.4GHz IEEE 802.11 b/g/n Wi-Fi ATWINC1500 SoC (embedded inside the ATSAMW25 module). It provides an integrated software solution with application and network services (integrated TCP/IP stack).

The Wi-Fi module firmware and driver are part of ASF. The Wi-Fi firmware and driver version used in the smart plug is 19.4.4.

For more details about the Wi-Fi firmware development, refer to Software Programming Guide for SAM W25 Xplained Pro and ATWINC1500 Wi-Fi Network Controller - Software Design Guide.
3.2 Security

Security is an integrated part of the smart plug reference design, as it is a key design factor for any IoT application. Both a hardware crypto device and software crypto library are used to achieve the desired security features. The same security functions are also implemented on the Android app side to allow them to communicate with each other.

3.2.1 Hardware Crypto Engine

Atmel CryptoAuthentication device ATECC508A is used to perform high-speed public key (PKI) algorithms: ECDH (Elliptic Curve Diffie-Hellman) key agreement and ECDSA (Elliptic Curve Digital Signature Algorithm). This offloads many heavy operations from the MCU, leaving more cycles for the application. The ATECC508A crypto device also have a secure hardware-based cryptographic key storage and cryptographic countermeasures which are more secure than software-based key storage. For more information about ATECC508A driver support, refer to http://www.atmel.com/tools/CryptoAuthLib.aspx.

Also note that ATECC508A should be provisioned and/or personalized in production before it’s used in a final product. For more detail, refer to the application notes below:

- ATECC Production Provisioning Guide
- ATSHA204A and ATECC508A Personalization Guide

Note: The SN of ATECC508A in SAMW25 module is different with normal commercial devices. Contact Atmel technical support channel for more information.

3.2.2 Software Crypto Library

Besides the hardware crypto engine, a software crypto library is also included to perform the AES algorithm. In the smart plug reference design, AES-128 CBC mode is used to encrypt and decrypt the data. A third party library LibTomCrypt from http://www.libtom.net/ is used for this purpose.

3.3 Sensor Control

The smart plug includes two sensors: one is an energy measurement sensor (ATME90E26), while the other is the internal temperature sensor of the SAM D21 MCU.

The ATM90E26 is a high-performance energy metering device that is accessed by the MCU via a UART interface. It is used to measure real time output current, voltage, power and accumulated electrical energy output by the smart plug. Check more about ATM90E26 usage in application note: http://www.atmel.com/Images/Atmel-46102-SE-M90E26-ApplicationNote.pdf.

The temperature sensor is integrated with MCU ADC peripheral. It’s easy to read the ADC result and calculate the temperature. Read more about this temperature sensor in ATSAMD21G18 datasheet.

3.4 QTouch PTC Library

The MCU (ATSAMD21G18) embedded in the ATSAMW25 module is equipped with a PTC (peripheral touch controller) module. Due to the autonomous operation, the PTC uses very little CPU resources and provides high quality touch performance. Atmel provides the QTouch PTC library to support code development, making it easy to add a touch button to smart plug.

For more about touch application based on the PTC, refer to:


The QTouch Library is part of Atmel Software Framework.
4 Function Blocks

In this chapter, the main function blocks of smart plug are explained in detail. For instruction on how to use the smart plug with the Android app, refer to Atmel AT15736: Atmel Smart Plug Getting Started Guide.

4.1 Wi-Fi Function

4.1.1 Wi-Fi Provision

By default, a new smart plug is set to SoftAP mode from factory. In order to add it to the user’s Wi-Fi network, the user needs to provision it via the Android app. Since it's a SoftAP, any smart phone with Wi-Fi connectivity can connect to the smart plug just like connecting to a regular Wi-Fi router.

Default smart plug SoftAP mode configuration,
- SSID: AtmelSmartPlug + SerialNumberLast3Bytes. Example: AtmelSmartPlug000001
- Security type: OPEN (No encryption)
- TCP server on Smart Plug
  - IP address: 192.168.xxx.1, where xxx is random number between 0~255.
  - Port number: 8899

After the Android app has connected to the smart plug (SoftAP mode), the phone will be assigned an IP address by the smart plug DHCP server. After a successful connection the Android app and the smart plug will go through an authentication process. After a successful authentication the Android app will send a “Wi-Fi Node Discovery” command to the smart plug. The smart plug responds with a “Wi-Fi Node Discovery Response” command. In the last step the Android app will send the Wi-Fi network information (SSID/password/security type) of the Wi-Fi network to connect to. If the provision procedure succeeds, the smart plug will switch to STA mode and connect to the target Wi-Fi network. The SSID, password, and security type will be stored in the MCU non-volatile memory (NVM) for future connections.

As the Wi-Fi provision flow is tightly coupled with the crypto process, refer to Figure 4-1 to understand the provision steps in detail.

4.1.2 Node Discovery

Once a plug has been provisioned successfully, it works in STA mode and will always try to connect to the target Wi-Fi network. After a power-on-reset (POR), the smart plug will read the SSID, password, and security settings from NVM and try to connect to the stored Wi-Fi network. To make the smart plug discoverable by the Android app, the smart plug sends a UDP broadcast packet at predefined intervals. When the Android app receives the broadcast packet it sends the “Wi-Fi Node Discovery” command to the smart plug who sent the broadcast packet. The smart plug responds with a “Wi-Fi Node Discovery Response” command. The port numbers used during node discovery are:
- UDP broadcast destination port: 8997
- TCP server port number on smart plug: 8899

If the smart plug passes the authentication process, the smart plug will be shown as online in the Android app and the plug is discovered.

The node discovery flow also involves crypto steps, refer to Figure 4-2 for details.

4.2 Security Features

IoT is the new frontier of technology. As any new frontier, the IoT also introduces vulnerabilities throughout the network. So designers should take care of all the security features demanded by system design. In the smart plug reference design, the following security features are included:

- Function Blocks
- Security Features
- Node Discovery
- Wi-Fi Provision
- Wi-Fi Function
• ECDSA for Android app and smart plug authentication
• ECDH for session key generation without ever transmitting the secret key outside the secure key storage
• AES-128 CBC mode for data encryption and decryption of communication on the network
• SHA-256 based MAC (Message Authentication Code) for data integrity check

Besides the above mentioned crypto algorithms, the logical flow of the security used in the smart plug is also important.

4.2.1 Security Terminology

In the security flows listed in the next section, the following security terminologies are used:

• Pub-d = Device Public Key
• Priv-d = Device Private Key
• Sig-d = Device Signature by Signer (Priv-s)
• Pub-s = Signer Public Key
• Priv-s = Signer Private Key
• Sig-s = Signer Signature by Root (Priv-root)
• Pub-root = Root Public Key from Root Authority
• randNum n = Random Challenge from Android app or smart plug
• authSig = Signature of Random Number used for Authentication
• Pub-host = Host Public Key from Android
• Priv-host = Host Private Key from Android
• Pub-x = ECDH eXchange Public Key
• Priv-x = ECDH eXchange Private Key
• preMasterKey = Result of ECDH
• IV = Initialization Vector used for Encryption
• DevInfo = Smart Plug device information
• DevMac = MAC of Smart Plug device information used for integrity
• DevData = Smart Plug data
• DevDataMac = MAC of Smart Plug data used for integrity
• MAC = Message Authentication Code

4.2.2 Main Security Flow

From the application point of view, there are two main security flows in the smart plug reference design. The first is the flow when adding a new smart plug as shown in Figure 4-1. The second is shown in Figure 4-2 and is used in the data exchange stage after the smart plug has been added to an existing Wi-Fi network and works in STA mode. The flows cover the three pillars of security (“CIA”) – Confidentiality, Integrity, and Authenticity. They are further divided into the following function blocks:

• Android app authenticates the new smart plug when adding a new smart plug
• Smart plug authenticates the user (Android app) whenever the Android app tries to discover the smart plug
• Encrypted communication between Android app and smart plug
• Integrity check of each packet

4.2.2.1 Authenticate Smart Plug

As shown in Figure 4-1 (marked by dotted line “Authenticity”), Android app will check the authenticity of the smart plug by verifying its key/signature pair and challenge it. The smart plug will first send its key/signature
pair (Pub-d + Sig-d and Pub-s + Sig-s) to the Android app. To make sure the smart plug public key is real, the Android app tries to verify Pub-d and its signature Sig-d with Pub-s and also verify Pub-s and its signature Sig-s with Pub-root.

If both verify steps pass, the Android app will challenge the smart plug with a random number which the smart plug will sign with its private key (Priv-d) which is securely stored in the ATECC508A and return its signature (authSig) to the Android app. To make sure the smart plug private key is real the Android app verifies the random challenge signature again. If all these steps pass, the smart plug is proven to be real (both public and private keys are verified) and the Android app can send its public key to the smart plug for further communication.

**Figure 4-1. Adding New Plug**
4.2.2.2 Authenticate User

Whenever the Android app tries to discover a smart plug, it will send the “Wi-Fi Node Discovery” command to the smart plug. As shown in Figure 4-2 (marked by dotted line “Authenticity”), smart plug will reply with its mac address and a random number challenge. The mac address is used as an ID for the smart plug. If the ID can be found in Android app’s device list, it will then sign the random challenge and return it to smart plug for user authenticity check. If the authenticity check passes, the smart plug will send an encrypted “Wi-Fi Node Discovery Response” command to the Android app and the smart plug will be displayed in the Android app.

Figure 4-2. Data Communication

1) The Root Public Key (Pub-root) has been loaded onto the APP

2) The APP has an ECC key pair (Priv-host & Pub-host)

Both APP and Smart Plug connect to the same Router

Wi-Fi Node Discovery

Random Challenge with node MAC addr: (randNum2) + MAC addr
authSig = Sign_sha2-randNum2, Priv-host

Random Challenge Resp with host/user random: authSig + host/user random

Verify user - ECDSA_sha2-randNum2, authSig, Pub-host

Read Pub-x

preMasterKey = ECDH(Priv-x, Pub-host)

sessionKey = MAC(randNum2, preMasterKey)

IV = Random()

Collect data: DevData

DevDataValid = DevData + DevDataMac

DevDataEnc = AES(DevDataValid, sessionKey, IV)

DevDataPacket = DevDataEnc + IV + Pub-x

Reply with DevDataPacket

DevMacCalc = MAC(DevData, preMasterKey)

 Decrypt = AES(DevDataEnc, sessionKey, IV)

DevMacCalc = MAC(DevData, preMasterKey)

DevMacCalc = DevDataMac — Show device in APP

Session established

New session if previous connection aborted

Wi-Fi Node Discovery

Send Random Challenge: (randNum3)

authSig = Sign_sha2-randNum3, Priv-host

Random Challenge Response: authSig + host/user random

Confidentiality

Integrity

Authenticity

Not verified
Drop request

Encrypted Wi-Fi Node Discovery resp
4.2.2.3 Encrypted Communication

After the smart plug and the user (Android app) authenticate each other, the encrypted information can be exchanged between them as shown in Figure 4-1 and Figure 4-2 (marked by the dotted line “Confidentiality”). The smart plug will reply with its mac address and a random number challenge. Because the Android app public key has been sent to the smart plug during adding new smart plug stage, ATECC508A will generate an ECDH pre-master key and randomize the key using the MAC algorithm to output a session key. This session key will be used as the AES key to encrypt the smart plug device information transmitted to the Android app in reply.

When the Android app receives the smart plug ECDH public key, it runs the same ECDH and MAC algorithm to extract the same AES session key to decrypt the data received. After integrity check, the real data can be recovered.

During adding new smart plug stage, Android app will send encrypted Wi-Fi network information to smart plug. And smart plug can get these data to join target Wi-Fi network after decrypting the message. The session key is different from session to session, as new random number is used to generate session key via MAC algorithm for each new session.

4.2.2.4 Integrity Check

As mentioned in Section 4.2.2.3, there will be MAC calculation of each data packet. As shown in Figure 4-1 and Figure 4-2 (marked by the dotted line “Integrity”), the calculated MAC will be compared to the received MAC. If the two match the integrity check is passed.

4.2.3 Smart Plug Sharing

In this reference design, the smart plug can be shared with other users (Android app). The sharing function involves both the smart plug and two or more Android apps. The Android app which first adds the smart plug will be the original user. The Android app on the other phones will be considered as new users. Basically, the new user gets a secret message from the original user by scanning a QR code and tries to establish a new secure session with the plug. In this new session, the smart plug and the new user will exchange information and keep each other’s information for future communication. The smart plug sharing flow is shown in Figure 4-3 and Figure 4-4.

**Figure 4-3. Smart Plug Sharing Between Android Apps**

1) The Root Public Key (Pub-root) has been loaded onto the APP
2) The APP has an ECC key pair (Priv-host & Pub-host)
3) The New User APP has an ECC key pair (Priv-host-new & Pub-host-new)

APP

Sharing node to new user

- Generate Random Number: ranNum
- authSig = Sign(sha2-ranNum, Priv-host)
- Share info. between APP

New User APP

Smart Plug

- Share ranNum
- Save ranNum
- Share ranNum Resp

-New User APP scan this QR code

Read ranNum

Read authSig

Extract Sharing info.
The shared smart plug discovery is quite similar to the flows in Section 4.2.2 except the new user (Android app) needs to send the Request Access command to finalize the smart plug sharing function (Figure 4-4). After this initial access request the discovery flow is similar to the security flow shown in Figure 4-2.

**Figure 4-4. Shared Smart Plug Discovery**

Besides sharing a smart plug, it’s also necessary to be able to remove a shared smart plug in case the original user doesn’t want to share it anymore. The function of removing a shared smart plug is provided in this reference design. As shown in Figure 4-5, the original user can send a Delete User Info command for this purpose.

**Figure 4-5. Remove Shared Smart Plug**
4.3 Communication Protocol
As this reference design is part of Atmel IoT system solution and mainly works in a WLAN, the communication protocol will be focused on this use case. Little endian is used in the communication protocol.

4.3.1 Smart Plug Commands
The plain text command format is shown below (before secure connection):

Table 4-1. Command Frame Format

<table>
<thead>
<tr>
<th>SOF</th>
<th>Data length</th>
<th>Seq. Num</th>
<th>CMD ID</th>
<th>Reserved</th>
<th>Command Payload</th>
<th>CRC32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint8</td>
<td>Uint16</td>
<td>Uint8</td>
<td>Uint8</td>
<td>Uint32</td>
<td>Variable</td>
<td>Uint32</td>
</tr>
<tr>
<td>0x5A</td>
<td>Packet length</td>
<td></td>
<td></td>
<td></td>
<td>Defined by specific commands</td>
<td>Include SOF</td>
</tr>
</tbody>
</table>

- SOF: 0x5A
- Data length: Packet length from Seq. Num. to Command Payload. Not including the CRC32.
- Seq. Num.: Command frame sequence number. Increase 1 for each frame. The same sequence number is replied in corresponding response frame. Starting from 0x00 to 0xFF.
- CMD ID: Command identifier
- Reserved: 32-bit reserved field
- CRC32: Checksum (CRC32: CCITT8023, Polynomial 0x04C11DB7) from SOF to end of Command payload

When secure communication is established, the command format will be changed as below.

Table 4-2. Encrypted Command Frame Format

<table>
<thead>
<tr>
<th>SOF</th>
<th>Data length</th>
<th>Encrypted data</th>
<th>Initialization Vector</th>
<th>ECDH Public Key</th>
<th>CRC32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uint8</td>
<td>Uint16</td>
<td>N x 16 bytes</td>
<td>16 bytes</td>
<td>64 bytes</td>
<td>Uint32</td>
</tr>
<tr>
<td>0x5B</td>
<td>Excluding CRC32</td>
<td>Data to be encrypted: Command frame in Table 4-1 + MAC</td>
<td>Node public key used for ECDH</td>
<td>Include SOF</td>
<td></td>
</tr>
</tbody>
</table>

- SOF: 0x5B (Encrypted Frame)
- Data length: Encrypted data length + Initialization vector length + ECDH public key length
- Initialization Vector: IV for AES
- ECDH public key: The smart plug public key used for ECDH
- CRC32: Checksum (CRC32: CCITT8023, Polynomial 0x04C11DB7) from SOF to ECDH public key
- During the “Add new smart plug” stage, the following commands are supported. For provision details, refer to Section 4.1.1.
- Network Entry: Used to discover and add new nodes
- Node Authentication: Used for authentication between Android app and nodes
- During the “Data Exchange” stage, the following commands are supported
- Query: Used to get attributes from a node
- Control: Used to change/control attributes of a node
- Report: Node reports attributes: events, alarm etc.

4.3.1.1 Network Entry Command
These commands are used when adding a new smart plug into a Wi-Fi network or discovering existing plugs.
Table 4-3. Network Entry Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi Node Discovery</td>
<td>0x01</td>
<td>“Atmel_WiFi_Discovery”</td>
<td>Android app sends this message to discover nodes</td>
</tr>
<tr>
<td></td>
<td>0x01</td>
<td>“CONFIG=WiFi info”</td>
<td>Android app unicasting Wi-Fi configuration to the node being provisioned</td>
</tr>
<tr>
<td></td>
<td>0x01</td>
<td>“CONDONE”</td>
<td>Android app unicasting this message to the node being provisioned before switching to SoftAP mode</td>
</tr>
</tbody>
</table>

Table 4-4. Network Entry Response Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi Node Discovery Response</td>
<td>0x81</td>
<td>Device_id(uint16) + Mac address(8 bytes)</td>
<td>Response to “Atmel_WiFi_Discovery”</td>
</tr>
<tr>
<td></td>
<td>0x81</td>
<td>“+ok\r\n\r\n”</td>
<td>ACK to “CONFIG=WiFi info”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ACK to “CONDONE”</td>
</tr>
</tbody>
</table>

4.3.1.2 Node Authentication Command

When authentication is needed. The following authentication commands are required before any other commands. These commands are used for mutual authentication between the Android app and the smart plug and smart plug sharing.

Table 4-5. Node Authentication Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Node Authentication</td>
<td>0x02</td>
<td>“AUTHREQ”</td>
<td>Request authentication packet</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
<td>“RAMCHAL” + random number</td>
<td>Send Random Challenge (random number length: 32 bytes)</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
<td>“RNCHMAC” + random number + MAC address</td>
<td>Send Random Challenge with the node MAC address (8 bytes)</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
<td>“PUBKEY” + public key</td>
<td>Send public key (64 bytes)</td>
</tr>
<tr>
<td></td>
<td>0x02</td>
<td>“RAMSHARE” + random number</td>
<td>Send Random Number (32 bytes) for node share</td>
</tr>
<tr>
<td>Request Access</td>
<td>0x02</td>
<td>“ACCREQ” + host rand + authSig + new Pub-host + new authSig</td>
<td>Request access from shared Android app (32 bytes + 64 bytes + 64 bytes + 64bytes)</td>
</tr>
<tr>
<td>Delete User Info</td>
<td>0x02</td>
<td>“DELSHARE” + random number</td>
<td>Deleted user info from shared node. (32 bytes)</td>
</tr>
</tbody>
</table>
Table 4-6. Node Authentication Response Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Node Authentication</td>
<td>0x82</td>
<td>&quot;AUTHREQ=&quot; + Pub-d + Sig-d + Pub-s + Sig-s</td>
<td>Response with authentication packet (64 bytes + 64 bytes + 64 bytes + 64 bytes)</td>
</tr>
<tr>
<td>Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x82</td>
<td>&quot;RAMCHAL=&quot; + Random challenge response</td>
<td>Random Challenge Response (64 bytes)</td>
</tr>
<tr>
<td></td>
<td>0x82</td>
<td>&quot;RANCRID=&quot; + Random challenge response + host/user random *</td>
<td>Random Challenge Response with host/user random number (64 bytes + 32 bytes)</td>
</tr>
<tr>
<td></td>
<td>0x82</td>
<td>&quot;+ok\r\n\r\n&quot;</td>
<td>ACK to “PUBKEY”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ACK to “RAMSHARE”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ACK to “DELSHARE”</td>
</tr>
</tbody>
</table>

Note: * Assigned as user ID during provisioning or node sharing. Check Figure 4-1 and Figure 4-3 for details.

4.3.1.3 Query Command

The supported query commands are listed in Table 4-7.

Table 4-7. Query Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query attribute</td>
<td>0x11</td>
<td>CID + Cluster index (n) + Attribute ID</td>
<td>After received this command, report specified attribute</td>
</tr>
<tr>
<td>Query cluster</td>
<td>0x12</td>
<td>CID + Cluster index (n)</td>
<td>After received this command, report all attributes under specified cluster</td>
</tr>
<tr>
<td>Query attribute</td>
<td>0x11</td>
<td>CID + Cluster index (n) + Attribute ID</td>
<td>After received this command, report specified attribute</td>
</tr>
</tbody>
</table>

Cluster index is the index number of clusters in a node if multiple clusters (the same cluster type) are supported on this single node. For example, a plug node type may support two On/Off clusters. Cluster 0 refers to the first On/Off cluster and Cluster 1 refers to the second On/Off cluster.

From the Report All command from node, the Android app will get the “Number of Clusters” of the same cluster type and the supported clusters (by CID) in the node. The Android app will then use this data to communicate with target node.
4.3.1.4 Control Command

The supported control commands are listed in Table 4-8. The OTAU commands are also listed here.

Table 4-8. Control Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control attribute</td>
<td>0x21</td>
<td>CID + Cluster index (n) + Attribute ID + attribute value</td>
<td>Change the specified attribute to the received attribute value</td>
</tr>
<tr>
<td>Control cluster</td>
<td>0x22</td>
<td>CID + Cluster index (n) + attribute values under cluster</td>
<td>Change all attributes in the specified cluster with received attribute values</td>
</tr>
<tr>
<td>OTAU MCU data frame</td>
<td>0x28</td>
<td>Sequence number (2 byte) + Max. 64 x 4 byte data for SAM D</td>
<td>Send OTAU data to device. The max data length is decided by the MCU row/page size.</td>
</tr>
<tr>
<td>OTAU state cmd</td>
<td>0x29</td>
<td>&quot;OTA_HOST&quot; &quot;CRC_HOST&quot; + CRC32 &quot;OTA_WIFI&quot; &quot;RUN_NEWF&quot;</td>
<td>Start host MCU OTAU Host MCU firmware CRC check Start Wi-Fi module OTAU Switch to new host MCU firmware</td>
</tr>
</tbody>
</table>

4.3.1.5 Report Command

Supported report commands are listed in Table 4-9. The OTAU response commands are also listed here.

Table 4-9. Report Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Command Identifier</th>
<th>Command parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report attribute</td>
<td>0x91</td>
<td>CID + Cluster index (n) + Attribute ID + attribute value</td>
<td>Report specified attribute. This command is sent in response to the “query attribute” and “control attribute” commands or used to report an attribute asynchronously.</td>
</tr>
<tr>
<td>Report cluster</td>
<td>0x92</td>
<td>CID + Cluster index (n) + attribute values</td>
<td>Report all attributes in specified cluster. This command is sent in response to the “query cluster” and “control cluster” commands or used to report clusters asynchronously.</td>
</tr>
<tr>
<td>OTAU MCU data frame</td>
<td>0xA8</td>
<td>&quot;+ok\r\n\r\n&quot;</td>
<td>ACK OTAU MCU data frame</td>
</tr>
<tr>
<td>OTAU state cmd</td>
<td>0xA9</td>
<td>&quot;+ok\r\n\r\n&quot;</td>
<td>ACK OTAU state cmd</td>
</tr>
</tbody>
</table>

Table 4-10 lists all available error codes. These error codes are reported to the Android app in case of wrong command, wrong command sequence, unsupported feature, or other failures.
### Table 4-10. Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0x00</td>
<td>Operation was successful</td>
</tr>
<tr>
<td>FAILURE</td>
<td>0x01</td>
<td>Operation was not successful</td>
</tr>
<tr>
<td>PROV_ERR</td>
<td>0x50</td>
<td>Wrong provision sequence</td>
</tr>
<tr>
<td>AUTH_ERR</td>
<td>0x58</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>SHARE_FAIL</td>
<td>0x59</td>
<td>Share node failed</td>
</tr>
<tr>
<td>OTAU_BUSY</td>
<td>0x60</td>
<td>OTAU is not ready</td>
</tr>
<tr>
<td>OTAU_SEQ_ERROR</td>
<td>0x61</td>
<td>Incorrect OTAU sequence</td>
</tr>
<tr>
<td>OTAU_WRITE_ERR</td>
<td>0x62</td>
<td>OTAU host programming error. OTAU should be restarted</td>
</tr>
<tr>
<td>OTAU_VERIFY_ERR</td>
<td>0x63</td>
<td>OTAU host verify error. OTAU should be restarted</td>
</tr>
<tr>
<td>OTAU_IMG_CRC_ERR</td>
<td>0x64</td>
<td>OTAU image file CRC check failed. OTAU should be restarted</td>
</tr>
<tr>
<td>NOT_AUTHORIZED</td>
<td>0x7E</td>
<td>The received command did not pass authentication check</td>
</tr>
<tr>
<td>UNSUPPORTED_CLUSTER</td>
<td>0x81</td>
<td>The specified cluster is not supported on the node</td>
</tr>
<tr>
<td>UNSUPPORTED_COMMAND</td>
<td>0x82</td>
<td>The specified command is not supported on the node</td>
</tr>
<tr>
<td>UNSUPPORTED_ATTRIBUTE</td>
<td>0x86</td>
<td>The specified attribute does not exist on the device</td>
</tr>
<tr>
<td>INVALID_VALUE</td>
<td>0x87</td>
<td>Attribute out of range, or set to a reserved value</td>
</tr>
<tr>
<td>READ_ONLY</td>
<td>0x88</td>
<td>Attempted to write a read-only attribute</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>0x94</td>
<td>The exchange was aborted due to timeout</td>
</tr>
</tbody>
</table>
4.3.2 Smart Plug Function Cluster

The smart plug is defined to be a plug node type with ID: 0x0600. And the plug node type has several function clusters as listed in Table 4-11.

Table 4-11. Function Clusters on Plug Node Type

<table>
<thead>
<tr>
<th>Cluster Supported</th>
<th>Cluster Identifier (CID)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On/Off</td>
<td>0x0001</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Device Temperature</td>
<td>0x0003</td>
<td>Optional</td>
</tr>
<tr>
<td>Time Calendar</td>
<td>0x0004</td>
<td>Optional</td>
</tr>
<tr>
<td>Schedule</td>
<td>0x0005</td>
<td>Optional</td>
</tr>
<tr>
<td>Historic Log</td>
<td>0x000A</td>
<td>Optional</td>
</tr>
<tr>
<td>System Status</td>
<td>0x000B</td>
<td>Optional</td>
</tr>
<tr>
<td>MAC Address</td>
<td>0x00FD</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Product Information</td>
<td>0x00FE</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Energy Measurement</td>
<td>0x0600</td>
<td>Optional</td>
</tr>
<tr>
<td>Plug socket standard</td>
<td>0x0601</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Each function cluster is a function entity. A device may have one or more function clusters of the same type. Here are the function cluster numbers in the smart plug.

```
#define ONOFF_CLST_NUM           1
#define DEVICE_TEMP_CLS_NUM      1
#define TIME_CALENDAR_CLST_NUM   1
#define SCHEDULE_CLST_NUM        10
#define HISTORIC_LOG_CLST_NUM    3
#define SYS_STATUS_CLST_NUM      1
#define ENERGY_MEAS_CLST_NUM     1
#define PLUG_TYPE_CLST_NUM       1
#define ALARM_CLST_NUM           1
```

Each function cluster contains different function attributes which are explained in the following sections. For more details about smart plug cluster and attribute definition, refer to the source code released along with this document.

4.3.2.1 On/Off Cluster

The On/Off cluster contains the following attributes.

Table 4-12. Attributes in On/Off Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>On/Off</td>
<td>Boolean</td>
<td>0x00/0x01</td>
<td>0x01</td>
<td>0 = Off, 1 = On</td>
<td>R/W</td>
</tr>
</tbody>
</table>

4.3.2.2 Device Temperature Cluster

The device temperature cluster contains the following attributes.
### Table 4-13. Attributes in Device Temperature Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Temperature</td>
<td>int16</td>
<td>-200~200</td>
<td>0x01</td>
<td>0xFF38::-200C 0x00C8:200C</td>
<td>R</td>
</tr>
<tr>
<td>Temperature Threshold Low</td>
<td>int16</td>
<td>-200~200</td>
<td>0x02</td>
<td>0xFF38::-200C 0x00C8:200C 0xF000: Threshold disabled</td>
<td>R/W</td>
</tr>
<tr>
<td>Temperature Threshold High</td>
<td>int16</td>
<td>-200~200</td>
<td>0x03</td>
<td>0xFF38::-200C 0x00C8:200C 0xF000: Threshold disabled</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Device temperature indicates the internal temperature of device. When the device temperature is above or below the set temperature thresholds, the device will generate an alarm.

### 4.3.2.3 Time/Calendar Cluster

The time/calendar cluster contains the following attributes.

### Table 4-14. Attributes in Time/Calendar Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Year</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x01</td>
<td>0x07DF: 2015</td>
<td>R/W</td>
</tr>
<tr>
<td>Weekday</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x02</td>
<td>0x01: Mon., … 0x07: Sun.</td>
<td>R/W</td>
</tr>
<tr>
<td>Time/Calendar Mask</td>
<td>Uint8</td>
<td>0x00~0x3F</td>
<td>0x03</td>
<td>bit[7][6][5][4][3][2][1][0] [Reserved] [Reserved] [year] [month] [date] [hour] [minute] [second] 0b00: disable, 0b01: enable</td>
<td>R</td>
</tr>
<tr>
<td>Time/Calendar</td>
<td>Uint32</td>
<td>0x00000000~0xFFFFFFFF</td>
<td>0x04</td>
<td>bit[31…0] [Year: 6bit</td>
<td>Month: 4bit</td>
</tr>
<tr>
<td>Running time</td>
<td>Uint32</td>
<td>0x00000000~0xFFFFFFFF</td>
<td>0x05</td>
<td>Unit: second</td>
<td>R</td>
</tr>
</tbody>
</table>

Note: 1. The Time/Calendar mask is used to mask supported time/calendar function on a node. If the bits in Time/Calendar mask are set to 0b00, the element is disabled. For example, 0b00011111 stands for “Year” is not supported.

### 4.3.2.4 Schedule Cluster

The schedule cluster contains the following attributes. The schedule cluster is defined to support a weekly schedule. For more complex schedule settings, it’s better to handle it in the cloud or in the Android app.

### Table 4-15. Attributes in Schedule Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduled Cluster ID</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x01</td>
<td>Target cluster ID (CID)</td>
<td>R/W</td>
</tr>
</tbody>
</table>
### 4.3.2.5 Historic Log Cluster

This cluster records a data log locally on the node. The data should be from clusters/attributes supported by the node.

#### Table 4-17. Historic Log Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log cluster ID</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x01</td>
<td>Cluster ID of the log</td>
<td>R</td>
</tr>
<tr>
<td>Log cluster index</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x02</td>
<td>Cluster index of the log</td>
<td>R</td>
</tr>
<tr>
<td>Log attribute ID</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x03</td>
<td>Attribute ID of the log</td>
<td>R</td>
</tr>
<tr>
<td>Number of logs</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x03</td>
<td>Number of historic logs</td>
<td>R</td>
</tr>
<tr>
<td>Log value list</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x04</td>
<td>The log value list of specified attribute value and its timestamp</td>
<td>R</td>
</tr>
</tbody>
</table>

As the log attribute value is different according to log cluster ID/log cluster index/log attribute ID, the “Type” of log attribute value may vary. The format of the log attribute value is show in Table 4-18.
Table 4-18. Log Value Format

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>As defined in Section 4.3.2.3</td>
<td>Time/Calendar attribute</td>
</tr>
<tr>
<td>Attribute Value</td>
<td>Decided by cluster ID/log</td>
<td>E.g. Onoff cluster index 0, attribute 1 set to off: 0(bool)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td></td>
<td></td>
<td>Timestamp (Node time)</td>
</tr>
<tr>
<td>Attribute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.2.6 System Status Cluster

The system status cluster contains the following attributes.

Table 4-19. Attributes in System Status Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of statuses</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x01</td>
<td>The number of system statuses supported by the node.</td>
</tr>
<tr>
<td>Status list 1…n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The system status cluster lists system errors or issues when the status code returned was not zero.

Table 4-20. Status List Format

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status code</td>
<td>Uint8</td>
<td>Status code as define in Table 4-21.</td>
</tr>
<tr>
<td>Priority</td>
<td>Uint8</td>
<td>The lower this number, the higher its priority.</td>
</tr>
</tbody>
</table>

Table 4-21. Status Code

<table>
<thead>
<tr>
<th>Status Name</th>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON/OFF hardware issue</td>
<td>0x01</td>
<td>The on/off control hardware is faulty.</td>
</tr>
</tbody>
</table>

4.3.2.7 MAC Address Cluster

The MAC Address cluster contains the following attributes.

Table 4-22. Attributes in MAC Address Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Connection Type</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x01</td>
<td>0x01: Wi-Fi 0x02: Bluetooth 0x03: ZigBee 0x04: Ethernet 0x05: Z-Wave 0x06: Power Line Communication</td>
</tr>
<tr>
<td>MAC Address</td>
<td>Uint64</td>
<td>0x0000000000000000~0xFFFFFFF</td>
<td>0x02</td>
<td>Wi-Fi MAC address (48bit) The 2 high bytes (16bit) are ignored</td>
</tr>
</tbody>
</table>
4.3.2.8 Product Information Cluster

The Product Information cluster contains the following attributes.

**Table 4-23. Attributes in Product Information Cluster**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Node Type</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x01</td>
<td>Product node type ID. For smart plug, this is 0x0600.</td>
<td>R</td>
</tr>
<tr>
<td>Manufacturer ID</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x02</td>
<td>0x0000: unknown</td>
<td>R</td>
</tr>
<tr>
<td>Firmware Version</td>
<td>Uint16</td>
<td>0x0000~0xFFFF</td>
<td>0x03</td>
<td>High byte: main version</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Low byte: sub-version</td>
<td></td>
</tr>
<tr>
<td>Serial Number Length</td>
<td>Uint8</td>
<td>0x00~0xFF</td>
<td>0x04</td>
<td>Unit: Byte</td>
<td>R</td>
</tr>
<tr>
<td>Serial Number</td>
<td>String</td>
<td>Defined by Serial Number Length above</td>
<td>0x05</td>
<td>Manufacturer defined unique serial number</td>
<td>R</td>
</tr>
</tbody>
</table>

4.3.2.9 Energy Measurement Cluster

The energy measurement cluster contains the following attributes.

**Table 4-24. Attributes in Energy Measurement Cluster**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Type</td>
<td>Uint8</td>
<td>0x01~0xFF</td>
<td>0x01</td>
<td>0x00: not specified</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: AC output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x02: DC output</td>
<td></td>
</tr>
<tr>
<td>Output Voltage</td>
<td>Uint8</td>
<td>0x00~0xF0</td>
<td>0x02</td>
<td>For AC output, RMS voltage</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit: V (AC), V/10 (DC)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFF: not supported</td>
<td></td>
</tr>
<tr>
<td>Output Current</td>
<td>Uint16</td>
<td>0x0000~0x7D00</td>
<td>0x03</td>
<td>For AC output, RMS current</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit: mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFFFF: not supported</td>
<td></td>
</tr>
<tr>
<td>Output Power</td>
<td>Uint16</td>
<td>0x0000~0xC350</td>
<td>0x04</td>
<td>Active power output</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit: 1/10 W</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFFFF: not supported</td>
<td></td>
</tr>
<tr>
<td>Active Energy</td>
<td>Uint16</td>
<td>0x0000~0x7D00</td>
<td>0x05</td>
<td>Accumulated active energy consumption after power on</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unit: 1/10 kWh</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFFFF: not supported</td>
<td></td>
</tr>
</tbody>
</table>
4.3.2.10 Plug Socket Standard Cluster

The plug socket standard cluster contains the following attributes. It specifies the standard plug and socket complying with, so it can be used in corresponding countries and areas.

Table 4-25. Attributes in Plug Socket Standard Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Range</th>
<th>Attribute ID</th>
<th>Description</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug Standard</td>
<td>Uint8</td>
<td>0x01~0xFF</td>
<td>0x01</td>
<td>0x00: not specified</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01: US</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x02: EU</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x03: China</td>
<td></td>
</tr>
</tbody>
</table>

4.4 Sensor Reading

4.4.1 Internal Temperature Sensor

The MCU internal temperature sensor is used to monitor the smart plug operating temperature. In case of a too high temperature, the output will be shut off to prevent potential damage.

4.4.2 Energy Measurement

The energy measurement is done by the Atmel metering AFE (Analog Front End) chip ATM90E26. The MCU interfaces the ATM90E26 via UART using the default settings of 9600 baud, no parity, and one stop bit. The following results are read out and displayed on the Android app every 2 seconds.

- Current – instant RMS current
- Voltage – instant RMS voltage
- Power – instant active power
- Electrical energy – accumulated electrical energy

For electrical energy, the ATM90E26 register will be cleared each time it’s read. The value must be accumulated in the MCU firmware to get the long term electrical energy data. To record historic electrical energy, the data is reported to the Android app every 2 hours.

To achieve better accuracy, the ATM90E26 is calibrated in factory and the calibration data is stored in the MCU. The calibration data is written back to the ATM90E26 during initialization before reading any measurement results from ATM90E26.

4.5 User Interface

4.5.1 Touch Button

Thanks to the PTC module embedded inside ATSAMD21G18, a touch button is added on smart plug. Each time the touch button is pressed, it will toggle the output. When pressing the touch button for ~5s, it will reset the smart plug to SoftAP state.

4.5.2 LED Indicator

There are five LEDs on the smart plug. Three of them are used by the Wi-Fi indicator, while the other two LEDs are used to indicate output state.

Wi-Fi indicator function:

- Yellow blink: Smart plug in SoftAP mode, no client connected
- Yellow: Smart plug in SoftAP mode, a Wi-Fi client is connected
- Green blink: Smart plug in STA mode, connecting to home Access Point (AP)
- Green: Smart plug in STA mode, connected to AP
- Red: Smart plug error

Smart plug output indicator function:
- Green: ON
- Red: OFF

The output status LED colors are the same as shown in the Android app. The LED blink interval is 500ms.

4.6 OTAU

For a typical IoT edge node, over the air upgrade (OTAU) is necessary and the Atmel smart plug includes this feature. The MCU (ATSAMD21G18) has an internal flash of 256KB, leaving enough space for a downloaded firmware image as well as keeping the original firmware unchanged. In this reference design, no external storage is used for OTAU.

4.6.1 MCU Memory Map

The MCU internal flash is divided into different sections as the memory map shown in Figure 4-6.

Figure 4-6. MCU Internal Flash Memory Map

In the smart plug reference design, the application firmware area is divided into two parts – Application 1 and Application 2. A bootloader as described in Section 4.6.3 is used to decide which application firmware to run after reset. At the last part of internal flash is an emulated EEPROM used to store user information, Wi-Fi network information (SSID/password/security type), and local historic records etc.
The two application areas start at 0x0004000 and 0x00020000 respectively. When generating a new application firmware, make sure the binary code size doesn’t go beyond 112KB (128KB half the internal flash size – 16KB Bootloader or EEPROM area).

4.6.2 OTAU Flow

OTAU function is enabled when smart plug is in STA mode. The new firmware image is stored in the Android app in advance and the binary will be sent from the Android app to the smart plug once the OTAU is triggered. OTAU commands are part of the control and report commands as listed in Table 4-8 and Table 4-9.

After a new image is successfully downloaded to the host MCU, the smart plug will do a warm reset to switch to the new firmware. In this reference design, the application firmware can only be upgraded - downgrading is not supported. The firmware version currently running can be checked from the Android app.

The OTAU flow is shown Figure 4-7.

Figure 4-7. OTAU Flow

4.6.3 Bootloader

As no external storage is used for image storage during the OTAU, the bootloader is quite simple. After reset, it will check the flags in the EEPROM and decide which application firmware should be started. The bootloader project files are included in the software package released with this document.
5  **Source Project Structure**

As shown in the Atmel Studio 7 screen shot in Figure 5-1, there are two projects included in this reference design. One is the smart plug (application firmware) project and the other is the bootloader project.

**Figure 5-1.  Source Projects**

The project structure of the bootloader is quite simple and the following part will focus on the smart plug project. Figure 5-2 shows the project structure of smart plug.

**Figure 5-2.  Smart Plug Project Structure**
The project is based on Atmel Software Framework. Besides the ASF folder and source files, there are several other folders and source files, listed in the order shown in Figure 5-2:

- atcalib: Atmel CryptoAuthentication support library
- Clusterlib: Header files of clusters and commands mentioned in communication protocol (Section 4.3)
- config: Project configuration files – including function block configuration files, e.g. Wi-Fi, Crypto chip, QTTouch library, and node setting etc.
- libtomcrypt: Third party crypto library. The AES CBC files are used.
- NodePlug: Plug node type function implementation
- RTT: SEGGER Real Time Transfer codes used for debug purpose. It can be removed in final project.

Source files:

- adc_sensor.c and adc_sensor.h: MCU internal temperature sensor reading
- button.c and button.h: QTTouch button initialization and handler
- ecc508a.c and ecc508a.h: The application code of ecc508a
- led.c and led.h: LED indicator initialization and control
- m90e26.c and m90e26.h: Metering AFE initialization and control
- main_samd21.c and main.h: The main file of smart plug
- winc1500.c and win1500.h: Wi-Fi application layer code

The above folders and files give a basic function division in the project. As some of the function blocks are tightly integrated, several function blocks may be mixed in application layer files.
6 Main API Introduction

In this chapter, the main APIs are introduced for better understanding of key function blocks. For more details about the function parameters and return values etc., refer to the comments in the source files.

6.1 Wi-Fi Connectivity API

These APIs can be found in winc1500.c.

- void configure_winc1500(void)
  Initialize winc1500 Wi-Fi SoC.
- uint8_t getWiFiMode(void)
  Get smart plug Wi-Fi working mode (STA or SoftAP).
- void wifiSwitchtoAP(void)
  Change Wi-Fi to SoftAP mode.
- void wifiSwitchtoSTA(void)
  Change Wi-Fi to STA mode.
- static void m2m_wifi_state_handler(uint8_t u8MsgType, void *pvMsg)
  Wi-Fi status update callback.
- static void m2m_wifi_socket_handler(SOCKET sock, uint8 u8Msg, void *pvMsg)
  Socket data and status callback.
- static void parse_cmd_frame(uint8_t session_num, uint8 *provbuffer)
  Parse received message to get commands for further handling.
- static void tcpsendresponse(SOCKET sock, tstrSocketRecvMsg *pstrRx)
  TCP socket response after receiving a message from the TCP socket.
- void winc1500_handler(void)
  Wi-Fi Routine tasks running at predefined interval.

6.2 Security APIs

The security APIs can be found in ecc508a.c:

- void configure_ecc508a(void)
  Initialize ATECC508A I2C interface.
- uint8_t cademo_sign_host_rand(uint8_t *challenge_sig, uint8_t *host_random);
  Sign the random number from Android app to generate its signature.
- uint8_t cademo_verify_challenge_resp(uint8_t* verified);
  Verify the random number challenge response from Android app.
- uint8_t cademo_enc_frame(encrypt_packet_t* msg_enc_report, uint8_t session_num);
  Encrypt a message to be sent to the Android app.
- uint8_t cademo_dec_frame(encrypt_packet_t* msg_enc_report, uint8_t session_num);
  Decrypt a message received from the Android app.

6.3 Communication Protocol APIs

The communication protocol is explained in Section 4.3. The APIs can be found in NodePlug\nodeplug.c:
• `void parse_iot_cmd(uint8_t session_num, uint8_t cmd, uint8_t *cmd_param, cmd_resp_t *cmd_resp);`
  Entry point to handle the commands defined in communication protocol.
• `void onoff_handler(cmd_param_hdr_t *param_hdr, uint8_t *ptr_value);`
  Callback function to handle on/off cluster from application layer.
• `void time_calendar_handler(cmd_param_hdr_t *param_hdr, uint8_t *ptr_value);`
  Callback function to handle time/calendar cluster from application layer.
• `iot_status_code_t report_XXX_cluster(cmd_param_hdr_t *param_hdr, cmd_resp_t *cmd_resp);`
  • `iot_status_code_t write_XXX_cluster(cmd_param_hdr_t *param_hdr, uint8_t *ptr_value);`
  Report and write command handlers. XXX refers to the function clusters defined in Section 4.3.
• `void node_plug_init(void);`
  Initialize node plug clusters.

6.4 Sensor Reading APIs
The internal temperature sensor APIs can be found in adc_sensor.c:
• `void configure_adc(void);`
  Initialize the ADC module to read internal sensor.
• `float get_internal_temperature(void);`
  Start ADC conversion to get ADC result and convert the ADC data to temperature.

The ATM90E26 sensor reading APIs can be found in m90e26.c:
• `void configure_m90e26(void);`
  Initialize UART interface and write calibration data to ATM90E26.
• `void read_data_m90e26(void);`
  Read measurement results from ATM90E26.

6.5 UI APIs
The UI includes QTouch button and LED indicator.
QTouch library is part of ASF and its APIs can be found in ASF. For library details, refer to Atmel QTouch Library User Guide. The application layer API is in button.c and main_samd21.c:
• `static void button_handler(buttons_state_code_t button_status)`
  In main_samd21.c. Handles the button action.
• `void timer_init(void)`
  Initializes the timers needed by the QTouch library.
• `void touch_button_handler(void)`
  Reads the touch sensor results using the QTouch library API and outputs the button status (short press or long press).
  LED indicator APIs can be found in LED.c. As the output indicator works together with the relay output, the APIs are for Wi-Fi indicators.
• `void set_wifi_led_mode(led_mode_t led_mode, led_index_t led_pin_num)`
  Set LED indicator working mode (on, off and toggle).
• `void exec_wifi_led_mode(void)`
  Refresh the LED indicator status at predefined interval.
7 Memory Footprint

Here is the smart plug project overall memory size generated in Atmel Studio 7 with GCC optimization option set to –o1.

- Program Memory Usage: 83244 bytes, 31.8% Full
- Data Memory Usage: 15632 bytes, 47.7% Full

Figure 7-1 and Figure 7-2 show the memory usage grouped by the smart plug function blocks.

Figure 7-1. Flash Memory Footprint

![Flash Memory Footprint Diagram]

Figure 7-2. SRAM Memory Footprint (Byte)

![SRAM Memory Footprint Diagram]
## Revision History

<table>
<thead>
<tr>
<th>Doc Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>